Computational Complexity of Evolutionary Algorithms, Hybridizations, and Swarm Intelligence

Dissertation

zur Erlangung des Grades eines
Doktors der Naturwissenschaften
der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Dirk Sudholt

Dortmund
2008

Tag der mündlichen Prüfung:	15. Dezember 2008
Dekan:	Prof. Dr. Peter Buchholz
Gutachter:	Juniorprof. Dr. Thomas Jansen
	Prof. Dr. Günter Rudolph

Dedicated to the memory of Ingo Wegener (1950–2008)

Summary

Bio-inspired randomized search heuristics such as evolutionary algorithms, hybridizations with local search, and swarm intelligence are very popular among practitioners as they can be applied in case the problem is not well understood or when there is not enough knowledge, time, or expertise to design problem-specific algorithms. Evolutionary algorithms simulate the natural evolution of species by iteratively applying evolutionary operators such as mutation, recombination, and selection to a set of solutions for a given problem. A recent trend is to hybridize evolutionary algorithms with local search to refine newly constructed solutions by hill climbing. Swarm intelligence comprises ant colony optimization as well as particle swarm optimization. These modern search paradigms rely on the collective intelligence of many single agents to find good solutions for the problem at hand. Many empirical studies demonstrate the usefulness of these heuristics for a large variety of problems, but a thorough understanding is still far away.

We regard these algorithms from the perspective of theoretical computer science and analyze the random time these heuristics need to optimize pseudo-Boolean problems. This is done in a mathematically rigorous sense, using tools known from the analysis of randomized algorithms, and it leads to asymptotic bounds on their computational complexity. This approach has been followed successfully for evolutionary algorithms, but the theory of hybrid algorithms and swarm intelligence is still in its very infancy. Our results shed light on the asymptotic performance of these heuristics, increase our understanding of their dynamic behavior, and contribute to a rigorous theoretical foundation of randomized search heuristics.

Acknowledgments

I would like to express my gratitude to several people that directly or indirectly contributed to this thesis. I thank Hans-Paul Schwefel and his former staff members for the opportunity to start my Ph. D. work at the chair *Systems Analysis/Algorithm Engineering* during the year 2005. I am indebted to my advisor Ingo Wegener for his steady support through many years and many lessons I could learn from him. He passed away in November 2008 after a long battle with cancer. He has always been a shining example to me in every respect. I warmly thank Thomas Jansen for his advice and support, in particular during the first years of my Ph. D. work.

I would like to thank my coauthors Benjamin Doerr, Tobias Friedrich, Thomas Jansen, Pietro Oliveto and, especially, Frank Neumann and Carsten Witt for introducing me to the runtime analysis of swarm intelligence. I thank all current and former members of the chair *Efficient Algorithms and Complexity Theory* for the enjoyable working atmosphere, and in particular Thomas Jansen and Carsten Witt for many fruitful discussions and insights.

Finally, I gratefully acknowledge financial support from the Deutsche Forschungsgemeinschaft (DFG) through the collaborative research center "Design and Management of Complex Technical Processes and Systems by Means of Computational Intelligence Methods" (SFB 531).

Nomenclature

b^i	String with i concatenations of the letter b
δ	The local search depth in memetic algorithms
$\mathrm{E}(X)$	Expectation of the random variable X
e	Euler's number $e = \exp(1) = 2.7182\ldots$
$f = \Omega(g)$	Function f grows at least as fast as g, see Definition 2.1.3
$f = \omega(g)$	Function f grows faster than g, see Definition 2.1.3
$f = \Theta(g)$	Functions f and g have the same order of growth, see Definition 2.1.3
$f = O(g)$	Function f grows at most as fast as g, see Definition 2.1.3
$f = o(g)$	Function f grows slower than g, see Definition 2.1.3
f_a	A function f transformed according to $f_a(x) = f(x \oplus a)$ for $x, a \in \{0,1\}^n$
$\mathrm{H}(x, y)$	Hamming distance between x and y for $x, y \in \{0,1\}^n$
λ	The number of offspring created in one generation
$\ln(x)$	Natural logarithm of x, i. e., logarithm of x to the base e
$\log(x)$	Logarithm of x to the base 2
μ	The size of the population
n	The number of bits in the search space, also called problem dimension
$N(x)$	Open Hamming neighborhood of x (excluding x)
$N^*(x)$	Closed Hamming neighborhood of x (including x)
N_k	Neighborhood containing all points with Hamming distance exactly k
$\mathbb{N}$	The set of natural numbers, $\mathbb{N} = \{1, 2, 3, \ldots\}$
$\mathbb{N}_0$	The set of natural numbers including 0

p_m	The mutation probability with which each bit is flipped in a mutation
$\text{Prob}(A)$	Probability of the event A
$\mathbb{R}$	The set of reals
$\mathbb{R}^+$	The set of positive reals
$\mathbb{R}_0^+$	The set of positive reals including 0
ρ	The evaporation factor in ant colony optimization
τ	Function indicating pheromones on edges in ant colony optimization
τ	Reciprocal of the local search frequency in memetic algorithms
$v_{\max}$	The maximum velocity value of a particle swarm optimizer
$\|x\|_1$	The number of bits with value 1 in x
$\|x\|_0$	The number of bits with value 0 in x
$\{X_t\}_{t\geq 0}$	An infinite sequence of random variables $X_0, X_1, X_2, \ldots$
$\mathbb{Z}$	The set of integers, $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$

Contents

1 Introduction

1.1 Motivation and Overview

Optimization is one of the most important topics in computer science, engineering, and many other disciplines. Theoretical computer science has brought forward a plethora of sophisticated algorithms to solve optimization problems, especially for combinatorial optimization. In combinatorial optimization the space S of possible solutions can be described as a finite combination of finite discrete subspaces: $S = S_1 \times \cdots \times S_m$. Famous examples are packing problems where a set of objects is given with associated values for weight and profit and one is looking for a selection of objects that maximizes the total profit of selected objects while respecting constraints such as a limit on the total weight. The (combinatorial) search space is then given by a combination of single bits, where $S_i = \{0, 1\}$ encodes the choice whether to select the i-th object. Another famous problem is the traveling salesman problem (TSP) where a complete weighted graph is given and a shortest tour is sought, i. e., a closed path through the graph of minimal cost that visits all vertices exactly once. This space can, again, be regarded as a combination of choices for the vertex to visit at the i-th position of the tour. As all single spaces of a combinatorial space are finite, they can be encoded as a binary string. Concatenating all these strings for a problem results in a so-called *pseudo-Boolean* function $f\colon \{0,1\}^n \to \mathbb{R}$. This class of functions therefore includes all problems from combinatorial optimization.

A binary encoding of a combinatorial problem resembles a genome in nature, except that the state space for a single gene consists of four symbols ("alleles") A, T, C, G instead of just two values 0 and 1. The Darwinian evolution of species has led to extremely well-developed organisms using simple mechanisms such as mutation and recombination of genes and selection. Selection includes environmental selection, a principle often referred to as "survival of the fittest", as well as the selection process of animals or humans looking for appropriate mating partners.

From today's point of view, it is not surprising that the simple yet powerful principle of evolution has been used to "evolve" good solutions for a combinatorial problem in an artificial environment. The so-called *Genetic Algorithms* have been established in the United States by Holland (1975). The main idea is to generate a binary encoding for the problem at hand and then to apply mutation, recombination, and selection to a multiset ("population") of solutions in an iterative process. Selection is done according to the "fitness" of a solution, that is, the objective value of a maximization problem. Given enough time, the hope is that the process is able to evolve good solutions with high

fitness, similar to natural evolution. Independently, so-called *Evolution Strategies* were founded in Germany by Rechenberg (1973) and Schwefel (1975). In contrast to Genetic Algorithms, which heavily relied on recombination, Evolution Strategies mainly or only used mutation. Moreover, Evolution Strategies have been formulated for optimization in continuous spaces. Nowadays, both Genetic Algorithms and Evolution Strategies are known by the name of *Evolutionary Algorithms* and they form the basis of a highly active and vivid area of research known as *Evolutionary Computation*.

Evolutionary algorithms have been applied to a large variety of practical problems, often with very good results reported in the literature. The popularity of these algorithms can be explained as they are easy to implement and they can be applied in complex settings where the problem to be optimized is not well enough understood to allow the design of problem-specific algorithms. In an extreme case, the problem is just a black box to the algorithm designer and knowledge on the problem at hand can only be obtained by simulations or evaluations of candidate solutions. Moreover, evolutionary algorithms can easily be applied when there is not enough time, knowledge, or expertise to design a customized algorithm.

In contrast to problem-specific algorithms, evolutionary algorithms are not tailored towards an analysis. Hence, assessing the performance of an evolutionary algorithm on a specific problem is a non-trivial task. It is widely agreed that a solid theoretical foundation of these algorithms is needed. First approaches towards a theoretical foundation mostly dealt with convergence results or simplified models of evolutionary algorithms. A convergence result states that as time goes to infinity, the probability of the algorithm to find a global optimum tends to 1. However, in discrete spaces common mutation operators trivially ensure convergence. As practical evolutionary algorithms represent complex dynamical systems, simplified models have been generated. One exemplary simplification is to assume a "population" of infinite size and then to compute fractions of solutions with a specific property at some point of time. The investigation of simplified models may result in statements that reflect the behavior of the real algorithm. However, the real algorithm may still differ from the model in an uncontrolled way, which makes it difficult to transfer results for the model to the real algorithm. In particular, without bounding the errors introduced by simplifications it is not possible to obtain proven theorems for the real algorithm.

In the late 1990s a new line of research emerged from theoretical computer science. Instead of creating a model of an algorithm, researchers analyzed the algorithm as a model of itself. Evolutionary algorithms were regarded as randomized algorithms and the random time until these algorithms find a global optimum was assessed with mathematical rigor using tools from the analysis of randomized algorithms. This led to first theorems proving (asymptotic) bounds on the expected time to find an optimum.

Initial studies of this kind considered simple evolutionary algorithms on simple example functions. A well-known algorithm is the so-called (1+1) EA using a "population" consisting of a single individual and no recombination (see the seminal article by Droste,

Jansen, and Wegener, 2002a). The example functions chosen represented simple benchmark problems, functions with an interesting characteristic, or examples constructed to demonstrate specific effects observed through the evolution. The analysis of these simple functions encouraged the development of new methods and tools for analyzing the runtime of evolutionary algorithms. Moreover, these works clarified some questions that were intensively discussed in the field, for instance whether all unimodal functions (a definition will be given on page 17) are easy to optimize (Droste et al., 2002a; Droste, Jansen, and Wegener, 2006), whether selection should accept solutions with equal fitness (Jansen and Wegener, 2001), or whether there is a problem for which the use of recombination can improve the performance significantly (Jansen and Wegener, 2005). This way, the described theoretical studies yielded new insights for the design of evolutionary algorithms and increased our understanding of the dynamic behavior of evolutionary algorithms.

The analysis methods gained through these initial studies were then applied to more complex evolutionary algorithms and to various classical combinatorial optimization problems. For simple problems it has been shown that appropriately designed evolutionary algorithms can solve the problem to optimality in expected polynomial time, for instance for shortest path problems (Scharnow, Tinnefeld, and Wegener, 2004), sorting (as the minimization of unsortedness) (Scharnow et al., 2004), Eulerian cycles (Doerr and Johannsen, 2007a; Neumann, 2004), minimum spanning trees (Neumann and Wegener, 2006, 2007) and matroid optimization (Reichel and Skutella, 2007). For other problems such as maximum matchings (Giel and Wegener, 2003), the NP-hard partition problem (Witt, 2005), or the NP-hard minimum multicut problem (Neumann and Reichel, 2008), simple evolutionary algorithms can provide good approximations in polynomial time or, in the case of maximum matchings, they constitute a PRAS (polynomial-time randomized approximation scheme). Although we cannot expect evolutionary algorithms to outperform problem-specific algorithms, these results underline the capabilities (and, in some cases, also the limitations) of evolutionary algorithms. Hence, the fundamental research on simple algorithms on simple example functions in the end led to very interesting results that demonstrate the usefulness of evolutionary algorithms from a well-founded theoretical perspective.

Motivation

The purpose of this thesis is to continue this fruitful line of research and to apply the techniques for the analysis of evolutionary algorithms to new classes of randomized search heuristics. In recent years practitioners have developed and established a variety of different extensions of evolutionary algorithms. One such extension is the use of additional mechanisms for maintaining diversity (i. e., dissimilarity of solutions) within the population. A large diversity is often essential for the exploration of a search space with different local optima. Furthermore, many new mutation and recombination operators

have been proposed, some of which are, intentionally or unintentionally, biased towards creating particular solutions more likely than others. The effect of such a bias on search is not well understood from a theoretical perspective.

Another very popular extension of evolutionary algorithms is to hybridize the evolutionary process with local search. The purpose is to apply local search to newly constructed solutions to quickly find good solutions. In an extreme case, the evolution is then focused on the space of local optima. These algorithms are also well suited for constrained optimization problems because local search can work as a repair mechanism: if mutation and/or recombination lead to constraint violations, local search may turn an infeasible solution with low fitness into a much more promising feasible solution with good fitness. It is plausible that the evolution benefits from using local search from a local point of view, but it is not obvious how the use of local search affects the global behavior and hence the performance of the algorithm. One important practical issue in the design of these hybrids is the balance between evolutionary search and local search. As local search is often computationally expensive, resources have to be spread adequately among the two components. Moreover, using too much local search may interfere with the exploration of the search space as the algorithm behaves too greedily. In the worst case, this may result in the same local optima being rediscovered over and over again or in the population quickly collapsing to the same local optimum. The theoretical understanding of these effects as well as the limits and capabilities of these hybrids is still in its infancy.

Another important topic addressed by this thesis is the analysis of new bio-inspired optimization paradigms. The area of *Swarm Intelligence* relies on the collective intelligence of agents that interactively explore the search space. The two best known areas of swarm intelligence are *Ant Colony Optimization* and *Particle Swarm Optimization*. Ant colony optimization imitates the foraging behavior of real ants. Certain ant species communicate by placing chemicals called *pheromones* on the ground. If an ant locates a food source, it leaves a trail of pheromones when walking back to the nest. These pheromones can be smelled by other ants and other ants are then attracted by the pheromone trail. The more ants follow this trail, the more pheromones are placed on this trail and the more ants are attracted. This reinforcement of paths then leads to well known ant trails. Moreover, when different ants discover different paths from the nest to a food source, short paths are likely to be invested with pheromones more quickly than longer paths. This way, ants are capable of finding short paths in nature, using a very simple form of communication.

Ant colony optimization mimics this simple mechanism to reinforce good solutions. The main idea is that artificial ants perform a random walk on a so-called construction graph. This graph is chosen such that the path taken by an ant can be mapped to a solution of the problem at hand. All edges receive a certain amount of artificial pheromone. The ant's random walk is then biased by the amount of pheromone placed on all edges; edges leading to currently unvisited vertices are chosen with a probability

proportional to their amount of pheromone. When the solution construction is completed, the constructed solution is evaluated and, in case of a sufficiently good solution, the pheromones on the chosen edges are increased to increase the probability that subsequent ants rediscover the same good solution or take similar paths.

Particle swarm optimization mimics the social behavior of a fish school or a bird flock. These algorithms have originally been defined for optimization in continuous (e. g., Euclidean) spaces. A "swarm" of particles is maintained, where every particle flies through the search space with a certain velocity. Moreover, each particle has a current position, corresponding to a candidate solution for the problem, and it memorizes the best position it has occupied up to now. In every step the velocity of each particle is then updated with respect to the particle's own experience and the experience of other particles in its neighborhood. More precisely, the velocity is updated in the direction of its own best position and the best position of any particle in its neighborhood. The motivation for this kind of update stems from psychology; it combines cognitive and social effects to determine the behavior of a particle. In the literature discrete variants of particle swarm optimizers have been defined, where the velocities are used in a probabilistic process to construct new (binary) positions for a particle. The velocities there play a similar role for particle swarm optimization as pheromones for ant colony optimization.

The theoretical analysis of swarm intelligence is a new and a major challenge as, similar to the first days of evolutionary algorithms, only convergence results and results for simplified models were known until 2006.

Overview

We give an overview of the thesis. Chapter 2 introduces the reader into evolutionary algorithms and some frequently used tools for their analysis. We present an introduction into evolutionary algorithms, including commonly used terms, definitions, and ways of speaking. Then some prominent fitness functions are defined. Most functions played a role in the theory of evolutionary algorithms so far and many functions will be revisited later on in the context of different algorithms. A major part of this chapter is a description of methods and tools from the analysis of randomized algorithms that are used for or adapted to the analysis of evolutionary algorithms. As these methods will be frequently used throughout this thesis, we use a textbook-like presentation, including illustrative examples, to familiarize the reader with their application. Moreover, the examples give a first impression on the asymptotic runtime of simple evolutionary algorithms on interesting example functions.

Chapter 3 covers extensions of simple evolutionary algorithms. In Section 3.1 we investigate a mutation operator that is biased towards solutions containing either few or many 1-bits. This operator is motivated by applications where some knowledge on the problem is available; in particular, where it is known that good solutions have either few or many 1-bits. We consider illustrative example functions where this operator leads

to a significant speed-up, but we also take a broader perspective including general and negative effects this operator can have.

Section 3.2 covers mechanisms to maintain diversity within the population. Maintaining diversity is non-trivial as fitness-based selection and a large diversity are often conflicting goals. Many different strategies have been proposed in the literature. We compare these mechanisms on a bimodal test function that reflects characteristics found in practical problems, but also allows for a theoretical analysis. The results show that some of these mechanisms can find the global optimum efficiently, while other mechanisms fail even with a fairly large population.

Chapter 4 contains the first analyses of hybridizations of evolutionary algorithms with local search. These algorithms are known by various names such as *memetic algorithms*, *genetic/evolutionary local search*, and *global-local search hybrids*. We first address the issue of finding a good parametrization of such a memetic algorithm, i. e., a proper balance between evolutionary search and local search. It is often believed that finding a good parametrization is always easy. Section 4.1 describes classes of worst-case examples for this task. For these functions certain parametrizations lead to an efficient optimization, but only slight changes of the parameters lead to a disastrous decline in performance. Moreover, for almost all parametrizations there is a function where this parametrization turns out to be very inefficient. This disproves the existence of simple a priori design guidelines that work well for every problem.

Still, memetic algorithms are powerful search heuristics that have been successfully applied for various combinatorial problems. Section 4.2 contains an initial study why these algorithms are so effective in combinatorial optimization. We investigate a memetic algorithm using a local search operator called *variable-depth search*, which resembles the well known Lin-Kernighan operator for the traveling salesman problem and the Kernighan-Lin operator for graph partitioning. We consider simple instances of three classical combinatorial problems, Mincut, Knapsack, and Maxsat. These instances all contain local optima that are very hard to overcome for common algorithms like the simple (1+1) EA (see page 14 for a definition), simulated annealing, and even a simple memetic algorithm using a more greedy kind of local search. While all these algorithms need exponential time with a fairly high probability, a simple memetic algorithm using variable-depth search turns out to be very effective. These results give a first impression on the reason why memetic algorithms are so effective in practice.

Last but not least, Chapter 5 deals with the analysis of swarm intelligence algorithms. The rigorous mathematical analysis of ant colony optimization was started in 2006, independently by Gutjahr (2008) and Neumann and Witt (2006). The latter authors defined a simple algorithm called 1-ANT and analyzed its performance on the simple example function OneMax that simply counts the number of bits set to 1 in the given solution. Section 5.1 brings forward the analysis of the 1-ANT on slightly more difficult example functions. Thereby, we develop new methods for the analysis of ant colony optimization algorithms and insights into the dynamic behavior of the 1-ANT. It turns out, as already

observed by Neumann and Witt (2006) for OneMax, that the 1-ANT is very sensitive to a parameter called ρ that determines the strength of a pheromone update. More precisely, there is a phase transition from polynomial runtimes to exponential runtimes as ρ decreases. Gutjahr and Sebastiani (2008) presented an analysis of a slightly different ant colony optimizer called MMAS* where this phase transition does not occur. This motivates us to study this algorithm and variants thereof in Section 5.2. In these algorithms, the best solution found so far is reinforced in every iteration. We contribute to the analysis of these algorithms, building upon previous results by Gutjahr and Sebastiani (2008) and thereby also demonstrate the importance of the question whether the best-so-far solution should be exchanged by new solutions with the same objective value.

Similar to hybrid evolutionary algorithms, a common strategy in ant colony optimization is to use local search to refine newly constructed solutions. Interestingly, the use of local search here has additional effects, compared to evolutionary algorithms, as the pheromones only slowly adapt to changes of the best-so-far solution. If local search creates a new best-so-far solution that is far away from the previous best one, this may result in the algorithm sampling in a region of the search space that has never been visited before. Section 5.3 describes this effect in more detail and demonstrates the impact on performance on two constructed functions where the use of local search is either beneficial or detrimental.

Finally, Section 5.4 contains the first rigorous runtime analysis of a particle swarm optimizer (PSO). We stick to the space of pseudo-Boolean functions and consider the binary PSO algorithm introduced by Kennedy and Eberhart (1997). We present a general lower bound for all functions with unique global optimum and provide a general method for proving upper bounds by transferring a method well-known from evolutionary algorithms to particle swarm optimization. A more detailed analysis on the simple test function OneMax yields additional insights into the dynamic behavior of the algorithm.

In addition to theoretical bounds on the (expected) time to find an optimum, Chapters 3 and 5 contain supplementary experimental results that accompany our theoretical findings and give a more concrete impression on the performance of the considered algorithms.

This thesis ends with conclusions and an outlook in Chapter 6 and an appendix containing some useful mathematical tools collected from the literature.

1.2 Underlying Publications

This thesis is based on the following publications, listed in the order in which they appear in this thesis. For all joint papers with k authors, this author's contribution in terms of ideas, proofs, and writing can roughly be quantified as $1/k$.

1. Jansen, T. and Sudholt, D. (2005). Design and analysis of an asymmetric mutation operator. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '05)*, pages 497–504. IEEE Press.

   Section 3.1 is based on the following extension.

   Jansen, T. and Sudholt, D. (2009). Analysis of an asymmetric mutation operator. *Evolutionary Computation*. To appear.

2. Friedrich, T., Oliveto, P. S., Sudholt, D., and Witt, C. (2008). Theoretical analysis of diversity mechanisms for global exploration. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '08)*, pages 945–952. ACM Press. Best paper award in the track "Genetic Algorithms."

   Section 3.2 is based on the following extension.

   Friedrich, T., Oliveto, P. S., Sudholt, D., and Witt, C. (submitted). Analysis of diversity mechanisms for global exploration.

3. Sudholt, D. (2006b). On the analysis of the (1+1) memetic algorithm. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '06)*, pages 493–500. ACM Press.

4. Sudholt, D. (2006a). Local search in evolutionary algorithms: the impact of the local search frequency. In: *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC '06)*, volume 4288 of *LNCS*, pages 359–368. Springer.

   Section 4.1 is based on the following extension that contains 4. and parts of 3.

   Sudholt, D. (submitted). The impact of parametrization in memetic evolutionary algorithms.

5. Sudholt, D. (2008). Memetic algorithms with variable-depth search to overcome local optima. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '08)*, pages 787–794. ACM Press. Nominated for a best paper award in the track "Formal Theory."

   Section 4.2 is based on this paper. An extended version has been invited to a special issue in *Algorithmica*.

6. Doerr, B., Neumann, F., Sudholt, D., and Witt, C. (2007b). On the runtime analysis of the 1-ANT ACO algorithm. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '07)*, pages 33–40. ACM Press. Best paper award in the track "Ant Colony Optimization, Swarm Intelligence, and Artificial Immune Systems."

   Section 5.1 is based on the following extended version.

   Doerr, B., Neumann, F., Sudholt, D., and Witt, C. (submitted). Runtime analysis of the 1-ANT ACO algorithm.

7. Neumann, F., Sudholt, D., and Witt, C. (2007). Comparing variants of MMAS ACO algorithms on pseudo-Boolean functions. In: *Proceedings of Engineering Stochastic Local Search Algorithms (SLS '07)*, volume 4638 of *LNCS*, pages 61–75. Springer.

   Section 5.2 is based on the following extended version.

   Neumann, F., Sudholt, D., and Witt, C. (2009). Analysis of different MMAS ACO algorithms on unimodal functions and plateaus. *Swarm Intelligence*. To appear.

8. Neumann, F., Sudholt, D., and Witt, C. (2008). Rigorous analyses for the combination of ant colony optimization and local search. In: *Proceedings of the Sixth International Conference on Ant Colony Optimization and Swarm Intelligence (ANTS '08)*, volume 5217 of *LNCS*, pages 132–143. Springer.

   Section 5.3 is based on this paper.

9. Sudholt, D. and Witt, C. (2008b). Runtime analysis of Binary PSO. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '08)*, pages 135–142. ACM Press.

   Section 5.4 is based on the following extended version.

   Sudholt, D. and Witt, C. (submitted). Runtime analysis of a binary particle swarm optimizer.

2 Preliminaries

2.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) are randomized search heuristics inspired by the natural evolution of species. The main idea is to simulate the evolution of candidate solutions for an optimization problem. A typical evolutionary algorithm maintains a multiset of solutions called *population*. The quality of the solutions in the population is given by the objective value of the function to be optimized; one speaks of the *fitness* of a solution and refers to the function as *fitness function*. The algorithm iteratively tries to evolve the current population into a population of higher fitness. This is done by selecting parents from the current population, applying mutation and/or recombination operators to create new solutions, called *offspring*, and finally selecting good solutions from the old population and the newly created offspring to form a new population. By mimicking the principle of "survival of the fittest", we can hope to evolve a population of good individuals in the long run.

Evolutionary algorithms and other such heuristics are popular among practitioners as they can be applied in a setting where the optimization problem is not well understood or when there is not enough time, money, or expertise to design a problem-specific algorithm. In the extreme case, nothing is known about the problem at hand and evaluations of solutions are the only way to gather knowledge about it. This setting is known as *black-box scenario* as here the problem can be seen as a black box. In such a scenario, randomized search heuristics like evolutionary algorithms are often the only possible way to deal with these problems.

This thesis covers the maximization of pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$. The search space $\{0,1\}^n$ represents a "natural" space for evolutionary algorithms; moreover, all problems from combinatorial optimization can be described as a pseudo-Boolean function. Throughout this thesis n always denotes the number of bits. The restriction to maximization problems is not essential as all minimization problems can be mapped to maximization problems by multiplying the objective function by -1. Elements of the search space are synonymously called *search points*, *solutions*, or *individuals* in the context of evolutionary algorithms. For a search point x we often denote $x = x_1 \dots x_n$ where x_i represents the value of the i-th bit in the bit string. Thereby, we use the string notation $x_1 \dots x_n$ as concatenation of $x_1, x_2, \dots,$ and x_n. In particular, we adopt the widely used notation b^i for the i concatenations of the letter b. Thus, the all-one bit

string of length n can be written as 1^n. The most natural metric underlying $\{0,1\}^n$ is the *Hamming distance*.

Definition 2.1.1 (Hamming distance)**.** For two search points $x, y \in \{0,1\}^n$ with bit values $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$ let

$$\mathrm{H}(x,y) := \sum_{i=1}^{n} |x_i - y_i|$$

be the *Hamming distance* between x and y. If $\mathrm{H}(x,y) = 1$, then y is called *Hamming neighbor* of x. For two non-empty sets of search points $X, Y \subseteq \{0,1\}^n$ let

$$\mathrm{H}(x,Y) := \min_{y \in Y}\{\mathrm{H}(x,y)\}$$

be the Hamming distance between the search point x and the set Y and

$$\mathrm{H}(X,Y) := \min_{x \in X}\{\mathrm{H}(x,Y)\}$$

be the Hamming distance between the sets X and Y.

We also define optima with respect to Hamming distance.

Definition 2.1.2 (Local and global optima)**.** A search point x is called *local optimum* of a function $f\colon \{0,1\}^n \to \mathbb{R}$ if $f(x) \geq f(y)$ holds for all Hamming neighbors y of x. In addition, x is called *global optimum* of f if $f(x) = \max\{f(y) \mid y \in \{0,1\}^n\}$.

The term *local optimum* will often be used for local optima that are not global optima, although every global optimum also represents a local optimum. In the language of evolutionary computation, one often speaks of local and global optima in the context of a *fitness landscape*. A fitness landscape is an imaginary "landscape" that reflects the search space. The height of search points in this landscape corresponds to their fitness. A local or global optimum can be seen as the top of a "hill" if it is surrounded by search points whose fitness increases towards the optimum. Such a "hill" then corresponds to the "basin of attraction" of a local or global optimum, another informal notion characterizing a region of the search space where the fitness points towards a specific optimum. Similar to hills, one can also speak of "valleys" in the landscape and even "ridges", i. e., paths of Hamming neighbors with high fitness, surrounded by points of lower fitness. A more frequently used term is that of a "plateau", a set of solutions with equal fitness value, usually connected in Hamming space. We do not present formal definitions for these terms as they are only used in informal discussions on a high level of abstraction.

Now, we describe the general scheme of an evolutionary algorithm. First, the population is initialized by choosing a multiset of μ individuals uniformly at random from $\{0,1\}^n$. Thereby μ is a common notation for the size of the population. Then a loop

is executed until some stopping criterion is fulfilled. In this loop, first a set of parents is selected; this process is called *parent selection*. These parents may then be mutated and/or recombined to create new offspring. The number of offspring created in one generation is denoted by λ. Finally, the next generation's population is created by selecting good individuals from the set of μ parents and the λ new offspring. This selection step is also referred to as *environmental selection*.

Algorithm 1 General scheme of an evolutionary algorithm

Initialize the population as a multiset of μ individuals.
repeat
    Select a multiset of parents from the population.
    Create λ offspring by mutating and/or recombining the selected parents.
    Select μ individuals from the union of the population and the offspring.
    Let these individuals replace the current population.
until stopping criterion.

For both kinds of selection, individuals may be selected uniformly at random (*uniform selection*) or with a probability proportional to their fitness (*fitness-proportional selection*). Other common strategies are simply to deterministically select the individuals with the best fitness (*cut selection*) or to simulate a tournament among randomly selected individuals and then to select the winner (*tournament selection*). Any selection mechanism that ensures the survival of the best individuals found so far is also termed *elitist selection*.

In case the new population is selected from both old population and offspring, the corresponding evolutionary algorithm (EA) is called a $(\mu+\lambda)$ EA as the new population is created out of $\mu + \lambda$ individuals. An alternative is to ignore the old population and to select μ individuals among the λ offspring. This requires $\lambda \gg \mu$ and it leads to algorithms denoted as (μ, λ) EAs. To emphasize the use of recombination, also called *crossover*, "EA" may be replaced by "GA" for *Genetic Algorithm*.

The runtime analysis of evolutionary algorithms usually focuses on $(\mu+\lambda)$ EAs with cut selection, where in case of equal fitness offspring usually are preferred over their parents. In such an algorithm it is ensured that the fitness of the best individual is monotone increasing over time. This also reflects the principle of "survival of the fittest." We take the same perspective in this thesis and consider a $(\mu+\lambda)$ EA with cut selection to be a "typical" evolutionary algorithm.

We also describe a few common mutation and recombination operators. Mutation is usually done by flipping each bit independently with a fixed parameter called *mutation probability* p_m. A common choice for this value is $p_m = 1/n$ as then the expected number of flipping bits equals 1.

A common recombination operator is 1-point crossover: when crossing two parents $x_1 \dots x_n$ and $y_1 \dots y_n$, a value $i \in \{1, \dots, n-1\}$ is chosen uniformly at random and an offspring is created by cutting the parents at position i and taking parts from different parents. The resulting offspring is then $x_1 \dots x_i y_{i+1} \dots y_n$; a second offspring may be constructed as $y_1 \dots y_i x_{i+1} \dots x_n$. Similarly, 2-point crossover or, generally, k-point crossover chooses multiple such cut points at random and then takes parts from alternating parents. These recombination operators depend on the ordering of bits. This does not hold for *uniform crossover*: uniform crossover chooses each bit value from a parent chosen uniformly at random. This implies, in particular, if both parents share a common value at some bit, the offspring will receive the same value at this position. If both parents have different bit values, the bit is assigned randomly in the offspring.

The analysis of evolutionary algorithms started with very simple variants. One very popular algorithm in the theory of evolutionary algorithms is the so-called (1+1) EA using a "population" of size $\mu = 1$ and an offspring population of size $\lambda = 1$. As there is no mating partner available, the (1+1) EA only uses mutation and no recombination. Still, there is evolution as the current search point constantly competes with its offspring. We give a formal definition of the (1+1) EA.

Algorithm 2 (1+1) EA

Choose x uniformly at random.
repeat
  Create y by flipping each bit in x independently with probability $1/n$.
  if $f(y) \geq f(x)$ **then** $x := y$.

We refrain from specifying a stopping criterion and consider the algorithm as an infinite stochastic process instead. This makes sense as we are mainly interested in the random time until a global optimum is found. As performance measure, denoted as *runtime* or *optimization time*, we often consider the number of generations needed to find a global optimum. In case of population-based algorithms or memetic algorithms using local search, it is often more fair to consider the number of fitness evaluations instead. This accounts for the computational effort in one generation. Relying on the number of fitness evaluations has become a standard for the analysis of randomized search heuristics. One reason is that the additional computational effort necessary to run an evolutionary algorithm is often polynomially related to the number of fitness evaluations. Moreover, in practice often fitness evaluations are the most expensive operations, for example if the fitness can only be assessed by simulations. Finally, in the following we often allow ourselves to neglect the cost of initialization as this only constitutes an additive term of μ.

Randomized search heuristics are usually not designed with an analysis in mind. When analyzing such algorithms, one therefore often considers simple variants. In the case

of evolutionary algorithms, a popular way to obtain a simple variant is to consider a population of size 1. Such an algorithm is often called a *trajectory-based algorithm* as the sequence of current solutions over time corresponds to a single trajectory through the search space.

This restriction to trajectory-based algorithms makes sense for several reasons. One is that the analysis for these algorithms is simpler than for "real" populations. The dynamics within a large populations are often quite difficult to control. If the population is well-spread in the search space, a new solution might be created by mutation from different parents at different locations, implying a large variance in the offspring construction. When considering a crossover of two or more parents, this further increases the variance. Moreover, the analysis has to take into account that with selection different individuals in the population compete against one another. This means that the analysis cannot simply focus on the trajectory of one specific individual as this individual might be suddenly removed from the population if other individuals evolve towards better solutions more quickly. Those effects can be quite challenging for an analysis. In particular, when analyzing new classes of heuristics, it therefore makes sense to start with a simple algorithm.

Second, the analysis of a trajectory-based algorithm can lead to valuable insights about the behavior of an algorithm and it encourages the development of analysis methods. These methods can then be used to analyze more complicated algorithms later on.

Third, often simple trajectory-based algorithms are somewhat representative for more complicated algorithms. The hints given by an objective function that determine the behavior of a trajectory-based algorithm also apply to larger populations in many cases. The behavior of these two classes of algorithms is then similar, unless the behavior of a large population is superimposed by other effects (like crossover). We can say that often the analysis of population-based algorithms is more complicated than the analysis of a trajectory-based algorithm and that it is simply not worthwhile to make this additional effort as conclusions can also be drawn from the analysis of a simple algorithm.

Last but not least, for many problems investigated theoretically trajectory-based algorithms perform surprisingly well. When taking the number of f-evaluations as performance measure, the following holds. If the objective function gives good hints towards global optima, algorithms with population size 1 are often more effective than variants with a larger population. When a better solution is found, an algorithm with population size 1 immediately occupies this new location in the search space. A larger population, however, may need more time to occupy this new spot with an adequate number of individuals (either by successful mutations or by offspring at the new location replacing worse individuals at the old location). A small population is able to adapt more quickly to the fitness landscape. This is an advantage for "easy" functions, but it may be a disadvantage on difficult multimodal problems where a more diverse search is needed. Many problems in applications are of this kind. On the other hand, many problems investigated theoretically are easier as the purpose of the analysis is often to better

understand the behavior of these algorithms. This partially explains why practitioners often use large populations and theoreticians favor much smaller populations.

We conclude this section with a bit of notation. Bounds on the (expected) optimization time are usually stated as asymptotic results. This reflects the order of growth of the bound and neglects constant factors or smaller order terms, leading to simplified terms for our bounds. We use the well-known asymptotic notation, also known as Landau notation or "big-Oh" notation (see for example Cormen, Leiserson, Rivest, and Stein, 2001).

Definition 2.1.3 (Asymptotic notation)**.** Let $f, g\colon \mathbb{N}_0 \to \mathbb{R}$ be two functions.

We say f grows at most as fast as g and write $f = O(g)$ if and only if there exist $n_0 \in \mathbb{N}$ and $c \in \mathbb{R}^+$ such that for all $n \geq n_0$ we have $f(n) \leq c \cdot g(n)$. We say f grows at least as fast as g and write $f = \Omega(g)$ if and only if $g = O(f)$.

We say f and g have the same order of growth and write $f = \Theta(g)$ if and only if $f = O(g)$ and $f = \Omega(g)$.

We say f grows faster than g and write $f = \omega(g)$ if and only if $\lim_{n\to\infty} g(n)/f(n) = 0$. We say f grows slower than g and write $f = o(g)$ if and only if $g = \omega(f)$.

When dealing with asymptotics, this always refers to the problem size n. Note that asymptotic bounds such as $O(n^2)$ or $\Omega(n \log n)$ hold trivially if n is bounded from above by a fixed constant; then every function of n is of order $\Theta(1)$. Therefore, for proofs of such bounds it is safe to assume that $n \geq n_0$ for a suitable constant n_0.

An important distinction is between polynomial and exponential (expected) optimization times. An algorithm with exponential optimization time is clearly inefficient for the considered problem. Contrarily, an algorithm with polynomial expected optimization time can be called efficient, when opposed to an exponential runtime. We also define notions for inverse polynomial and inverse exponential functions that will be used mostly for probabilities.

Definition 2.1.4 (Polynomial/superpolynomial/exponential)**.** Let $f\colon \mathbb{N} \to \mathbb{R}^+$.

- We say that f is polynomial and write $f = \text{poly}(n)$ if $f(n) = O(n^k)$ for some constant $k \in \mathbb{R}_0^+$.
- We say that f is superpolynomial if $f(n) = \omega(n^k)$ for every constant $k \in \mathbb{R}_0^+$.
- We say that f is exponential if $f(n) = \Omega\left(2^{n^\varepsilon}\right)$ for some constant $\varepsilon \in \mathbb{R}^+$.

- We say that f is polynomially small and write $f = 1/\text{poly}(n)$ if $1/f$ is polynomial.
- We say that f is superpolynomially small if $1/f$ is superpolynomial.
- We say that f is exponentially small if $1/f$ is exponential.

Concerning probabilities, we also define a handy notation for events that are very likely.

Definition 2.1.5 (Notions for probabilities)**.** We say that an event A occurs with high probability (w. h. p.) if $1 - \text{Prob}(A) = O(n^{-k})$ for some $k \in \mathbb{R}^+$.

We say that an event A occurs with overwhelming probability (w. o. p.) if $1 - \text{Prob}(A)$ is exponentially small.

2.2 Fitness Functions

The theoretical investigation of evolutionary algorithms started with simple example functions. This makes sense since, unlike many problem-specific algorithms, evolutionary algorithms are usually not designed to support an analysis. Often example functions were chosen that were simple enough to be attacked by rigorous arguments, had interesting properties, and reflected characteristics of practical problems. Analyzing such example functions allowed researchers to build up a reservoir of methods and insights. This then helped to attack more complicated problems. Nowadays, this approach can be called a success story since analyses are now possible for more realistic problems from combinatorial optimization. When analyzing extensions of evolutionary algorithms and new classes of search heuristics, we will try to follow this path and to develop new analysis methods with simple example functions. We present some of these functions here.

The number of bits with value 1 is denoted by $|x|_1$ and the number of bits with value 0 is denoted by $|x|_0$. The most simple test function is the function

$$\textsc{OneMax}(x) := |x|_1$$

that simply counts the number of ones in x. OneMax is often chosen to test whether a heuristic algorithm can hit a specific target point efficiently. Here the fitness function gives best possible hints towards the optimum and small steps of length 1 are sufficient to optimize the function. The latter also holds for a more general class of functions, so-called *unimodal functions*.

Definition 2.2.1. A function f is called *unimodal* if and only if for every search point x that is not a global optimum there is a Hamming neighbor y of x with $f(y) > f(x)$.

Remark 2.2.2. An alternative definition is that f contains exactly one local optimum. A unimodal function from this definition is also unimodal under Definition 2.2.1. The latter definition is more general as it allows multiple local optima with the same f-value. In particular, a constant function is unimodal as there is no x that is not globally optimal.

Functions with several local and global optima of different fitness are commonly called *multimodal* or *bimodal* in the case of two such optima.

Some example functions have been proposed as examples or counterexamples to common beliefs. One such example is the function

$$\textsc{LeadingOnes}(x_1 \dots x_n) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_i$$

that counts the number of leading ones, i. e., the length of the longest prefix of the bit string that contains only 1-bits. Observe that this function is unimodal as flipping only the leftmost 0-bit results in a Hamming neighbor with larger fitness. The function LeadingOnes was proposed by Rudolph (1997a) as a counterexample for the belief that all unimodal functions can be optimized by simple hill-climbers like the (1+1) EA in expected time $O(n \log n)$.

Another interesting function is Ridge (Quick, Rayward-Smith, and Smith, 1998), also known as SPI, short path with increasing fitness, in Jansen and Wegener (2001). The most interesting part of the function is a path, a sequence of Hamming neighbors, called *ridge*, that ends with the unique global optimum. In the rest of the search space the function gives hints to reach the start of the ridge. An evolutionary algorithm typically reaches the ridge close to the starting point and then has to climb up the ridge towards the optimum. In particular, the (1+1) EA optimizes Ridge in expected time $\Theta(n^2)$ (Jansen and Wegener, 2001).

$$\textsc{Ridge}(x) := \begin{cases} n - |x|_1 & \text{if } x \notin \{1^i 0^{n-i} \mid 0 \leq i \leq n\} \\ n + i & \text{if } x \in \{1^i 0^{n-i} \mid 0 \leq i \leq n\} \end{cases}$$

The concept of guiding an algorithm towards a path with increasing fitness is interesting. The benefit of such a construction is that the course of the evolution can be predicted quite well once the whole population reaches the path. If all individuals close to the path (but not belonging to it) have low fitness, the creation of search points that are far away from the path is very unlikely. Similar constructions will reappear in Sections 4.1 and 5.3.

All functions defined so far are unimodal. One difficulty with non-unimodal functions is that there may be regions of the search space of equal fitness. Such a region is called a *plateau*. Depending on size and structure of the plateau, it may be hard for an algorithm to leave the plateau towards better individuals as the fitness function does not give useful hints. A simple example of a plateau is given in the following function Plateau, which is also known as SPC in Jansen and Wegener (2001).

$$\textsc{Plateau}(x) := \begin{cases} n - |x|_1 & \text{if } x \notin \{1^i 0^{n-i} \mid 0 \leq i \leq n\} \\ n + 1 & \text{if } x \in \{1^i 0^{n-i} \mid 0 \leq i < n\} \\ n + 2 & \text{if } x = 1^n \end{cases}$$

The function PLATEAU resembles RIDGE, except that the path of increasing fitness is now a plateau of equal fitness. However, the plateau is small enough to be efficiently overcome by simple algorithms like the (1+1) EA, whose expected optimization time is of order $\Theta(n^3)$ (Brockhoff, Friedrich, Hebbinghaus, Klein, Neumann, and Zitzler, 2007; Jansen and Wegener, 2001).

In the most extreme case the whole search space is a plateau, except for a single global optimum. In such a setting, search is like looking for a needle in a haystack. This is reflected in the function

$$\textsc{Needle}(x) := \prod_{i=1}^{n} x_i$$

for which Droste, Jansen, Tinnefeld, and Wegener (2003) proved that *every* black-box algorithm needs $\Omega(2^n)$ function evaluations on average (with respect to a natural generalization to a class of functions).

2.3 Methods for the Analysis of Randomized Search Heuristics

In order to set the stage for the upcoming analyses, we describe some of the methods frequently used in the analysis of randomized search heuristics. Many tools are accompanied by examples showing their concrete application for the (1+1) EA. Parts of this presentation are freely adapted from Wegener (2002), Oliveto, He, and Yao (2007), and lectures on evolutionary algorithms by Wegener (Summer term 2002) and Jansen (Winter term 2006/2007).

2.3.1 Standard Mutations

The most common mutation operator, called *standard mutation*, flips each bit independently with probability $1/n$. In order to understand evolutionary algorithms and their extensions, it is crucial to gain some intuition about the characteristics of this operator.

Consider a standard mutation of a parent x. Fix a specific target point y with $\mathrm{H}(x, y) = k \geq 1$. Then the probability that the mutation creates y is

$$\left(\frac{1}{n}\right)^k \cdot \left(1 - \frac{1}{n}\right)^{n-k}.$$

Obviously, this probability is bounded from above by $1/n^k$. We obtain a lower bound $1/(en^k)$ using $(1 - 1/n)^{n-k} \geq (1 - 1/n)^{n-1} \geq 1/e$ with $e = \exp(1) = 2.7182\ldots$ by Lemma A.13 in the appendix.

The probability of generating y from x by a standard mutation only depends on the Hamming distance $\mathrm{H}(x, y)$. Therefore, all search points with Hamming distance k from x

have the same probability of being generated. The probability of creating an arbitrary search point with Hamming distance k is therefore

$$\binom{n}{k} \cdot \left(\frac{1}{n}\right)^k \cdot \left(1 - \frac{1}{n}\right)^{n-k}.$$

Using $\binom{n}{k} \leq n^k/k!$ (cf. Lemma A.10) yields $1/k!$ as upper bound. We summarize our findings in the following lemma.

Lemma 2.3.1. *Consider a standard mutation of a search point x and let $1 \leq k \leq n$. The probability that a specific search point with Hamming distance k to x is created is bounded from below and above by*

$$\frac{1}{en^k} \leq \left(\frac{1}{n}\right)^k \cdot \left(1 - \frac{1}{n}\right)^{n-k} \leq \frac{1}{n^k}.$$

The probability that an arbitrary search point with Hamming distance k is created is

$$\binom{n}{k} \cdot \left(\frac{1}{n}\right)^k \cdot \left(1 - \frac{1}{n}\right)^{n-k} \leq \frac{1}{k!}.$$

In every generation of the (1+1) EA there is a chance that the events from Lemma 2.3.1 happen. The expected time until one such event with probability p happens is therefore geometrically distributed with parameter p. By Lemma A.1, the expected waiting time equals $1/p$. It follows that the expected waiting time for a specific k-bit mutation is at least n^k and at most en^k. The expected waiting time for a mutation flipping k bits is at least $k!$.

2.3.2 The Fitness-Level Method

The (μ+λ) EA and all other evolutionary algorithms considered in this thesis have in common that the fitness of the best individual in the population cannot decrease over time. The expected optimization time can therefore be bounded from above if we have bounds for the expected waiting times until the fitness of the best individual increases. The following theorem is known as *method of f-based partitions* (Wegener, 2002) or *fitness-level method*.

Theorem 2.3.2 (Fitness-level method)**.** *For two sets $A, B \subseteq \{0,1\}^n$ and fitness function f let $A <_f B$ if $f(a) < f(b)$ for all $a \in A$ and all $b \in B$. Consider the (μ+λ) EA and a partition of the search space into non-empty sets $A_1, \ldots, A_m$ such that*

$$A_1 <_f A_2 <_f \cdots <_f A_m$$

and A_m only contains global optima. Let s_i be a lower bound on the probability of creating a new offspring in $A_{i+1} \cup \cdots \cup A_m$, provided the population contains a search point in A_i.

Then the expected number of generations for the $(\mu+\lambda)$ EA *to find an optimum is bounded by*

$$\sum_{i=1}^{m} \frac{1}{s_i}.$$

Note that this bound is independent of the initialization as we pessimistically assume that all fitness-level sets have to be passed. In Wegener (2002) a more precise bound is presented taking into account the random initialization. However, in most applications the initialization does not have a significant impact, therefore we stick to this simplified formulation of the method.

We exemplarily apply the fitness-level method to the analysis of the (1+1) EA on OneMax, following Droste et al. (2002a).

Theorem 2.3.3 (Application: upper bound for OneMax)**.** *The expected optimization time of the* (1+1) EA *on* OneMax *is* $O(n \log n)$.

Proof. We choose the canonical partition $A_0, \ldots, A_n$ where A_i contains all search points x with $|x|_1 = i$. Let $0 \leq i < n$. Given that the current solution x is in A_i, there are $n - i$ bits with value 0. Flipping one such bit and not flipping the other $n - 1$ bits results in a search point in $A_{i+1} \cup \cdots \cup A_n$. For a specific 0-bit, this event has probability $1/n \cdot (1-1/n)^{n-1} \geq 1/(en)$, recalling Lemma 2.3.1. As the considered events for all $n-i$ 0-bits are disjoint, we have $s_i \geq (n-i) \cdot 1/(en)$. Theorem 2.3.2 leads to the bound

$$\sum_{i=0}^{n-1} \frac{1}{s_i} \;\leq\; \sum_{i=0}^{n-1} \frac{en}{n-i} \;=\; en \cdot \sum_{i=1}^{n} \frac{1}{i} \;\leq\; en \cdot ((\ln n) + 1).$$

For the last inequality we have used Lemma A.11. Therefore, the expected runtime of the (1+1) EA on OneMax is $O(n \log n)$. □

It is also easy to arrive at a general upper bound for all unimodal functions.

Theorem 2.3.4 (Application: general upper bound for unimodal functions)**.** *The expected optimization time of the* (1+1) EA *on a unimodal function* f *attaining* d *different function values is* $O(nd)$.

Proof. As f contains d different function values, there is an f-based partition $A_1, \ldots, A_d$ where every set contains search points of equal fitness. For every set A_i, $i < d$, every search point has at least one Hamming neighbor on a better fitness level. It follows $s_i \geq 1/(en)$ and the expected optimization time is at most $(d-1) \cdot en = O(nd)$. □

2.3.3 Analyzing Typical Runs

The fitness-level method is a simple yet powerful method, but it does not need to reflect a typical behavior of the algorithm. It may happen that many fitness level sets are skipped or that the values s_i are too pessimistic. For example, s_i may be dominated by a few challenging search points with a low probability of leaving A_i, which are unlikely to be reached when passing through A_i.

Therefore, it is often necessary to track the typical behavior of an evolutionary algorithm. This can be done by (explicitly or implicitly) identifying "phases" in the run of the algorithm. We usually identify subsequent phases where Phase i ends when some goal is achieved, and then Phase $i+1$ starts. The total runtime can be estimated by the sum of times spent in all phases. The main benefit of such a phase argumentation is that a carefully specified goal for Phase i allows a more detailed analysis of the following phases. For example, when looking for an upper bound, the goal of Phase i may imply that then the goal of Phase $i+1$ is reached efficiently. By a careful specification of phases and their corresponding goals, we can obtain strong results on the runtime distribution of an evolutionary algorithm.

Such a phase argumentation may contain situations where unlikely events may "disturb" the search dynamics in a typical run. For example, a very unlucky initialization may lead the algorithm to regions of the search space that are never visited in a typical run. Other "disturbing" events may be that standard mutation causes overly large jumps that result in a very untypical behavior. We often speak of these unlikely events as "errors" or "failures" and its probabilities as "error probabilities" and "failure probabilities", respectively.

In a typical analysis, we bound the total error probability p_{error} by the sum of all error probabilities (cf. the union bound from Lemma A.4). By the law of total probability (cf. Lemma A.2), we have for the runtime T

$$\mathrm{E}(T) \;=\; \mathrm{E}(T \mid \text{error}) \cdot p_{\text{error}} + \mathrm{E}(T \mid \text{no error}) \cdot (1 - p_{\text{error}})$$

In order to obtain lower bounds, we can use the trivial lower bound $\mathrm{E}(T \mid \text{error}) \geq 0$, resulting in

$$\mathrm{E}(T) \;\geq\; \mathrm{E}(T \mid \text{no error}) \cdot (1 - p_{\text{error}}). \tag{2.1}$$

For upper bounds on $\mathrm{E}(T)$, we require an upper bound for $\mathrm{E}(T \mid \text{error})$, a smaller upper bound for $\mathrm{E}(T \mid \text{no error})$ and an upper bound for the total error probability. In case there is no suitable upper bound for $\mathrm{E}(T \mid \text{error})$, we at least obtain a statement on the conditional expected runtime in a typical run without errors. Moreover, if this conditional expectation is concentrated, we can prove runtime bounds that hold with high probability.

Theorem 2.3.5 (Application: lower bound for OneMax)**.** *The expected optimization time of the* (1+1) EA *on* OneMax *is* $\Omega(n \log n)$.

Proof. Partition a run of the (1+1) EA into two phases. Phase 1 only contains the initialization and Phase 2 comprises the rest of the run. The goal of Phase 1 is that initialization creates a search point x with at least $n/3$ zeros. The expected number of initial zeros is $\mathrm{E}(|x|_0) = n/2$. Chernoff bounds (cf. Lemma A.6) state that the probability of having less than $n/3$ zeros is at most

$$\mathrm{Prob}\left(|x|_0 \leq \frac{n}{3}\right) \;=\; \mathrm{Prob}\left(|x|_0 \leq \left(1-\frac{1}{3}\right)\mathrm{E}(|x|_0)\right) \;\leq\; e^{-\mathrm{E}(|x|_0)\cdot 1/18} \;=\; e^{-n/36}.$$

Assume that Phase 1 has been successful and consider Phase 2. The goal here is not to reach the optimum within $t := (n-1)\cdot\ln(n/3) = \Omega(n\log n)$ steps. Consider the opposite event that the optimum is reached within t steps. A necessary condition is that every 0-bit has to be flipped at least once during t steps. The probability that a specific bit is not flipped during t steps is at least

$$\left(1-\frac{1}{n}\right)^t \;=\; \left(\left(1-\frac{1}{n}\right)^{n-1}\right)^{\ln(n/3)} \;\geq\; e^{-\ln(n/3)} \;=\; \frac{1}{n/3}.$$

The probability that this bit is flipped at least once is therefore at most $1-1/(n/3)$ and the probability that $n/3$ bits are flipped at least once in t steps is at most

$$\left(1-\frac{1}{n/3}\right)^{n/3} \;\leq\; \frac{1}{e}.$$

By the union bound (cf. Lemma A.4) the total error probability in both phases is $e^{-n/36} + 1/e \leq 1/2$ if n is large enough. Hence with probability at least $1/2$ a run is typical and the goal of Phase 2 implies a lower bound of t steps. By Equation 2.1 we obtain the lower bound $t/2 = \Omega(n\log n)$. □

For a typical run, we often demand that during a phase of random length an undesired event must not happen. In order to compute the error probability for the whole phase, we can simply multiply the error probability for a single generation with the expected length of the phase.

Lemma 2.3.6 (Error probability in a phase)**.** *Consider a phase of random length T. If in every generation there is a probability of at most p that some undesired event happens, the probability that the event happens at least once during the phase is bounded by*

$$\sum_{t=0}^{\infty} \mathrm{Prob}(T=t)\cdot t\cdot p \;=\; \mathrm{E}(T)\cdot p.$$

In order to obtain bounds that hold with high probability the following argumentation often proves useful. We describe this argument for a general stochastic process—an infinite sequence of random variables $X_0, X_1, X_2, \ldots$, denoted by $\{X_t\}_{t\geq 0}$. Such a process is called *Markov chain* or *Markov process* if

$$\text{Prob}(X_t = x_t \mid X_0 = x_0, \ldots, X_{t-1} = x_{t-1}) \;=\; \text{Prob}(X_t = x_t \mid X_{t-1} = x_{t-1})$$

holds, i. e., the current state only depends on the previous state.

Lemma 2.3.7 (Independent phases)**.** *Consider a Markov chain $\{X_t\}_{t\geq 0}$ and the first hitting time T of some desired event. If $\mathrm{E}(T \mid X_0) \leq \alpha$ for all X_0 and some integer $\alpha \in \mathbb{N}_0$, then $\text{Prob}(T \geq \alpha \cdot 2k) \leq 2^{-k}$ for every $k \in \mathbb{N}$.*

Proof. By Markov's inequality (cf. Lemma A.5) $\text{Prob}(T \geq 2\alpha \mid X_0) \leq 1/2$ for all X_0. Consider k phases of length 2α each. As the above statement holds for every initialization and the process is Markovian, each phase independently contains the desired event with probability at least $1/2$. Hence the probability that the event does not occur in all k phases is at most 2^{-k}. □

Note that Lemma 2.3.7 is applicable for runtime bounds obtained by the fitness-level method as the bound from Theorem 2.3.2 holds for every initialization.

2.3.4 Drift Analysis

Often the fitness value of the individuals in the population does not provide enough information to obtain good runtime bounds. For example, there may be plateaus of search points with equal fitness where fitness alone does not give any useful information about the algorithm's progress. It then makes sense to consider other measures than fitness. Let g be a function defined over the space of possible populations with codomain $\mathbb{R}_0^+$. Consider g as a sort of distance measure to some target population with g-value 0. We are then interested in the first hitting time T of a target state with g-value 0. A natural example is to choose g as the Hamming distance of the population to a global optimum. One can also think of other applications, for example in the context of typical runs where g may denote some sort of distance to the goal of the current phase.

There are various settings where such a consideration proves useful. It may be that the algorithm has a natural tendency to move towards the desired state with g-value 0. Imagine a simple deterministic scenario where every step decreases the current g-value by some $\delta \in \mathbb{R}^+$, with probability 1. Then, starting with state X_0, we would clearly have $\mathrm{E}(T) = g(X_0)/\delta$, provided $g(X_0)/\delta \in \mathbb{N}_0$. The same statement holds if the *expected* g-decrease in one step equals δ, i. e., if $\mathrm{E}(g(X_t) - g(X_{t+1}) \mid X_t) = \delta$ holds for $g(X_t) > 0$. This relationship between $\mathrm{E}(T)$ and δ is known as Wald's equation (Mitzenmacher and Upfal, 2005; Wald, 1944).

We will often speak of the expected g-decrease as *drift*. In order to derive upper or lower bounds on $\mathrm{E}(T)$, it is not necessary that δ describes the drift exactly. If δ is a lower bound for the drift and nevertheless $\delta > 0$, then we can derive an upper bound $\mathrm{E}(T) \leq \mathrm{E}(g(X_0))/\delta$. Symmetrically, if δ is an upper bound for the drift and $\delta > 0$, a lower bound $\mathrm{E}(T) \geq \mathrm{E}(g(X_0))/\delta$ can be shown.

Although the above-mentioned bounds were derived independently in other studies (see Wegener and Witt (2005, Lemma 6) for an upper bound and Jägersküpper (2007, Lemma 12) for a lower bound), we refer to He and Yao (2004) for bounds on the conditional expectation $\mathrm{E}(T \mid X_0)$. Using the law of total expectation (cf. Lemma A.3) yields $\mathrm{E}(T) = \mathrm{E}(\mathrm{E}(T \mid X_0))$. This is used to extend results from He and Yao (2004) with bounds for the unconditional expected first hitting time $\mathrm{E}(T)$.

Theorem 2.3.8 (Drift arguments, He and Yao (2004))**.** *Consider a Markov process $\{X_t\}_{t\geq 0}$ with state space S and a function $g\colon S \to \mathbb{R}_0^+$. Let $T := \inf\{t \geq 0\colon g(X_t) = 0\}$.*

1. *If there exists $\delta > 0$ such that for every time $t \geq 0$ and every state X_t with $g(X_t) > 0$ the condition $\mathrm{E}(g(X_t) - g(X_{t+1}) \mid X_t) \geq \delta$ holds, then*

$$\mathrm{E}(T \mid X_0) \;\leq\; \frac{g(X_0)}{\delta} \qquad \text{and} \qquad \mathrm{E}(T) \;\leq\; \frac{\mathrm{E}(g(X_0))}{\delta}.$$

2. *If there exists $\delta \in \mathbb{R}^+$ such that for every time $t \geq 0$ and every state X_t with $g(X_t) > 0$ the condition $\mathrm{E}(g(X_t) - g(X_{t+1}) \mid X_t) \leq \delta$ holds, then*

$$\mathrm{E}(T \mid X_0) \;\geq\; \frac{g(X_0)}{\delta} \qquad \text{and} \qquad \mathrm{E}(T) \;\geq\; \frac{\mathrm{E}(g(X_0))}{\delta}.$$

Theorem 2.3.9 (Application: upper and lower bounds for LeadingOnes)**.** *The expected optimization time of the* (1+1) EA *on* LeadingOnes *is $\Theta(n^2)$.*

Proof. We choose as distance function the fitness difference to an optimal fitness value, i. e., $g(x_t) := n - \textsc{LeadingOnes}(x_t)$ where x_t denotes the current search point in generation t. First, we bound the expected g-value of the initial solution x_0. The probability that the initial solution contains i leading ones is 2^{-i-1} for $0 \leq i < n$ as this requires the first i bits to be set to 1 and the $(i+1)$-st bit to be 0. We have

$$\mathrm{E}(g(x_0)) \;=\; n - \left(\sum_{i=0}^{n-1} i \cdot 2^{-i-1} + n \cdot 2^{-n}\right) \;=\; n - 1 + 2^{-n},$$

proving $n - 1 \leq \mathrm{E}(g(x_0)) \leq n$.

For the upper bound it is sufficient for a fitness increase that all i leading ones are reproduced and an $(i+1)$-st 1-bit is appended. This event has probability at least

$(1-1/n)^i \cdot 1/n \geq 1/(en)$ and then $g(x_{t+1}) = g(x_t) - 1$. As the LeadingOnes-value cannot decrease, this implies $\mathrm{E}(g(x_t) - g(x_{t+1})) \leq 1/(en)$ if $\textsc{LeadingOnes}(x_t) > 0$. Invoking Theorem 2.3.8 with $\delta := 1/(en)$ yields the upper bound $en^2 = O(n^2)$.

We now strive at proving a lower bound of the same order. A necessary event to decrease the g-value is to flip the leftmost 0-bit. This happens with probability $1/n$. However, the LeadingOnes-value may increase by more than 1 if some of the following bits happen to have value 1. These bits are called *free riders* by Droste, Jansen, and Wegener (2002a). The bits following the leftmost zero have never yet contributed to the fitness value, hence, intuitively, these bits are due to a uniformly random distribution. (The interested reader may consult Droste et al. (2002a) for a formal proof.) If $t+1$ is the first generation where such a bit is first relevant for the fitness value, it attains value 1 in x_{t+1} with probability 1/2. Note that we consider an unconditional probability, i. e., we do not condition on the bit's value in any previous search point, including initialization. Instead, we pretend that the random decision about the bit's value at time $t+1$ is first made at time $t+1$. This way of arguing is known as *principle of deferred decisions* (Mitzenmacher and Upfal, 2005; Motwani and Raghavan, 1995). After generation $t+1$, all considered bits either belong to the block of leading ones or represent the leftmost 0-bit, hence we cannot make multiple deferred decisions for the same bit.

By these arguments the probability that the first j bits following the leftmost 0-bit all become free riders is 2^{-j}. Repeating the calculations for $\mathrm{E}(x_0)$ from above, the expected number of free riders gained in a step increasing the LeadingOnes-value is at most 1 and the expected fitness increase is at most 2. We conclude $\mathrm{E}(g(x_t) - g(x_{t+1})) \leq 2/n$, hence Theorem 2.3.8 results in the bound $(n-1) \cdot n/2 = \Omega(n^2)$. □

Another scenario where a consideration of a distance function g proves useful is when there is a tendency to move away from the target point, i. e., when there is a drift to increase the distance g. As we are dealing with finite search spaces, the distance g cannot become arbitrarily large. So a tendency to increase g cannot be present in the whole search space. However, if we restrict our considerations to an interval $[a, b]$ on the scale of g, it is possible that for this interval we have a tendency to increase the distance g. It then may be hard to decrease the g-value below a.

A drift in the interval $[a, b]$ alone is not sufficient to derive lower bounds as the interval may be skipped or traversed very quickly by large jumps on the g-scale. To avoid this, we need the additional assumption that the probability of large jumps decreases fast enough. Hajek (1982) presented a *drift theorem* where lower bounds could be derived from the two above-mentioned assumptions. This theorem was used for the runtime analysis of evolutionary algorithms (Friedrich, Oliveto, Sudholt, and Witt, 2008; Giel and Wegener, 2003). Oliveto and Witt (2008) recently presented a simplified theorem that is much easier to apply than the previous formulation. To simplify the notation, we consider a general Markov process $\{X_t\}_{t \geq 0}$ that includes the case where $X_t = g(x_t)$.

Theorem 2.3.10 ((Simplified) Drift Theorem, (Oliveto and Witt, 2008)). *Consider a Markov process $\{X_t\}_{t\geq 0}$ over the state space $S = \{0, 1, \ldots, N\}$ for some integer $N \in \mathbb{N}$. Let $0 \leq a < b \leq N$ and $T = \min\{t \geq 0 \colon X_t \leq a \mid X_0 \geq b\}$. For $i \in S$ and $t \geq 0$ let $\Delta_t(i) := (X_{t+1} - X_t \mid X_t = i)$. Suppose there are constants $\delta, \varepsilon, r > 0$ such that for all $t \geq 0$*

1. $\mathrm{E}(\Delta_t(i)) \geq \varepsilon$ *for* $a < i < b$ *and*
2. $\mathrm{Prob}(\Delta_t(i) = -j) \leq 1/(1+\delta)^{j-r}$ *for* $i > a$ *and* $j \geq 1$,

then there is a constant $c > 0$ such that $\mathrm{Prob}\big(T \leq 2^{c(b-a)}\big) = 2^{-\Omega(b-a)}$.

The following example is adapted from Oliveto and Witt (2008).

Theorem 2.3.11 (Application: lower bound for NEEDLE). *The probability that the* (1+1) EA *optimizes* NEEDLE *in 2^{cn} generations is $2^{-\Omega(n)}$ for some constant $c > 0$.*

Proof. Let x_t be the current search point of the (1+1) EA in generation t and let $X_t = |x_t|_0$. We use the following observation: if the number of zeros in the current search point is low, i. e., smaller than $n/4$, it is likely that more 1-bits flip to 0 than 0-bits flip to 1. Therefore, for these search points there is a drift towards an increased number of zeros.

To apply Theorem 2.3.10, we set $a := 0$ and $b := n/4$. With probability $1 - 2^{-\Omega(n)}$ the initial search point contains at least $n/4$ zeros and then the condition $X_0 \geq b$ in the definition of T is fulfilled. We first estimate $\mathrm{E}(\Delta_t(i)) = \mathrm{E}(|x_{t+1}|_0 - |x_t|_0 \mid |x_t|_0 = i)$ for $0 \leq i < b$. Given $|x_t|_0 = i$, the expected number of zeros flipping to 1 is i/n, while the expected number of ones flipping to 0 is $(n-i)/n$. Together, we have

$$\mathrm{E}(\Delta_t(i)) \;=\; \frac{n-i}{n} - \frac{i}{n} \;=\; \frac{n-2i}{n} \;\geq\; \frac{1}{2}$$

and choosing $\varepsilon := 1/2$, the first condition of Theorem 2.3.10 holds. For the second condition the number of newly created 0-bits is clearly bounded by the number of flipping bits. Recall that flipping k bits simultaneously has probability at most $1/k! \leq (e/k)^k \leq 2^{3-k}$ for $k \in \mathbb{N}$. Therefore, choosing $\delta := 1$ and $r := 3$, we have $\mathrm{Prob}(\Delta_t(i) = -j) \leq 2^{3-j} = 1/(1+\delta)^{3+j}$ and the second condition has been verified as well. Theorem 2.3.10 then yields the claim. □

2.3.5 Random Walks and Martingales

In case there is no drift, neither towards nor pointing away from the target state, the considered algorithm performs an unbiased random walk through the state space. It may happen that for almost all states the expected progress equals 0. Such a process is known in probability theory as *martingale*.

Definition 2.3.12 (Martingales, submartingales, and supermartingales)**.** Let $\{X_t\}_{t\geq 0}$ be a stochastic process and $\{Y_t\}_{t\geq 0}$ be a real-valued stochastic process such that Y_t is a function of $X_0, \ldots, X_t$ and $\mathrm{E}(|Y_t|) < \infty$ for all $t \geq 0$.

Then $\{Y_t\}_{t\geq 0}$ is called a *martingale* with respect to $\{X_t\}_{t\geq 0}$ if

$$\mathrm{E}(Y_{t+1} \mid X_0, \ldots, X_t) \;=\; Y_t.$$

Moreover, $\{Y_t\}_{t\geq 0}$ is called a *submartingale* with respect to $\{X_t\}_{t\geq 0}$ if

$$\mathrm{E}(Y_{t+1} \mid X_0, \ldots, X_t) \;\geq\; Y_t$$

and *supermartingale* if

$$\mathrm{E}(Y_{t+1} \mid X_0, \ldots, X_t) \;\leq\; Y_t.$$

Note that we do not require $\{X_t\}_{t\geq 0}$ to be Markovian. However, in the Markovian case we can replace $\mathrm{E}(Y_{t+1} \mid X_0, \ldots, X_t)$ by $\mathrm{E}(Y_{t+1} \mid X_t)$ in Definition 2.3.12. Also note that a martingale is a submartingale and a supermartingale at the same time. If $\{Y_t\}_{t\geq 0}$ is a submartingale, then $\{-Y_t\}_{t\geq 0}$ is a supermartingale, and vice versa. Finally, it is possible that $\{X_t\}_{t\geq 0} = \{Y_t\}_{t\geq 0}$, i. e., a martingale can be defined with respect to itself.

Of course, an expected progress of 0 does not imply that the algorithm is unable to reach specific target states. This is due to the variance in the stochastic process. We are interested in the random time until a target state is hit. This time is commonly denoted as *stopping time*.

Definition 2.3.13. A *stopping time* T of a stochastic process $\{X_t\}_{t\geq 0}$ is a random variable with codomain $\mathbb{N}_0 \cup \{\infty\}$ such that for all $t \geq 0$ the event $T = t$ does only depend on $X_0, \ldots, X_t$ (and not on $X_{t+1}, X_{t+2}, \ldots$).

By definition of a martingale, its expected progress in one step equals 0. Moreover, its expected progress in $t \geq 0$ steps still equals 0 if t is fixed. Consider the expected progress within T steps, where T is a stopping time and hence a random variable that may depend on the process itself. One might think that the statement for all fixed t implies the statement for T, but this need not be the case. Consider a symmetric random walk $\{X_t\}_{t\geq 0}$ on $\mathbb{Z}$, starting with $X_0 = 0$, where $\mathrm{Prob}(X_{t+1} = X_t + 1) = 1/2$ and $\mathrm{Prob}(X_{t+1} = X_t - 1) = 1/2$. Define the stopping time $T := \inf\{t \mid X_t = 1\}$. It is known that $\mathrm{E}(T) < \infty$, hence $\mathrm{E}(X_T)$ is defined, but $\mathrm{E}(X_T) = 1 \neq X_0$ by definition of T. However, under certain conditions $\mathrm{E}(X_T) = \mathrm{E}(X_0)$ still holds. The following *martingale stopping theorem* is also known as optional stopping theorem or optional sampling theorem.

Theorem 2.3.14 (Martingale Stopping Theorem (Williams, 1991, Theorem 10.10))**.** *Let $\{Y_t\}_{t\geq 0}$ be defined with respect to a process $\{X_t\}_{t\geq 0}$ and T be a stopping time of $\{X_t\}_{t\geq 0}$.*

If $\mathrm{E}(T) < \infty$ *and there is a constant* $c \in \mathbb{R}_0^+$ *such that* $\mathrm{E}(|Y_{t+1} - Y_t| \mid X_0, \ldots, X_t) \leq c$ *for all* $0 \leq t < T$*, then*

$$\begin{array}{lll} \mathrm{E}(Y_T) \geq \mathrm{E}(Y_0) & & \text{if } \{Y_t\}_{t\geq 0} \text{ is a submartingale,} \\ \mathrm{E}(Y_T) \leq \mathrm{E}(Y_0) & & \text{if } \{Y_t\}_{t\geq 0} \text{ is a supermartingale, and} \\ \mathrm{E}(Y_T) = \mathrm{E}(Y_0) & & \text{if } \{Y_t\}_{t\geq 0} \text{ is a martingale.} \end{array}$$

We will use the martingale stopping theorem to estimate the expected time until a real-valued stochastic progress first hits some boundary values. The application is not immediate. After all, all statements related to martingales seen so far tell us that such a stochastic process does *not* move, in expectation. The trick is to introduce another martingale process that incorporates the variance of the original process. Then the martingale stopping theorem is applied for both processes and the relationship between the two processes yields the desired bound. We make this argumentation precise for the following simple random walk.

Theorem 2.3.15 (Application: analysis of a simple random walk)**.** *Let* $\{X_t\}_{t\geq 0}$ *by a stochastic process on* $\mathbb{Z}$ *where* $X_{t+1} := X_t + 1$ *with probability* $1/2$ *and* $X_{t+1} := X_t - 1$ *otherwise. Define* $T := \inf\{t \geq 0 \mid X_t \in \{0, n\}\}$ *and observe that* T *is a stopping time with respect to* $\{X_t\}_{t\geq 0}$*. If* $0 \leq X_0 \leq n$*, then* $\mathrm{E}(T) = X_0(n - X_0)$*.*

Proof. Observe that $\{X_t\}_{t\geq 0}$ is a martingale:

$$\mathrm{E}(X_{t+1} \mid X_0, \ldots, X_t) \;=\; \frac{1}{2} \cdot (X_t + 1) + \frac{1}{2} \cdot (X_t - 1) \;=\; X_t.$$

Define a stochastic process $\{Y_t\}_{t\geq 0}$ with respect to $\{X_t\}_{t\geq 0}$ by

$$Y_t \;:=\; (X_t)^2 - t.$$

We claim that $\{Y_t\}_{t\geq 0}$ is a martingale with respect to $\{X_t\}_{t\geq 0}$:

$$\begin{aligned} \mathrm{E}(Y_{t+1} \mid X_0, \ldots, X_t) \;&=\; (X_{t+1})^2 - t - 1 \\ &=\; \frac{1}{2} \cdot (X_t + 1)^2 + \frac{1}{2} \cdot (X_t - 1)^2 - t - 1 \\ &=\; (X_t)^2 - t \;=\; Y_t. \end{aligned}$$

We apply the martingale stopping theorem to $\{Y_t\}_{t\geq 0}$. As there is always a probability of at least $2^{-n/2}$ that the next $\lfloor n/2 \rfloor$ steps reach either 0 or n, we have $\mathrm{E}(T) \leq n \cdot 2^{n/2} < \infty$. Along with $|Y_{t+1} - Y_t| \leq (X_t)^2 \leq n^2$, the second condition of Theorem 2.3.14 is fulfilled. This implies $\mathrm{E}(Y_T) = \mathrm{E}(Y_0) = (X_0)^2$. On the other hand, we have by definition of Y_t that $\mathrm{E}(Y_T) = \mathrm{E}\big((X_T)^2 - T\big) = \mathrm{E}\big((X_T)^2\big) - \mathrm{E}(T)$. Putting the two statements together yields

$$\mathrm{E}(T) \;=\; \mathrm{E}\big((X_T)^2\big) - (X_0)^2. \tag{2.2}$$

Observe $\mathrm{E}\big((X_T)^2\big) = \mathrm{Prob}(X_T = n) \cdot n^2$. Applying the martingale stopping theorem again for $\{X_t\}_{t\geq 0}$ yields $\mathrm{E}(X_T) = X_0$, which implies $\mathrm{Prob}(X_T = n) = X_0/n$ as $\mathrm{E}(X_T) = \mathrm{Prob}(X_T = n) \cdot n$. We conclude $\mathrm{E}\big((X_T)^2\big) = X_0 \cdot n$ and plugging this into Equation 2.2 yields

$$\mathrm{E}(T) \;=\; X_0 \cdot n - (X_0)^2 \;=\; X_0(n - X_0).$$

□

Note that the term $-t$ in the definition of $\{Y_t\}_{t\geq 0}$ reflects the total variance in t steps of the process $\{X_t\}_{t\geq 0}$. We will see in Section 3.1 how the result from Theorem 2.3.15 can be generalized to random walks with arbitrary step lengths.

2.3.6 The Role of Experiments

Although the main focus of this thesis is on rigorous theoretical results, some analyses are accompanied by experimental supplements. We would like to explain how these results relate to our theoretical results. It should be noted that the explanatory power of experiments is limited. Clearly, experimental results cannot prove positive general results (i. e., "for all" statements). To disprove a general result, one counterexample is sufficient. However, when dealing with purely asymptotical results, we cannot find a counterexample by experiments. One reason is that asymptotic results only hold for $n \geq n_0$ for some unknown constant $n_0 \in \mathbb{R}_0^+$. When doing experiments for problem dimensions up to n', we cannot exclude that $n_0 > n'$, i. e., the asymptotic result under consideration does not hold for the problem dimension investigated experimentally. Even if n_0 is known, statements on random variables mostly are of stochastic nature and thus cannot be proved or disproved by experiments for logical reasons.

On the other hand, experiments can be a useful enhancement to theoretical results. First, they can provide information for small problem dimensions that may not be covered by asymptotic results. Second, experiments can reveal insight into the constant factors and lower-order terms hidden in the asymptotic notation. In particular, experiments can provide hints whether the asymptotic runtime bounds are tight. It may thus happen that theory and experiments cross-fertilize.

We often add experiments for illustrative purposes. We think that a combination of theoretical results and experiments can deliver a more complete picture. Moreover, if visualized properly, experimental data is readable and easy to grasp. Another benefit of experiments is therefore that the scope of a scientific work is broadened and a wider audience is addressed. As a final consequence, this may help to narrow the gap between theory and practice.

All experiments in this thesis have been conducted by the author using FrEAK, the "**Fr**ee **E**volutionary **A**lgorithm **K**it" (Briest, Brockhoff, Degener, Englert, Gunia, Heering, Jansen, Leifhelm, Plociennik, Röglin, Schweer, Sudholt, Tannenbaum, and Wegener,

2004b), developed within the student project group 427 (Briest, Brockhoff, Degener, Englert, Gunia, Heering, Jansen, Leifhelm, Plociennik, Röglin, Schweer, Sudholt, Tannenbaum, and Wegener, 2004a). The list of authors includes the two supervisors Thomas Jansen and Ingo Wegener. FrEAK is a platform for running evolutionary algorithms and other randomized search heuristics, released under the GNU General Public License (GPL). It has been designed such that it is easily extendable to incorporate new fitness functions, new operators and many other new components of randomized search heuristics. The process of designing new modules is described in the Module Developer's Guide (Briest et al., 2004b). Moreover, the user can specify and add so-called observer modules that collect information from the run of the algorithm. This information can then be displayed on a graphical interface, which enables a detailed and customized view into the dynamic behavior of the algorithm.

For the experiments in this thesis version 0.2 of FrEAK has been used and some additional modules have been written that have not been published. As pseudo-random number generator the *Mersenne Twister* was used (Matsumoto and Nishimura, 1998).

In some cases, we apply statistical hypothesis tests to the experimental data to test the hypothesis whether one algorithm outperforms another one. For this purpose, we use the non-parametric *Mann-Whitney test* (Mann and Whitney, 1947) to test the hypothesis whether two sets of samples originate from the same probability distribution. Unlike other common statistical tests such as Student's t-test, a non-parametric test does not require the underlying distribution to be normal. This is important as, in contrast to typical statistical tests in other disciplines (arts, biology, medicine, ...), measured runtimes are not necessarily normally distributed. Instead of using the absolute sampled values, the Mann-Whitney test operates on the ranks of the sampled data to avoid the normality requirement, so that this test can be safely applied in our settings.

3 Extensions of Evolutionary Algorithms

The (1+1) EA and the (μ+1) EA using a parent population constitute the most simple evolutionary algorithms. These algorithms are most interesting for a theoretical analysis as they are a clean and straight representation of basic working principles of evolutionary algorithms. Yet these algorithms can be attacked by rigorous arguments. Evolutionary algorithms used in practical applications, however, often contain various extensions of these basic mechanisms. In this chapter we consider two such extensions in detail: the use of biased operators and mechanisms to maintain diversity within the population. The extended algorithms are compared with their simple variants (1+1) EA and (μ+1) EA, respectively. All other aspects of algorithms design are kept fixed, which allows us to isolate the effects of the extension under consideration.

Evolutionary algorithms are usually applied in a black-box scenario where nothing is known about the fitness function at hand and fitness evaluations of search points are the only way to gather knowledge about the problem. In such a setting it makes sense to use operators that are not biased towards specific regions of the search space. The only bias in the search process is supposed to result from fitness evaluations of the search points found so far. In a practical setting, some problem-specific knowledge may be known beforehand. Then it might be advantageous to apply search operators that are tailored towards the problem to focus the search on more promising parts of the search space. In Section 3.1 we investigate a specialized mutation operator biasing the search towards solutions with either few 1-bits or few 0-bits. The use of this operator can speed up the evolution considerably in case the bias gives helpful hints. On the other hand, the bias may also slow down the search drastically if it leads the search into the wrong direction. Another aspect is that the biased mutation operator renders the algorithm vulnerable to simple transformations of the fitness function that do not affect search for the simple (1+1) EA. Such a transformation can determine whether the hints given by the biased operator are helpful or misleading. Therefore, the use of such a transformation can also be beneficial or detrimental in this context.

When using populations, one typically aims at maintaining a certain amount of diversity within the population. This enables the algorithm to discover different local or global optima. Many different approaches are known how this goal can be achieved, for example avoiding duplicates, crowding, and fitness sharing. In Section 3.2 we compare the (μ+1) EA incorporating these mechanisms with a plain (μ+1) EA without diversification on a simple bimodal test function. Our results show that some mechanisms

are able to find both optima quite efficiently, while other mechanisms fail to find both optima, even with a fairly large population.

3.1 Analysis of a Biased Mutation Operator

Evolutionary algorithms are often applied in the context of optimization when there is not enough knowledge, time, or expertise to design problem-specific algorithms. When analyzing such algorithms, one typically assumes that nothing is known about the fitness function at hand and that fitness evaluations are the only way to gather knowledge about it. This optimization scenario is called black-box optimization (Droste et al., 2006) and then it makes sense to use search operators that are not biased towards predetermined regions of the search space (see Droste and Wiesmann, 2003). When taken to its extreme, the black-box scenario leads to the well-known no free lunch theorem (NFL): when there is no structural knowledge at all, then all algorithms have equal average performance (Igel and Toussaint, 2004). In applications, such a scenario is hardly ever realistic since there is almost always some knowledge about typical solutions. It is well known that incorporating problem-specific knowledge can be crucial for the success and the efficiency of evolutionary algorithms (see Doerr, Hebbinghaus, and Neumann (2007a); Doerr and Johannsen (2007a); Raidl, Koller, and Julstrom (2006) for some recent studies).

Here, we consider one specific mutation operator for binary strings that is plausible when good solutions contain either very few bits with value 0 or very few bits with value 1. Many real-world problems share this specific property. One example is the problem of computing a minimum spanning tree (Cormen et al., 2001) on a graph $G = (V, E)$. A bit string $x \in \{0,1\}^{|E|}$ represents an edge set where each bit corresponds to exactly one edge of the graph and the selected edges correspond to bits with value 1. Most graphs with $|V|$ nodes contain $\Theta(|V|^2)$ edges whereas trees contain only $|V| - 1$ edges. So in this case all feasible solutions contain only a very small number of bits with value 1.

The most common mutation operator for bit strings of length $|E|$ flips each bit independently with probability $1/|E|$. In case the current population contains bit strings that represent spanning trees and we wait for another spanning tree to be generated by mutation, the operator tends to create offspring with more than $|V| - 1$ edges. This suggests to bias the search towards strings with few 1-bits. Neumann and Wegener (2004) introduced the asymmetric mutation operator analyzed here and proved that it leads to a significant speed-up for the minimum spanning tree problem. This mutation operator tends, on average, to preserve the number of 1-bits.

Our aim is to present a broad and informative analysis of this mutation operator. We consider its performance on illustrative example functions and on interesting classes of functions. In particular, we also consider functions beyond the class of functions the operator was originally designed for. All example functions considered here have been introduced elsewhere and for completely different reasons. Thus, they are not designed

with this mutation operator in mind. With this approach we are able to demonstrate the assets and drawbacks of this specific mutation operator in a clear and intuitive, yet rigorous way. We accompany our asymptotic results with the results of empirical investigations. We believe that the combination of theoretical results with empirical findings delivers a more complete picture. While presenting a concrete analysis for one concrete mutation operator we hope that this analysis can serve as an example of how a thorough analysis of new operators and variants of evolutionary algorithms can be presented.

In the search space $\{0,1\}^n$ one can think of all points with exactly i 1-bits as forming the i-th level. Clearly, for $i = O(1)$ and $i = n - O(1)$ the levels contain only a polynomial number of points whereas the levels with $i \approx n/2$ are exponentially large (up to size $\binom{n}{n/2}$ for even n). Imagine a random walk on $\{0,1\}^n$ induced by repeated standard mutations. Standard mutations flip each bit independently with some fixed mutation probability, typically $1/n$. Thus, standard mutations tend to sample the search space uniformly. This implies that the random walk induced by repeated standard mutations spends most of the time on levels with $i \approx n/2$. When reaching a search point x with either very few or lots of 1-bits, there is a strong tendency to return to levels $i \approx n/2$ since these levels have a much larger size.

The asymmetric mutation operator considered here is likely to preserve the current level, on average. However, considering a random walk induced by repeated asymmetric mutations, variance lets the random walk change the current level. Since there is no tendency to the medium levels, the random walk is more likely to reach levels with very few or lots of 1-bits than the random walk based on standard mutations. We will investigate the influence of this bias on the search by considering two extreme cases. The mutations operator's bias becomes most visible when analyzing its behavior in the absence of a fitness-driven selection. On the other hand, we discover how this bias can be counterbalanced by guiding the evolutionary search via fitness. To this end, we consider a class of functions where fitness values point in the direction of a unique global optimum. Thereby, we also consider a simple transformation of the fitness function that allows us to choose the position of the optimum arbitrarily in the search space. This transformation is irrelevant to evolutionary algorithms using unbiased mutation operators, but with asymmetric mutations the transformation may alter the performance drastically.

In Section 3.1.1, we define the asymmetric mutation operator and the mentioned transformation. We also derive some simple properties and prove general upper and lower bounds. In Section 3.1.2, we concentrate on the performance on simply structured example functions and demonstrate that the operator shows increased efficiency as expected. This helps to build a more concrete intuition of the properties of the asymmetric mutation operator. In a more general context, we explain in Section 3.1.3 that the performance on a broad and interesting class of functions does not differ from that of an unbiased mutation operator. Section 3.1.4 presents an example where the bias intro-

duced by the mutation operator has an immense negative impact. Finally, we conclude in Section 3.1.5.

3.1.1 Asymmetric Mutations

Recall that the mutation operator of the (1+1) EA is called standard mutation. We give a formal definition of this mutation operator.

Operator 3 Standard Mutation(x)

Independently for each bit in $x \in \{0,1\}^n$, flip the bit with probability $1/n$.

The asymmetric mutation operator aims at leaving the number of bits with value 1 unchanged. This can be achieved by letting the probability of mutating a bit depend on its value.

Operator 4 Asymmetric Mutation(x)

Independently for each bit in $x \in \{0,1\}^n$, flip the bit with probability $1/(2\,|x|_1)$ if it has value 1 and with probability $1/(2\,|x|_0)$ otherwise.

In the following, we refer to the (1+1) EA with standard mutations as the *standard* (1+1) EA and to the (1+1) EA using asymmetric mutations instead as the *asymmetric* (1+1) EA.

Remark 3.1.1. We use $1/(2|x|_i)$ as mutation probability instead of $1/|x|_i$ to prevent the mutation operator from becoming deterministic in the special case of exactly one bit with value 0 or 1. In this deterministic case the property that any $y \in \{0,1\}^n$ can be reached from any $x \in \{0,1\}^n$ in one mutation is not preserved. The value 2 is a straightforward choice since for every x with $0 < |x|_1 < n$ the expected number of flipping bits is $|x|_0 \cdot 1/(2\,|x|_0) + |x|_1 \cdot 1/(2\,|x|_1) = 1$. This coincides with the expected number of flipping bits for standard mutations.

For many objective functions, mutations of single bits turn out to be important leading from x to a Hamming neighbor $y \in N(x)$. Using standard mutations, steps to specific Hamming neighbors have probability $\Theta(1/n)$. Asymmetric mutations do not decrease the probability of such mutations significantly. They may, however, increase the probability for such steps significantly.

Lemma 3.1.2. *Let $x, y \in \{0,1\}^n$ with $y \in N(x)$. The probability of mutating x into y in one asymmetric mutation is bounded from below by $1/(8\,|x|_i)$ if a bit with value i needs to flip.*

Proof. Assume that one 0-bit in x needs to flip; the other case is symmetric. For $x \neq 0^n$, the probability of flipping only this bit equals

$$\frac{1}{2\,|x|_0}\left(1-\frac{1}{2\,|x|_0}\right)^{|x|_0-1}\left(1-\frac{1}{2\,|x|_1}\right)^{|x|_1} \;\geq\; \frac{1}{8\,|x|_0}$$

since $(1-1/(2k))^k \geq 1/2$ for $k \in \mathbb{N}$. For $x = 0^n$ we obtain $1/(4\,|x|_0) > 1/(8\,|x|_0)$ in the same way. □

For standard mutations, we have seen in Lemma 2.3.1 that the probability of flipping k bits simultaneously is at most $1/k! \leq (e/k)^k$. The latter, weaker bound also holds for asymmetric mutations.

Lemma 3.1.3. *For every $k \in \mathbb{N}$ the probability that one asymmetric mutation flips k bits simultaneously is bounded from above by $(e/k)^k$.*

Proof. If the search point x to be mutated is either 0^n or 1^n, the probability that k bits flip is at most

$$\binom{n}{k}\cdot\left(\frac{1}{2n}\right)^k \;\leq\; \frac{n^k}{k!}\cdot n^{-k} \;=\; \frac{1}{k!} \;\leq\; \left(\frac{e}{k}\right)^k.$$

Let Z be the random variable describing the number of bits flipping in the asymmetric mutation of x. If $x \notin \{0^n, 1^n\}$ the expected number of flipping bits equals $\mathrm{E}(Z) = 1$. By Chernoff bounds (Lemma A.6),

$$\mathrm{Prob}(Z \geq k) \;\leq\; \frac{e^{k-1}}{k^k} \;\leq\; \left(\frac{e}{k}\right)^k.$$

□

We start our analysis with some very general bounds on the expected optimization time of the asymmetric (1+1) EA that only rely on basic properties of the asymmetric mutation operator. The same considerations have also been made for the standard (1+1) EA, so we have a direct comparison of both mutation operators. For the standard (1+1) EA it is known that the expected optimization time is bounded from above by n^n for all functions (Droste, Jansen, and Wegener, 2002a) and bounded from below by $\Omega(n \log n)$ if the global optimum is unique (Jansen, De Jong, and Wegener, 2005). It is easy to extend this lower bound to functions with multiple global optima. We present corresponding bounds for the asymmetric (1+1) EA.

Theorem 3.1.4. *The expected optimization time of the asymmetric* (1+1) EA *on every function $f\colon \{0,1\}^n \to \mathbb{R}$ is at most $(2n)^n$. If f has at most $2^{n/2}$ global optima, the expected optimization time is bounded below by $\Omega(n)$.*

Proof. As long as the current search point x is not a global optimum, the probability of hitting an optimum x^* by a direct jump is bounded below by $(1/(2n))^n$. Hence the expected time to hit a global optimum is at most $(2n)^n$.

For the lower bound let OPT be the set of all global optima and consider the Hamming distance $\mathrm{H}(x, \mathrm{OPT})$ from the current search point x to the closest point in OPT. By Chernoff bounds, the probability that the initial search point has Hamming distance at most $n/24$ to a specific global optimum is at most

$$\left(\frac{e^{1/12-1}}{(1/12)^{(1/12)}}\right)^{n/2} = 2^{-n/2-\Omega(n)}.$$

Taking the union bound for at most $2^{n/2}$ global optima, the probability that initially $\mathrm{H}(x, \mathrm{OPT}) \leq n/24$ holds is $2^{-\Omega(n)}$. The expected initial distance is therefore at least $\left(1 - 2^{-\Omega(n)}\right) \cdot n/24 \geq n/25$ if n is large enough. We now apply drift arguments w. r. t. the potential $\mathrm{H}(x, \mathrm{OPT})$. Observe that the expected number of flipping bits when mutating x equals 1 for $x \notin \{0^n, 1^n\}$ and 1/2 for $x \in \{0^n, 1^n\}$. Therefore, the distance to the closest global optimum decreases by at most 1 in expectation. The lower bound $n/25$ then follows by drift arguments from Theorem 2.3.8. □

Comparing standard mutations to asymmetric mutations, Theorem 3.1.4 does not show a clear advantage for any operator. We also see that the resulting bounds are weak due to the weak assumptions made on the fitness function. This motivates the investigation of concrete functions for which much stronger results can be obtained.

Moreover, we will investigate the impact of simple transformations of the fitness function that only affect asymmetric mutations. We adopt the definition and notation of Droste, Jansen, Tinnefeld, and Wegener (2003) where the generalization of objective functions is considered in the context of black-box complexity.

Definition 3.1.5. For $f\colon \{0,1\}^n \to \mathbb{R}$ and $a \in \{0,1\}^n$ we define $f_a\colon \{0,1\}^n \to \mathbb{R}$ by $f_a(x) := f(x \oplus a)$ for all $x \in \{0,1\}^n$ where $x \oplus a$ denotes the bit-wise exclusive or of x and a.

Since the standard (1+1) EA is insensitive to the number of 1-bits in the current bit string and since it treats 1-bits and 0-bits symmetrically, it exhibits the same behavior on f as on f_a for every a. So, the class of functions f_a is a straightforward generalization of f. When we use asymmetric mutations instead, this is not necessarily the case. Transforming x to $x \oplus a$ does in general change the number of 1-bits and therefore alters the mutation probabilities. Note, however, that the (1+1) EA with asymmetric mutations behaves the same on f_a and $f_{\overline{a}}$ where $\overline{a}$ denotes the bit-wise complement of a. This is due to the symmetrical roles of 0 and 1 as bit values if one replaces all 0s by 1s and vice versa.

In the following, we will compare the performance of the asymmetric (1+1) EA on f and f_a for various transformations. In particular, we will see that the performance gap between f and f_a can be exponentially large.

3.1.2 Assets of the Asymmetric Mutation Operator

The asymmetric mutation operator preserves, on average, the number of 1-bits in the parent. This makes this mutation operator very different from standard mutations if the number of 1-bits is either very small or very large. Thus, we expect to obtain best results when good search points have this property and when good search points lead the algorithm to the global optimum. The well-known fitness function OneMax has all these properties. It is therefore not surprising that the asymmetric mutation operator leads to a considerable speed-up.

Theorem 3.1.6. *The expected optimization time of the asymmetric* (1+1) EA *on* OneMax *is* $\Theta(n)$.

Proof. As long as the current search point x differs from the global optimum, there are $|x|_0$ Hamming neighbors with a larger fitness value. Due to Lemma 3.1.2, the probability of increasing the fitness value is at least $|x|_0 \cdot 1/(8\,|x|_0) = 1/8$ and the expected time to increase the fitness is at most 8. Since the fitness value has to be increased at most n times, $8n$ is an upper bound on the expected optimization time. The lower bound follows from Theorem 3.1.4. □

Asymmetric mutations outperform standard mutations by a factor of order $\log n$ here. However, this relies heavily on the fact that the unique global optimum is the all-one bit string. Clearly, the objective function OneMax can be described as minimizing the Hamming distance to the unique global optimum. This is equivalent to maximizing the Hamming distance to the bit-wise complement of the unique global optimum. We can preserve this property but move the global optimum x^* somewhere else by defining the fitness as $n - \mathrm{H}(x, x^*)$. This function equals $\textsc{OneMax}_a$ with $a = \overline{x^*}$. One may fear that the advantage of asymmetric mutations for OneMax is counterbalanced by a disadvantage when the global optimum is far away from 1^n. However, this is not the case if one considers asymptotic expected optimization times.

Theorem 3.1.7. *For every* a *the expected optimization time of the asymmetric* (1+1) EA *on* $\textsc{OneMax}_a$ *is* $\Theta(n \log(\min\{|a|_0, |a|_1\} + 2))$.

Proof. W. l. o. g., $|a|_1 \leq n/2$. The unique global optimum of $\textsc{OneMax}_a$ is $\overline{a}$. We begin with a proof of the upper bound and partition a run into two phases: the first phase starts with the beginning of the run and ends when we have a search point with at most $2\,|\overline{a}|_0$ 0-bits for the first time. The second phase starts after the first phase and ends when the global optimum is found.

Let x be some current search point in the first phase, then there are at least $|x|_0 - |\overline{a}|_0$ positions i where $x_i = 0$ and $\overline{a}_i = 1$. Clearly, a 1-bit mutation flipping such a bit increases the fitness. During the first phase $|x|_0 > 2\,|\overline{a}|_0$ holds. Thus, $|x|_0 - |\overline{a}|_0 > |x|_0/2$ and by Lemma 3.1.2 the probability of a 1-bit mutation flipping one of the considered bits is at least $|x|_0/2 \cdot 1/(8\,|x|_0) = 1/16$. Therefore, the expected length of Phase 1 is $O(n)$.

At the beginning of the second phase, $|x|_0 \leq 2\,|\overline{a}|_0$ holds. We have $\mathrm{H}(x, \overline{a}) \leq 3\,|\overline{a}|_0$ since $\mathrm{H}(x, 1^n) = |x|_0 \leq 2\,|\overline{a}|_0$ and $\mathrm{H}(1^n, \overline{a}) = |\overline{a}|_0$. This implies $\textsc{OneMax}_a(x) \geq n - 3\,|\overline{a}|_0$ for the rest of Phase 2. Thus, the fitness has yet to be increased at most $3\,|\overline{a}|_0 = 3\,|a|_1$ times. There are $n - \textsc{OneMax}_a(x)$ Hamming neighbors with function value larger than x. By Lemma 3.1.2, the probability of reaching a specific Hamming neighbor by a direct mutation is bounded below by $1/(8n)$. Thus, the expected length of the second phase is bounded above by

$$\sum_{i=1}^{3|a|_1} \frac{8n}{i} \;=\; 8n \sum_{i=1}^{3|a|_1} \frac{1}{i} \;=\; O(n \log(|a|_1 + 2)).$$

For the lower bound we distinguish three cases with respect to $|a|_1$. In case $|a|_1 \leq 1$ the lower bound $\Omega(n)$ follows from Theorem 3.1.4. Consider the case $2 \leq |a|_1 \leq n/4$. Since the mutation operator does not incorporate bit positions, we can assume $a = 1^{|a|_1}0^{|a|_0}$ without loss of generality. With probability at least $1/2$, the initial search point contains at least $|a|_1/2$ 1-bits among the first $|a|_1$ positions. Analogously, it contains at least $|a|_0/2$ 1-bits among the last $|a|_0$ positions with probability at least $1/2$. Consider the case where both events occur, which happens with probability at least $1/4$. Since the Hamming distance to the global optimum $\overline{a} = 0^{|a|_1}1^{|a|_0}$ cannot increase, the number of 1-bits in the current search point is bounded below by $|a|_0/2 - |a|_1 \geq n/8$ during the run. Thus, the probability that a specific 1-bit is flipped is bounded above by $4/n$. We have at least $|a|_1/2$ 1-bits that all need to flip at least once. The probability that not all of these bits flip within $((n/4) - 1)\ln|a|_1$ mutations is bounded below by

$$\begin{aligned} &1 - \left(1 - \left(1 - \frac{4}{n}\right)^{((n/4)-1)\ln|a|_1}\right)^{|a|_1/2} \\ &\geq 1 - \left(1 - e^{-\ln|a|_1}\right)^{|a|_1/2} \geq 1 - e^{-1/2}. \end{aligned}$$

Thus, the expected optimization time is bounded below by

$$\frac{1}{4} \cdot \left(1 - e^{-1/2}\right) \cdot \left(\frac{n}{4} - 1\right) \ln|a|_1 = \Omega(n \log(|a|_1 + 2))$$

in this case.

Finally, we consider the case $n/4 < |a|_1 \leq n/2$. Chernoff bounds yield that with probability $1 - 2^{-\Omega(n)}$ the initial Hamming distance to the unique global optimum is

bounded below by $n/8$. Moreover, by Lemma 3.1.3 with probability $1 - 2^{-\Omega(n)}$ we only have mutations with at most $n/16$ bits flipping simultaneously within the first $O(n \log n)$ generations. Thus, we may consider the situation at the end when the Hamming distance to the optimum is in the interval $[n/16, n/8]$. As the Hamming distance cannot increase during a run, the number of 1-bits in the current search point is always bounded below by $n/8$ and bounded above by $7n/8$. This implies that for each bit (regardless of its value) the probability of flipping it is bounded above by $4/n$. Now we are in a situation very similar to the second case. Repeating the line of thought from there completes the proof. □

We see that, asymptotically, there is no disadvantage for the (1+1) EA with asymmetric mutations in comparison with standard mutations on $\textsc{OneMax}_a$. We complement these asymptotical bounds by the results of some experiments. All reported results are averages of 100 independent runs. For $\textsc{OneMax}_a$, we choose a with $|a|_1 = cn$ and $c \in \{0, .05, .1, .15, .2, \ldots, .95, 1\}$. We choose n, the length of the bit strings, from $\{100, 200, 300, \ldots, 1900, 2000\}$. The average optimization times can be seen in Figure 3.1. Note that these empirical findings have illustrative purposes. Therefore, we refrain from a statistical analysis that does not yield additional insights.

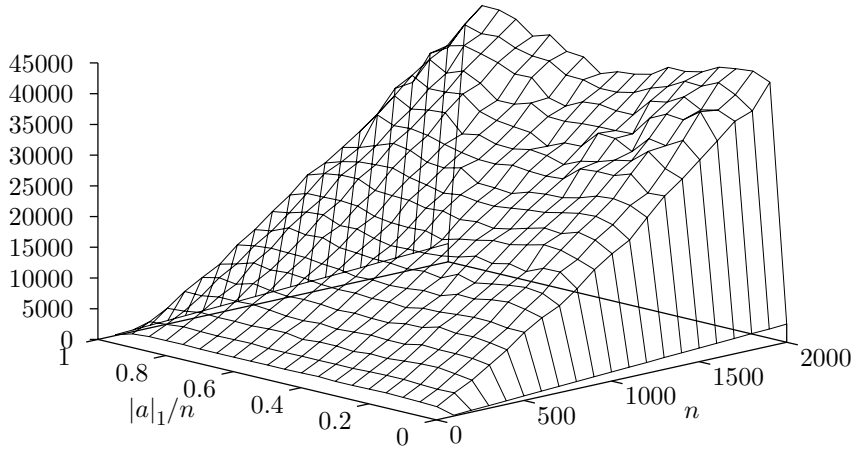

Figure 3.1: Average runtimes on $\textsc{OneMax}_a$ in 100 independent runs for $n \in \{100, 200, \ldots, 2000\}$ and $|a|_1 \in \{0, .05n, .1n, \ldots, .95n, n\}$.

Two aspects of the empirical data displayed in Figure 3.1 deserve some explanation. First, we observe that the average runtimes are maximal in some distance to the extreme values $|a|_1 = 0$ and $|a|_1 = n$ and decrease towards $|a|_1 = n/2$. This effect is not contained in our bound. Note that the observed differences are small enough not to be visible in

the asymptotic notation $\Theta(\cdot)$. Observing concrete runs for $|a|_1 = cn$ for small values of c, the asymmetric mutation operator has the tendency to lead the algorithm too close to 1^n, making final steps back towards the unique global optimum necessary. This causes an increased optimization time.

Second, the average optimization time at the boundaries seems to increase from a value very small for $|a|_1 \in \{0, n\}$ to something clearly larger almost immediately. This seems to contradict our theoretical bound $\Theta(n \log(\min\{|a|_0, |a|_1\} + 2))$. Remembering that $c = .05$ implies $|a|_1 = n/20 = \Theta(n)$ we see that this is not really the case. More values for $|a|_1$ close to 0 and n are helpful to see this more clearly. We present average runtimes for $|a|_1 \in \{0, 1, 2, \ldots, 9\}$ in Figure 3.2. The runtime increases with $|a|_1$ quite smoothly as our bound predicts. We learn that the investigation of observed runtimes alone may be misleading.

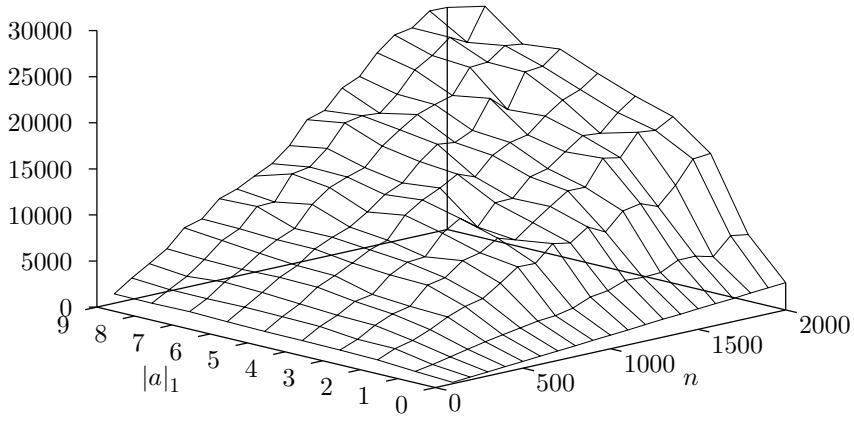

Figure 3.2: Average runtimes on $\textsc{OneMax}_a$ in 100 independent runs for $n \in \{100, 200, \ldots, 2000\}$ and $|a|_1 \in \{0, 1, 2, \ldots, 9\}$.

The reader might conclude from our findings that the search is not too clearly biased by asymmetric mutations. However, for $\textsc{OneMax}_a$, the function values point in the direction of the global optimum so clearly that the relatively small bias introduced by the asymmetric mutations is not important when compared to the clear bias introduced by selection. Thus, optimization is efficient regardless of the location of the unique global optimum. This is a strong hint that the asymmetric mutation operator can successfully be applied far beyond the class of problems it was originally designed for.

In the following, we show that there is a clear bias due to asymmetric mutations that can have a great impact on the performance of the asymmetric (1+1) EA. We consider the asymmetric (1+1) EA on a flat fitness function: we consider Needle. Since all non-optimal search points are equally fit, we exclude the effects of selection

on the optimization process and, as long as the needle is not found, the search process equals a random walk induced by repeated asymmetric mutations. So, by considering the function NEEDLE with the needle in 1^n, we can learn more about the bias induced by asymmetric mutations.

Note that the asymmetric (1+1) EA treats all bits symmetrically, regardless of their position in the bit string. Hence, the behavior of the asymmetric (1+1) EA is determined only by the number of 1-bits in the current search point. We describe the asymmetric (1+1) EA on NEEDLE as a discrete Markov chain where the current state represents the number of 1-bits in the current bit string. The main observation is that, as long as no search point in $\{0^n, 1^n\}$ is reached, this process is a martingale (see Definition 2.3.12), i. e., the expected change in the number of 1-bits over time is 0: let $x \in \{0,1\}^n$ with $0 < |x|_1 < n$ and x' be a random variable describing the result of an asymmetric mutation of x. Then

$$\mathrm{E}\big(|x'|_1 \mid |x|_1\big) = |x|_1 \cdot \left(1 - \frac{1}{2\,|x|_1}\right) + |x|_0 \cdot \frac{1}{2\,|x|_0} = |x|_1 .$$

We therefore observe a Markov chain that is similar to the simple symmetric random walk on $\{0, 1, \ldots, n\}$ investigated in Theorem 2.3.15. The main differences are that arbitrary step sizes may occur, including steps of length 0. For the simple symmetric random walk starting with state X_0, the bound $X_0(n - X_0)$ holds (Theorem 2.3.15). We are able to generalize this result to a large class of stochastic processes using a lower bound on the one-step variance. The following lemma is of independent interest.

Lemma 3.1.8. *Consider a stochastic process $\{X_t\}_{t\geq 0}$ on $\{0, \ldots, n\}$ for $n \in \mathbb{N}$. Let $T := \inf\{t \mid X_t \in \{0, n\}\}$. If $\{X_t\}_{t\geq 0}$ is a martingale and $\mathrm{E}\big((X_{t+1} - X_t)^2 \mid X_0, \ldots, X_t\big) \geq \delta$ for all $0 \leq t < T$ and some $\delta \in \mathbb{R}^+$, then $\mathrm{E}(T) \leq X_0(n - X_0)/\delta$ and $\mathrm{Prob}(X_T = n) = X_0/n$.*

Note that for the simple symmetric random walk Lemma 3.1.8 with $\delta = 1$ implies Theorem 2.3.15.

Proof of Lemma 3.1.8. We follow the line of thought from the proof of Theorem 2.3.15. Let $\mathfrak{F}_t$ abbreviate $X_0, \ldots, X_t$. We define $\{Y_t\}_{t\geq 0}$ by

$$Y_t \;:=\; (X_t)^2 - \sum_{k=0}^{t-1} \mathrm{E}\left((X_{k+1} - X_k)^2 \mid \mathfrak{F}_k\right)$$

and claim that $\{Y_t\}_{t\geq 0}$ is a martingale with respect to $\{X_t\}_{t\geq 0}$. Consider

$$\mathrm{E}\left(Y_{t+1} \mid \mathfrak{F}_t\right) \;=\; \mathrm{E}\left((X_{t+1})^2 \mid \mathfrak{F}_t\right) - \sum_{k=0}^{t} \mathrm{E}\left(\mathrm{E}\left((X_{k+1} - X_k)^2 \mid \mathfrak{F}_k\right) \mid \mathfrak{F}_t\right). \tag{3.1}$$

Regarding the $\sum$-term, the summand for $k = t$ equals

$$\mathrm{E}\left(\mathrm{E}\left((X_{k+1} - X_k)^2 \mid \mathfrak{F}_k\right) \mid \mathfrak{F}_t\right) = \mathrm{E}\left((X_{k+1} - X_k)^2 \mid \mathfrak{F}_k\right) \tag{3.2}$$

by definition. For $k < t$ we have

$$\mathrm{E}\left(\mathrm{E}\left((X_{k+1} - X_k)^2 \mid \mathfrak{F}_k\right) \mid \mathfrak{F}_t\right) = \mathrm{E}\left((X_{k+1} - X_k)^2 \mid \mathfrak{F}_k\right) \tag{3.3}$$

since the right-hand side is $\mathfrak{F}_t$-measurable. Secondly, by the formula $\mathrm{E}\big(Z^2\big) = (\mathrm{E}(Z))^2 + \mathrm{E}\big((Z - \mathrm{E}(Z))^2\big)$ we have

$$\begin{aligned} \mathrm{E}\left((X_{t+1})^2 \mid \mathfrak{F}_t\right) &= (\mathrm{E}\left(X_{t+1} \mid \mathfrak{F}_t\right))^2 + \mathrm{E}\left((X_{t+1} - \mathrm{E}\left(X_{t+1} \mid \mathfrak{F}_t\right))^2 \mid \mathfrak{F}_t\right) \\ &= (X_t)^2 + \mathrm{E}\left((X_{t+1} - X_t)^2 \mid \mathfrak{F}_t\right). \end{aligned} \tag{3.4}$$

Plugging Equations (3.2),(3.3), and (3.4) into Equation (3.1),

$$\begin{aligned} \mathrm{E}\left(Y_{t+1} \mid \mathfrak{F}_t\right) &= (X_t)^2 + \mathrm{E}\left((X_{t+1} - X_t)^2 \mid \mathfrak{F}_t\right) - \sum_{k=0}^{t} \mathrm{E}\left((X_{k+1} - X_k)^2 \mid \mathfrak{F}_k\right) \\ &= (X_t)^2 - \sum_{k=0}^{t-1} \mathrm{E}\left((X_{k+1} - X_k)^2 \mid \mathfrak{F}_k\right) \\ &= Y_t. \end{aligned}$$

Thus, $\{Y_t\}_{t\geq 0}$ is a martingale with respect to $\{X_t\}_{t\geq 0}$.

In every state $0 < X_t < n$ we have a positive one-step variance, so there is a positive probability of getting closer to the closest state from $\{0, n\}$. This implies $\mathrm{E}(T) < \infty$. Since for $t < T$

$$|Y_{t+1} - Y_t| \;\leq\; |(X_{t+1})^2 - (X_t)^2| + \mathrm{E}\left((X_{t+1} - X_t)^2 \mid \mathfrak{F}_t\right) \;<\; 2n^2$$

holds, we can apply the martingale stopping theorem (Theorem 2.3.14). This yields $\mathrm{E}(Y_T) = \mathrm{E}(Y_0) = (X_0)^2$ on one hand and, along with $\mathrm{E}\left((X_{k+1} - X_k)^2 \mid \mathfrak{F}_k\right) \geq \delta$,

$$\mathrm{E}(Y_T) \;=\; \mathrm{E}\big((X_T)^2\big) - \mathrm{E}\left(\sum_{k=0}^{T-1} \mathrm{E}\left((X_{k+1} - X_k)^2 \mid \mathfrak{F}_k\right)\right) \;\leq\; \mathrm{E}\big((X_T)^2\big) - \delta \cdot \mathrm{E}(T)$$

on the other hand, which implies

$$\mathrm{E}(T) \;\leq\; \frac{\mathrm{E}\big((X_T)^2\big) - (X_0)^2}{\delta} \;=\; \frac{\mathrm{Prob}(X_T = n) \cdot n^2 - (X_0)^2}{\delta}. \tag{3.5}$$

Applying the martingale stopping theorem again w. r. t. $\{X_t\}_{t\geq 0}$ yields $\mathrm{E}(X_T) = X_0$. Along with $\mathrm{E}(X_T) = \mathrm{Prob}(X_T = n) \cdot n$, we obtain $\mathrm{Prob}(X_T = n) = X_0/n$ and plugging this into (3.5) yields the bound $\mathrm{E}(T) \leq X_0(n - X_0)/\delta$. □

In order to apply Lemma 3.1.8 to the analysis of the asymmetric (1+1) EA on NEEDLE we have to prove a lower bound δ on the one-step variance. Moreover, the stopping time T from Lemma 3.1.8 does not distinguish between the two absorbing states 0 and n, so we have to argue how long it takes on average to arrive at n in case state 0 is reached.

Theorem 3.1.9. *For every constant $k \in \mathbb{N}_0$ and all $a \in \{0,1\}^n$ with either at most k 0-bits or at most k 1-bits, the expected optimization time of the asymmetric* (1+1) EA *on* NEEDLE_a *is bounded above by $O(n^2 + n^{k+1})$.*

Proof. W. l. o. g. we assume that the needle has $n-k$ 1-bits. Let X_t denote the number of 1-bits in the current search point in generation t. A lower bound on the one-step variance is $\mathrm{E}\big((X_{t+1} - X_t)^2 \mid X_0, \ldots, X_t\big) \geq \mathrm{Prob}(X_{t+1} \neq X_t \mid X_t)$. For $0 < X_t < n$ we have $\mathrm{Prob}(X_{t+1} = X_t + 1 \mid X_t) \geq 1/8$ and $\mathrm{Prob}(X_{t+1} = X_t - 1 \mid X_t) \geq 1/8$ by Lemma 3.1.2. This proves $\mathrm{E}\big((X_{t+1} - X_t)^2 \mid X_0, \ldots, X_t\big) \geq 1/4$ for $0 < X_t < n$ and the conditions of Lemma 3.1.8 are fulfilled for $\delta := 1/4$. Thus, starting with an initial search point with i ones the expected number of essential steps until some x^* with $|x^*|_1 \in \{0, n\}$ is reached is $4i(n-i) \leq n^2$.

With probability 1/2, by symmetry of the initialization, we have $|x^*|_1 = n$ and are done. Otherwise, let $T_{0\to n}$ be the random time for the asymmetric (1+1) EA to reach 1^n, starting from 0^n. The expected time until an asymmetric mutation flips at least one bit in 0^n is at most $1/(1-(1-1/(2n))^n) < 4$. Let $i \geq 1$ be the number of ones of the first such offspring. A second application of Lemma 3.1.8 shows that the expected number of steps to next reach some x^{**} with $|x^{**}|_1 \in \{0, n\}$ is bounded by $4i(n-i)$. Moreover, $\mathrm{Prob}(x^{**} = n) = i/n$, hence we return to 0^n with probability $1 - i/n$. This implies the following recursion, $p_{i,j}$ being a compact notation for $\mathrm{Prob}(X_{t+1} = j \mid X_t = i)$.

$$\mathrm{E}(T_{0\to n}) \leq 4 + \sum_{i=1}^{n} p_{0,i} \left(4i(n-i) + \left(1 - \frac{i}{n}\right) \cdot \mathrm{E}(T_{0\to n}) \right)$$

By rearranging we obtain

$$\mathrm{E}(T_{0\to n}) \;\leq\; \frac{4 + \sum_{i=1}^{n} p_{0,i} \cdot 4i(n-i)}{\sum_{i=1}^{n} p_{0,i} \cdot \frac{i}{n}} \;\leq\; 4n + \frac{\sum_{i=1}^{n} p_{0,i} \cdot 4in}{\sum_{i=1}^{n} p_{0,i} \cdot \frac{i}{n}} = 4n^2 + 4n.$$

This implies for $k = 0$ the bound $n^2 + 1/2 \cdot \mathrm{E}(T_{0\to n}) \leq 3n^2 + 2n$.

For $k > 0$ the time until the needle is found is clearly bounded by the expected time to reach the needle by a direct jump from 1^n. The probability for such a jump is $(1/(2n))^k \cdot (1-1/(2n))^{n-k} \geq e^{-1/2} \cdot (2n)^{-k}$, hence the expected number of trials is

$O((2n)^k)$. The expected return time from 1^n to 1^n is at most

$$\begin{aligned}
&4 + \sum_{i=0}^{n} p_{n,i} \left(4i(n-i) + \left(1 - \frac{i}{n}\right) \cdot \mathrm{E}(T_{0 \to n}) \right) \\
&\leq 4 + \sum_{i=0}^{n-1} p_{n,i} \left(4i(n-i) + (n-i)(4n+4)\right) \\
&\leq 4 + 8n \cdot \sum_{i=0}^{n-1} p_{n,i}(n-i).
\end{aligned}$$

The $\sum$-term describes the expected number of flipping bits when mutating 1^n, which can be more easily computed as $n \cdot 1/(2n) = 1/2$. This results in the bound $4n + 4$. As the expected number of steps between two trials is $O(n)$, we obtain the bound $O(n^2 + n \cdot (2n)^k) = O(n^2 + n^{k+1})$ for constant k. □

Remark 3.1.10. For an alternative proof of Theorem 3.1.9 for $k = 0$, we may exploit that the number of ones is a submartingale, until stopped at time $T := \inf\{t \mid X_t = n\}$. Similarly to Lemma 3.1.8, the following statement can be shown. If $\{X_t\}_{t \geq 0}$ is a submartingale and δ is a lower bound on the one-step variance, then $\mathrm{E}(T) \leq n^2/\delta$. An extension of this bound will be presented in Section 5.2.

In order to get a closer picture of the actual performance of the asymmetric (1+1) EA, we consider the results of 100 independent runs for $k = 0$ and $n \in \{100, 200, \ldots, 2000\}$. We report the number of runs where the unique global optimum was found within cn^2 steps for $c \in \{1, 2, \ldots, 6\}$ in Table 3.1.

Let $N := \{\textsc{Needle}_a \mid a \in \{0,1\}^n\}$ be the class of needle functions with the global optimum at some point $\overline{a}$ in the search space. It is known from the black-box complexity of function classes (Droste et al. (2003)) that every search heuristic needs at least $2^{n-1} + 1/2$ function evaluations on N on average. Thus, while the asymmetric (1+1) EA performs very well on $\textsc{Needle}_a$ with a close to 0^n or 1^n, it performs poorly on other functions $\textsc{Needle}_a$ with a far from 0^n and 1^n. This is another hint that the search process of the asymmetric (1+1) EA is clearly biased.

Note that the class $N = \{\textsc{Needle}_a \mid a \in \{0,1\}^n\}$ is closed under permutations of the search space. Thus, the same conclusion seems to be implied by the NFL: averaged over all such functions, all algorithms make an equal number of different function evaluations (Igel and Toussaint (2004)). However, this result has only limited relevance with respect to the expected optimization time since it does not take into account re-sampling of points in the search space.

n	$1n^2$	$2n^2$	$3n^2$	$4n^2$	$5n^2$	$6n^2$
100	72	90	95	100	100	100
200	73	95	100	100	100	100
300	75	90	97	99	100	100
400	78	97	99	99	100	100
500	74	92	96	99	99	100
600	73	92	96	99	100	100
700	74	93	97	100	100	100
800	74	96	100	100	100	100
900	73	96	99	100	100	100
1000	71	94	98	100	100	100
1100	71	90	95	97	100	100
1200	80	94	96	98	99	100
1300	67	90	97	99	100	100
1400	79	93	98	99	100	100
1500	70	91	99	100	100	100
1600	79	91	97	99	100	100
1700	71	89	99	99	100	100
1800	83	90	95	98	99	100
1900	75	94	97	98	99	100
2000	76	92	100	100	100	100

Table 3.1: Number of runs among 100 independent runs where the needle was found within the given time bound.

3.1.3 Analysis for Unimodal Functions

The results from Section 3.1.2 proved the asymmetric mutation operator to be advantageous for objective functions where good bit strings have either many or few 1-bits. Clearly, OneMax and Needle are both artificial examples that do not have much in common with problems encountered in applications. In order to gain a broader perspective, results on more general function classes are needed. We compare the asymmetric (1+1) EA with the standard (1+1) EA on the class of all unimodal functions. The class of unimodal functions is closed under the transformation of objective functions considered here. That is, for every $a \in \{0,1\}^n$, f_a is unimodal if and only if f is. Thus, again we can move the unique global optimum anywhere in the search space.

Recall from Definition 2.2.1 that a function is unimodal if and only if every non-optimal search point has a Hamming neighbor with strictly larger fitness. This implies that unimodal functions can be optimized via mutations of single bits, i. e., hill climbers are guaranteed to be successful. Starting with an arbitrary search point, there is a path of

Hamming neighbors to the unique global optimum with strictly increasing fitness. Note, however, that paths to the unique global optimum may be exponentially long, making such functions difficult to optimize. In fact, it is known that every search heuristic needs in the worst case an exponential number of function evaluations to optimize a unimodal function (Droste et al. (2006)).

Also recall the upper bound $O(nd)$ for the standard (1+1) EA on unimodal functions with d different function values from Theorem 2.3.4. Using Lemma 3.1.2, it is easy to obtain the same general upper bound for the asymmetric (1+1) EA.

Theorem 3.1.11. *Let $f\colon \{0,1\}^n \to \mathbb{R}$ be a unimodal function with d different function values. The expected optimization time of the asymmetric* (1+1) EA *on f is bounded above by $O(nd)$.*

Proof. We know from Lemma 3.1.2 that the probability of increasing the function value of the current search point is bounded below by $1/(8n)$. This yields $8n$ as upper bound on the expected time to increase the fitness. Clearly, at most $d-1$ fitness increases are sufficient to reach the global optimum. □

Asymmetric mutations deliver the same upper bound on an important class of functions as standard mutations. Of course, in both cases, the upper bound is not necessarily tight. However, it is known to be tight for standard mutations for some functions. One example of such a function is LeadingOnes, an example function where the standard (1+1) EA has expected optimization time $\Theta(n^2)$ (see Theorem 2.3.9). Moreover, deviations from the expected value by some constant factors are extremely unlikely. The use of asymmetric mutations, however, leads to a considerable speed-up. This is surprising as the bits following the leftmost 0-bit are subject to a random process and the standard (1+1) EA typically has to make $\Omega(n)$ improvements where a specific bit has to be flipped. For the asymmetric (1+1) EA, however, the situation is different. After some leading ones have been gathered, these bits bias the random process of the following bits towards bit value 1. This decreases the expected optimization time significantly.

Theorem 3.1.12. *The expected optimization time of the asymmetric* (1+1) EA *on* LeadingOnes *is $O(n^{3/2})$.*

Proof. We partition a run into two phases: the first phase starts with the beginning of the run and ends when some search point x^* with $\textsc{LeadingOnes}(x^*) \geq n^{1/2}$ is reached for the first time. The second phase starts after the first phase and ends when the global optimum is found.

Due to Lemma 3.1.2, the expected length of Phase 1 is bounded by $O(n^{3/2})$ since there always are Hamming neighbors with larger fitness and the fitness has to be increased at most $n^{1/2}$ times to reach the end of Phase 1.

For the investigation of Phase 2, we apply drift arguments to the number of zeros in the current search point. Let x be the current population and x' be the population of

the next generation. Then $x' = x$ or x' is an accepted mutant of x. A necessary and sufficient condition for the acceptance of x' is that the first LeadingOnes(x) 1-bits do not flip. Thus, we have

$$p := \text{Prob}\big(x' \text{ is accepted}\big) = \left(1 - \frac{1}{2\,|x|_1}\right)^{\text{LeadingOnes}(x)} \geq \frac{1}{2}$$

since $\text{LeadingOnes}(x) \leq |x|_1$. Moreover,

$$\text{E}\big(|x'|_0\big) \;=\; \text{E}\big(|x'|_0 \mid x' \text{ is accepted}\big) \cdot p + |x|_0 \cdot (1-p).$$

Since the first LeadingOnes(x) 1-bits cannot flip to 0 in an accepted step,

$$\begin{aligned}\text{E}\big(|x'|_0 \mid x' \text{ is accepted}\big) \;&=\; |x|_0 \cdot \left(1 - \frac{1}{2\,|x|_0}\right) + (|x|_1 - \text{LeadingOnes}(x)) \cdot \frac{1}{2\,|x|_1} \\ &=\; |x|_0 - \frac{\text{LeadingOnes}(x)}{2\,|x|_1}.\end{aligned}$$

Together, this results in

$$\begin{aligned}\text{E}\big(|x'|_0\big) \;&=\; \left(|x|_0 - \frac{\text{LeadingOnes}(x)}{2\,|x|_1}\right) \cdot p + |x|_0 \cdot (1-p) \\ &=\; |x|_0 - p \cdot \frac{\text{LeadingOnes}(x)}{2\,|x|_1} \\ &\leq\; |x|_0 - \frac{\text{LeadingOnes}(x)}{4\,|x|_1}.\end{aligned}$$

Hence,

$$\text{E}\big(|x|_0 - |x'|_0\big) \;\geq\; \frac{\text{LeadingOnes}(x)}{4\,|x|_1} \;\geq\; \frac{1}{4n^{1/2}}$$

and Theorem 2.3.8 yields an upper bound $n/(1/(4n^{1/2})) = 4n^{3/2}$ on the expected time to complete Phase 2. □

Like OneMax, the function LeadingOnes has the property that the unique global optimum is the all-one bit string 1^n. Obviously, this fosters the finding of the global optimum using asymmetric mutations. Therefore, it makes sense to investigate the expected optimization time on LeadingOnes_a. We would like to see whether there are some $a \in \{0,1\}^n$ such that the expected optimization time of the (1+1) EA using asymmetric mutations is $\omega(n^{3/2})$.

We consider a where the optimum has linear Hamming distance to both 0^n and 1^n. We will see that then the asymmetric (1+1) EA's tendency towards some kind of bits is more hindering than helpful.

Theorem 3.1.13. *Given some constant $0 < c < 1$, let $a \in \{0,1\}^n$ be chosen uniformly at random among all search points with cn 1-bits. The expected (w. r. t. the random bits of a and the algorithm's decisions) optimization time of the asymmetric* (1+1) EA *on* $\textsc{LeadingOnes}_a(x)$ *is* $\Theta(n^2)$.

Remark 3.1.14. If a is drawn uniformly at random from the whole search space $\{0,1\}^n$, a has a linear number of 1-bits and 0-bits with overwhelming probability and then Theorem 3.1.13 applies. If this is not the case, we may use the known bounds 0 and $O(n^2)$. We conclude that the bound $\Theta(n^2)$ also holds in this setting.

Proof of Theorem 3.1.13. The upper bound follows from Theorem 3.1.11. The main proof idea for the lower bound is to consider pairs of neighbored bits in a, say $a_i a_{i+1}$, where $a_i \neq a_{i+1}$. As long as the current LeadingOnes-value is less than $i-1$, for the current search point x the assignment $x_i x_{i+1} = 01$ has the same probability as $x_i x_{i+1} = 10$. Hence, both bits are gained as free riders only with probability at most 1/2. Using drift arguments similar to Theorem 2.3.9 yields the claimed lower bound.

W. l. o. g., n is even and $c \leq 1/2$. Let $k = cn$. Call a pair of bits a_i, a_{i+1} in a a transition if $a_i \neq a_{i+1}$. Let T be the number of transitions in a and T_ℓ, T_r be the number of transitions within the bits $a_1 \dots a_{n/2}$ and $a_{n/2+1} \dots a_n$, resp. We first estimate $\mathrm{E}(T)$. We have $\binom{n}{k}$ possibilities to choose a. The event $a_i \neq a_{i+1}$ for some $1 \leq i \leq n-1$ occurs if exactly one of these variables is 1 and the remaining $k-1$ 1-bits are distributed among the remaining $n-2$ bits. Hence

$$\mathrm{Prob}(a_i \neq a_{i+1}) \;=\; 2\binom{n-2}{k-1}\binom{n}{k}^{-1} \;=\; \frac{2k(n-k)}{n(n-1)}.$$

By the linearity of expectation, along with $k \leq n/2$, we have

$$\mathrm{E}(T) \;=\; (n-1)\cdot\frac{2k(n-k)}{n(n-1)} \;=\; \frac{2k(n-k)}{n} \;\geq\; k.$$

Due to symmetry, T_ℓ and T_r are due to the same probability distribution and since $(a_{n/2}, a_{n/2+1})$ is excluded, we have $\mathrm{E}(T_\ell) = \mathrm{E}(T_r) \geq \mathrm{E}(T)/2 - 1 \geq k/2 - 1$.

We claim that T_ℓ and T_r are strongly concentrated. Observing

$$|\mathrm{E}(T_\ell \mid a_1, \dots, a_i) - \mathrm{E}(T_\ell \mid a_1, \dots, a_{i-1})| \;\leq\; 1$$

we can apply the method of bounded martingale differences (Lemma A.8) yielding

$$\mathrm{Prob}\left(T_\ell \leq \frac{k}{2} - \frac{k}{4}\right) \;\leq\; e^{-\left(\frac{k}{4}-1\right)^2/n} \;=\; e^{-\Omega(n)}.$$

Hence, with overwhelming probability $T_\ell \geq k/4$ and $T_r \geq k/4$.

Assuming $T_\ell \geq k/4$ implies that both the number of 1-bits and the number of 0-bits among the bits $a_1, \ldots, a_{n/2}$ is bounded below by $k/8$ as every bit contributes to at most 2 transitions. Recalling that $\overline{a}$ is the global optimum, a direct consequence is that whensoever a search point x with $\textsc{LeadingOnes}_a(x) \geq n/2$ is mutated, the mutation probability for a specific bit is at most $4/k$.

Let x^* be the first search point reached during the optimization process, where $\textsc{LeadingOnes}_a(x^*) \geq n/2$. We bound the expected optimization time by the expected time to find the optimum starting with x^*. Let

$$d(x) := |\{i \mid \textsc{LeadingOnes}_a(x) + 1 < i < n \wedge a_i \neq a_{i+1}\}|$$

be the number of transitions to the right of the leftmost bit differing in x and $\overline{a}$. We now apply drift analysis arguments in order to estimate the expected time until the current search point's d-value has decreased to 0, which is necessary to find the optimum.

Let x be the current search point and x' be the search point in the next generation. A necessary condition for $d(x') < d(x)$ is that the leftmost differing bit flips, which has probability at most $4/k$. Moreover, the positions of the bits to the right have not yet had any influence on the fitness up to now, hence for every j with $\textsc{LeadingOnes}_a(x) + 1 < j < n$

$$\text{Prob}(x_j x_{j+1} = 01) \;=\; \text{Prob}(x_j x_{j+1} = 10) \;\leq\; \frac{1}{2}.$$

We observe that, in case the $\textsc{LeadingOnes}_a$-value increases, the two bits of the following transition both match the optimum $\overline{a}$ with probability at most $1/2$. Transitions may overlap, however, two matching bits can decrease the d-value by at most 2. In case both these bits match, we consider the first transition in $x_{j+2} \ldots x_n$ and repeat the argumentation with independent events. If the considered bits do not both match, the d-value decreases by at most 2. We arrive at

$$\mathrm{E}\big(d(x) - d(x')\big) \;\leq\; \frac{4}{k} \cdot \left(2 \sum_{i=0}^{\infty} 2^{-i}\right) \;=\; \frac{16}{k}.$$

By the same arguments, $\mathrm{E}(d(x^*)) \geq T_r - 4 \geq k/4 - 4$ follows and drift analysis arguments from Theorem 2.3.8 yield the bound

$$\frac{\mathrm{E}(d(x^*))}{16/k} \;\geq\; \left(\frac{k}{4} - 4\right) \cdot \frac{k}{16} \;=\; \frac{c^2 n^2}{64} - O(n).$$

Note that a bound $\Omega(n^2)$ also holds when multiplying with the probability $1 - e^{-\Omega(n)}$ for $T_\ell, T_r \geq k/4$ and we have proved the theorem. □

Here, we consider experiments for $\textsc{LeadingOnes}_a$ for $n \in \{40, 80, \ldots, 600\}$ and different bit strings a. We fix $|a|_1 \in \{0, .05n, .1n, \ldots, .95n, n\}$ and choose one such a uniformly

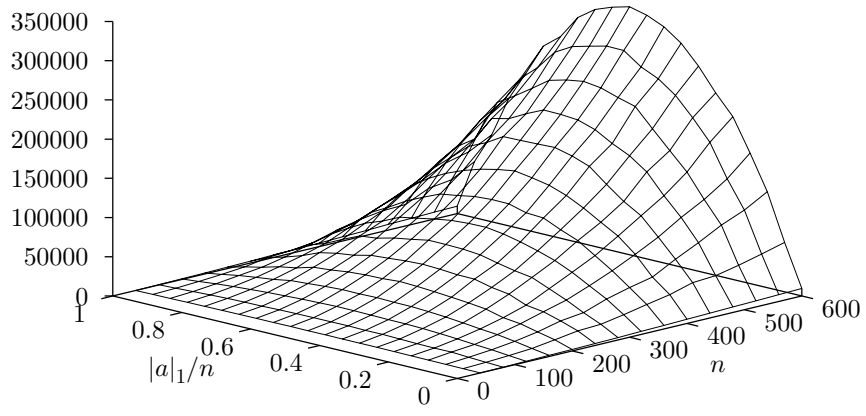

Figure 3.3: Average runtimes on LeadingOnes$_a$ in 100 independent runs for $n \in \{40, 80, \ldots, 600\}$ and $|a|_1 \in \{0, .05n, .1n, \ldots, .95n, n\}$.

at random for each value of $|a|_1$. In Figure 3.3 we observe a clear increase of the average optimization time as $|a|_1$ moves towards $n/2$.

When looking for a known example function where the general upper bound for unimodal functions is tight, without applying any transformation, one may think of Ridge (Quick et al. (1998)). Whereas the unique global optimum is 1^n (like for OneMax and LeadingOnes), the algorithm cannot benefit from additional 1-bits that happen to be present in the trailing bits. The definition of Ridge forces these bits to be 0 on the ridge. This makes it seem unlikely that the asymmetric (1+1) EA can outperform the standard (1+1) EA on Ridge. We prove this intuition to be correct.

Theorem 3.1.15. *The expected optimization time of the asymmetric* (1+1) EA *on* Ridge *is* $\Theta(n^2)$. *The same holds for* Ridge$_a$ *and every* $a \in \{0,1\}^n$.

Proof. The upper bound follows from Theorem 3.1.11. With probability $1 - 2^{-\Omega(n)}$ the initial search point has Hamming distance at least $n/3$ from the unique global optimum. Offspring closer to the optimum with a fitness value smaller than $n + 1$ are rejected. Thus, with probability $1 - 2^{-\Omega(n)}$ the first accepted search point x^* where $f(x^*) \geq n+1$ has Hamming distance $\Omega(n)$ to the unique global optimum.

Let $S = (s_0, \ldots, s_{n-1})$ be the sequence of Hamming neighbors such that $f(s_i) = n + i$ for all $0 \leq i \leq n - 1$. Then for every a there is a coherent subsequence $S' = (s'_1, \ldots, s'_m)$ of S of length $m = \Omega(n)$ such that $f(s'_1) \geq f(x^*)$ and both $|s'_i|_1 = \Omega(n)$ and $|s'_i|_0 = \Omega(n)$ hold for all $1 \leq i \leq m$. Due to the definition of Ridge$_a$, this subsequence has to be traversed in order to optimize Ridge$_a$. The expected decrease in Hamming distance to

the global optimum on this subsequence in one mutation is $O(1/n)$. Hence, applying drift arguments yields the lower bound $\Omega(n^2)$ on the expected optimization time. □

Considering the results of experiments for RIDGE, designed in the same way as the experiments for LEADINGONES, our asymptotical bounds are confirmed (see Figure 3.4). There is hardly any difference for different choices of a.

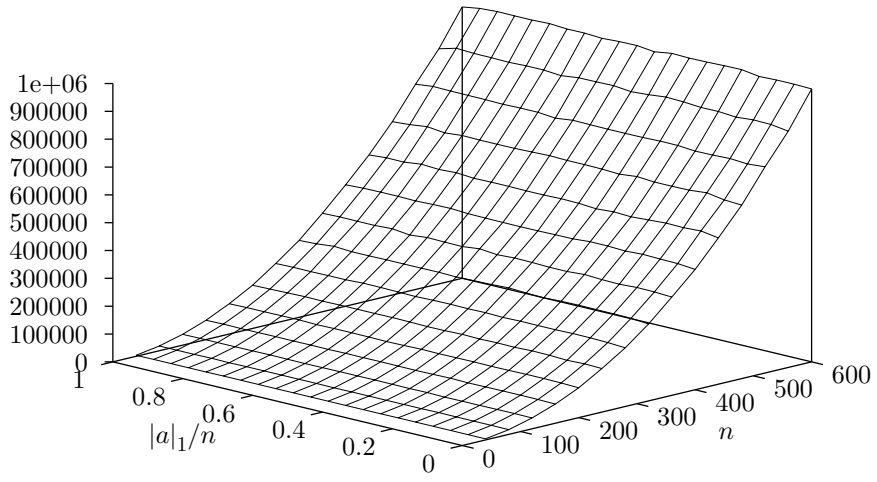

Figure 3.4: Average runtimes on RIDGE_a in 100 independent runs for $n \in \{40, 80, \ldots, 600\}$ and $|a|_1 \in \{0, .05n, .1n, \ldots, .95n, n\}$.

The performance of the standard (1+1) EA and the asymmetric (1+1) EA are asymptotically equal on RIDGE_a. Even the proofs of the bounds are very similar (Jansen and Wegener (2001)). So far, we have seen only advantages for asymmetric mutations and many similarities to standard mutations. In the following section, we consider an example where the asymmetric mutation operator leads to an extreme decline in performance.

3.1.4 Drawbacks of the Asymmetric Mutation Operator

The function PLATEAU is very similar to RIDGE. The function values differ only for n out of 2^n points in the search space. These n points are the most important ones, though. For RIDGE, the increase in function values on the ridge leads towards the global optimum. For PLATEAU, these points have equal function values and the evolutionary algorithm has to perform a kind of blind random walk on this plateau. It is known that standard mutations complete this random walk successfully on average in $\Theta(n^3)$ steps. Asymmetric mutations fail to be efficient in any sense here.

Theorem 3.1.16. *The probability that the asymmetric* (1+1) EA *optimizes* PLATEAU *within* 2^{cn} *steps is bounded above by* $2^{-\Omega(n)}$ *for some constant* $c > 0$.

Proof. We first show that with probability $1 - 2^{-\Omega(n)}$ the first point on the plateau reached by the algorithm has at most $n/16$ 1-bits. As long as the plateau has not been hit, the asymmetric (1+1) EA behaves as on the function $\textsc{ZeroMax}(x) := |x|_0$. Due to the symmetry between x and $\overline{x}$, we can apply bounds for ONEMAX also for ZEROMAX. We know from Theorem 3.1.6 that the expected time until 0^n is found or the plateau is hit is $O(n)$. As long as the current search point of the asymmetric (1+1) EA on PLATEAU has the same distribution as on ZEROMAX the following holds. Let x_t be the search point in generation t. If we know that x_t has i 1-bits, then x_t is uniform over all search points with i 1-bits. Therefore, the probability that $x_t = 1^i 0^{n-i}$ is $1/\binom{n}{i}$, which is $2^{-\Omega(n)}$ if $i \geq n/32$ and $i \leq n - n/32$. A situation where $i > n - n/32$ is very unlikely as the probability that initialization creates a search point with at most $n - n/16$ 1-bits is $1 - 2^{-\Omega(n)}$ and then $n/32$ bits are required to flip simultaneously, which has probability $2^{-\Omega(n)}$ as well. By the last argument, also in case a search point with at most $n/32$ 1-bits is reached, at least $n/32$ bits would be required to flip in order to reach a plateau point with at least $n/16$ 1-bits. Considering all mentioned error probabilities for the initialization and using Lemma 2.3.6 for $O(n)$ expected generations results in a total error probability of $2^{-\Omega(n)}$. Therefore, the first plateau point has at most $n/16$ 1-bits with probability $1 - 2^{-\Omega(n)}$.

Assume in the following that the above event happened. We show that then there is a drift towards decreasing the number of 1-bits. Let x_t be the current search point on the plateau in generation t and x_{t+1} be the current search point in the following generation. Abbreviate $\Delta_i := (|x_{t+1}|_0 - |x_t|_0 \mid |x_t|_0 = i)$. For all $7n/8 < i < 15n/16$, we have

$$\begin{aligned} \mathrm{E}(\Delta_i) &= \sum_{d=-|x_t|_0}^{n-|x_t|_0} d \cdot \mathrm{Prob}(\Delta_i = d) \\ &\geq \mathrm{Prob}(\Delta_i = 1) - \sum_{d=1}^{|x_t|_0} d \cdot \mathrm{Prob}(\Delta_i = -d). \end{aligned}$$

For $\Delta_i = 1$ it is sufficient to reach a specific Hamming neighbor by flipping a 1-bit. Lemma 3.1.2 therefore yields $\mathrm{Prob}(\Delta_i = 1) \geq 1/(8(n-i))$. A necessary condition for $\Delta_i = -d$ is that d specific 0-bits flip, which has probability $(1/(2i))^d$. Together, using $i > 7n/8$ and $d \leq (\sqrt{2})^d$ for $d \in \mathbb{N}$,

$$\begin{aligned} \mathrm{E}(\Delta_i) &\geq \frac{1}{8(n-i)} - \sum_{d=1}^{|x_t|_0} d \cdot \left(\frac{1}{2i}\right)^d \\ &\geq \frac{1}{8(n-i)} - \sum_{d=1}^{\infty} \left(\frac{1}{\sqrt{2}i}\right)^d \end{aligned}$$

$$= \frac{1}{8(n-i)} - \frac{1}{\sqrt{2}i - 1} \geq \frac{1}{n} - \frac{8}{\sqrt{2} \cdot 7n} = \Omega\left(\frac{1}{n}\right).$$

We conclude that for the observed part of the plateau there is a slight tendency to decrease the number of 1-bits. Call a step *relevant* if the offspring y replaces its parent $x \neq y$ on the plateau. In all non-relevant steps the asymmetric (1+1) EA does not change its current search point, i. e., a relevant step is equivalent to $\Delta_i \neq 0$. The probability of a relevant step is of order $\Theta(1/n)$ due to the conditions on i. By the law of total probability (Lemma A.2)

$$\mathrm{E}(\Delta_i \mid \Delta_i \neq 0) = \frac{\mathrm{E}(\Delta_i)}{\mathrm{Prob}(\Delta_i \neq 0)} \geq \varepsilon$$

for all considered i and a suitable constant $\varepsilon > 0$.

We aim for an application of the simplified Drift Theorem to the number of 0-bits in relevant steps. Let $a := 7n/8$ and $b := 15n/16$, then the first condition has been shown. The unconditional probability of a step decreasing the number of 0-bits by $1 \leq d \leq i$ is $(1/(2i))^d \leq (4/(7n))^d$. The conditional probability of this event, given a relevant step, is by a factor of $\Theta(n)$ larger. Still, $(4/(7n))^d \cdot \Theta(n) \leq 2^{-d+O(1)}$ for $d \geq 1$, hence also the second condition of Theorem 2.3.10 is fulfilled for $\delta := 1$ and an appropriate constant r. Theorem 2.3.10 yields that the optimization time is at least 2^{cn} for some constant $c > 0$ with probability $1 - 2^{-\Omega(n)}$. This still holds if we take into account the error probability for the first plateau point and the theorem has been proven. □

Note, however, that this immense drawback is due to the special definition of Plateau. In particular, we can transform the landscape in a way that does not influence the standard (1+1) EA at all but is important for the asymmetric (1+1) EA. This leads to a function for which we can prove bounds of equal order on the expected optimization time of the two algorithms.

Theorem 3.1.17. *For even n we define $a_{01} := 010101 \cdots 01 \in \{0,1\}^n$. The expected optimization time of the asymmetric* (1+1) EA *on* $\textsc{Plateau}_{a_{01}}$ *is* $\Theta(n^3)$.

Proof. It follows from the result on $\textsc{OneMax}_a$ (Theorem 3.1.7) that some point on the plateau will be found on average within the first $O(n \log n)$ steps. Then, the plateau cannot be left again. For each $i \in \{0, \ldots, n-1\}$ and some search point x on the plateau we show the following claim. If it is possible to create search points x^{+i}, x^{-i} on the plateau out of x such that the Hamming distance to the unique global optimum is increased or decreased by i, resp., then x^{+i} and x^{-i} are reached with equal probability. This is due to the choice of a_{01} since all plateau points of $\textsc{Plateau}_{a_{01}}$ have either $n/2$ or $(n/2)+1$ 1-bits and points with $n/2$ and $(n/2)+1$ 1-bits are alternating on the plateau. Therefore, $\left|x^{+i}\right|_1 = \left|x^{-i}\right|_1$ holds and both the number of flipping 1-bits and the number of flipping 0-bits is the same for x^{+i} and x^{-i}. Furthermore, the choice of a_{01} implies that

$|x|_1 = n/2 + O(1)$ for all x on the plateau yielding a probability of $\Theta(1/n^i)$ to create x^{+i} or x^{-i}.

By these arguments, a bound $\Theta(n^3)$ on the expected optimization time can be shown analogously to the results by Jansen and Wegener (2001) and Brockhoff et al. (2007). □

We consider experiments for PLATEAU_{a01} measuring the average runtime of the asymmetric (1+1) EA over 100 runs for each $n \in \{10, 20, \ldots, 200\}$. The results are shown in Figure 3.5. A regression analysis with functions cn^3 yielded a good fit for $c = 1.33951$.

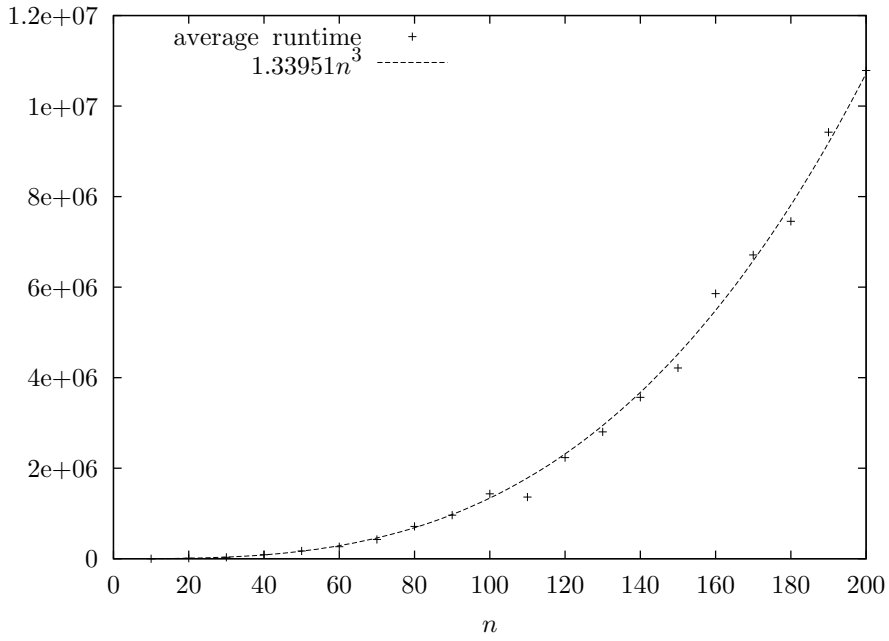

Figure 3.5: Average runtimes on PLATEAU_{a01} in 100 independent runs for $n \in \{10, 20, \ldots, 200\}$. The dashed line shows the fitted function $1.33951n^3$.

Next, we consider experiments for the asymmetric (1+1) EA on the function PLATEAU. Since our theoretical result predicts overly large runtimes, we measure the number of runs where the optimum is found within $16n^3$ steps. The factor 16 is chosen for the following reasons according to the empirical results on PLATEAU_{a01}. First, this factor is more than ten times larger than the factor $c = 1.33951$ of the fitted function for PLATEAU_{a01}. Moreover, all runs on PLATEAU_{a01} found the global optimum within $16n^3$ steps.

The results on PLATEAU are shown in Table 3.2. The search space dimension $n = 10$ is too small for the bias of asymmetric mutations to have a significant impact on the plateau. However, for larger n the asymmetric (1+1) EA clearly fails on PLATEAU.

n	successful runs
10	60
20	0
30	0
40	0
50	0
60	0
70	0
80	0
90	0
100	0

n	successful runs
110	0
120	0
130	0
140	0
150	0
160	0
170	0
180	0
190	0
200	0

Table 3.2: Number of runs among 100 independent runs where the optimum of PLATEAU was found within $16n^3$ generations.

3.1.5 Conclusions

We presented a mutation operator for bit strings that flips bits with a probability depending on the number of 1-bits. The operator is designed in a way that on average the number of 1-bits is not changed. This helps to bias the search towards bit strings containing either very few 0-bits or very few 1-bits. Such a mutation operator is motivated by applications where good solutions are known or at least thought of having this property.

We analyzed this mutation operator by comparing it with standard mutations that flip each bit independently with probability $1/n$. Table 3.3 gives an overview on bounds for the corresponding expected optimization times. For ONEMAX, a speed-up of order $\log n$ is proved. For NEEDLE, there is even an exponential advantage for asymmetric mutations. For the class of unimodal functions, we proved the same general upper bound as known for standard mutations. For LEADINGONES, a speed-up of order $n^{1/2}$ is proved in comparison to standard mutations. However, both mutation operators lead to runtimes of equal order on a simple transformation of LEADINGONES and on a class of ridge functions. These results show that the general upper bound for unimodal functions can be tight and that both algorithms can have similar performance on broad classes of functions.

Contrarily, we demonstrated a clear weakness of asymmetric mutations on a function where an unbiased random walk on a plateau is needed in order to be successful. We showed that there is an exponential performance gap between asymmetric mutations and standard mutations. However, a simple transformation of the landscape lets both mutation operators lead to polynomial expected optimization times.

	standard	asymmetric	asymmetric and transformed	
General lower bound	$\Omega(n \log n)$	$\Omega(n)$	$\Omega(n)$	(every a)
General upper bound	n^n	$(2n)^n$	$(2n)^n$	(every a)
OneMax	$\Theta(n \log n)$	$\Theta(n)$	$\Theta(n \log(k+2))$	(every a)
Needle	$\Theta(2^n)$	$O(n^2)$	$O(n^2 + n^{k+1})$	(constant k)
Unimodal functions	$O(nd)$	$O(nd)$	$O(nd)$	(every a)
LeadingOnes	$\Theta(n^2)$	$O(n^{3/2})$	$\Theta(n^2)$	(random a)
Ridge	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	(every a)
Plateau	$\Theta(n^3)$	$2^{\Omega(n)}$	$\Theta(n^3)$	($a = 0101\ldots$)

Table 3.3: Bounds on the expected optimization times for the standard (1+1) EA, the asymmetric (1+1) EA, and the asymmetric (1+1) EA on functions transformed according to a with $k := \min\{|a|_0, |a|_1\}$. Conditions on the choice of a are indicated. Recall that bounds for the standard (1+1) EA also hold for every transformation. The general lower bound applies to functions with a bounded number of global optima and the general upper bound holds for all functions. For unimodal functions d denotes the number of fitness values.

Our results show that the asymmetric mutation operator introduces a remarkable bias on the search process that is, nevertheless, easy to counteract by fitness-based guidance. The results on the flat fitness landscape Needle_a (Theorem 3.1.9) and Plateau (Theorem 3.1.16) demonstrate that, in the absence of a fitness-based guidance, the random search induced by asymmetric mutations is different from the random search induced by standard mutations. On the other hand our result for OneMax_a (Theorem 3.1.7) shows that fitness-based guidance is easily able to counteract this search bias. There, asymmetric mutations improve the performance if the number of ones in the target string is either very large or very small. But from the perspective of asymptotic optimization times, the performance is never worse than with standard mutations. In this sense the asymmetric mutation operator is a successful example of a variation operator inspired by a specific class of applications that is not limited in its use to this class.

3.2 Diversity Mechanisms for Global Exploration

The term *diversity* indicates dissimilarities of individuals in evolutionary computation and is considered an important property. In a population-based evolutionary algorithm without diversity mechanism there is a risk of the best individual taking over the whole population before the fitness landscape is explored properly. When the population becomes completely redundant, the algorithm basically reduces to a trajectory-based al-

gorithm while still suffering from high computational effort and space requirements for the whole population.

Diversity mechanisms can help the optimization in two ways. On one hand, a diverse population is able to deal with multimodal functions and can explore several hills in the fitness landscape simultaneously. Diversity mechanisms can therefore support global exploration and help to locate several local and global optima. In particular, this behavior is welcome in dynamic environments as the algorithm is more robust with respect to changes of the fitness landscape. Moreover, the algorithm can offer several good solutions to the user, a feature desirable in multiobjective optimization. On the other hand, a diverse population gives higher chances to find dissimilar individuals and to create good offspring by recombining different "building blocks." Diversity mechanisms can thus enhance the performance of crossover.

Up to now, the use of diversity mechanisms has been assessed mostly by means of empirical investigations (e. g., Chaiyaratana, Piroonratana, and Sangkawelert, 2007; Ursem, 2002). Theoretical runtime analyses involving diversity mechanisms mostly use these mechanisms to enhance the performance of crossover. Jansen and Wegener (2005) presented the first proof that crossover can make a difference between polynomial and exponential expected optimization times. They used a very simple diversity mechanism that only shows up as a tie-breaking rule: when there are several individuals with minimal fitness among parents and offspring, the algorithm removes those individuals with a maximal number of genotype duplicates. Nevertheless, this mechanism makes the individuals spread on a certain fitness level such that crossover is able to find suitable parents for recombination. Storch and Wegener (2004) presented a similar result for populations of constant size. They used a stronger mechanism that prevents duplicates from entering the population, regardless of their fitness value.

Niching methods encourage the exploration of "niches", that is, they aim at the survival of individuals far apart from the other individuals. The first theoretical runtime analysis considering niching methods was presented by Fischer and Wegener (2005) for a fitness function derived from a generalized Ising model on ring graphs. The authors compare the well-known (1+1) EA with a (2+2) GA with fitness sharing. Fitness sharing (Mahfoud, 1997) derates the real fitness of an individual x by a measure related to the similarity of x to all individuals in the population, hence encouraging the algorithm to decrease similarity in the population. Fischer and Wegener prove that their genetic algorithm outperforms the (1+1) EA by a polynomial factor. Sudholt (2005) extended this study for the Ising model on binary trees, where the performance gap between GAs and EAs is even larger. While a broad class of $(\mu+\lambda)$ EAs has exponential expected optimization time, a (2+2) GA with fitness sharing finds a global optimum in expected polynomial time.

In all these studies diversity is used to assist crossover. Contrarily, Friedrich, Hebbinghaus, and Neumann (2007) focused on diversity mechanisms as a means to enhance the global exploration of EAs without crossover. Using rigorous runtime analyses, the

authors compare a mechanism avoiding genotype duplicates with a strategy avoiding duplicate fitness values to spread individuals on different fitness levels. It is shown for artificial functions that both mechanisms can outperform one another drastically.

Friedrich et al. (2007) were the first to focus on the use of diversity mechanisms for global exploration, with respect to rigorous runtime analyses. However, their test functions are clearly tailored towards one particular diversity mechanism. We want to obtain a broader perspective including a broader range of diversity mechanisms. Therefore, we compare several well-known diversity mechanisms on the simplest bimodal function that may also appear as part of a real-world problem. On the considered function simple hill climbers find the global optimum with constant probability, hence a restart strategy is sufficient for optimization. We focus on well-known diversity mechanisms that do not restart the algorithm. Firstly, we rigorously prove that diversity mechanisms are necessary for our function since populations of almost linear size without diversification fail to find both peaks, with high probability. Then we analyze common diversity mechanisms and show that not all of them are effective to prevent the population from collapsing to the local optimum, even for such a simple landscape. As a result, we hope to get a more objective and more general impression of the capabilities and limitations of common diversity mechanisms.

In the remainder of this section, we first present our bimodal test function in Section 3.2.1. Negative results for a plain (μ+1) EA in Section 3.2.2 show that diversification is needed. In Sections 3.2.3 and 3.2.4 we investigate the strategies previously analyzed in Friedrich et al. (2007) to avoid genotype duplicates and fitness duplicates, respectively. Section 3.2.5 deals with the well-known deterministic crowding strategy (Mahfoud, 1997) where offspring directly compete with their associated parents. Fitness sharing, which turns out to be the strongest mechanism, is analyzed in Section 3.2.6. Section 3.2.7 contains experimental results showing how well our theoretical results match with empirical results and revealing additional insight on the dynamic behavior of the algorithms. We present our conclusions in Section 3.2.8.

3.2.1 A Simple Bimodal Function

We consider a simple bimodal function called TwoMax that has already been investigated in the context of genetic algorithms by Pelikan and Goldberg (2000) and Goldberg, Van Hoyweghen, and Naudts (2002). The function TwoMax is essentially the maximum of OneMax and ZeroMax. Local optima are solutions 0^n and 1^n where the number of zeros or the number of ones, respectively, is maximized. Hence, TwoMax can be seen as a bimodal equivalent of OneMax. The fitness landscape consists of two hills with symmetric slopes, i. e., an unbiased random search heuristic cannot tell in advance which hill is more promising. In contrast to Pelikan and Goldberg (2000) and Goldberg et al. (2002) we modify the function slightly such that only one hill contains the global

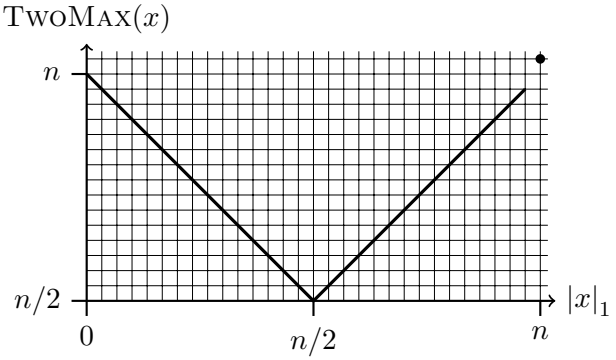

Figure 3.6: Sketch of TwoMax. The dot indicates the global optimum.

optimum, while the other one leads to a local optimum. This is done by simply adding an additional fitness value for 1^n, turning it into a unique global optimum.

For $x = x_1 x_2 \dots x_n$

$$\textsc{TwoMax}(x) \;:=\; \max\{|x|_0, |x|_1\} + \prod_{i=1}^{n} x_i.$$

Figure 3.6 shows a sketch of TwoMax. Among all search points with more than $n/2$ 1-bits, the fitness increases with the number of ones. Among all search points with less than $n/2$ 1-bits, the fitness increases with the number of zeros. We refer to these sets as *branches* and the algorithms as climbing these two branches of TwoMax.

The TwoMax function does appear in well-known combinatorial optimization problems. For example, the Vertex Cover bipartite graph analyzed in Oliveto, He, and Yao (2008) consists of two branches, one leading to a local optimum and the other to the minimum cover. In fact similar proof techniques as those used in this paper have also been applied in the Vertex Cover analysis of the (μ+1) EA for the bipartite graph. Another function with a similar structure is the Mincut instance analyzed in Section 4.2.

Simple hill climbers like the (1+1) EA find the global optimum of TwoMax before the local one with probability 1/2, hence they are efficient with probability 1/2. Considering μ independent parallel runs of the (1+1) EA (or μ starts of the algorithm with a suitable restart scheme) yields that the probability of not finding the global optimum efficiently is $2^{-\mu}$. When using populations of size μ, we expect the search to be more focused than with μ independent runs. Hence without diversification the probability of finding the optimum should be significantly worse. However, from a good diversity mechanism we expect that the probability of not finding the optimum decreases significantly with growing μ. We will see in the following that diversity mechanisms avoiding duplicates do not fulfill this property and that the probability of finding the global optimum efficiently

is at most $1/2 + o(1)$, hence comparable to the simple (1+1) EA. We can conclude for these mechanisms that populations are nearly useless on TwoMax. On the other hand, deterministic crowding shows a behavior very similar to μ independent runs and fitness sharing even finds the optimum efficiently with probability 1.

3.2.2 No Diversity Mechanism

In order to obtain a fair comparison of different diversity mechanisms, we keep one algorithm fixed as much as possible. The basic algorithm, the following (μ+1) EA, has already been investigated by Witt (2006). The (μ+1) EA uses random parent selection and elitist selection for survival. As parents are chosen randomly, the impact of selection is quite weak. Nevertheless, the (μ+1) EA is not able to maintain individuals on both branches for a long time. We now show that if μ is not too large, the individuals on one branch typically get extinct before the top of the branch is reached. Thus, the global optimum is found only with probability close to 1/2 and the expected optimization time is very large.

Algorithm 5 (μ+1) EA

Let $t := 0$.
Initialize P_0 with μ individuals chosen uniformly at random.
repeat
  Choose $x \in P_t$ uniformly at random.
  Create y by flipping each bit in x independently with probability $1/n$.
  Choose $z \in P_t$ with minimal fitness.
  if $f(y) \geq f(z)$ **then** $P_{t+1} := P_t \setminus \{z\} \cup \{y\}$
    else $P_{t+1} := P_t$.
  Let $t := t + 1$.

Theorem 3.2.1. *The probability that the* (μ+1) EA *with no diversity mechanism and* $\mu = o(n/\log n)$ *optimizes* TwoMax *in time* n^{n-1} *is at most* $1/2 + o(1)$. *Its expected optimization time is* $\Omega(n^n)$.

Proof. The probability that during initialization either 0^n or 1^n is created is bounded by $\mu \cdot 2^{-n+1}$, hence exponentially small. In the following, we assume that such an atypical initialization does not happen as this assumption only introduces an error probability of $o(1)$.

Consider the algorithm at the first point of time T^* where either 0^n or 1^n is created. Due to symmetry, the local optimum 0^n is created with probability 1/2. We assume in the following that 0^n is created and keep in mind an error probability of 1/2. We now show that then with high probability 0^n takes over the population before the global

optimum 1^n is created. Let i be the number of copies of 0^n in the population. From the perspective of extinction, a good event G_i is to increase this number from i to $i+1$. We have

$$\text{Prob}(G_i) \;\geq\; \frac{i}{\mu} \cdot \left(1 - \frac{1}{n}\right)^n \;\geq\; \frac{i}{4\mu}$$

since it suffices to select one out of i copies and to create another copy of 0^n. On the other hand, the bad event B_i is to create 1^n in one generation. This probability is maximized if all $\mu - i$ remaining individuals contain $n-1$ ones:

$$\text{Prob}(B_i) \;\leq\; \frac{\mu - i}{\mu} \cdot \frac{1}{n} \;<\; \frac{1}{n}.$$

Together, the probability that the good event G_i happens before the bad event B_i is

$$\text{Prob}(G_i \mid G_i \cup B_i) \;\geq\; \frac{\text{Prob}(G_i)}{\text{Prob}(G_i) + \text{Prob}(B_i)}. \tag{3.6}$$

Plugging in bounds for $\text{Prob}(B_i)$ and $\text{Prob}(G_i)$ yields the bound

$$\frac{i/(4\mu)}{i/(4\mu) + 1/n} \;=\; 1 - \frac{1/n}{i/(4\mu) + 1/n} \;\geq\; 1 - \frac{4\mu}{in}.$$

The probability that 0^n takes over the population before the global optimum is reached is therefore bounded by

$$\prod_{i=1}^{\mu} \text{Prob}(G_i \mid G_i \cup B_i) \;\geq\; \prod_{i=1}^{\mu} \left(1 - \frac{4\mu}{in}\right).$$

Using $4\mu/n \leq 1/2$ and $1 - x \geq e^{-2x}$ for $x \leq 1/2$ (cf. Lemma A.12), we obtain

$$\begin{aligned}
\prod_{i=1}^{\mu} \left(1 - \frac{4\mu}{in}\right) \;&\geq\; \prod_{i=1}^{\mu} \exp\left(-\frac{8\mu}{in}\right) \;=\; \exp\left(-\frac{8\mu}{n} \cdot \sum_{i=1}^{\mu} \frac{1}{i}\right) \\
&\geq\; \exp(-O((\mu \log \mu)/n)) \\
&\geq\; 1 - O((\mu \log \mu)/n) \;=\; 1 - o(1).
\end{aligned}$$

If the population consists of copies of 0^n, mutation has to flip all n bits to reach the global optimum. This event has probability n^{-n} and, by the union bound, the probability of this happening in a phase consisting of n^{n-1} generations is at most $1/n$. The sum of all error probabilities is $1/2 + o(1)$, which proves the first claim.

For the second claim, observe that the conditional expected optimization time is n^n once the population has collapsed to copies of 0^n. As this situation occurs with probability at least $1/2 - o(1)$, the unconditional expected optimization time is $\Omega(n^n)$. □

3.2.3 No Genotype Duplicates

It has become clear that diversity mechanisms are very useful to optimize even a simple function such as TwoMax. The simplest way to enforce diversity within the population is not to allow genotype duplicates. The following algorithm has been defined and analyzed by Storch and Wegener (2004). It prevents identical copies from entering the population as a natural way of ensuring diversity. We will, however, show that this mechanism is not powerful enough to explore both branches of TwoMax.

Algorithm 6 (μ+1) EA with genotype diversity

Let $t := 0$.
Initialize P_0 with μ individuals chosen uniformly at random.
repeat
  Choose $x \in P_t$ uniformly at random.
  Create y by flipping each bit in x independently with probability $1/n$.
  if $y \notin P_t$ **then**
    Choose $z \in P_t$ with minimal fitness.
    if $f(y) \geq f(z)$ **then** $P_{t+1} := P_t \setminus \{z\} \cup \{y\}$
      else $P_{t+1} := P_t$.
  Let $t := t + 1$.

We prove that if the population is not too large, the algorithm can be easily trapped in a local optimum.

Theorem 3.2.2. *The probability that the* (μ+1) EA *with genotype diversity and* $\mu = o(n^{1/2})$ *optimizes* TwoMax *in time* n^{n-2} *is at most* $1/2+o(1)$. *Its expected optimization time is* $\Omega(n^{n-1})$.

Proof. We use a similar way of reasoning as in the proof of Theorem 3.2.1. With probability $1/2 - o(1)$, 0^n is the first local or global optimum created at time T^*. Call x *good* (from the perspective of extinction) if $|x|_1 \leq 1$ and *bad* if $|x|_1 \geq n - 1$. At time T^* the number of good individuals is at least 1. In the worst case (again from the perspective of extinction) the population at time T^* consists of 0^n and $\mu - 1$ bad individuals with $n - 1$ ones. Provided that the (μ+1) EA does not flip $n - 2$ bits at once, we now argue that the number of good individuals is monotone unless the unique 0-bit in a bad individual is flipped.

Due to the assumptions on the population only offspring with fitness at least $n - 1$ are accepted, i. e., only good or bad offspring. In order to create a bad offspring, the unique 0-bit has to be flipped since otherwise a clone or an individual with worse fitness is obtained. Hence the number of good individuals can only decrease if a bad individual is chosen as parent and its unique 0-bit is flipped. If there are i good individuals, we

denote this event by B_i and have

$$\text{Prob}(B_i) \;\leq\; \frac{\mu - i}{\mu} \cdot \frac{1}{n}.$$

On the other hand, the number of good individuals is increased from i to $i+1$ if a good offspring is created and a bad individual is removed from the population. We denote this event by G_i. A good offspring is created with probability at least $1/(3\mu)$ for the following reasons. The point 0^n is selected with probability at least $1/\mu$ and then there are $n-(i-1) = n - o(n^{1/2}) \geq (e/3) \cdot n$ 1-bit mutations (provided n is large enough) creating good offspring that are not yet contained in the population. Along with the fact that a specific 1-bit mutation has probability $1/n \cdot (1-1/n)^{n-1} \geq 1/(en)$, the bound $1/(3\mu)$ follows. After creating such a good offspring, the algorithm removes an individual with fitness $n-1$ uniformly at random. As there are $i-1$ good individuals with this fitness and $\mu - i$ bad individuals, the probability of removing a bad individual equals $(\mu-i)/(\mu-1) \geq (\mu-i)/\mu$. Together, the probability that a generation increases the number of good individuals is at least

$$\text{Prob}(G_i) \;\geq\; \frac{1}{3\mu} \cdot \frac{\mu - i}{\mu} \;=\; \frac{\mu - i}{3\mu^2}.$$

Along with inequality (3.6), the probability that G_i happens before B_i is at least

$$\frac{\frac{\mu-i}{3\mu^2}}{\frac{\mu-i}{3\mu^2} + \frac{\mu-i}{\mu n}} \;=\; \frac{1}{1+3\mu/n} \;=\; 1 - \frac{3\mu/n}{1+3\mu/n} \;\geq\; 1 - \frac{3\mu}{n}.$$

The probability that the number of good individuals increases to μ before the global optimum is reached is

$$\prod_{i=1}^{\mu} \text{Prob}(G_i \mid G_i \cup B_i) \;\geq\; \left(1 - \frac{3\mu}{n}\right)^{\mu} \;\geq\; 1 - \frac{3\mu^2}{n} \;=\; 1 - o(1).$$

The probability of creating a global optimum, provided the population contains only search points with at most one 1-bit, is at most $n^{-(n-1)}$. The probability of this happening in a phase of n^{n-2} generations is still at most $1/n$. Adding up all error probabilities, the first claim follows.

The claim on the expected optimization time follows as the last situation is reached with probability at least $1/2 - o(1)$ and the conditional expected optimization time is at least n^{n-1} then. □

3.2.4 No Fitness Duplicates

Avoiding genotype duplicates does not help much to optimize TwoMax as individuals from one branch are still allowed to spread on a certain fitness level and take over the

population. A more restrictive mechanism is to avoid fitness duplicates, i. e., multiple individuals with the same fitness. Such a mechanism has been defined and analyzed by Friedrich et al. (2007) for plateaus of constant fitness. In addition, this resembles the idea of fitness diversity proposed by Hutter and Legg (2006).

The following (μ+1) EA with fitness diversity avoids that multiple individuals with the same fitness are stored in the population. If at some time t a new individual x is created with the same fitness value as a pre-existing one $y \in P_t$, then x replaces y.

Algorithm 7 (μ+1) EA with fitness diversity

Let $t := 0$.
Initialize P_0 with μ individuals chosen uniformly at random.
repeat
  Choose $x \in P_t$ uniformly at random.
  Create y by flipping each bit in x independently with probability $1/n$.
  if there is a $z \in P_t$ such that $f(y) = f(z)$
  then $P_{t+1} := P_t \setminus \{z\} \cup \{y\}$
  else
    Choose $z \in P_t$ with minimal fitness.
    if $f(y) \geq f(z)$ **then** $P_{t+1} := P_t \setminus \{z\} \cup \{y\}$
      else $P_{t+1} := P_t$.
  Let $t := t + 1$.

From the analysis of Friedrich et al. (2007) it can be derived that if the population size μ is a constant, then the runtime on a simple plateau is exponential in the problem size n. Only if μ is very close to n the expected runtime is polynomial. In particular, if $\mu = n$ then the same upper bound as that of the (1+1) EA for plateaus of constant fitness (Jansen and Wegener, 2001) can be obtained (i. e., $O(n^3)$). In the following, by analyzing the mechanism on TwoMax, we show how also on a simple bimodal landscape, fitness diversity does not help the (μ+1) EA to avoid getting trapped on a local optimum.

The following theorem proves that if the population is not too large, then with high probability the individuals climbing one of the two branches will be extinguished before any of them reaches the top. Since the two branches of the TwoMax function are symmetric, this also implies that the global optimum will not be found in polynomial time with probability $1/2 - o(1)$.

Theorem 3.2.3. *The probability that the* (μ+1) EA *with fitness diversity and* $\mu = \text{poly}(n)$ *optimizes* TwoMax *in time* 2^{cn}, $c > 0$ *an appropriate constant, is at most* $1/2 + o(1)$. *Its expected optimization time is* $2^{\Omega(n)}$.

Proof. Obviously the second claim follows from the first one. W. l. o. g. $n/6 \in \mathbb{N}$. By Chernoff bounds, the probability that initialization creates a search point with at most

$n/3$ 1-bits or at most $n/3$ 0-bits is at most $\mu \cdot 2^{-\Omega(n)} = o(1)$. We assume in the following that only search points x with $n/3 < |x|_1 < 2n/3$ are created and keep in mind an error probability of $o(1)$.

Let individuals with $i < n/2$ 0-bits be called x_i and individuals with $i < n/2$ 1-bits be called y_i. Initially there is neither x_i nor y_i in the population for $0 \leq i \leq n/3$. Because of the fitness diversity mechanism there may be only one such x_i or one y_i in the population at the same time for $1 \leq i \leq n/3$ as x_i and y_i have the same fitness. For the current population P_t at time t define a potential $\varphi = \varphi(P_t)$ as follows:

$$\varphi \;=\; \varphi(P_t) \;:=\; \min\left\{\min\{i\colon y_{i+1} \notin P_t\},\; \frac{n}{6}\right\}.$$

The potential is capped at $n/6$ for technical reasons that will become obvious later on. As $n/6 < n/3$, we can conclude from potential i that $x_0, \ldots, x_i \notin P_t$. Intuitively, the potential is then a lower bound for the Hamming distance from the closest point in the population to the global optimum.

The following arguments are easier for $\mu \leq n/3$ as then we do not have to deal with the fact that the $n/3$ best fitness levels may contain multiple individuals. For the sake of simplicity and readability, we first only consider this case. At the end of the proof, we then argue how to deal with larger population sizes.

We now partition the run into phases and call such a phase good (for the lower bound) if the global optimum is not found during the phase. Phase 1 ends when a search point in $\{0^n, 1^n\}$ is found for the first time. After the initialization only one new individual is created in each generation. By symmetry of the TwoMax function, the probability that 0^n is found first equals the probability that 1^n is found first. Hence Phase 1 is good with probability $1/2 - o(1)$.

At the end of Phase 1 we have a current potential of $\varphi \geq 1$. Let $m_1 := \min\{\mu - 1, \sqrt[3]{n}\}$ and $m_2 := \min\{\mu - 1, n/6\}$ be the maximum potential. Define Phase 2 to start after a good Phase 1 and to end when either the global optimum is found or the potential increases to $\varphi \geq m_1$.

We inspect changes of the potential. It may happen that the next step creates $y_{\varphi+1}$. Then $x_{\varphi+1}$ is removed from the population and φ increases by 1. Such a step is called a *good step*. On the other hand, if x_i for $i \leq \varphi$ is created, then y_i is removed and φ decreases to $i - 1$. This is referred to as a *bad step*. All other steps do not change the potential. For proving a lower bound, given a current potential φ, we consider the following population as a worst case:

$$P_t \;=\; \{y_0, \ldots, y_\varphi,\; x_{\varphi+1}, \ldots, x_{\mu-1}\}.$$

In P_t we have $y_0, \ldots, y_\varphi \in P_t$ by definition of the potential φ and the x-individuals are stacked one after another. The latter maximizes the probability of reaching the global optimum in a single step. More general, for every $i \leq \varphi$ the probability of

creating x_i is maximized—this event is equivalent to a bad step decreasing the potential to i. Furthermore, the probability a good step is minimized as all x-individuals have the largest possible Hamming distance to search points $y_{\varphi+1}$. Recall that every fitness level contains at most one search point, hence there cannot be a worse constellation for the x-individuals.

We now simplify the analysis by making pessimistic assumptions that reduce the algorithm towards a simple Markov chain. After a step has been made, we compute the new potential and then pessimistically replace the resulting population by the worst-case population corresponding to the new potential. The current state of the algorithm can then be characterized by the current potential only and all transition probabilities to smaller or larger potentials have been estimated pessimistically.

Even in this pessimistic setting φ increases to m_1 with high probability before a bad step happens. If $x_{\varphi+j}$ is chosen as parent, a necessary condition for a bad step is that j out of $\varphi + j$ 0-bits flip. Using $\binom{n}{k} \leq (ne/k)^k$ (Lemma A.9), the probability of a bad step is at most

$$\begin{aligned}
\frac{1}{\mu} \sum_{j=1}^{n/3-\varphi} \binom{j+\varphi}{j} \left(\frac{1}{n}\right)^j &\leq \frac{1}{\mu} \sum_{j=1}^{n/3-\varphi} \binom{j+m_1-1}{j} \left(\frac{1}{n}\right)^j \\
&\leq \frac{1}{\mu} \sum_{j=1}^{n/3-\varphi} \left(\frac{e(j+m_1-1)}{jn}\right)^j \\
&\leq \frac{1}{\mu} \sum_{j=1}^{\infty} \left(\frac{em_1}{n}\right)^j = \frac{1}{\mu} \cdot \frac{em_1}{n-em_1} < \frac{3}{\mu} \cdot \frac{m_1}{n}
\end{aligned}$$

as $m_1 = o(n)$. If $\varphi < m_1$, the probability of a good step is at least

$$\frac{1}{\mu} \cdot \frac{n-\varphi}{en} \geq \frac{1}{\mu} \cdot \frac{n-m_1}{en} \geq \frac{1}{2e\mu}$$

since it suffices to select y_φ and to flip exactly one out of $n - \varphi$ bits. So, the probability that a good step happens before a bad step is at least

$$\frac{1/(2e\mu)}{1/(2e\mu) + 3/\mu \cdot m_1/n} = \frac{1}{1+6em_1/n} = 1 - \frac{6em_1/n}{1+6em_1/n} \geq 1 - \frac{6em_1}{n}.$$

The probability of increasing φ from 1 to m_1 by subsequent good steps before a bad step happens is bounded by

$$\left(1 - \frac{6em_1}{n}\right)^{m_1-1} \geq 1 - \frac{6em_1^2}{n} = 1 - o(1)$$

since $m_1^2 = o(n)$.

The next Phase 3 starts after a good Phase 2 and ends when either the global optimum is found or the potential increases to $\varphi \geq m_2$. In case $m_1 = \mu$ the goals of Phase 2 and Phase 3 coincide and Phase 3 is empty. We consider the expected increase of the potential in one generation. Let $\Delta_\varphi = \Delta_\varphi(t) := \varphi(P_{t+1}) - \varphi(P_t)$. If the current potential is not maximal, the probability of increasing the potential by 1 can be estimated by the probability of selecting y_φ and making one out of $n - \varphi$ 1-bit mutations increasing the number of 1-bits. Define $\Delta_\varphi^+ := \Delta_\varphi \cdot \mathbb{I}_{\Delta_\varphi > 0}$ and $\Delta_\varphi^- := \Delta_\varphi \cdot \mathbb{I}_{\Delta_\varphi < 0}$ where $\mathbb{I}_A = 1$ if condition A is true and 0 otherwise. Then for $\varphi < m_2$

$$\mathrm{E}\big(\Delta_\varphi^+\big) \;\geq\; \frac{1}{\mu} \cdot \frac{n-\varphi}{en}.$$

If $x_{\varphi+j}$ is selected, in order to decrease the potential it is necessary that j out of $\varphi + j$ 0-bits flip. The probability for this event is clearly bounded by $1/(j!)$. Under the condition that at least j 0-bits flip, the expected number of flipping 0-bits among $\varphi + j$ 0-bits is bounded by $j + \varphi/n$, the term φ/n representing the expected number of flipping 0-bits among φ 0-bits. The conditional expected decrease of the potential is then at most φ/n, leading to an unconditional expectation of

$$\begin{aligned}\mathrm{E}\big(\Delta_\varphi^-\big) \;&\leq\; \frac{1}{\mu} \sum_{j=1}^{n/3-\varphi-1} \frac{1}{j!} \cdot \frac{\varphi}{n} \\ &\leq\; \frac{1}{\mu} \cdot \frac{\varphi}{n} \sum_{j=1}^{\infty} \frac{1}{j!} \;=\; \frac{1}{\mu} \cdot \frac{(e-1)\varphi}{n}\end{aligned}$$

using $e = \sum_{j=0}^{\infty} 1/(j!)$. Putting $\mathrm{E}\big(\Delta_\varphi^+\big)$ and $\mathrm{E}\big(\Delta_\varphi^-\big)$ together, along with $\varphi < m_2 \leq n/6$,

$$\mathrm{E}(\Delta_\varphi) \;\geq\; \frac{1}{\mu} \cdot \left(\frac{n-\varphi}{en} - \frac{(e-1)\varphi}{n} \right) \;=\; \frac{1}{\mu} \cdot \frac{n - (e^2 - e + 1)\varphi}{en} \;\geq\; \frac{\varepsilon}{\mu}$$

for some $\varepsilon > 0$. One conclusion is that by drift arguments (Theorem 2.3.8) the expected time until the potential has reached its maximum value m_2 or the optimum is found beforehand is bounded by $O(n\mu)$. In other words, Phase 3 ends in expected time $O(n\mu)$.

In order to estimate the error probability in this phase, we apply the Drift Theorem 2.3.8 to the potential in the interval between $a := 0$ and $b := \sqrt[3]{n}$. Note that b cannot be chosen larger as the Drift Theorem requires the starting point to be at least b. Another obstacle is that the drift ε/μ decreases with μ. So we consider the potential only in relevant steps, defined as steps where $\Delta_\varphi \neq 0$. The arguments from our estimation of $\mathrm{E}\big(\Delta_\varphi^+\big)$, along with $\varphi \leq n/6$, yield the lower bound

$$\mathrm{Prob}(\Delta_\varphi \neq 0) \;\geq\; \frac{1}{\mu} \cdot \frac{n-\varphi}{en} \;\geq\; \frac{5}{6e\mu}.$$

We also need an upper bound on this probability. If $y_{\varphi-k}$ is selected for $0 \leq k \leq \varphi$, at least k bits have to flip in order to have a relevant step. The same holds if $x_{\varphi+1+k}$ is selected for $0 \leq k \leq \mu - \varphi - 2$. As all these solutions are selected with probability $1/\mu$, the probability of a relevant step is bounded from above by

$$\text{Prob}(\Delta_\varphi \neq 0) \;\leq\; \frac{1}{\mu} \cdot 2 \sum_{k=0}^{\infty} \frac{1}{k!} \;=\; \frac{2e}{\mu}.$$

This yields for $i < n/6$

$$\text{E}(\Delta_\varphi \mid \Delta_\varphi \neq 0) \;=\; \frac{\text{E}(\Delta_\varphi)}{\text{Prob}(\Delta_\varphi \neq 0)} \;\geq\; \frac{\varepsilon}{2e}$$

and the first condition for Theorem 2.3.10 is fulfilled. For the second condition, it is necessary for $\Delta_\varphi = -j$, $j \in \mathbb{N}$, to select some $x_{\varphi+k}$ and to flip $k+j$ bits simultaneously. Using $(k+j)! \geq k! \cdot j!$ for $k, j \in \mathbb{N}_0$ yields

$$\text{Prob}(\Delta_\varphi = -j) \;\leq\; \frac{1}{\mu} \cdot \sum_{k=0}^{\infty} \frac{1}{(k+j)!} \;\leq\; \frac{1}{\mu} \cdot \frac{1}{j!} \sum_{k=0}^{\infty} \frac{1}{k!} \;=\; \frac{e}{\mu} \cdot \frac{1}{j!}.$$

For the conditional probabilities in relevant steps this means

$$\text{Prob}(\Delta_\varphi = -j \mid \Delta_\varphi \neq 0) \;=\; \frac{\text{Prob}(\Delta_\varphi = -j)}{\text{Prob}(\Delta_\varphi \neq 0)} \;\leq\; \frac{6e^2}{5j!} \;\leq\; 2^{-j+5}$$

and the second condition holds for $\delta := 1$ and $r := 5$. The Drift Theorem shows that the probability of reaching the global optimum within $2^{c\sqrt[3]{n}}$ steps is $2^{-\Omega(\sqrt[3]{n})} = o(1)$ for an appropriate constant $c > 0$. By Markov's inequality the probability that Phase 3 is not finished after this number of steps is also $o(1)$. Concluding, Phase 3 is good with probability $1 - o(1)$.

The last Phase 4 starts after a good Phase 3 and ends when the global optimum has been found. Phase 4 therefore starts with a maximum potential of m_2. If $\mu \leq n/6 + 1$, then at least $2n/3$ bits have to flip simultaneously in order to create an accepted x-individual. The probability of this event is $n^{-\Omega(n)}$ and the claim follows. If $\mu > n/6+1$, then we apply the Drift Theorem to the larger interval from $a := 0$ to $b := n/6$. In the analysis of Phase 3 we have already shown the preconditions for this larger interval, hence the algorithm needs at least $2^{\Omega(n)}$ steps with probability $o(1)$. Summing up all error probabilities proves the claim for $\mu \leq n/3$.

Finally, we argue how to deal with larger population sizes, $\mu > n/3$. We relax our condition on worst-case populations P_t to

$$P_t \;\supseteq\; \{y_0, \ldots, y_\varphi,\; x_{\varphi+1}, \ldots, x_{n/3}\},$$

where the remaining individuals all have a number of ones in between $n/3$ and $2n/3$. These individuals therefore have Hamming distance at least $n/6$ to all search points that determine the current potential. This is the reason why we have capped the potential to $n/6$. The probability that such a search point interferes with our previous arguments is therefore at most $1/(n/6)!$. Reinspecting the analysis for Phases 2–4, we see that compared to the setting for $\mu \leq n/3$ the probabilities for bad and relevant steps only increase by additive terms of $n^{-\Omega(n)}$. Similarly, $\mathrm{E}(\Delta_\varphi)$ is only decreased by $n^{-\Omega(n)}$ for $0 < \varphi < n/6$. In particular, Phase 2 remains good with probability $1 - o(1)$ and the application of the Drift Theorem in Phases 3 and 4 remains possible for appropriate constants ε, δ, r. □

3.2.5 Deterministic Crowding

In the deterministic crowding mechanism offspring compete directly with their respective parents. According to Mahfoud (1997), in a genetic algorithm with crossover deterministic crowding works as follows. In every generation the population is partitioned into $\mu/2$ pairs of individuals, assuming μ to be even. These pairs are then recombined and mutated. Every offspring then competes with one of its parents and may replace it if the offspring is not worse.

As we do not consider crossover, we adapt the main idea of offspring competing with their parents for a mutation-based algorithm. More precise, in the following algorithm an offspring replaces its parent if its fitness is at least as good.

Algorithm 8 (μ+1) EA with deterministic crowding

Let $t := 0$.
Initialize P_0 with μ individuals chosen uniformly at random.
repeat
  Choose $x \in P_t$ uniformly at random.
  Create y by flipping each bit in x independently with probability $1/n$.
  if $f(y) \geq f(x)$ **then** $P_{t+1} := P_t \setminus \{x\} \cup \{y\}$
    else $P_{t+1} := P_t$.
  Let $t := t + 1$.

The algorithm closely resembles a parallel (1+1) EA since μ individuals explore the landscape independently. However, in contrast to parallel runs, interactions between the individuals may be obtained by using other operators together with mutation. Recently, the mechanism together with crossover has proved to be useful for the Vertex Cover problem by making the difference between polynomial and exponential runtimes for some instances (Oliveto et al., 2008). Here we concentrate on the capabilities of guaranteeing diversity of the mechanism by analyzing the (μ+1) EA with deterministic crowding on

the TwoMax function. For sufficiently large populations the algorithm can easily reach both local optima.

Theorem 3.2.4. *The* $(\mu+1)$ EA *with deterministic crowding and* $\mu = \text{poly}(n)$ *reaches on* TwoMax *a population consisting of only local or global optima in expected time* $O(\mu n \log n)$. *In that case the population contains at least one global optimum with probability at least* $1 - 2^{-\mu}$.

Proof. The main observation for the second statement is that the individuals of the population are *independent* due to the crowding mechanism. Due to the symmetry of TwoMax, the i-th individual in the population reaches the global optimum with probability 1/2. The probability that at least one individual finds the global optimum is $1 - (1/2)^{\mu}$.

Let T be the random time until all the individuals have reached local or global optima on TwoMax. It is easier to find a local optimum on TwoMax than to find a global optimum on OneMax, hence we estimate $\mathrm{E}(T)$ by the expected global optimization time on OneMax.

Consider the following game of balls and bins, where bins represent bits and balls represent 1-values. Imagine a bin for every bit in the initial population, i. e., a set of μn bins labelled with their respective bits. Place a ball in each bin if the associated bit is set to 1 in the initial population. In the following generations balls may be put into empty bins according to certain rules. The game ends when all bins contain a ball—this corresponds to a population where all bits have been set to 1 and every individual represents the optimum 1^n.

Consider a generation where some individual x is selected as parent. The probability that mutation only flips a specific 0-bit x_i into a 1-bit is $1/\mu \cdot 1/n \cdot (1-1/n)^{n-1} \geq 1/(en\mu)$. As a consequence of this mutation, the offspring replaces its parent x. Compared to the previous population, the bit x_i is being switched to value 1 and we imagine that in this case a new ball is put into the empty bin for x_i. If mutation creates an offspring y with $\textsc{OneMax}(y) = \textsc{OneMax}(x)$, y again replaces x and we imagine the bins for x being rearranged so that their occupancy matches the 1-bits in y. Note that rearranging bins of an individual, i. e., rearranging bits within an individual, does not make a difference for the algorithm as all bits are treated equally. In the case where $\textsc{OneMax}(y) > \textsc{OneMax}(x)$, we imagine that $\textsc{OneMax}(y) - \textsc{OneMax}(x)$ balls are added to arbitrary empty bins of x and afterwards the bins are again rearranged to match the offspring y.

Fix an arbitrary empty bin. The probability that it receives a ball in one generation is at least $1/(en\mu)$. The probability that the bin does not receive a ball within $t := en\mu \cdot \ln(2n\mu)$ steps is bounded by

$$\left(1 - \frac{1}{en\mu}\right)^{en\mu \cdot \ln(2n\mu)} \leq e^{-\ln(2n\mu)} = \frac{1}{2n\mu}.$$

By the union bound, the probability that there is still an empty bin left in the game after t steps is at most $1/2$. In case the game has not ended after t steps, we consider another period of t steps and continue waiting for the game to end. The expected number of periods until the game ends is at most 2. Hence the expected time of the game is bounded by $\mathrm{E}(T) \leq 2t = O(\mu n \log n)$ since $\ln(2n\mu) = O(\log n)$ follows from $\mu = \mathrm{poly}(n)$. □

3.2.6 Fitness Sharing

Fitness sharing (see, e. g., Mahfoud, 1997) derates the real fitness of an individual x by an amount that represents the similarity of x to other individuals in the population. The similarity between x and y is measured by a so-called *sharing function* $\mathrm{sh}(x, y) \in [0, 1]$ where a large value corresponds to large similarities and value 0 implies no similarity. The idea is that if there are several copies of the same individual in the population, these individuals have to share their fitness. As a consequence, selection is likely to remove such clusters and to keep the individuals apart. We denote the *shared fitness* of x in the population P as

$$f(x, P) \;=\; \frac{f(x)}{\sum_{y \in P} \mathrm{sh}(x, y)}.$$

The fitness of the population is then defined as

$$f(P) \;=\; \sum_{x \in P} f(x, P).$$

It is common practice to use a so-called *sharing distance* σ such that individuals only share fitness if they have distance less than σ. Given some distance function d, a common formulation for the sharing function is

$$\mathrm{sh}(x, y) \;=\; \max\{0, 1 - (d(x, y)/\sigma)^{\alpha}\}$$

where α is a positive constant that regulates the shape of the sharing function. We use the standard setting $\alpha = 1$ and, following Mahfoud (1997), we set the sharing distance to $\sigma = n/2$ as this is the smallest value allowing discrimination between the two branches. As TwoMax is a function of unitation, we allow the distance function d to depend on the number of ones: $d(x, y) := \big|\, |x|_1 - |y|_1 \,\big|$. Such a strategy is known as *phenotypic sharing* (Mahfoud, 1997). Our precise sharing function is then

$$\mathrm{sh}(x, y) \;=\; \max\left\{0,\; 1 - 2\,\frac{\big|\, |x|_1 - |y|_1 \,\big|}{n}\right\}.$$

We now incorporate fitness sharing into the (μ+1) EA. Our goal is to evolve a good population, hence selection works by comparing candidates for next generation's population with respect to their $f(P)$-values. This selection strategy has already been analyzed by Sudholt (2005).

Algorithm 9 (μ+1) EA with fitness sharing

Let $t := 0$.
Initialize P_0 with μ individuals chosen uniformly at random.
repeat
  Choose $x \in P_t$ uniformly at random.
  Create y by flipping each bit in x independently with probability $1/n$.
  Let $P_t^* := P_t \cup \{y\}$.
  Choose $z \in P_t^*$ such that $f(P_t^* \setminus \{z\})$ is maximized.
  Let $P_{t+1} := P_t^* \setminus \{z\}$ and $t := t + 1$.

Note that when evaluating $f(P_t^* \setminus \{z\})$ the shared fitness values have to be recomputed for all these populations. However, with the use of dictionaries it suffices to compute $f(y)$ and the sharing values $\mathrm{sh}(x, y)$ for $x \in P_t$ only once. In addition, fitness evaluations are often the most expensive operations in evolutionary computation, so the additional effort is negligible.

We now show that the (μ+1) EA with fitness sharing can find both optima on TwoMax. Imagining all parents and the new offspring on a scale of $|x|_1$, the individuals with the smallest and the largest number of ones have the largest distance to all individuals in the population. Therefore, fitness sharing makes these outer individuals very attractive in terms of shared fitness, hence these individuals are taken over to the next generation. This even holds if an outer individual has the worst fitness in the population.

Lemma 3.2.5. *Consider the* (μ+1) EA *with fitness sharing and* $\mu \geq 2$ *on* TwoMax. *Let* P_t^* *be the enlarged parent population at some point of time* t *and w. l. o. g. let* $P_t^* = \{x_1, \ldots, x_{\mu+1}\}$ *with* $|x_1|_1 \leq |x_2|_1 \leq \cdots \leq |x_{\mu+1}|_1$. *If* $|x_\mu|_1 < |x_{\mu+1}|_1$ *then* $x_{\mu+1} \in P_{t+1}$. *Also, if* $|x_1|_1 < |x_2|_1$ *then* $x_1 \in P_{t+1}$.

Proof. We only prove $x_{\mu+1} \in P_{t+1}$ for $|x_\mu|_1 < |x_{\mu+1}|_1$. The second claim for the case $|x_1|_1 < |x_2|_1$ follows by symmetry. Let $P^- := P_t^* \setminus \{x_{\mu+1}\}$ be a "bad" new population without $x_{\mu+1}$, contradicting our claim that $x_{\mu+1}$ remains in the population. Let $P^+ := P_t^* \setminus \{x_\mu\}$ be one "good" new population where x_μ is removed instead and let $P^\cap := P^+ \cap P^-$ contain all other individuals. We will prove $f(P^-) < f(P^+)$. This implies that the bad population P^- does not have maximal population fitness among all possible next populations $P_t^* \setminus \{z\}$ examined in the selection step, hence P^+ or another "good" population is chosen and $x_{\mu+1}$ remains in the population.

We first deal with the case $f(x_\mu) \leq f(x_{\mu+1})$. Intuitively, $x_{\mu+1}$ is better than x_μ both in terms of real fitness and in terms of sharing distance to the other individuals in $P^\cap$. More precise, we have $\mathrm{sh}(x_\mu, x) > \mathrm{sh}(x_{\mu+1}, x)$ for $x \in P^\cap$ due to the ordering of the x_i and the definition of the sharing function. Hence, $\sum_{y \in P^-} \mathrm{sh}(x, y) > \sum_{y \in P^+} \mathrm{sh}(x, y)$ for

$x \in P^{\cap}$ and $\sum_{y\in P^-} \mathrm{sh}(x_\mu, y) > \sum_{y\in P^+} \mathrm{sh}(x_{\mu+1}, y)$. Together, we obtain

$$\begin{aligned} f(P^-) &= \sum_{x\in P^{\cap}} \frac{f(x)}{\sum_{y\in P^-} \mathrm{sh}(x,y)} + \frac{f(x_\mu)}{\sum_{y\in P^-} \mathrm{sh}(x_\mu,y)} \\ &< \sum_{x\in P^{\cap}} \frac{f(x)}{\sum_{y\in P^+} \mathrm{sh}(x,y)} + \frac{f(x_{\mu+1})}{\sum_{y\in P^+} \mathrm{sh}(x_{\mu+1},y)} = f(P^+). \end{aligned}$$

It remains to show $f(P^-) < f(P^+)$ for the case $f(x_\mu) > f(x_{\mu+1})$. This means that x_μ is better than $x_{\mu+1}$ in terms of real fitness, but worse with respect to the shared distance to all individuals in $P^{\cap}$. We want to examine the impact of the distance between the two individuals on the fitness and the sharing function. Therefore, we define

$$d := \min\left\{ |x_{\mu+1}|_1 , \frac{n}{2} \right\} - |x_\mu|_1 .$$

Note that $d \geq 0$ as $|x_\mu|_1 < n/2$ follows from $f(x_\mu) > f(x_{\mu+1})$. The minimum in the definition of d implies the following inequality.

$$\forall\, x \in P^{\cap}\colon\ \mathrm{sh}(x, x_{\mu+1}) \leq \mathrm{sh}(x, x_\mu) - \frac{2d}{n}, \tag{3.7}$$

which is immediate from the definition of $\mathrm{sh}(\cdot,\cdot)$ if $x_{\mu+1}$ is within sharing distance from x. Otherwise, it follows from

$$\mathrm{sh}(x, x_{\mu+1}) = 0 \leq \mathrm{sh}(x, x_\mu) - \frac{2(\frac{n}{2} - |x_\mu|_1)}{n} = \mathrm{sh}(x, x_\mu) - \frac{2d}{n}.$$

Inequality (3.7) can now be used to relate the shared fitness $f(x, P^-)$ to the shared fitness $f(x, P^+)$ for $x \in P^{\cap}$. This reflects the gain in total shared fitness for the individuals in $P^{\cap}$ if P^+ is selected instead of P^-. Using inequality (3.7) and the fact that the sharing function is at most 1,

$$\begin{aligned} \forall\, x \in P^{\cap}\colon\ \frac{f(x,P^-)}{f(x,P^+)} &= \frac{\sum_{y\in P^+} \mathrm{sh}(x,y)}{\sum_{y\in P^-} \mathrm{sh}(x,y)} \\ &\leq \frac{\sum_{y\in P^{\cap}} \mathrm{sh}(x,y) + \mathrm{sh}(x,x_\mu) - 2d/n}{\sum_{y\in P^{\cap}} \mathrm{sh}(x,y) + \mathrm{sh}(x,x_\mu)} \\ &\leq \frac{\mu - 2d/n}{\mu} = 1 - \frac{2d}{n\mu}. \end{aligned} \tag{3.8}$$

As $f(x_{\mu+1}) < f(x_\mu)$, the total real fitness of the individuals is worse for P^+ than for P^-. We also relate the shared fitness $f(x_\mu, P^-)$ to $f(x_{\mu+1}, P^+)$ in order to estimate the loss of shared fitness. By definition of d and TwoMax it is easy to see that $f(x_\mu) - d \leq f(x_{\mu+1})$. Along with $f(x) \geq n/2$ for every $x \in \{0,1\}^n$, we have

$$\frac{f(x_{\mu+1})}{f(x_\mu)} \geq \frac{f(x_\mu) - d}{f(x_\mu)} \geq \frac{n/2 - d}{n/2} = 1 - \frac{2d}{n}.$$

Also, using inequality (3.7), we obtain

$$\begin{aligned}\frac{\sum_{y\in P^+}\mathrm{sh}(x_{\mu+1},y)}{\sum_{y\in P^-}\mathrm{sh}(x_\mu,y)} &= \frac{1+\sum_{y\in P^\cap}\mathrm{sh}(x_{\mu+1},y)}{1+\sum_{y\in P^\cap}\mathrm{sh}(x_\mu,y)}\\ &\le \frac{1+\sum_{y\in P^\cap}(\mathrm{sh}(x_\mu,y)-2d/n)}{1+\sum_{y\in P^\cap}\mathrm{sh}(x_\mu,y)}\\ &\le \frac{1+(\mu-1)(1-2d/n)}{\mu}.\end{aligned}$$

Taking the last two estimations together,

$$\begin{aligned}\frac{f(x_\mu,P^-)}{f(x_{\mu+1},P^+)} &= \frac{\sum_{y\in P^+}\mathrm{sh}(x_{\mu+1},y)}{\sum_{y\in P^-}\mathrm{sh}(x_\mu,y)}\cdot\frac{f(x_\mu)}{f(x_{\mu+1})}\\ &\le \frac{1+(\mu-1)(1-2d/n)}{\mu(1-2d/n)}\\ &= \frac{1-\frac{2d}{n}+\frac{2d}{n\mu}}{1-\frac{2d}{n}} \;<\; 1+\frac{2d}{n\mu}.\end{aligned} \tag{3.9}$$

So, when comparing P^- with P^+, we have a gain of shared fitness for all $x\in P^\cap$ and a loss of shared fitness for the remaining individual when exchanging x_μ for $x_{\mu+1}$. Putting inequalities (3.8) and (3.9) together yields

$$\begin{aligned}f(P^-) &= \sum_{x\in P^\cap} f(x,P^-)+f(x_\mu,P^-)\\ &< \sum_{x\in P^\cap} f(x,P^+)\cdot\left(1-\frac{2d}{n\mu}\right)+f(x_{\mu+1},P^+)\cdot\left(1+\frac{2d}{n\mu}\right)\\ &\le f(P^+)-\frac{2d}{n\mu}\left(\sum_{x\in P^\cap} f(x,P^+)-f(x_{\mu+1},P^+)\right).\end{aligned}$$

Now $f(P^-)<f(P^+)$ follows if we can show that the term in parentheses is non-negative, i. e.,

$$\sum_{x\in P^\cap} f(x,P^+) \;\ge\; f(x_{\mu+1},P^+). \tag{3.10}$$

Recall $|x_\mu|_1 \le n/2$ and consider the left-hand side of inequality (3.10). This term is minimized if all $x\in P^\cap$ equal $x_{\mu-1}$ since then for all individuals in $P^\cap$ fitness is

minimized and sharing is maximized. Along with $f(x_{\mu-1}) > f(x_{\mu+1})$ and $\mu - 1 \geq 1$,

$$\begin{aligned}\sum_{x \in P^{\cap}} f(x, P^+) &\geq \frac{(\mu-1) \cdot f(x_{\mu-1})}{\mu - 1 + \mathrm{sh}(x_{\mu-1}, x_{\mu+1})} \\ &> \frac{f(x_{\mu+1})}{1 + \mathrm{sh}(x_{\mu-1}, x_{\mu+1})} \\ &\geq \frac{f(x_{\mu+1})}{\sum_{y \in P^+} \mathrm{sh}(x_{\mu+1}, y)} = f(x_{\mu+1}, P^+).\end{aligned}$$

□

Now it is easy to prove an upper bound on TwoMax. To the best of our knowledge, the following theorem provides the first runtime analysis of an EA with fitness sharing for population sizes greater than 2.

Theorem 3.2.6. *The* (μ+1) EA *with fitness sharing and* $\mu \geq 2$ *reaches on* TwoMax *a population containing both optima in expected optimization time* $O(\mu n \log n)$.

Proof. For a population P, we consider the following characteristic values as potential functions: $m_0(P)$ denotes the maximum number of zeros and $m_1(P)$ the maximum number of ones for the individuals in P. We are interested in the expected time until both potentials become n.

According to Lemma 3.2.5, both potentials cannot decrease. If $m_0(P) = k$ then we wait for an individual with k zeros to be chosen and for the number of zeros to be increased. The expected time for this to happen is bounded from above by $O(\mu \cdot n/(n-k))$. Hence, the expected time until the m_0-potential reaches its maximum value n is $O(\mu n \log n)$. A symmetrical statement holds for the m_1-potential, hence the expected time until both optima are found is bounded by $O(\mu n \log n)$. □

3.2.7 Experiments

Our negative results for the (μ+1) EA without diversification and the genotype diversity mechanism only hold for relatively small population sizes. We believe that the same results also hold for larger values of μ, but a theoretical analysis is challenging. Initial experiments have shown that also with larger μ extinction is still likely, but the question which branch gets extinct and when extinction happens is determined by long-term dynamics that are difficult to understand.

A typical behavior is that one branch starts lagging behind a little. Then chances to create offspring on higher fitness levels are lower for this branch, while the other branch has an advantage in this respect. When climbing the next fitness levels, this effect may intensify until the branch that is behind gets extinct. However, it may also happen that by chance the branch that is behind creates an offspring on a good fitness level ℓ and

then the branch is safe from extinction until level ℓ becomes the worst fitness level in the population. This gives the branch some time to recover, which makes it hard to predict when a branch will get extinct. It is even hard to tell which branch is "behind." One branch may consist of few good individuals and the other one of many inferior solutions. Which one is more likely to survive in the long run? We feel that completely new methods have to be developed in order to understand these long-term dynamics.

We rely on experiments to find out how large a population has to be to avoid extinction. This also allows a more detailed comparison of diversity mechanisms. We consider exponentially increasing population sizes $\mu = 2, 4, 8, \ldots, 1024$ for $n = 30$ and perform 100 runs in each setting. An obvious performance measure is to record the number of runs where the global optimum is found. However, for hill climbers like the (1+1) EA this measure fluctuates around 50 runs. In order to obtain a more clear picture without this random fluctuation, we instead consider the number of runs where both 0^n and 1^n were present in the population at the same time. Such a run is called *successful* hereinafter. In all non-successful runs we have a conditional probability of exactly 1/2 that the global optimum was found due to the symmetry of TwoMax, provided $\mu \geq 2$. Moreover, we record the maximum progress on the branch that gets extinct, computed as

$$\min \{m_0(P_t),\ m_1(P_t)\},$$

$m_0(P_t)$ ($m_1(P_t)$) denoting the maximum number of zeros (ones) in the population P_t at time t. In case both 0^n and 1^n are contained in the population, the maximum progress equals n. For fitness sharing we present our theoretical results as the outcome of experiments is predetermined. The number of successful runs is clearly 100 and the maximum progress is $n = 30$.

The choice of the stopping criterion is non-trivial. A natural design choice is to stop the algorithm after the whole population has reached local or global optima. One often speaks of "convergence", although this term differs from convergence in a mathematical sense. For fitness sharing, deterministic crowding, and the (μ+1) EA without diversification, we stop the run when either the run is successful or when the whole population only consists of copies of 0^n or 1^n. For genotype diversity and, especially, fitness diversity there is no such "convergence" as copies of 0^n and 1^n are not allowed. The genotype diversity mechanism gets stuck when all individuals are as close to 0^n or 1^n as possible. The same happens for fitness diversity when $\mu \ll n/2$ and all individuals are stacked one after another on one branch. However, the fitness diversity mechanism for $\mu > n/2$ does not converge at all because the population performs a random walk with a drift pointing away from the optimum when it gets close enough. Therefore, we stop a run for these two mechanisms after 3600μ generations. The time bound 3600μ was chosen for the following reason. For $n = 30$ we have $360\mu > \mu \cdot en \ln(n + 1)$. The latter is an upper bound for the expected time of an algorithm to reach a local or global optimum by relying only on steps selecting the current best individual and performing a 1-bit

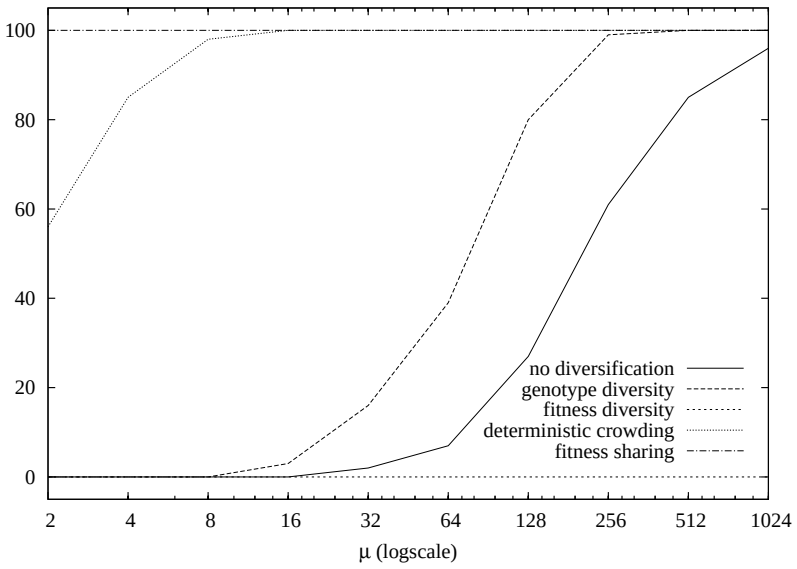

Figure 3.7: The number of successful runs among 100 runs for $n = 30$ and $\mu = 2, 4, 8, \ldots, 1024$. A run is called successful if 0^n and 1^n were present in the population at the same time.

mutation towards the considered optimum. This upper bound applies to the $(\mu+1)$ EA without diversification, genotype diversity, deterministic crowding, and fitness sharing. Moreover, the random time is concentrated around the expectation. Our bound 3600μ is more than ten times larger than the expectation. This should be enough time for the two algorithms to "converge" or to reach a meaningful equilibrium state.

Figure 3.7 shows the number of successful runs. While fitness sharing is always successful, fitness diversity was never found to be successful. Using deterministic crowding, the success probability increases very steeply compared to the scenarios of no diversification and genotype diversity. Although genotype diversity is a rather weak mechanism, it turns out to be more successful than no diversification.

Figure 3.8 shows the average maximum progress in 100 runs. It is obvious that fitness sharing and deterministic crowding perform well due to their high success probabilities. But here also fitness diversity has an effect as its progress indicator increases with increasing μ, although more slowly than the indicator of all the other algorithms. Concerning the $(\mu+1)$ EA without diversification and with genotype diversity, we can see from Figure 3.8 that for small population sizes extinction occurs very early and for low fitness values. Contrarily, our theoretical arguments were based on the very last fitness levels where our estimates of the extinction probabilities were best. This strengthens

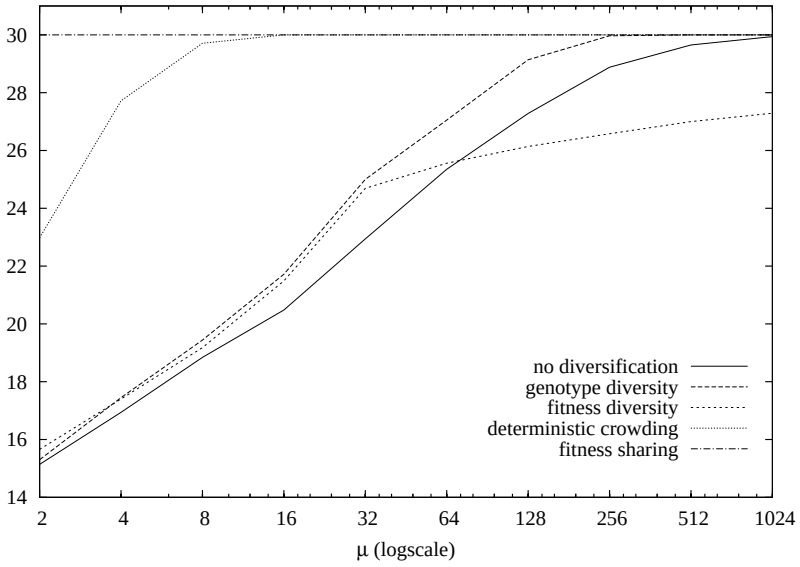

Figure 3.8: The average maximum progress on the branch that is behind, measured among 100 runs, for $n = 30$ and $\mu = 2, 4, 8, \ldots, 1024$.

our impression that extinction is due to much more complex long-term effects than used in our proofs.

3.2.8 Conclusions

We have examined the behavior of different diversity mechanisms on a fitness landscape consisting of two hills with symmetric slopes. The results are summarized in Table 3.4. We rigorously proved that without any diversification the whole population of the (μ+1) EA runs into the local optimum with probability almost 1/2 (Theorem 3.2.1). This still holds if we avoid genotype duplicates or fitness duplicates. An implication is that for these algorithms the population is nearly useless as we experience the same performance as for simple hill climbers like local search or the (1+1) EA.

On the other hand, stronger diversity mechanisms like fitness sharing and deterministic crowding allow the (μ+1) EA to find both optima of our test function TwoMax with high probability. Deterministic crowding performs as well as independent runs and the probability of not finding both optima decreases exponentially with μ. Fitness sharing using a phenotypic distance function always finds both optima efficiently for arbitrary populations of size $\mu \geq 2$.

Our theoretical results and the experiments from Section 3.2.7 have also revealed important open problems. Theorems 3.2.1 and 3.2.2 apply only to sublinear population

Mechanism	Population size	Success probability in T generations
No diversification	$\mu = o(n/\log n)$	$1/2 + o(1)$ for $T = n^{n-1}$
Genotype diversity	$\mu = o(\sqrt{n})$	$1/2 + o(1)$ for $T = n^{n-2}$
Fitness diversity	$\mu = \text{poly}(n)$	$1/2 + o(1)$ for $T = 2^{cn}$
Deterministic crowding	$\mu = \text{poly}(n)$	$1 - 2^{-\mu}$ for $\text{E}(T) = O(\mu n \log n)$
Fitness sharing	$\mu \geq 2$	1 for $\text{E}(T) = O(\mu n \log n)$

Table 3.4: Summary of the results for variants of the (μ+1) EA w. r. t. conditions on the population size μ. The success probability denotes (a bound on) the probability of finding a global optimum within T generations for specific conditions on the variable T; $c > 0$ is an appropriate constant.

sizes. Our experimental results indicate similar behavior also for larger populations, but a theoretical analysis is difficult. It seems that a more thorough understanding of the long-term dynamics is required to strengthen the theorems.

4 Memetic Evolutionary Algorithms

Over the past years, it has become increasingly popular to include local search techniques into the random search process of evolutionary algorithms. These hybrid algorithms are known by various names such as memetic (evolutionary) algorithms (MAs), evolutionary local search, genetic local search, global-local search hybrids, large-step Markov chains, and others. The term *memetic algorithms* has been coined by Pablo Moscato (Moscato, 1989) who relates local search to a process of cultural evolution; the word "memetic" is derived from biologist Richard Dawkins's notion of "memes" as the cultural and social equivalent of "genes" (Dawkins, 1976). We stick to this name in the following.

Memetic algorithms apply local search to newly created offspring to quickly find high-fitness individuals and to discover promising regions of the search space. Researchers consistently report very good results for memetic algorithms on practical problems and there is much experimental evidence where this approach outperforms common evolutionary algorithms, see Moscato (1999) for a survey or the book by Hart, Krasnogor, and Smith (2004b). The benefit of local search is manifold. Firstly, we can expect high-fitness solutions to be found more quickly. Moreover, there may be low-fitness offspring located in the basin of attraction of a high-fitness local optimum. In a standard evolutionary algorithm, such solutions are likely to get lost immediately in the selection process. In a memetic approach local search may improve upon such solutions and reach a local optimum with high fitness. This effect is particularly visible in constraint optimization problems where often infeasible solutions are penalized with respect to their fitness and the penalty decreases towards feasible regions. If mutation and/or crossover create an infeasible offspring, local search can work as repair mechanism and find a feasible solution. Lastly, there is an option to include problem-specific knowledge into the search as local search may be adopted to the problem at hand. Such an approach is often possible since local search strategies are typically easy to design, even in cases where no global problem-specific strategy is known.

An extreme example of a memetic algorithm is known as *iterated local search* (see, e. g., Lourenço, Martin, and Stützle, 2002) where local search is performed for every created offspring until a local optimum is found. This means that the population always contains only local optima and the algorithm is said to search within the subspace of local optima. In order to escape from the current local optimum, usually a large mutation is used, which is also called *perturbation*.

In recent years many different kinds and variants of memetic algorithms have emerged. Krasnogor and Smith (2005) present a taxonomy of memetic algorithms and discuss sev-

eral design issues. One trend is to employ multiple local search operators. These algorithms are sometimes called *multimeme algorithms* (Neri, Toivanen, Cascella, and Ong, 2007); each local search operator is called a "meme." The choice of memes is often made adaptively or even self-adaptively, see the survey by Ong, Lim, Zhu, and Wong (2006). Furthermore, memetic approaches have been proposed for optimization in continuous spaces (Hart, 2003) and for other algorithmic paradigms such as estimation of distribution algorithms (Aickelin, Burke, and Li, 2007) and ant colony optimization (Dorigo and Stützle, 2004; Levine and Ducatelle, 2004). One "memetic" ant colony optimizer will be analyzed in Section 5.3.

Hybridization of search heuristics poses new challenges to both theory and practice. For theory it is hard to keep track with the state-of-the-art as the algorithms become more and more elaborate. It is widely acknowledged that a solid theoretical foundation of these algorithms is needed. Many operators commonly used in such a hybrid are fairly simple and easy to implement; this holds for standard genetic operators such as mutation, crossover, and selection as well as for many local search strategies. However, the dynamics of an algorithm and the interplay of these components is very hard to tackle analytically. Therefore, studies on memetic algorithms are mostly empirical (e. g. Boughaci and Drias (2005); Cotta (2005); Lourenço et al. (2002)) or rely on non-rigorous arguments (e. g. Sinha, Chen, and Goldberg (2004)). We again take a rigorous perspective to address this topic.

In Section 4.1 we consider the parametrization of memetic algorithms and the balance between global and local search. In practical applications, the available computational resources have to be spread among global and local search, but finding a good balance is not always easy. We consider a simple memetic algorithm that captures basic working principles of memetic algorithms—the interplay of genetic operators like mutation and selection with local search. We then present artificial examples where good parameter settings are very hard to find. Moreover, only small changes to such a good parametrization lead to a phase transition from polynomial optimization times to superpolynomial or even exponential times, with high probability. Our results show exemplarily that parametrizing memetic evolutionary algorithms can be extremely hard. Moreover, they rule out simple and effective design guidelines that do not depend on the problem at hand.

Section 4.2 then considers the use of memetic algorithms for combinatorial problems. Our motivation is to give an explanation why memetic algorithms are so effective for many practical problems. To this end, we present instances for several problems from combinatorial optimization where a memetic algorithm with so-called variable-depth search drastically outperforms many other algorithms like the (1+1) EA, memetic algorithms with a simple local search operator and simulated annealing. The negative results even hold if we allow a broad range of parameter settings for these algorithms. In addition, we prove that in one case the memetic algorithm is more powerful than its single components.

4.1 The Impact of Parametrization

Memetic evolutionary algorithms integrate local search into the random search process of evolutionary algorithms. As computational resources have to be spread adequately among local and evolutionary search, one has to care about when to apply local search and how much computational effort to devote to local search. Often local search is called with a fixed frequency, the *local search frequency*, and then run for a fixed number of iterations, the *local search depth*. There is empirical evidence that these parameters have a significant impact on the performance of memetic algorithms, but a theoretical understanding as well as concrete design guidelines are missing.

From the practitioner's perspective one has to think carefully about how to spread the available resources adequately among evolutionary (global) search and local search. If the effect of local search is too weak, we fall back to standard evolutionary algorithms. If the effect of local search is too strong, the algorithm may quickly get stuck in local optima of bad quality. Moreover, the algorithm is likely to rediscover the same local optimum over and over again, wasting computational effort. Lastly, too much local search quickly leads to a loss of diversity within the population.

The importance of the parametrization has already been recognized by Hart (1994) who investigated empirically the impact of the local search frequency and the local search depth on three artificial test functions. Later on, this study was extended to combinatorial optimization by Land (1998), referring to the balance between global search and local search as the local/global ratio. In a more recent study Ishibuchi, Yoshida, and Murata (2003) considered a hybrid algorithm where local search is called with a fixed probability and investigated the impact of this parameter on a flowshop scheduling problem. Another line of research was to adapt the parametrization of a memetic algorithm to the problem at hand according to an analysis of the problem structure (Merz, 2004; Watson, Howe, and Whitley, 2003). Finally, there were theoretical investigations of simplified models of memetic algorithms explaining how to balance global and local search (Sinha et al., 2004). However, as their arguments were non-rigorous, their work could not lead to formal proofs.

At present, still no guidelines are available for the parametrization of memetic algorithms and the empirical and theoretical knowledge in this respect is limited. Quoting from Hart, Krasnogor, and Smith (2004a):

> "The question of *when* to apply local improvement heuristics, to *which* individuals in EAs population and *how much* computational effort to devote to them remains unanswered, and more research effort is required to gain the understanding and insights that may lead to guidelines for the design of efficient and effective algorithms."

In this section, we address the first and the third question, that is, the choice of the local search frequency and the local search depth. Our purpose is to stress the importance

of these parameters from a rigorous theoretical perspective and to show that finding simple guidelines to design efficient memetic algorithms is a difficult task. To this end, we consider a simple memetic algorithm that reflects basic working principles of memetic algorithms: the interplay of evolutionary operators like mutation and selection with local search. Note that we do not consider crossover; we will, however, discuss possible effects of introducing crossover at the end of Section 4.1.3. We present examples where the algorithm is very sensitive to even small changes of the local search depth and the local search frequency in a sense that the optimization time can change from polynomial to superpolynomial and vice versa, with high probability. Concerning the local search depth, we present a class of functions allowing us to specify an ideal value for the local search depth that leads to a polynomial optimization time, with high probability. If the local search depth differs from this ideal value by only a polylogarithmic additive term in either direction, this is likely to imply superpolynomial optimization times. A second hierarchy result implies that doubling or halving the local search frequency can decide between polynomial versus exponential optimization times with overwhelming probability. These hierarchy results imply that for every parametrization, apart from mild restrictions, there is a function on which the algorithm is inefficient, although the runtime can be polynomial with a better parametrization. These results rule out the existence of simple and effective design guidelines for the considered memetic algorithm that do not depend on the function at hand.

The reader might object that such a conclusion was already implied by the well-known no free lunch theorem (NFL) (Igel and Toussaint, 2004). The NFL theorem states that all parameter settings for an algorithm lead to equal average performance when the performance is averaged over all functions (with respect to a finite domain such as $\{0,1\}^n$ and a finite codomain). This means that if there is a function where parametrization A beats parametrization B with respect to the number of different search points evaluated (the NFL theorem does not consider resampling), then functions exist where B beats A. However, considering the class of all functions is not a realistic scenario. As argued by Droste, Jansen, and Wegener (2002b), the vast majority of functions can neither be stated effectively nor be evaluated in reasonable time. These functions therefore are of no significance and no relevance for optimization. When making restrictions towards functions with reasonable complexity (with respect to size and time for evaluation), a no free lunch result probably cannot exist (Droste et al., 2002b). A second fundamental limitation of the NFL theorem is that it does not make any statements regarding the magnitude of the performance difference between two parameter settings. An optimistic algorithm designer might conjecture that the choice of the parametrization affects the expected runtime at most by factors of polynomial order. This conjecture cannot be disproven by the NFL theorem, but we will disprove it in the following and thereby show that the parametrization can indeed have a tremendous impact on performance.

The remainder of this section is structured as follows. In Section 4.1.1 we introduce the considered memetic algorithm, the $(\mu+\lambda)$ Memetic Algorithm, shortly $(\mu+\lambda)$ MA.

We prepare ourselves for the analysis of the (μ+λ) MA with some useful tools for the analysis of population-based algorithms in Section 4.1.2. Section 4.1.3 deals with the impact of the local search depth on the (μ+λ) MA while in Section 4.1.4 we focus on the impact of the local search frequency. Finally, we finish with conclusions in Section 4.1.5.

4.1.1 The (μ+λ) Memetic Algorithm

The (μ+λ) Memetic Algorithm, shortly (μ+λ) MA, is defined for maximization, although it can be easily adapted to minimization. We first describe two operators used by the (μ+λ) MA. One is the standard mutation operator that flips each bit in the search point independently from the other bits with probability $1/n$. The result of a standard mutation applied to x is denoted as mutation(x).

The local search operator employed in the (μ+λ) MA is defined as follows. Throughout this section, $N(x)$ denotes the open Hamming neighborhood of x and $N^*(x)$ the closed Hamming neighborhood that also includes x itself. For a set $X \subseteq \{0,1\}^n$ we denote $N(X) := \bigcup_{x \in X} N(x)$ and $N^*(X) := N(X) \cup X$. The local search depth is denoted by δ.

Operator 10 Local search(y)

for δ iterations **do**
  if there is a $z \in N(y)$ with $f(z) > f(y)$ **then** $y := z$
  else stop and **return** y.
return y.

This local search strategy is generic in that no pivot rule is specified how to choose z from the set of all better Hamming neighbors. It generalizes several concrete local search strategies such as first ascent or steepest ascent. The choice of the pivot rule will turn out to be immaterial to the results obtained hereinafter. Hence every pivot rule is suitable as long as each $z \in N(y)$ is evaluated at most once.

The (μ+λ) MA (see page 88) operates with a population of size μ and creates λ offspring in each generation. This is done by choosing randomly a parent, then mutating it, and, every τ generations, additionally applying local search to the result of the mutation. The population for the next generation is selected among the parents and the offspring, i. e., among $\mu + \lambda$ individuals, which explains the name (μ+λ) MA. Formally, the (μ+λ) MA represents a class of algorithms as different pivot rules may be used in the local search; it is parametrized by μ, λ, δ, and τ.

Note that the (μ+λ) MA does not accept worsenings at the end of a generation as only individuals with maximal fitness in $P_t \cup P'_t$ are chosen for the next generation. Such a selection strategy is called *elitist selection*.

The time until a global optimum is found—the optimization time—is defined by the number of f-evaluations until a global optimum is evaluated. Considering only

Algorithm 11 (μ+λ) Memetic Algorithm

Let $t := 0$.
Initialize P_0 with μ individuals chosen uniformly at random.
repeat
  $P'_t := \emptyset$.
  Do λ times:
    Choose $x \in P_t$ uniformly at random. // *parent selection*
    Create y by flipping each bit in x independently with probability $1/n$. // *mutation*
    if $t \bmod \tau = 0$ **then** $y :=$ local search(y). // *local search*
    $P'_t := P'_t \cup \{y\}$.
  Let P_{t+1} contain μ individuals from the multiset $P_t \cup P'_t$ with maximal f-value. (Break ties in favor of P'_t.) // *selection*
  $t := t + 1$.

f-evaluations is motivated by the fact that in practice these evaluations dominate the runtime. Local search constitutes an inner loop within the main loop, hence the number of f-evaluations within local search must be accounted for, too. Whatever pivot rule is used, the number of f-evaluations in one local search call is trivially bounded by δn. During t generations the number of f-evaluations is bounded by $t\lambda(1 + \delta n/\tau)$. This implies that if λ and δ are polynomial, then the optimization time is polynomial if and only if the number of generations until an optimum is found is polynomial.

Several well-known randomized search heuristics can be identified as special cases of the (μ+λ) MA. The (μ+λ) MA without local search, i. e., $\delta = 0$ or $\tau = \infty$, equals the (μ+λ) EA. The (1+1) MA with $\tau = 1$ represents an iterated local search algorithm (see, e. g., Lourenço et al., 2002).

4.1.2 Analysis of Population-based Evolutionary Algorithms

We prepare ourselves for the analysis of the (μ+λ) MA by first considering the (μ+λ) EA without local search. We analyze the dynamics within its population by means of so-called *family trees* as this will help us in Section 4.1.3 to analyze and understand the behavior of the (μ+λ) MA. Moreover, the results on the (μ+λ) EA are of independent interest.

The following results need to be put in an appropriate perspective. The analysis of evolutionary algorithms with family trees as well as the following results have been introduced by Witt (2006) for the analysis of the (μ+1) EA. Here, we show that Witt's approach can be generalized to the (μ+λ) EA with moderate technical effort. We think

that the generalized results are important enough to be presented in this separate section, although the improvements over Witt (2006) are minor from a technical point of view.

According to Witt, a family tree is a directed acyclic graph whose nodes represent individuals and edges represent direct parent-child relations created by a mutation-based evolutionary algorithm. After initialization, for every initial individual x^* there is a family tree containing only x^*. We say that x^* is the *root* of the family tree $T(x^*)$. Afterwards, whenever the algorithm chooses an individual $x \in T(x^*)$ as parent and creates an offspring y out of x, a new node representing y is added to $T(x^*)$ along with an edge from x to y. That way, $T(x^*)$ contains all descendants from x^* obtained by direct and indirect mutations.

As the $(\mu+\lambda)$ EA creates an infinite number of individuals, the growth of the family trees is infinite as well. Note, however, that there may be family trees containing only individuals that have been deleted from the current population. As μ individuals survive in every selection, at least one tree is guaranteed to grow. A subtree of a family tree is, again, a family tree. A (directed) path within a family tree from x to y represents a sequence of mutations creating y out of x. The number of edges on a longest path from the root x^* to a leaf determines the *depth* of $T(x^*)$.

Family trees can be used to derive lower bounds on the optimization time of mutation-based evolutionary algorithms as follows. Suppose that after some time t the depth of a family tree $T(x^*)$ is still small. Then typically the leaves are still quite similar to the root, which implies that the individuals in $T(x^*)$ are still concentrated around x^*. If the distance from x^* to all global optima is not too small, then it is unlikely that an optimum has been found after t steps.

The intuitive reasoning that $T(x^*)$ is concentrated around x^* is made rigorous in the following lemma. The proof is adapted from Witt (2006, Lemma 2 and proof of Theorem 4).

Lemma 4.1.1. *Let x^* be an individual entering the population in some generation t^*. The probability that within the following t/λ generations some $z^* \in T(x^*)$ emerges with $H(x^*, z^*) \geq 8t/\mu$ is $2^{-\Omega(t/\mu)}$.*

Proof. Consider the family tree $T(x^*)$ rooted with x^*. After generation t^*, $T(x^*)$ contains only x^*. In the following t/λ generations, t offspring are created. Let these offspring be $x_1, \ldots, x_t$, given in the order of creation. When the $(\mu+\lambda)$ EA is about to create a new individual, each individual in $T(x^*)$ is chosen for reproduction either with probability $1/\mu$ (if it is contained in the current population) or with probability 0 (if it has been deleted or if it is contained in the current offspring population). Hence, the probability that, given a fixed index sequence $1 \leq s_1 < \cdots < s_\ell \leq t$, a path is created in $T(x^*)$ containing the individuals $x^* = x_{s_0}, x_{s_1}, \ldots, x_{s_\ell}$ is at most

$$\prod_{i=0}^{\ell-1} \frac{\mathrm{mut}(x_{s_i}, x_{s_{i+1}})}{\mu} = \left(\frac{1}{\mu}\right)^{\ell} \cdot \prod_{i=0}^{\ell-1} \mathrm{mut}(x_{s_i}, x_{s_{i+1}})$$

where $\mathrm{mut}(x, y) := \mathrm{Prob}(y = \mathrm{mutation}(x))$. Observe that the latter product describes a process of ℓ consecutive independent mutations. Let F_ℓ describe the random number of bit flips during ℓ consecutive independent mutations. Clearly, this number is an upper bound for the Hamming distance $\mathrm{H}(x^*, s_\ell)$. Hence, the probability that a fixed path $x^*, x_{s_1}, \ldots, x_{s_\ell}$ emerges with a fixed index sequence $1 \leq s_1 < \cdots < s_\ell \leq t$ such that $\mathrm{H}(x^*, x_\ell) \geq 8t/\mu$ is bounded above by

$$\left(\frac{1}{\mu}\right)^\ell \cdot \mathrm{Prob}(F_\ell \geq 8t/\mu).$$

The number of possible paths of length exactly ℓ that may be created during t/λ generations is bounded by $\binom{t}{\ell}$, the number of ways to choose an index sequence $1 \leq s_1 < \cdots < s_\ell \leq t$. Summing over all values of ℓ, the probability that during t/λ generations some $z^* \in T(x^*)$ emerges with Hamming distance at least $8t/\mu$ to x^* is bounded by

$$\sum_{\ell=1}^{\infty} \binom{t}{\ell} \cdot \left(\frac{1}{\mu}\right)^\ell \cdot \mathrm{Prob}(F_\ell \geq 8t/\mu).$$

Using $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$ by Lemma A.10, we obtain the bound

$$\sum_{\ell=1}^{\infty} \left(\frac{et}{\ell\mu}\right)^\ell \cdot \mathrm{Prob}(F_\ell \geq 8t/\mu).$$

Due to random parent selection, the average length of a path is bounded by $t/\lambda \cdot 1/\mu \ll 3t/\mu$. We can bound the summands with $\ell > 3t/\mu$ if we exploit that such long paths are very unlikely (cf. Witt, 2006, Lemma 2).

$$\begin{aligned}
\sum_{\ell=3t/\mu+1}^{\infty} \left(\frac{et}{\ell\mu}\right)^\ell \cdot \mathrm{Prob}(F_\ell \geq 8t/\mu) \;&\leq\; \sum_{\ell=3t/\mu+1}^{\infty} \left(\frac{e}{3}\right)^\ell \\
&=\; \left(\frac{e}{3}\right)^{3t/\mu} \cdot \sum_{\ell=1}^{\infty} \left(\frac{e}{3}\right)^\ell \\
&=\; \left(\frac{e}{3}\right)^{3t/\mu} \cdot \frac{e}{3-e} = 2^{-\Omega(t/\mu)}.
\end{aligned}$$

Considering shorter paths of length $\ell \leq 3t/\mu$, it is very unlikely that more than $8t/\mu$ bits flip during ℓ mutations. The expected number of flipping bits equals $\mathrm{E}(F_\ell) = \ell < 4t/\mu$ and applying Chernoff bounds (Lemma A.6) with $\mathrm{E}(F_\ell)$ replaced by the upper bound $4t/\mu$, we obtain

$$\mathrm{Prob}(F_\ell \geq 8t/\mu) \;\leq\; (e/4)^{4t/\mu} \;\leq\; e^{-4t/(3\mu)}.$$

Now,

$$\sum_{\ell=1}^{3t/\mu} \left(\frac{et}{\ell\mu}\right)^{\ell} \cdot \text{Prob}(F_\ell \geq 8t/\mu) \;\leq\; e^{-4t/(3\mu)} \cdot \sum_{\ell=1}^{3t/\mu} \left(\frac{et}{\ell\mu}\right)^{\ell}$$
$$\leq\; e^{-4t/(3\mu)} \cdot 3t/\mu \cdot \max_{\ell=1}^{3t/\mu} \left(\frac{et}{\ell\mu}\right)^{\ell}.$$

Since the maximum of the term $(et/(\ell\mu))^\ell$ is attained for $\ell = t/\mu$, we arrive at the bound

$$e^{-4t/(3\mu)} \cdot 3t/\mu \cdot e^{t/\mu} \;=\; 3t/\mu \cdot e^{-t/(3\mu)} \;=\; 2^{-\Omega(t/\mu)}.$$

□

Another property of populations in mutation-based evolutionary algorithms is that with elitist selection at the end of the generation, high-fitness solutions tend to spread in the population very quickly. The following result was known before for $\lambda = 1$ (Witt, 2006), but the case $\lambda > 1$ requires different proof ideas.

Lemma 4.1.2. *Let x be an individual in the $(\mu+\lambda)$ MA's current population at some point of time and $n, \mu \geq 2$. The expected number of following generations until the population contains only individuals with fitness at least $f(x)$ is $O(\mu/\lambda \cdot \log \mu + \log \mu)$.*

Proof. Let x be a current elitist and call an individual *fit* if it has fitness at least $f(x)$. We now estimate the expected number of generations until the population is taken over by fit individuals, which we call the *expected takeover time*. As fit individuals are always preferred to non-fit individuals in the selection, the expected takeover time equals the expected number of generations until $\mu - 1$ fit individuals have been created, starting with one fit individual.

Let T_i describe the random number of generations needed to increase the number of fit individuals from 5^i to 5^{i+1}. Starting with a new generation with $5^i < \mu$ fit individuals in the parent population, we now consider a phase of 32μ offspring creations, disregarding generation boundaries. In one offspring creation process the $(\mu+\lambda)$ MA selects a fit individual as parent with probability at least $5^i/\mu$. Then with probability at least $(1-1/n)^n \geq 1/4$ for $n \geq 2$ mutation creates a clone. As local search can only improve the fitness, the resulting offspring is a fit individual, regardless whether local search has been called after the mutation or not. Let N_i denote the random number of new fit offspring created in the phase, then $\text{E}(N_i) \geq 32\mu \cdot 5^i/(4\mu) = 8 \cdot 5^i$ and by Chernoff bounds

$$\text{Prob}\big(N_i < 4 \cdot 5^i\big) \;\leq\; \exp(-\text{E}(N_i)/8) \;\leq\; \exp(-5^i) \;\leq\; \exp(-1).$$

If $N_i < 4 \cdot 5^i$ the phase is called unsuccessful and we consider another phase of 32μ offspring creations. The expected waiting time for a successful phase is $1/(1-\exp(-1))$

and the expected number of offspring creations until $N_i \geq 4 \cdot 5^i$ is $32\mu/(1-\exp(-1))$. Taking into account waiting times for the next generation boundary, this implies $\mathrm{E}(T_i) < 32\mu/(\lambda(1-\exp(-1))) + 1$ and the expected takeover time is at most

$$\begin{aligned}\sum_{i=0}^{\lceil \log_5 \mu \rceil - 1} \mathrm{E}(T_i) &\leq \lceil \log_5 \mu \rceil \cdot \left(\frac{32\mu}{\lambda(1-\exp(-1))} + 1 \right) \\ &= O(\mu/\lambda \cdot \log \mu + \log \mu).\end{aligned}$$

□

4.1.3 The Impact of the Local Search Depth

We now want to investigate the impact of the local search depth on the performance of the $(\mu+\lambda)$ MA. Our goal is to define a class of functions where there is only a small critical window for values of the local search depth δ in which an efficient optimization is possible. More precisely, there is an ideal value $D = D(n)$ for the local search depth in a sense that the choice $\delta = D$ guarantees an efficient optimization with high probability, while even small deviations of δ from D lead to superpolynomial runtimes. The value of D can be chosen almost arbitrarily given a fixed value of n.

It makes sense to restrict μ, λ, and the local search depth δ to polynomial values since, of course, a memetic algorithm cannot run in polynomial time if the initialization or a single generation with local search takes superpolynomial time. For important combinatorial problems, e. g. the traveling salesman problem (TSP), we do not expect that local search always finds a local optimum in polynomial time due to PLS-completeness results (Michiels, Aarts, and Korst, 2007). Moreover, artificial instances are known on which local search takes even exponential time, see Michiels et al. (2007) for the metric TSP and Englert, Röglin, and Vöcking (2007) for the Euclidean TSP.

Similar results also hold for pseudo-Boolean optimization. For a PLS-completeness result we refer to Michiels et al. (2007, Theorem 6.4). Specialized PLS-completeness results for memetic algorithms were presented by Krasnogor and Smith (2008). Rudolph (1997b) presented so-called *long k-paths*, paths of Hamming neighbors with increasing fitness whose length can be exponential (depending on k). An exponential length implies that the path has to be "folded" in $\{0,1\}^n$ in a sense that there are $i < j$ such that the i-th and the j-th point on the path have Hamming distance smaller than $j - i$. Standard mutations have a positive probability of jumping from the i-th to the j-th point, hence there is a chance to skip large parts of the path by taking a shortcut. However, long k-paths are constructed in such a way that at least k bits have to flip simultaneously in order to take a shortcut. The probability of such an event is exponentially small if $k = \Theta(\sqrt{n})$, in which case the path still has exponential length. A rigorous analysis for the (1+1) EA on long k-paths was presented by Droste et al. (2002a).

Long k-paths turn out to be very useful for our purposes. If we consider the first points of a long k-path and assign increasing fitness values to them, we obtain a fitness-increasing path of any desired length. We adapt the definition of long k-paths from Rudolph (1997b).

Definition 4.1.3. Let $k \in \mathbb{N}$ and n be a multiple of k. The long k-path $\mathcal{P}_n^k$ of dimension n is a sequence of bit strings from $\{0,1\}^n$ defined recursively as follows. The long k-path of dimension 0 is the empty bit string. Assume the long k-path of dimension $n-k$ is given by the sequence $\mathcal{P}_{n-k}^k = (p_1, \ldots, p_\ell)$ with bit strings $p_1, \ldots, p_\ell \in \{0,1\}^{n-k}$ and $\ell \in \mathbb{N}$. Then the long k-path of dimension n is defined by prepending k bits to these search points: let $S_0 := (0^k p_1, 0^k p_2, \ldots, 0^k p_\ell)$, $S_1 := (1^k p_\ell, 1^k p_{\ell-1}, \ldots, 1^k p_1)$, and $B := (0^{k-1}1 p_\ell, 0^{k-2}1^2 p_\ell, \ldots, 01^{k-1} p_\ell)$. The search points in S_0 and S_1 differ in the k leading bits and the search points in B represent a bridge between them. The long k-path of dimension n, $\mathcal{P}_n^k$, is the concatenation of S_0, B, and S_1.

Remark 4.1.4. This definition differs slightly from Rudolph's definition as he bases his construction on a long k-path $(0,1)$ of dimension 1. This implies that the rightmost bit of a long k-path is treated differently from the other bits as these bits form blocks of size k. In particular, this enforces $n-1$ to be a multiple of k. We think that the definition presented here is a more consistent way to define long k-paths and adapt the following results to this small change.

We only remark that, given some search point x, it is possible to check if x belongs to the long k-path and to compute its index on the path in linear time. Given two points $\mathrm{P}_s, \mathrm{P}_{s+i}$ for $i > 0$ and some index s, P_{s+i} is called the i-th *successor* of P_s and P_s is called a *predecessor* of P_{s+i}. Long k-paths have the following important properties. The proof is adapted from Droste et al. (2002a).

Lemma 4.1.5.

1. *The length of the long k-path of dimension n is defined as the number of bit strings in $\mathcal{P}_n^k$ and denoted by $\left|\mathcal{P}_n^k\right|$. We have $\left|\mathcal{P}_n^k\right| = k \cdot 2^{n/k} - k + 1$. All points on the path are different.*

2. *Let $\mathrm{P}_s \in \mathcal{P}_n^k$ and $\mathrm{P}_{s+i} \in \mathcal{P}_n^k$ with $s, i \in \mathbb{N}_0$.*
   If $i < k$ then $H(\mathrm{P}_s, \mathrm{P}_{s+i}) = i$, otherwise $H(\mathrm{P}_s, \mathrm{P}_{s+i}) \geq k$.

Proof. For $n = 0$ the length of the long k-path is $k \cdot 2^0 - k + 1 = 1$, the other statements are trivial.

Let the lemma hold for all dimensions smaller than n. The length of the long k-path of dimension n is $2 \cdot \left|\mathcal{P}_{n-k}^k\right| + k - 1 = 2 \cdot (k \cdot 2^{(n-k)/k} - k + 1) + k - 1 = k \cdot 2^{k/n} - k + 1$. By definition, all points on the path are different.

If $\mathrm{P}_s = 0^k p_a$ for $p_a \in S_0$, then the statement for $\mathrm{P}_{s+i} \in S_0$ follows from the induction hypothesis as the first k bits are fixed to zeros in all points of S_0. For bridge points,

we observe $\mathrm{H}\left(0^k p_a, 0^{k-j}1^j p_\ell\right) = \mathrm{H}\left(0^k p_a, 0^k p_\ell\right) + j$ as $0^k p_a$ and $0^k p_\ell$ only differ in the trailing $n-k$ bits and $0^k p_\ell$ and $0^{k-j}1^j p_\ell$ differ only in the k leading bits. Applying the induction hypothesis to p_a and p_ℓ proves the claim for $\mathrm{P}_{s+i} \in B$. Finally, all points S_1 differ from all points in S_0 by k leading bits, hence the claim follows for $\mathrm{P}_{s+i} \in S_1$.

If P_s is a point on the bridge, then the Hamming distance to the i-th successor on the bridge is i. For the other successors, observe $\mathrm{H}\left(0^j 1^{k-j} p_\ell, 1^k p_b\right) = \mathrm{H}\left(1^k p_\ell, 1^k p_b\right) + j$ and applying the induction hypothesis to p_ℓ and p_b yields the result.

Lastly, if $\mathrm{P}_s \in S_1$ the claim follows directly from the induction hypothesis. □

Throughout this section, k will be chosen as $\sqrt{n}$ (Section 4.1.3) or $\sqrt{n/2}$ (Section 4.1.4), both values yielding an exponential length for the long k-path. The long k-path will be simply referred to as long path if the value of k is obvious. In the following, we assume that n is not too small as our results are asymptotic and notions like polynomial times versus exponential times collapse if n is bounded above by a fixed constant. We also allow ourselves to write $c_1 \leq c_2 - o(1)$ for constants $c_1 < c_2$, assuming that n is large enough.

Now we are ready to define functions where the local search depth has a large impact on the $(\mu+\lambda)$ MA. The basic idea is to enforce the $(\mu+\lambda)$ MA to climb several paths with increasing fitness, each one ending in a local optimum. The paths are connected, such that mutation and local search may reach the next path from a local optimum. However, the first points on the new path have a very low fitness, meaning that local search has to run for a long time until it finds a solution that is competitive to the old local optimum. If the local search depth is too small, local search stops with an individual of worse fitness which is rejected immediately in the selection step. Hence, for an efficient runtime behavior the local search depth must not be too small.

On the other hand, we want to design our functions in such a way that a large local search depth also yields an inefficient behavior. If the local search depth is large, the next local optimum on the path is found deterministically. This behavior is bad if promising regions in the search space are found on the way to (but not too close to) the local optimum. We define regions of target points representing global optima somewhere on the way to the next local optimum. We thereby exploit that mutation and local search employ different neighborhoods: the target is placed with Hamming distance 2 to the path. All other search points, in particular those in between the path and the target, are assigned a very low fitness. This barrier prevents local search from traversing between the path and the target. For standard mutations flipping 2 bits are still very likely, hence the target can be reached by mutation if the parent is a path point close to the target. By these ideas, we obtain a function where the local search depth must not be too large if we want to find the target. However, when climbing the path the target can be missed easily. To ensure that the target is reached with high probability during a run, we repeat the construction several times with separate target regions such that

one target region is likely to be hit during a run. A sketch of this function is shown in Figure 4.1.

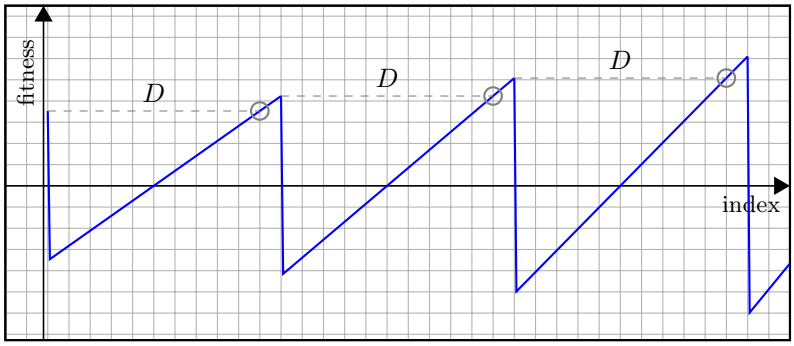

Figure 4.1: Sketch of the function f_D. The x-axis shows the index on the long k-path. The y-axis shows the fitness. The thick solid line shows the fitness of the points on the long k-path. Encircled path points are close to a target region with respect to Hamming distance. The long k-path can be separated into n disjoint paths with increasing fitness, each one ending with a local optimum. For the sake of clarity, only the first out of n paths are shown.

We now formalize this intuitive construction. Let P_i be the i-th point of the long k-path $\mathcal{P}_n^k$. For path points $\mathrm{P}_i, \mathrm{P}_j \in \mathcal{P}_n^k$ we define the *path distance* between P_i and P_j as $|i-j|$, i. e., the absolute index difference. Note that according to Lemma 4.1.5 the path distance coincides with the Hamming distance if $|i-j| < k$. Let $m = \log^3 n$ and choose some value $D = D(n)$ such that $D \geq m$. We now identify n paths as disjoint parts of the long k-path. On each path the fitness increases with growing index and the path ends with a local optimum. Each local optimum has a path distance of $D+m$ to the local optimum on the next path. The union of all such points is defined as

$$\text{Paths} := \bigcup_{i=0}^{n(D+m)} \{\mathrm{P}_i\}.$$

The fitness of a point on the long k-path is specified by the following height function. Define $q_i := \lceil i/(D+m) \rceil$ and

$$\text{height}(i) := \left(1 + \frac{2m}{D}\right)^{q_i} \cdot \left(i - q_i \cdot (D+m) + \frac{D}{2} + m\right).$$

This function is piecewise linear on intervals $i \in [j(D+m)+1, (j+1)(D+m)]$ on which $q_i = j+1$ and the height function simplifies to

$$\left(1+\frac{2m}{D}\right)^{j+1} \cdot \left(i - \frac{D}{2}\right).$$

As such a linear function is strictly increasing and $\text{height}(i(D+m)+D/2) = 0$, the points $\text{P}_{i\cdot(D+m)}$ are local optima. These points are also called *peaks*. The first points whose fitness is competitive to the fitness of the previous peak are called *competitive points*. They have a path distance of D from the previous peak as

$$\begin{aligned}\text{height}(i(D+m)) &= \left(1+\frac{2m}{D}\right)^{i} \cdot \left(\frac{D}{2}+m\right) \\ &= \left(1+\frac{2m}{D}\right)^{i+1} \cdot \frac{D}{2} = \text{height}(i(D+m)+D).\end{aligned}$$

It is therefore correct to define

$$\text{Peaks} := \bigcup_{i=0}^{n} \{\text{P}_{i(D+m)}\} \quad \text{and} \quad \text{Comp} := \bigcup_{i=0}^{n} \{\text{P}_{i(D+m)+D}\}.$$

The path distance from a competitive point to the next peak equals m and due to the properties of the long k-path, along with $m \leq k$, we also have that the Hamming distance to the next peak equals m. The target regions are defined such that they have Hamming distance 2 to the set of path points and Hamming distance at most $m/2$ to all competitive points. The latter condition immediately implies that the Hamming distance from every peak to every target point is at least $m/2$.

$$\text{Target} := \{x \mid \text{H}(x, \text{Paths}) = 2 \wedge \text{H}(x, \text{Comp}) \leq m/2\}.$$

Finally, the fitness function f_D is defined as follows.

Definition 4.1.6. Let $M := \max_{i=0}^{n(D+m)} \{|\text{height}(i)|\} + 1$ and

$$f_D(x) := \begin{cases} \text{height}(i) & \text{if } x = \text{P}_i \in \text{Paths}, \\ +M & \text{if } x \in \text{Target}, \\ -M & \text{otherwise.} \end{cases}$$

Obviously, all points in Target with fitness M are global optima. All points with fitness $-M$ form a plateau of equal fitness. Hence, if mutation creates an offspring y with $\text{H}(y, \text{Paths} \cup \text{Target}) > 1$, then y is surrounded by equally bad neighbors and local search stops with y. If the $(\mu+\lambda)$ MA starts with an initial population of larger fitness, such an offspring is rejected immediately.

Randomized search heuristics are usually initialized uniformly at random. The following theorem, however, considers a deterministic initialization where all individuals in the population occupy the first point on the path, P_0. This modification is not essential as the results can be adapted to hold for random initialization. One way to do this is described in detail in Section 3 of Sudholt (2006b); this involves an additional construction step, a slightly larger number of bits (without affecting polynomial runtime bounds), and a more complex fitness function. As this additional construction step does not give new insights, we allow ourselves to deal with a deterministic initialization for the sake of clarity and simplicity. The interested reader may consult Sudholt (2006a,b) for preliminary results that hold for random initialization.

Theorem 4.1.7. *Let $D \geq 2\log^3 n$, $\lambda = O(\mu)$, and $\mu, \delta, \tau = \mathrm{poly}(n)$. Initialize the $(\mu+\lambda)$ MA with μ copies of P_0, then the following holds:*

- *if $\delta = D$, the $(\mu+\lambda)$ MA optimizes f_D in polynomial time, w. h. p.*
- *if $|\delta - D| \geq \log^3 n$, the $(\mu+\lambda)$ MA needs superpolynomial time on f_D, w. h. p.*

Theorem 4.1.7 describes a small critical window of size only $m = \log^3 n$ where an efficient optimization is possible; outside this window the runtime is superpolynomial, w. h. p. We remark that relaxing the size of the critical window allows for stronger results. When replacing $m = \log^3 n$ by $m = n^\varepsilon$ for some $0 < \varepsilon \leq 1/2$ in the construction of f_D and Theorem 4.1.7, we obtain the following statement with slight adaptations of the proofs. If $\delta = D$, the $(\mu+\lambda)$ MA optimizes f_D in polynomial time, but if $|\delta - D| \geq n^\varepsilon$, the runtime is exponential. These claims then hold with overwhelming probability (cf. Sudholt, 2006a).

Proof of Theorem 4.1.7. Recall $m = \log^3 n$. We have already argued that all peaks have Hamming distance at least $m/2$ to all global optima. The probability of flipping at least $m/2$ bits in a single mutation is bounded by

$$\binom{n}{m/2} \cdot \left(\frac{1}{n}\right)^{m/2} \leq \frac{1}{(m/2)!} = 2^{-\Omega(m \log m)}.$$

By the union bound, w. h. p. even within $2^{cm \log m}$ mutations the probability that $m/2$ bits flip simultaneously in at least one mutation is still superpolynomially small if the constant $c > 0$ is small enough. In the following, we assume that less than $m/2$ bits flip in the first $2^{cm \log m}$ mutations and keep in mind the superpolynomially small error probability.

We first consider the case $\delta \leq D - m$ and observe that P_D is the first successor of P_0 such that $f_D(\mathrm{P}_D) \geq f_D(\mathrm{P}_0)$. Since all initial search points have fitness $f_D(\mathrm{P}_0)$, along with elitist selection, the only way to alter the current population is to create an offspring P_i with $i \geq D$ by mutation and/or local search. As $\delta \leq D - m$,

it is necessary for mutation to create some $y \in N^*(\{\mathrm{P}_{m+1}, \ldots, \mathrm{P}_{D+m}\})$. However, $\mathrm{H}(\mathrm{P}_0, N^*(\{\mathrm{P}_{m+1}, \ldots, \mathrm{P}_{D+m}\})) \geq m$. Hence, apart from a superpolynomially small error probability the $(\mu+\lambda)$ MA cannot find an optimum within a polynomial number of steps.

For the case $\delta = D$ we estimate the expected number of generations until the path to the next peak is climbed, as long as neither the target nor the very last peak has been hit. Let x be a current elitist, i. e., an individual of maximal fitness. If $x \in \text{Paths} \setminus (\text{Peaks} \cup \text{Target})$, there is a Hamming neighbor with larger fitness and the probability of choosing x and producing an offspring with larger fitness is at least $1/(en\mu)$. The probability that this event happens at least once among λ offspring creations can be estimated using $1 - x \leq e^{-x}$ for $x \in \mathbb{R}$, $e^{-x} \leq 1 - x/2$ for $0 \leq x \leq 1$ (see Lemma A.12), and $\lambda/(en\mu) \leq 1$ (recall $\lambda = O(\mu)$) if n is large enough.

$$1 - \left(1 - \frac{1}{en\mu}\right)^{\lambda} \;\geq\; 1 - e^{-\frac{\lambda}{en\mu}} \;\geq\; 1 - \left(1 - \frac{\lambda}{2en\mu}\right) \;=\; \frac{\lambda}{2en\mu}. \tag{4.1}$$

Hence, the expected number of generations until the next peak or a better individual is reached is at most $2en\mu/\lambda \cdot (D+m) = O(n\mu\delta/\lambda)$. Contrarily, if x is a peak, then every τ generations there is a chance to create an individual on the next path by mutation and then to climb the path by local search. If mutation creates the first successor of x, local search may run back into x, depending on the pivot rule. However, if the second successor of x is created, then the path is climbed and the resulting offspring has larger fitness. As this holds regardless of the pivot rule, we rely on the latter event in the following. Similar to (4.1), the expected number of generations until the best fitness increases is bounded by $\tau \cdot 2en^2\mu/\lambda$. Together, the expected number of generations until the path to the next peak is climbed is $O(n^2\mu\tau/\lambda)$.

We see that there are ways to climb the next path efficiently. In order to show that the target is likely to be hit, we have to consider all possible ways to climb the path, which requires a more careful argumentation. We divide a run into phases. Phase 0 starts with the initialization. Phase i, $1 \leq i \leq n-1$, starts when the $(i-1)$-th phase has ended. For $0 \leq i \leq n-1$, Phase i ends when an individual in $S_i := \{\mathrm{P}_{i(D+m)+D}, \ldots, \mathrm{P}_{i(D+m)+D+m/2-2}\}$ is mutated for the first time. Assume that at most $m/2-2$ bits flip in one mutation during all phases, which happens with high probability. We claim that then we have a constant probability that the target is hit at the end of each phase.

Consider some Phase i. We already argued that a path cannot be climbed by mutation, only, hence we consider a generation with local search. In such a generation, the path can only be climbed if mutation creates a point $y \in \{\mathrm{P}_{i(D+m)+1}, \ldots, \mathrm{P}_{i(D+m)+m/2-2}\}$ or a Hamming neighbor thereof that does not belong to the path. In both cases local search ends with a point $z \in S_i$. All points in S_i are selected for the next generation, hence there is a chance that a search point in S_i will be chosen as parent in a future

generation, resulting in the end of Phase i. Note that such an event is necessary in order to advance to the next path.

Consider the mutation of $\mathrm{P}_j \in S_i$ ending Phase i. Due to the definition of Target, all search points with Hamming distance 2 from P_j either belong to Target or to $N(\{\mathrm{P}_{j-3}, \mathrm{P}_{j-1}, \mathrm{P}_{j+1}, \mathrm{P}_{j+3}\})$. The number of search points where the latter condition holds is bounded by $4n$. Thus, among all search points with Hamming distance 2 from P_j, a $(1 - O(1/n))$-fraction belongs to Target, i. e., almost every 2-bit mutation will hit the target. The probability of an arbitrary 2-bit mutation is $\binom{n}{2}(1/n^2)(1-1/n)^{n-2} \geq 1/(2e)$. Hence the probability of hitting the target at the end of Phase i is $\Omega(1)$. We conclude that the probability of not hitting the target in all n phases is exponentially small. Moreover, the expected number of phases before hitting the target or the very last peak is $O(1)$. Given that no error occurs, which happens with a superpolynomially small probability, the conditional expected number of generations is $O(n^2\mu\tau/\lambda)$. Recall that the computational effort within one generation is polynomial. Markov's inequality then implies that the optimization time is polynomial with high probability.

Now consider the case $\delta \geq D + m$. Since the longest fitness-increasing sequence of Hamming neighbors in $\{0,1\}^n$ contains $D + m \leq \delta$ transitions, generations with local search are guaranteed to create peaks, global optima, or local optima with fitness $-M$ that are immediately rejected in the following selection. We will show that a sequence of mutations cannot lead us far away from all peaks in a typical run. In case $\mu = 1$ the population always contains a peak, hence we assume $\mu \geq 2$.

Once a new peak x is created, it may happen that mutation of x creates some predecessor y on the long path. Despite the use of elitist selection, as long as the population still contains older individuals with lower fitness, selection prefers y over these worse individuals and y remains in the population. This way, if the population size is not too small, the path leading to x may be climbed down and the Hamming distance to the target can decrease.

Let x be such a new peak, then by Lemma 4.1.2 the expected number of generations until the population consists of individuals with fitness at least $f_D(x)$ only, called the expected takeover time, is $O(\mu/\lambda \cdot \log \mu + \log \mu) = O(\mu/\lambda \cdot \log \mu)$ since $\lambda = O(\mu)$. By Markov's inequality, the probability that the takeover time is larger than $c\mu/\lambda \cdot \log \mu$ is at most $1/2$ for a sufficiently large constant $c > 0$. Considering independent phases of length $\lceil c\mu/\lambda \cdot \log \mu \rceil$ each, the probability that the takeover time is larger than $\lfloor \mu m/(24\lambda) \rfloor$ is $2^{-\Omega(m/(\log \mu))} = 2^{-\Omega(\log^2 n)}$ and hence superpolynomially small.

It is unlikely that the $(\mu+\lambda)$ MA reaches the target by climbing down the path before the population is taken over by good individuals. We first consider the behavior of the $(\mu+\lambda)$ MA in generations without local search, i. e., the behavior of the $(\mu+\lambda)$ EA, and then take into account generations with local search. Fix a search point $x \in$ Peaks, then applying Lemma 4.1.1 with $t/\lambda := \lfloor \mu m/(24\lambda) \rfloor$ yields that the probability of moving away from the peak by Hamming distance at least $m/2$ is $2^{-\Omega(m)}$. If, in a generation

with local search, a peak is created, we simply apply Lemma 4.1.1 anew for all those offspring. By the union bound, the error probability for all applications of Lemma 4.1.1 in polynomial time is still superpolynomially small.

With high probability, at some point of time the population contains μ copies of the last peak $\mathrm{P}_{n(D+m)}$. In this case at least $m/2$ bits have to flip in one mutation in order to find the target. Recall that the time until such an event occurs is superpolynomial with high probability. □

An interesting question is whether the results will change if the $(\mu+\lambda)$ MA is enhanced by a crossover operator that is applied prior to mutation, with some constant probability p_c, $0 < p_c < 1$. Uniform crossover copies the bit value for each bit either from the first or from the second parent; this choice is made independently and uniformly for each bit. This means that bits where both parent differ are assigned randomly. When a uniform crossover of two parents with Hamming distance h is performed, the offspring is uniform in the space of 2^h possible offspring. If both parents are very similar (i. e., if h is small), an offspring created by crossover also has a good chance to be created by mutation of the closer parent. If both parents are path points and h is not too small, it is very unlikely that the offspring is again a path point (or a global optimum) as, intuitively, the "density" of these points in the search space is quite low. It is therefore very likely that crossover creates a search point with fitness $-M$, local search cannot find an improvement, and the offspring is finally rejected by selection.

Other operators like ℓ-point crossover (this crossover randomly cuts both parents into ℓ pieces and concatenates pieces for the offspring from alternating parents) may exploit the block structure underlying long k-paths, so that shortcuts on the path cannot be excluded any more. However, we can simply change the binary encoding of our function by shuffling all bits to destroy the linkage between these bits. In this case ℓ-point crossover probably will not have a significant advantage over mutation either. We therefore conjecture that Theorem 4.1.7 can be adapted for a $(\mu+\lambda)$ MA with uniform or ℓ-point crossover while preserving a critical window of polylogarithmic size. A proof for this conjecture, however, remains open.

4.1.4 The Impact of the Local Search Frequency

The local search frequency is another essential parameter in the design of memetic algorithms as it can be used to control the influence of the hybrid's two components. As local search is a very greedy strategy, the local search frequency can also be regarded to control the amount of greediness. From the perspective of evolutionary computation, one often speaks of balancing exploration and exploitation.

We will show that the choice of the local search frequency can also have a tremendous impact on the performance of the $(\mu+\lambda)$ MA, similar to the results on the local search depth in the previous section. The analysis of the constructed functions is more involved

than the one made for the local search depth. Therefore, in the following we will only deal with the (1+1) MA. We define two functions called Race^{con} and $\text{Race}^{\text{uncon}}$ according to given values for n, δ, and τ. The (1+1) MA is efficient on Race^{con}, but inefficient on $\text{Race}^{\text{uncon}}$. Now, if the local search frequency is halved, the (1+1) MA suddenly becomes inefficient on Race^{con}, but efficient on $\text{Race}^{\text{uncon}}$.

The functions Race^{con} and $\text{Race}^{\text{uncon}}$, which we call *race functions*, are constructed in similar ways, so we describe them both at once. First of all, we partition all bit strings into their left and right halves, which form two subspaces $\{0,1\}^{n/2}$ within the original space $\{0,1\}^n$ for even n. Each subspace contains a part of a long path. Except for special cases, the fitness is the (weighted) sum of the positions on the two paths. This way, climbing either path is rewarded and the (1+1) MA is encouraged to climb both paths in parallel.

The difference between the two paths in the left and right half of the bit string is that they are adapted to the two neighborhoods used by mutation and local search, respectively. In the left half, we have a connected path consisting of the first $\Theta(n^4 \cdot \delta/\tau)$ points of a long k-path, $k = \sqrt{n}/2$. The right half contains a similar path with path distance (i. e., absolute index difference) only $\Theta(n)$ to the end of the path, but only every third point of the path is present. Instead of a connected path, we have a sequence of isolated peaks where the closest peaks have Hamming distance 3. As the peaks form a path of peaks, we speak of an *unconnected path*. While the unconnected path cannot be climbed by local search, mutation can jump from peak to peak as a mutation of 3 specific bits has probability at least $1/(en^3)$. Concluding, local search is well suited to climb the connected path while mutation is well suited to climb the unconnected path.

Now, the main idea is as follows: if the local search frequency is high, we expect the (1+1) MA to optimize the connected path prior to the unconnected path. Contrarily, if the local search frequency is low, the (1+1) MA is likely to optimize the unconnected path prior to the connected one. Which path is optimized first can make a large performance difference. In the special cases where the end of any path is reached, we define separate fitness values for Race^{con} and $\text{Race}^{\text{uncon}}$. For Race^{con}, if the connected path is optimized first (i. e., wins the race), a global optimum is found. However, if the unconnected path wins the race, Race^{con} turns into a so-called deceptive function that gives hints to move away from all global optima and to get stuck in a local optimum. In this situation, the expected time to reach a global optimum is exponential. For $\text{Race}^{\text{uncon}}$, the (1+1) MA gets trapped in the same way if the connected path wins and a global optimum is found in case the unconnected path wins. A similar construction has been presented by Witt (2008) for a function where the choice of the population size decides whether a global optimum or a trap is reached.

We formalize the above-mentioned race functions. The connected path has length $\ell = \Theta(n^4 \cdot \delta/\tau)$. The precise value of ℓ including constant factors and terms of smaller order will become obvious during the analysis of the (1+1) MA. The unconnected path

has length $r = n^5 + \Theta(n)$, but for technical reasons the (1+1) MA starts with the n^5-th point on the path.

Definition 4.1.8. Let $n = 2k^2$ for some $k \in \mathbb{N}$, k a multiple of 3, and let $\mathcal{P}^k_{n/2} = (\mathrm{P}_0, \mathrm{P}_1, \dots)$ be the long k-path of dimension $n/2$. For $w \in \mathcal{P}^k_{n/2}$ let $p(w) = i$ if $w = \mathrm{P}_i$.

For a search point $x \in \{0,1\}^n$ we denote $x = x'x''$ with $x', x'' \in \{0,1\}^{n/2}$ and call x *well-formed* if and only if $x', x'' \in \mathcal{P}^k_{n/2}$ and $p(x'')/3 \in \mathbb{N}_0$.

Given appropriate lengths $\ell = \Theta(n^4 \cdot \delta/\tau)$ and $r = n^5 + \Theta(n)$, we define

$$\mathrm{Race}^{\mathrm{con}}(x) := \begin{cases} n \cdot p(x') + p(x'') & \text{if } x \text{ well-formed}, p(x') < \ell, p(x'') < r, \\ 2^n - p(x') & \text{if } x \text{ well-formed}, p(x') < \ell, p(x'') \geq r, \\ 3^n & \text{if } x \text{ well-formed}, p(x') \geq \ell, \\ -1 & \text{otherwise.} \end{cases}$$

$$\mathrm{Race}^{\mathrm{uncon}}(x) := \begin{cases} n \cdot p(x') + p(x'') & \text{if } x \text{ well-formed}, p(x') < \ell, p(x'') < r, \\ 2^n - p(x'') & \text{if } x \text{ well-formed}, p(x') \geq \ell, p(x'') < r, \\ 3^n & \text{if } x \text{ well-formed}, p(x'') \geq r, \\ -1 & \text{otherwise.} \end{cases}$$

Now we state the main result of this section. The preconditions $\delta \geq 36$, $\delta/\tau \geq 2/n$, and $\tau = O(n^3)$ require that "enough" iterations of local search are performed during a polynomial number of generations. The reason is that local search must be a visible component in the algorithm for the different local search frequencies to take effect. The condition $\tau = n^{\Omega(1)}$ is required for technical reasons.

Theorem 4.1.9. *Let $\delta = \mathrm{poly}(n)$, $\delta \geq 36$, $\delta/\tau \geq 2/n$, $\tau = n^{\Omega(1)}$, and $\tau = O(n^3)$. If the* (1+1) MA *starts with $\mathrm{P}_0\,\mathrm{P}_{n^5}$, then with overwhelming probability*

- *the* (1+1) MA *with local search frequency $1/\tau$ optimizes* $\mathrm{Race}^{\mathrm{con}}$ *in polynomial time while the* (1+1) MA *with local search frequency $1/(2\tau)$ needs exponential time on* $\mathrm{Race}^{\mathrm{con}}$ *and*
- *the* (1+1) MA *with local search frequency $1/\tau$ needs exponential time on* $\mathrm{Race}^{\mathrm{uncon}}$ *while the* (1+1) MA *with local search frequency $1/(2\tau)$ optimizes* $\mathrm{Race}^{\mathrm{uncon}}$ *in polynomial time.*

To prove this theorem, we will show that in one parameter setting one specific path is optimized in a fixed amount of time and the other one is not. Therefore, we investigate the progress of the algorithm on the two paths, i. e., the distance traveled on the path. If $x_t = x'_t x''_t$ is the current search point in generation t, the progress in generation t on the connected (unconnected) path is defined as $p(x'_{t+1}) - p(x'_t)$ ($p(x''_{t+1}) - p(x''_t)$).

The progress in a set of generations is the sum of the progress values for all considered generations.

In the following, we derive upper and lower bounds on the progress in generations with and without local search, both for the connected and the unconnected path, respectively. In progress estimates we disregard situations where the end of a path is reached and assume that the fitness of a search point x is always $n \cdot p(x') + p(x')$ if x is well-formed and -1 otherwise. Applying the results to Race^{con} and $\text{Race}^{\text{uncon}}$ will be straightforward.

The task of estimating the progress on a single long path in the whole search space is not too difficult. However, with two long paths, the random progress variables for both paths are not independent as selection is based on both paths. One requirement for non-zero progress is to create a well-formed offspring. Then the fitness of the current search point is increased if the offspring advances on both paths at the same time or if the offspring advances on one path and stands still on the other one. Moreover, the fitness is increased if the offspring advances on the connected path and steps back on the unconnected one (assuming no shortcut is taken) as the position on the connected path is weighted with the factor n. In all other cases the (1+1) MA does not change the current solution.

Comparing the probability of reaching the d-th successor with the probability of reaching a further successor on a long path for $d < k$, we observe that the latter probability is by a factor of $O(1/n)$ smaller than the former one since at least one additional bit is required to flip. Based on this simple fact, we can derive a simple and useful bound on the progress applicable to both the connected and the unconnected path.

Lemma 4.1.10. *Let $X_1, \ldots, X_m$ be a sequence of independent and identically distributed random variables over $\mathbb{Z}$ with $X_i \leq k = n^{1/2}/2$ for $1 \leq i \leq m$ and let $X = \sum_{i=1}^{m} X_i$. Given a constant $d \in \mathbb{N}$, let $p := \text{Prob}(0 < X_i \leq d)$. If $\text{Prob}(X_i > d) = O(p/n)$, then for every constant $\varepsilon > 0$*

$$\text{Prob}\Big(X \geq \max\{(1+\varepsilon)dmp,\ \varepsilon n^{3/4}\}\Big) \ = \ e^{-\Omega(n^{1/4})}.$$

Proof. We can assume $\varepsilon < 1$, otherwise we prove a stronger bound by replacing ε with $0 < \varepsilon' < 1$. A random variable X_i is called a *small step* if $0 < X_i \leq d$ and a *large step* if $X_i > d$. The expected number of small steps is mp and the expected number of large steps is $O(mp/n)$. Consider the case $mp \geq \varepsilon n^{1/4}$. By Chernoff bounds, the number of small steps is at least $(1 + \varepsilon(1 - n^{-1/4})) \cdot mp$ with probability $\exp(-\Omega(mp)) = \exp(-\Omega(n^{1/4}))$. Moreover, the number of large steps is at least $\varepsilon n^{1/4} \cdot mp/n$ with probability $(n^{-1/4})^{\Omega(n^{1/4})} = \exp(-\Omega(n^{1/4}))$. Together, using the bounds $X_i \leq d$ for small steps, $X_i \leq k \leq n^{1/2}$ for large steps, and $X_i \leq 0$ for all other steps, with probability $1 - \exp(-\Omega(n^{1/4}))$

$$X \ \leq \ (1 + \varepsilon(1 - n^{-1/4})) \cdot dmp + \varepsilon n^{3/4} \cdot mp/n \ \leq \ (1+\varepsilon) \cdot dmp.$$

Now let $mp \leq \varepsilon n^{1/4}$ and observe $\text{Prob}(X_i > 0) = p(1 + O(1/n))$, i. e., the expected number of random variables with value greater than 0 is at most $\varepsilon n^{1/4}(1+O(1/n))$. The probability that at least $2\varepsilon n^{1/4}$ random variables are greater than 0 is $\exp(-\Omega(n^{1/4}))$ by Chernoff bounds. Applying the trivial upper bound k to these variables, we have $X \leq 2\varepsilon n^{1/4} \cdot k = \varepsilon n^{3/4}$ with probability $1 - \exp(-\Omega(n^{1/4}))$. □

For our progress estimates, we first consider the progress by mutations in a set of T generations without local search. The expected progress on the connected path is about $T/(en)$ as the probability of hitting the next path point in one step is about $1/(en)$. Similarly, the expected progress on the unconnected path is about $3T/(en^3)$ since the next peak is hit with probability about $1/(en^3)$ and such a step yields a progress of 3. Hence the following lemma should not come as a surprise.

Lemma 4.1.11. *Let $\Delta_{\text{mut}}^{\text{con}}$ ($\Delta_{\text{mut}}^{\text{uncon}}$) be the progress on the connected (unconnected) path in $T = \Theta(n^4)$ generations without local search. For every constant $0 < \varepsilon < 1$ w. o. p.*

$$(1-\varepsilon) \cdot \frac{T}{en} \;<\; \Delta_{\text{mut}}^{\text{con}} \;<\; (1+\varepsilon) \cdot \frac{T}{en}$$

and

$$(1-\varepsilon) \cdot \frac{3T}{en^3} \;<\; \Delta_{\text{mut}}^{\text{uncon}} \;<\; (1+\varepsilon) \cdot \frac{3T}{en^3}.$$

Proof. The probability that at least k bits flip simultaneously in one of T generations is exponentially small, hence it suffices to consider mutations of less than k bits if we keep in mind an exponentially small error probability.

Consider either one of the two subspaces and a mutation of a well-formed parent. Then reaching the d-th successor on the path in the considered subspace has probability $n^{-d} \cdot (1-1/n)^{n/2-d}$. In order to obtain a well-formed offspring, in the other subspace an appropriate path point has to be created, which happens with probability $(1-1/n)^{n/2} \cdot (1 + o(1))$, dominated by the probability $(1-1/n)^{n/2}$ to clone the current path point. Together, the probability of reaching a well-formed offspring and advancing by $1 \leq d \leq k$ on a fixed long path (provided $d/3 \in \mathbb{N}$ in case of the unconnected path) has probability $(1 + o(1)) \cdot (1-1/n)^{n-d} \cdot n^{-d} = (1 + o(1)) \cdot n^{-d}/e$.

Let x_t be the current search point in generation t. We see that, if Lemma 4.1.10 is applied to random variables $p(x'_{t+1}) - p(x'_t)$ describing the progress on the connected path, all necessary conditions are fulfilled and we obtain the progress bound $(1+\varepsilon) \cdot T/(en)$ if Lemma 4.1.10 is applied w. r. t. an appropriate $\varepsilon' = \varepsilon - o(1)$ and $d = 1$. Similarly, Lemma 4.1.10 with $d = 3$ yields the progress bound $(1+\varepsilon) \cdot 3T/(en^3)$ on the unconnected path.

For a lower bound on the connected path, observe that the probability of having progress 1 in one generation is at least $1/(en)$. By a direct application of Chernoff bounds, the progress in T steps is bounded below by $(1-\varepsilon) \cdot T/(en)$ with probability

$\exp(-\Omega(T/n)) = \exp(-\Omega(n^3))$. We hereby exploit that, as long as no shortcuts are taken, the progress on the connected path cannot be negative since a step from P_s to $P_{s+i}, i \neq 0$, is accepted if and only if $i > 0$.

The progress on the unconnected path, however, depends on the progress on the connected path since the position on the connected path dominates the fitness. This implies that a mutation may step back on the unconnected path and advance on the connected one and the offspring will nevertheless be accepted. Such a regressing mutation has probability $O(1/n^4)$. Applying Lemma 4.1.10 to random variables $\max\{0, p(x''_t) - p(x''_{t+1})\}$, the progress by these steps is at least $-n^{3/4}$. On the other hand, we have progress 3 with probability at least $1/(en^3)$ and by Chernoff bounds, we have at least $(1 - \varepsilon + cn^{-1/4}) \cdot T/(en^3) \geq (1 - \varepsilon) \cdot T/(en^3) + c \cdot \Omega(n^{3/4})$ such steps with probability $\exp(-\Omega(T/n)) = \exp(-\Omega(n))$ for every constant $c > 0$. If c is large enough, the progress is at least $(1 - \varepsilon) \cdot 3T/(en^3)$ w. o. p. □

In each iteration, local search flips a single bit in one of the two subspaces. As we did not specify a pivot rule, we cannot tell a priori which subspace will be concerned by which iteration of local search. But a closer look at the fitness function reveals that we can indeed make exact predictions on the behavior of local search.

In the subspace containing the unconnected path, local search can have at most one fitness-improving iteration, namely moving from a Hamming neighbor of a peak to the peak itself. On the other hand, local search can spend many iterations climbing the connected path in the other subspace. Consider a generation with local search where the resulting offspring is accepted. We claim that either δ or $\delta - 1$ iterations of local search concern x' and in the latter case, the first iteration alters x''.

Consider a well-formed parent x, a mutation y of x and the outcome z of local search applied to y. It is obvious that, if y has Hamming distance larger than 1 to all well-formed search points, local search stops with $z = y$, resulting in an offspring with fitness -1 that is rejected by selection. If y has Hamming distance 1 to the set of all well-formed search points, y has exactly one well-formed Hamming neighbor due to the structure of long paths and the fact that the Hamming distance between two peaks is larger than 2. Hence, there is a unique "wrong" bit in x' or x'' that is flipped by the first iteration of local search. The remaining $\delta - 1$ iterations then climb the connected path. Lastly, if y is well-formed, then all δ iterations climb the connected path.

The difference between $\delta - 1$ and δ iterations of local search climbing the connected path is not large. In both cases it is very likely that $p(z') > p(x')$ (unless mutation created some predecessor far behind x') and selection typically accepts z regardless of the progress on the unconnected path. Contrarily, for the unconnected path one iteration of local search can make a large difference. The probability of reaching the next peak on the unconnected path by a direct mutation is approximately $1/(en^3)$. However, in a generation with local search, the same peak is also reached if mutation hits one of its Hamming neighbors as then the first iteration of local search will climb the peak. Three

of these Hamming neighbors have only Hamming distance 2 to x, yielding a probability of approximately $3/(en^2)$ or reaching the next peak. If $x'' \neq \mathrm{P}_0$, we also have a probability of around $3/(en^2)$ of reaching the previous peak. We conclude that on the one hand, one iteration of local search significantly increases the probability of reaching the next or the previous peak. On the other hand, selection typically does not distinguish between these two cases, hence the expected progress in these steps on the unconnected path is close to 0. In other words, the random progress on the unconnected path in these steps is almost a martingale. In the following lemma we use a concentration result from martingale theory to show that the total progress in these steps is close to 0.

Lemma 4.1.12. *Let $\Delta_{\mathrm{ls}}^{\mathrm{uncon}}$ be the progress on the unconnected path in $T = O(n^4)$ generations with local search, provided that the parents are well-formed and their position on the unconnected path is greater than 0. Let $\delta \geq 6$, then for every $0 < \varepsilon \leq 1/4$ w. o. p.*

$$-6(T/n^2)^{1/2+\varepsilon} - n^{3/4} \;<\; \Delta_{\mathrm{ls}}^{\mathrm{uncon}} \;<\; 6(T/n^2)^{1/2+\varepsilon} + n^{3/4}.$$

Proof. Like in the proof of Lemma 4.1.11, we only consider mutations flipping less than k bits at once.

Let $x = x' \mathrm{P}_s$ be the current well-formed search point. We first derive a fairly tight bound on the probability of having progress $3i$ on the unconnected path for some $1 \leq i < k/3$ in a generation with local search. We have progress $3i$ if and only if local search traverses $y' \mathrm{P}_{s+3i}$ for some y' on the connected path and the outcome of local search is accepted. The argumentation preceding this lemma showed that such a search point has to be traversed either before or after the first iteration of local search. This is equivalent to the event that mutation creates a point in $N^*(y' \mathrm{P}_{s+3i})$.

First consider the case $y' = x'$. With probability $(1/n)^{3i} \cdot (1-1/n)^{n-3i}$, $x' \mathrm{P}_{s+3i}$ is reached directly by mutation. Moreover, there are $3i$ Hamming neighbors of $x' \mathrm{P}_{s+3i}$ with Hamming distance $3i-1$ to x and $n-3i$ Hamming neighbors with Hamming distance $3i+1$. Thus, the probability of creating a mutant in $N^*(x' \mathrm{P}_{s+3i})$ is

$$n^{-3i}\left(1-\frac{1}{n}\right)^{n-3i} + 3i \cdot n^{-(3i-1)}\left(1-\frac{1}{n}\right)^{n-3i+1} + (n-3i) \cdot n^{-(3i+1)}\left(1-\frac{1}{n}\right)^{n-3i-1}.$$

Note that for $1 \leq j \leq k$, we have $1/e \leq (1-1/n)^{n-j} \leq 1/e \cdot (1-1/n)^{n-n^{1/2}} = (1+o(1))/e$ and the above bound is of order

$$\frac{1+o(1)}{e} \cdot n^{-3i+1}\left[\frac{1}{n} + 3i + \frac{n-3i}{n^2}\right] \;=\; \frac{1+o(1)}{e} \cdot 3i \cdot n^{-3i+1}.$$

The probability of creating a point in $N^*(y' \mathrm{P}_{s+3i})$ for a different path point $y' \neq x'$ such that the outcome of local search is accepted is by a factor of $O(1/n)$ smaller. We conclude that progress $3i$ on the unconnected path has probability $(1+o(1)) \cdot 3i/e \cdot n^{-3i+1}$.

Observe that $y' \mathrm{P}_{s-3i}$ has the same probability of being traversed as $y' \mathrm{P}_{s+3i}$ in case $s - 3i \geq 0$ (otherwise, the former probability is 0). Regarding the effect of selection, $y' \mathrm{P}_{s+3i}$ is more or equally likely to yield an accepted search point at the end of the generation than $y' \mathrm{P}_{s-3i}$. Thus, if α is w. o. p. an upper bound for the progress in T generations with local search, $-\alpha$ is a lower bound on the progress w. o. p. It suffices to prove the claimed upper bound.

To this end, we focus on generations with progress ± 3 since these steps have the largest impact on the total progress. In all other generations, the probability of having non-zero progress is $O(n^{-5})$ and by Lemma 4.1.10, the progress in these generations is bounded by $n^{3/4}/2$.

Let S be the number of ± 3-steps in T generations, then $\mathrm{E}(S) = (1 + o(1)) \cdot 3/e \cdot T/n^2$. If $T = O(n^{5/2})$ then Chernoff bounds imply $S < n^{3/4}/6$ w. o. p. and the bound $n^{3/4}/2$ for the progress by ± 3-steps is immediate, proving the claim. On the other hand, if $T = \Omega(n^{5/2})$ then $\mathrm{Prob}\big(S \geq 2T/n^2\big) = \exp(-\Omega(T/n^2)) = \exp(-\Omega(n^{1/2}))$. Assume in the following that $S < 2T/n^2$.

Let $X_1, \ldots, X_S \in \{-1, 0, +1\}$ be random variables such that $3X_j$ indicates the progress by the j-th ± 3-step. Then

$$|\mathrm{E}(\Delta_{\mathrm{ls}}^{\mathrm{uncon}} \mid X_1, \ldots, X_j) - \mathrm{E}(\Delta_{\mathrm{ls}}^{\mathrm{uncon}} \mid X_1, \ldots, X_{j-1})| \leq 1$$

and we can apply the method of bounded martingale differences (Lemma A.8). Let $X := X_1 + \cdots + X_S$ and $\alpha = S^{1/2+\varepsilon} + n^{1/2+\varepsilon}/6$, then

$$\begin{aligned} \mathrm{Prob}(X \geq \mathrm{E}(X) + \alpha) &\leq \exp(-\alpha^2/(2S)) \\ &\leq \exp(-\Omega(S^{2\varepsilon} + n^{1+2\varepsilon}/S)) \\ &\leq \exp(-\Omega(n^{2\varepsilon})) \end{aligned}$$

as $S^{2\varepsilon} + n^{1+2\varepsilon}/S \geq n^{2\varepsilon}$ both if $S \geq n$ and if $S < n$. It remains to bound $\mathrm{E}(X)$. Let $x = x' \mathrm{P}_s$ be the current well-formed search point, $y' \mathrm{P}_{s+3i}$ be the first well-formed search point reached during local search and $z = z' \mathrm{P}_{s+3i}$ be the outcome of local search. The assumption $s > 0$ implies that $y' \mathrm{P}_{s+3}$ and $y' \mathrm{P}_{s-3}$ have the same probability of being reached. Selection only treats $z' \mathrm{P}_{s+3}$ and $z' \mathrm{P}_{s-3}$ differently if $p(z') = p(x')$. Since at least $\delta - 1$ iterations of local search climb the connected path, a necessary condition for $p(z') = p(x')$ is that mutation creates the $(\delta - 1)$-st or δ-th predecessor of x' on the connected path. By assumption $\delta \geq 6$ the probability for such a mutation is $O(n^{-5})$. It follows that $\mathrm{Prob}(X_i = 1) = 1/2 + O(n^{-5})$ and $\mathrm{Prob}(X_i = -1) = 1/2 - O(n^{-5})$, yielding $\mathrm{E}(X) = S \cdot O(n^{-5}) = o(1)$.

Adding up the bound $3X \leq 3\alpha + o(1)$ for the progress by ± 3-steps and $n^{3/4}/2$ for all other steps, recalling $\varepsilon \leq 1/4$, w. o. p.

$$\begin{aligned}\Delta_{\text{ls}}^{\text{uncon}} &\leq n^{3/4}/2 + 3\alpha + o(1)\\ &\leq n^{3/4}/2 + 3S^{1/2+\varepsilon} + n^{1/2+\varepsilon}/2 + o(1)\\ &\leq n^{3/4} + 3(2T/n^2)^{1/2+\varepsilon} + o(1)\\ &\leq n^{3/4} + 6(T/n^2)^{1/2+\varepsilon}.\end{aligned}$$

□

Recall that in a generation with local search, if the offspring is well-formed, then the connected path is climbed for $\delta - 1$ or δ iterations. On the unconnected path, it is essential whether mutation jumps up or down the path. For the connected path, this issue is not that important as local search clearly dominates the progress on the connected path if δ is not very small.

Lemma 4.1.13. *Let $\Delta_{\text{ls}}^{\text{con}}$ be the progress on the connected path in $T = n^{\Theta(1)}$ generations with local search and $\delta \geq 1$, then w. o. p.*

$$(1-\varepsilon)\cdot 2T/e\cdot(\delta-2) - n^{3/4} \;<\; \Delta_{\text{ls}}^{\text{con}} \;<\; (1+\varepsilon)\cdot 2T/e\cdot(\delta+2) + n^{3/4}.$$

Proof. A generation only yields non-zero progress if the outcome of local search is accepted. We call such a generation an accepting generation. A sufficient condition for an accepting generation is to clone the parent or to flip a single bit, in which case local search creates a well-formed search point with larger or equal fitness. The probability for such an event is $(1-1/n)^n + (1-1/n)^{n-1} = 2/e - o(1)$. On the other hand, a generation is accepting only if mutation creates a point with Hamming distance at most 1 to a well-formed search point. As there are only $O(n)$ points with a fixed Hamming distance $i \geq 2$ from the parent that may lead to acceptance, the probability of an accepting generation is bounded from above by $2/e + o(1)$.

Let A be the random number of accepting generations among the first T ones, then by Chernoff bounds

$$\text{Prob}(A \leq (1-\varepsilon)\cdot 2T/e) \;=\; \exp(-\Omega(T))$$

and

$$\text{Prob}(A \geq (1+\varepsilon)\cdot 2T/e) \;=\; \exp(-\Omega(T)).$$

In an accepting generation, a mutation creating an offspring y out of x with $\text{H}(x', y') \geq 2$ is called a large step. In a large step, assuming that less than k bits flip, the progress on the connected path is at least $-(k-1) + (\delta - 1) \geq -k + \delta$ and at most $k + \delta$. The probability of a large step is $O(1/n)$ as it is dominated by the probability of creating

one out of $O(n)$ points with Hamming distance 2 to x' that lead to acceptance. Let L denote the number of large steps, then by Chernoff bounds

$$\text{Prob}\Big(L \geq n^{1/4} + A/n^{1/2}\Big) \;\leq\; (n^{-1/2})^{-\Omega(n^{1/4})},$$

which implies $Lk \leq n^{3/4} + A$ w. o. p.

In all other accepting generations, if P_s is the parent, local search creates one of the search points $\{\mathrm{P}_s, \mathrm{P}_{s+1}, \mathrm{P}_{s+2}\}$ in the first iteration. Thus, the progress in such a step is bounded below by $\delta - 1$ and above by $\delta + 1$. Together, w. o. p. the total progress is bounded above by

$$L \cdot (k + \delta) + (A - L) \cdot (\delta + 1) \;\leq\; A(\delta + 1) + Lk \;\leq\; A(\delta + 2) + n^{3/4}$$

and below by

$$L \cdot (-k + \delta) + (A - L) \cdot (\delta - 1) \;\geq\; A(\delta - 1) - Lk \;\geq\; A(\delta - 2) - n^{3/4}.$$

Along with the bounds on A, this proves the claim. □

Using the progress bounds, we are now able to prove Theorem 4.1.9.

Proof of Theorem 4.1.9. We first give precise values for the lengths $\ell, r \in \mathbb{R}$ of the two paths. Note that ℓ and r need not be integral as race functions allow well-formed search points x with $p(x') > \ell$ or $p(x'') > r$. Let $\varepsilon > 0$ be a constant small enough w. r. t. constraints that will arise in the following. Let

$$\ell := \frac{1 - \varepsilon}{e} \cdot \left(n^3 + \frac{2n^4}{\tau} \cdot (\delta - 3)\right) - n^{3/4}$$

and

$$r := n^5 + \frac{3(1 + \sqrt{2})}{2e} \cdot n.$$

First, we investigate a period of n^4 generations of the (1+1) MA with local search frequency $1/\tau$ and show that typically the end of the connected path is reached within the period, while the end of the unconnected path is not. The number of generations with local search is about n^4/τ. To improve readability, we ignore rounding issues and assume that n^4/τ is the exact value. As $\lceil n^4/\tau \rceil - \lfloor n^4/\tau \rfloor = O(1/n) \cdot n^4/\tau$ (following from $\tau = O(n^3)$), the error is negligible.

The end of the connected path is reached within n^4 generations if the progress in n^4 generations is at least ℓ. On the other hand, the end of the unconnected path is not reached within the period if for *every* $T \leq n^4$ the progress in T steps is less than $r - n^5$. The probability of reaching the end of the unconnected path is maximized (apart from rounding issues with T/τ) if T attains the maximal value n^4. By the union bound,

multiplying the probability in case $T = n^4$ with a factor of n^4 yields an upper bound for the probability of reaching the end at any point of time in the period.

Now, all that remains is to bound the progress in n^4 generations. The following statements hold w. o. p. Let Δ^{con} be the total progress on the left path and Δ^{uncon} be the total progress on the right path in n^4 generations. Applying Lemma 4.1.11 for $n^4 - n^4/\tau$ generations without local search and Lemma 4.1.13 for n^4/τ generations with local search yields

$$\begin{aligned}\Delta^{\text{con}} &\geq (1-\varepsilon) \cdot \frac{n^4 - n^4/\tau}{en} + (1-\varepsilon) \cdot \frac{2n^4}{e\tau} \cdot (\delta - 2) - n^{3/4} \\ &= \frac{1-\varepsilon}{e} \cdot \left(n^3 - \frac{n^3}{\tau} + \frac{2n^4}{\tau} \cdot (\delta - 2) \right) - n^{3/4} \\ &\geq \frac{1-\varepsilon}{e} \cdot \left(n^3 + \frac{2n^4}{\tau} \cdot (\delta - 3) \right) - n^{3/4} \;=\; \ell.\end{aligned}$$

For the unconnected path we show that after n^4 generations we are still by a distance of at least k away from the end, that is, $n^5 + \Delta^{\text{uncon}} \leq r - k$. The fact that the (1+1) MA starts with P_{n^5} on the unconnected path enables us to apply Lemma 4.1.12 as n^4 steps can only decrease the position by $n^4 \cdot k$ implying that all considered parents differ from P_0 on the unconnected path. By Lemmas 4.1.11 and 4.1.12

$$\begin{aligned}\Delta^{\text{uncon}} &\leq (1+\varepsilon) \cdot \frac{3(n^4 - n^4/\tau)}{en^3} + 6(n^2/\tau)^{1/2+\varepsilon} + n^{3/4} \\ &\leq \frac{3(1+\varepsilon)}{e} \cdot n + 6n^{1+2\varepsilon} \cdot \tau^{-1/2-\varepsilon} + n^{3/4}.\end{aligned}$$

Recall $\tau = n^{\Omega(1)}$ and choose ε small enough to make $\tau \geq n^{4\varepsilon}$. Then $n^{1+2\varepsilon} \cdot \tau^{-1/2-2\varepsilon} \leq n^{1-8\varepsilon^2} = o(n)$ implying

$$n^5 + \Delta^{\text{uncon}} \;\leq\; n^5 + \frac{3(1+\varepsilon)}{e} \cdot n + o(n) \;\leq\; n^5 + \frac{3(1+\sqrt{2})}{2e} \cdot n - k \;=\; r - k$$

if ε is small and n large enough.

Together, the (1+1) MA with local search frequency $1/\tau$ reaches the end of the connected path within n^4 generations. This implies that on Race^{con}, a global optimum is found in n^4 generations, w. o. p. On $\text{Race}^{\text{uncon}}$, however, the objective is now turned to minimizing the position on the unconnected path. Since $n^5 + \Delta^{\text{uncon}} \leq r - k$ the Hamming distance to each point $x' \, \mathrm{P}_{r+i}$ for $i \geq 0$ is at least k and all points with smaller Hamming distance have worse fitness. The only way to reach a global optimum is a direct jump flipping at least k bits. The probability for such an event is at most $1/(k!) = 2^{-\Omega(n^{1/2} \log n)}$. Moreover, the probability of finding the optimum within $2^{-cn^{1/2} \log n}$ generations is still exponentially small if $c > 0$ is small enough, meaning that the (1+1) MA needs exponential time w. o. p.

The argumentation for the (1+1) MA with local search frequency $1/(2\tau)$ is similar. We now consider a period of $\sqrt{2}n^4$ generations of the (1+1) MA with local search frequency $1/(2\tau)$ and redefine Δ^{con} and Δ^{uncon} according to this new period. The number of generations with local search is about $\sqrt{2}/(2\tau) = 1/(\sqrt{2}\tau)$. Again, we ignore rounding issues to improve readability. Compared to the previous setting, the number of generations with local search decreases by a factor of $\sqrt{2}$. On the other hand, the number of generations without local search increases by approximately a factor of $\sqrt{2}$. This is a clear disadvantage for the connected path and an advantage for the unconnected path. We claim that here the unconnected path wins the race, i. e., $n^5 + \Delta^{\text{uncon}} \geq r$ and $\Delta^{\text{con}} \leq \ell - k$. Repeating the line of thought from above, the (1+1) MA needs polynomial time on $\text{Race}^{\text{uncon}}$ and exponential time on Race^{con} w. o. p.

First, we prove $n^5 + \Delta^{\text{uncon}} \geq r$ using Lemmas 4.1.11 and 4.1.12. We have

$$\begin{aligned}
\Delta^{\text{uncon}} &\geq (1-\varepsilon) \cdot \frac{3(\sqrt{2}n^4 - \sqrt{2}n^4/(2\tau))}{en^3} - 6(n^2/(\sqrt{2}\tau))^{1/2+\varepsilon} - n^{3/4} \\
&= \frac{3(1-\varepsilon)}{e} \cdot (\sqrt{2}n - \sqrt{2}n/(2\tau)) - 6(n^2/(\sqrt{2}\tau))^{1/2+\varepsilon} - n^{3/4} \\
&= \frac{3(1-\varepsilon)}{e} \cdot \sqrt{2}n - o(n),
\end{aligned}$$

the last equality following from the same arguments as above, along with $n/\tau = o(n)$. If ε is small enough such that $3(1-\varepsilon)\sqrt{2} > 3(1+\sqrt{2})/2$ and n is large enough then $n^5 + \Delta^{\text{uncon}} \geq r$ follows.

Finally, we show $\Delta^{\text{con}} \leq \ell - k$ using Lemmas 4.1.11 and 4.1.13.

$$\begin{aligned}
\Delta^{\text{con}} &\leq (1+\varepsilon) \cdot \frac{\sqrt{2}n^4 - \sqrt{2}n^4/(2\tau)}{en} + (1+\varepsilon) \cdot \frac{2\sqrt{2}n^4}{2e\tau} \cdot (\delta+2) + n^{3/4} \\
&= \frac{\sqrt{2}(1+\varepsilon)}{e} \cdot \left(n^3 - n^3/(2\tau) + \frac{n^4}{\tau} \cdot (\delta+2) \right) + n^{3/4} \\
&\leq \frac{\sqrt{2}(1+\varepsilon)}{e} \cdot \left(n^3 + \frac{n^4}{\tau} \cdot (\delta+2) \right) + n^{3/4}
\end{aligned}$$

If ε is small enough such that $(1+\varepsilon)\cdot\sqrt{2} < (1-\varepsilon)\cdot 3/2$ and n is large enough to dominate small order terms,

$$\begin{aligned}
\Delta^{\text{con}} &\leq \frac{1-\varepsilon}{e} \cdot \left(\frac{3}{2} \cdot n^3 + \frac{3}{2} \cdot \frac{(\delta+2)n^4}{\tau} \right) - n^{3/4} - k \\
&= \frac{1-\varepsilon}{e} \cdot \left(n^3 + \frac{\tau}{2n} \cdot \frac{n^4}{\tau} + \frac{3}{2} \cdot \frac{(\delta+2)n^4}{\tau} \right) - n^{3/4} - k
\end{aligned}$$

By hypothesis, $\delta/\tau \geq 2/n$ and $\delta \geq 36$ implying $\tau \leq \delta/2 \cdot n = (\delta - \delta/2)n \leq (\delta - 18)n$. Plugging this into the above equality yields

$$\begin{aligned}\Delta^{\text{con}} &\leq \frac{1-\varepsilon}{e} \cdot \left(n^3 + \frac{\delta - 18}{2} \cdot \frac{n^4}{\tau} + \frac{3}{2} \cdot \frac{(\delta+2)n^4}{\tau} \right) - n^{3/4} - k \\ &= \frac{1-\varepsilon}{e} \cdot \left(n^3 + \frac{2n^4}{\tau} \cdot (\delta - 3) \right) - n^{3/4} - k \;=\; \ell - k.\end{aligned}$$

□

Finally, let us discuss some possible extensions. Theorem 4.1.9 can be extended to a memetic algorithm calling local search probabilistically with probability $1/\tau$ as the number of local search calls in cn^4 generations is concentrated around cn^4/τ. Applying Chernoff bounds to estimate the number of local search calls introduces additional factors $(1-\varepsilon)$ and $(1+\varepsilon)$ into the preceding proof and these factors can easily be dealt with.

On the other hand, extending the result to offspring populations of size $\lambda > 1$ is not that easy. Recall that for $\lambda = 1$ the progress in generations with local search on the unconnected path is almost a martingale as typically jumps to the next and the previous peak are treated equally by selection. Now, if $\lambda > 1$ there is at least a constant probability that the current peak is cloned and then (assuming equal positions on the connected path) the clone is preferred over the previous peak. As a consequence, the martingale property is lost and the influence of local search on the unconnected path grows, changing the behavior of the algorithm significantly. If $\tau = o(n)$, the progress on the unconnected path might even be dominated by this effect. We conjecture that if the local search frequency is fixed to $1/\tau$ and $\tau = o(n)$, then the connected path wins the race if $\lambda = 1$, but the unconnected path wins in case $\lambda = 2$, say. Such a behavior would yield another example where the choice of the offspring population size is essential, as presented by Jansen, De Jong, and Wegener (2005).

4.1.5 Conclusions

We presented a rigorous theoretical analysis of a simple memetic evolutionary algorithm, the (μ+λ) MA. Much research effort has been spent in recent years to parametrize memetic algorithms and to find a proper balance between local search and evolutionary search. We have stressed the importance of this subject by presenting function classes where both the choice of the local search depth and the choice of the local search frequency have a tremendous impact on the optimization time of the (μ+λ) MA. For almost every reasonable parameter setting, we have constructed a function for which these parameters lead to an efficient optimization while most other parameter settings are very inefficient. More precisely, for the local search depth even a tiny additive term of $\log^3 n$ can turn a polynomial optimization time into a superpolynomial one and vice

versa. Regarding the local search frequency, variations by a small factor of 2 can even decide between polynomial and exponential optimization times.

The analyses presented here are the first rigorous runtime analyses of memetic evolutionary algorithms. Such analyses are challenging since these metaheuristics are typically not designed to support an analysis. This section shows that a rigorous analysis is possible and worthwhile as we have gained valuable insights into the dynamic behavior of memetic algorithms and into the interplay of genetic operators with local search. In addition, we have extended Witt's analysis of population dynamics for the $(\mu+1)$ EA to the $(\mu+\lambda)$ EA with larger offspring populations, which is of independent interest.

4.2 Memetic Algorithms with Variable-Depth Search

In the previous section we have analyzed memetic algorithms on artificial functions. In this section, we want to concentrate on combinatorial problems instead. Thereby, we also investigate broader classes of memetic algorithms. One aspect is the design of the two operators, mutation and local search. A local search procedure searching only for Hamming neighbors with larger fitness behaves similarly to a sequence of mutations with mutation probability $1/n$. The previous section has shown that the difference is large enough to allow the construction of an artificial function where this difference is essential, but in a more practical combinatorial setting these two operators might still behave similarly. When looking for a hybrid algorithm that is more powerful than its single components, such an algorithm is probably not the best choice.

Therefore, in this section we investigate memetic algorithms where the two operators use very different strategies. On one hand, mutation may use significantly larger mutation probabilities. This accounts for the fact that in iterated local search algorithms the perturbation move usually makes quite large jumps, i. e., a large mutation. On the other hand, we consider a different local search procedure that does not use a strict elitist strategy, but instead is able to traverse search points with inferior fitness. We will see in the following that the latter extension is able to perform well in simple combinatorial settings where many other common search heuristics fail dramatically.

Regarding problems from combinatorial optimization, a plethora of randomized search heuristics has been proposed, analyzed, and applied over the past decades. Besides evolutionary and memetic algorithms, simulated annealing belongs to the best known heuristics. Simulated annealing is a simple hill climber that may, however, accept worse solutions with a probability monotone in the current *temperature*. The temperature is typically decreased over time, which gradually changes the focus from exploration to exploitation.

All these heuristics have to deal with the possibility of reaching a poor local optimum. Population-based heuristics usually rely on diversifying the search to explore different local optima. However, it may happen that the whole population converges to non-global

local optima and then the situation is not too different from *trajectory-based algorithms*, that is, algorithms maintaining only a single current solution. These algorithms have to rely on different mechanisms to overcome local optima.

We describe some common approaches to overcome local optima and give pointers to related theoretical works, with a focus on combinatorial optimization.

1. **Accept solutions with inferior fitness:**
   Simulated annealing may accept solutions with inferior fitness, enabling the algorithm to climb down a hill. This well-known strategy proved to be effective for Graph Bisection (Jerrum and Sorkin, 1998), the two-dimensional Ising model (Fischer, 2004), and minimum spanning trees (Wegener, 2005b).

2. **Decrease the attractiveness of local optima:**
   Tabu search maintains a *tabu list* of solutions that are "taboo" for the algorithm, hence making local optima less attractive. Diversity mechanisms like fitness sharing also decrease the attractiveness of local optima. If many individuals are concentrated on one local optimum, they are forced to "share" their fitness according to their similarity. Hence, the algorithm is encouraged to decrease similarity in the population. The effectiveness of fitness sharing has been shown in Section 3.2.6 for the toy problem TwoMax that, however, has a similar global structure as the Mincut instance we will investigate in Section 4.2.3.

3. **Use larger or multiple neighborhoods:**
   Evolutionary algorithms like the (1+1) EA use a stochastic neighborhood where every search point has a positive probability of being created. This simple property enables the approximation of maximum matchings (Giel and Wegener, 2003) and balanced partitions (Witt, 2005). In addition, evolutionary algorithms may use crossover to recombine different local optima. The usefulness of crossover was first shown for toy problems (Jansen and Wegener, 2005). Moreover, crossover together with fitness sharing is effective for (a problem equivalent to) 2-coloring binary trees (Sudholt, 2005). The idea of using different neighborhoods is dominant in multimeme algorithms (Neri et al., 2007) and variable neighborhood search algorithms (see, e. g., Mladenović and Hansen, 1997) exploiting that a local optimum w. r. t. one neighborhood need not be a local optimum w. r. t. another one. Memetic algorithms like iterated local search also fall into this category as they combine mutation and local search. Their usefulness so far has only been proven for artificial problems (see Section 4.1).

Another common strategy is to restart the algorithm after convergence to local optima. This can be seen as a very large perturbation and hence falls into the third category.

In this section we consider a special local search operator for use within the framework of iterated local search, so-called *variable depth search* (VDS). Variable-depth search is

well-known for the TSP as *Lin-Kernighan* strategy (Lin and Kernighan, 1973) and for GRAPH BISECTION as *Kernighan-Lin* (Kernighan and Lin, 1970). The idea is to perform a sequence of local moves. The next local move is chosen in a greedy fashion. If there is a local move that increases fitness, then a move with maximal fitness gain is chosen. Otherwise, a move with minimal loss in fitness is selected. To prevent the algorithm from looping, certain parts of the search space are made "tabu". For binary search spaces this means that if VDS flips some bit, then this bit cannot be flipped again during the run of VDS. The output of VDS is then a best solution encountered during the sequence of local moves.

Iterated local search using VDS combines approaches from all three mentioned categories to overcome local optima. Firstly, it easily traverses solutions with inferior fitness if no fitness-improving move is available. Secondly, starting with a local optimum this local optimum is made "tabu" like in tabu search since steps moving back towards the local optimum are not allowed. A difference to classical tabu search is that we do not keep a tabu list of single individuals, but render large parts of the search space tabu. Finally, we employ different neighborhoods: VDS and mutation as perturbation.

This combination of strategies makes it easy for VDS to overcome local optima. Another remarkable aspect is the greedy component in VDS since we always choose a best move among the feasible local moves. We will in the following give examples where this greedy behavior provides a good guidance for the algorithm in order to find a global optimum.

We investigate instances of problems from combinatorial optimization, namely MINCUT, KNAPSACK, and MAXSAT. Definitions and descriptions of these problems are postponed to the following sections. The chosen instances contain non-global local optima with large basins of attraction that are hard to overcome. We will see that memetic algorithms with VDS are efficient on these functions while common trajectory-based algorithms like the (1+1) EA, traditional iterated local search algorithms, and simulated annealing fail to find a global optimum, even if they are given exponential time.

The instances we consider have a very simple structure. This helps to keep the argumentation simple and to focus on the essentials. We assume that the reader is familiar with basic knowledge on common combinatorial optimization problems. For details, we refer to appropriate text books (e. g., Cormen et al., 2001; Papadimitriou, 1994; Wegener, 2005a).

The remainder of this section is structured as follows. First, we define all investigated algorithms in Section 4.2.1. Section 4.2.2 contains lower bounds on the runtime after the whole population has reached or gotten close to local optima. In Sections 4.2.3, 4.2.4, and 4.2.5 we then deal with instances for the problems MINCUT, KNAPSACK, and MAXSAT, respectively. We conclude in Section 4.2.6.

4.2.1 Algorithms

We first define two local search operators used throughout this section. Both operators can use multiple neighborhoods, but we restrict ourselves to neighborhoods defined as follows.

Definition 4.2.1. Let the radius of a neighborhood N be the maximum distance from any point to any of its neighbors, i. e., $\max_{x,y}\{\mathrm{H}(x,y) \mid y \in N(x)\}$. Then N is called a *proper neighborhood* if it only depends on the Hamming distance, formally

$$\forall x, y_1, y_2 \colon \ \mathrm{H}(x,y_1) = \mathrm{H}(x,y_2) \ \Rightarrow \ (y_1 \in N(x) \Leftrightarrow y_2 \in N(x)),$$

its radius is $O(1)$, and $x \notin N(x)$ for all x.

The constant radius implies that the size $|N|$ of the neighborhood is bounded by a polynomial. Note that the Hamming neighborhood is a proper neighborhood. In the following, we denote by N_k the neighborhood of all points with Hamming distance exactly k. We reuse notions for a neighborhood N extended to a set $X \subseteq \{0,1\}^n$, i. e., $N(X) := \bigcup_{x \in X} N(x)$ and $N^*(X) := N(X) \cup X$.

The first local search operator we define repeatedly chooses neighbors with strictly larger fitness. In contrast to the operator from Section 4.1 where the choice of the pivot rule was left open, we here specify the pivot rule to be random. In order to distinguish the operator from the algorithm called randomized local search, we refrain from the name random local search, but instead call it *standard local search*. Standard local search accepts any neighbor with strictly larger fitness and stops whenever a local optimum is reached or the number of iterations reaches the local search depth $\delta(n)$.

Operator 12 Standard local search(y)

for $\delta(n)$ iterations **do**
  Choose $z \in N(y)$ with $f(z) > f(y)$ uniformly at random
    or stop and **return** y if no such z exists.
  Let $y := z$.
return y.

We already explained the concept of VDS. In the following procedure S denotes the sequence of solutions encountered during VDS and L is a set of indices for all bits that have been locked.

Operator 13 Variable-depth search(y)

$S, L := \emptyset$.
while $V_y := \{z \in N(y) \mid \forall i\colon (y_i \neq z_i \Rightarrow i \notin L)\} \neq \emptyset$ **do**
  Choose $z \in V_y$ with maximal f-value uniformly at random. // *choose neighbor*
  $S := S \cup \{z\}$. // *add to sequence*
  $L := L \cup \{i \mid y_i \neq z_i\}$. // *lock flipped bits*
  $y := z$.
return $z \in S$ with maximal f-value chosen uniformly at random.

Note that, given a proper neighborhood, one run of VDS takes at most n iterations since in every iteration of the loop at least one index is added to L.

We now take over the definition of the (1+1) Memetic Algorithm, shortly (1+1) MA, from Section 4.1, but we redefine it as a generic framework for various algorithms. One generalization is that any local search procedure may be used. Moreover, we may have an arbitrary criterion that decides whether local search is to be used in the current generation. Finally, the mutation probability p_m is not fixed to $1/n$, but instead it is a parameter of the algorithm now.

Algorithm 14 (1+1) Memetic Algorithm

Choose x uniformly at random.
repeat
  Create y by flipping each bit in x independently with prob. p_m. // *mutation*
  if criterion for using local search **then** $y :=$ local search(y). // *local search*
  if $f(y) \geq f(x)$ **then** $x := y$. // *selection*

The (1+1) MA never using local search equals the (1+1) EA with mutation probability p_m. Alternatively, local search may be applied periodically as in Section 4.1. We may also choose to apply local search probabilistically with a fixed probability as done in Ishibuchi et al. (2003). The algorithm where local search is called in each generation equals *iterated local search* (Lourenço et al., 2002). In particular, we will often refer to *iterated VDS* as the (1+1) MA calling VDS in every generation.

All algorithms considered so far have in common that the best-so-far fitness cannot decrease. Simulated annealing always accepts better solutions, but it also allows worse solutions to be accepted. This decision is made dependent on the size of the fitness decrease and a parameter called *temperature*. If the temperature equals 0, simulated annealing behaves like a hill climber, i. e., it does not accept worsenings. The larger the temperature, the more likely it is to accept worse solutions. It is common practice to start with a high temperature and then to decrease the temperature over time. This way, simulated annealing can explore the search space in the beginning and then grad-

ually turns into a hill climber focusing on exploitation. A strategy to turn down the temperature is called a *cooling schedule*. Unless otherwise noted, simulated annealing uses the Hamming neighborhood $N = N_1$. It is formulated for maximization to match previously defined algorithms.

Algorithm 15 Simulated Annealing

Let $t := 0$.
Choose x uniformly at random.
repeat
  Choose $y \in N(x)$ uniformly at random.
  Set $x := y$ with probability $\min\{1, \exp((f(y) - f(x))/T(t))\}$.
  $t := t + 1$.

Simulated annealing with a fixed temperature is called *Metropolis algorithm*. It was long unknown whether cooling down the temperature is essential for natural problems, i. e., whether simulated annealing outperforms the Metropolis algorithm with an optimal temperature. This question was recently solved in the affirmative by Wegener (2005b) for the natural problem of computing a minimum spanning tree.

For the efficiency of an algorithm a plausible performance measure is the number of generations until a global optimum is found. We will also consider the number of function evaluations, referred to as *optimization time*. Thereby, we in particular account for the computational effort of local search. Note that the number of function evaluations in a generation with local search is bounded above by $\delta(n) \cdot |N|$ for greedy local search and by $n \cdot |N|$ for VDS.

4.2.2 Lower Bounds when Stuck in Local Optima

We start our investigations with lower bounds for different algorithms after having reached (or gotten close to) a local optimum. Combinatorial fitness landscapes often contain several local optima that are close to one another. Therefore, it makes sense to group such local optima into sets and to consider the Hamming distance to such a set. A set of local optima is difficult for an algorithm if it has a large basin of attraction. This is especially true if the fitness decreases with every local move leading away from the local optimum. If this holds up to a Hamming distance of α, the set is called α-*difficult*.

Definition 4.2.2. A non-empty set $S^* \subseteq \{0,1\}^n$ is called α-*difficult* for $\alpha = \alpha(n)$ w. r. t. the function f and a neighborhood N if $y \in N(x)$ and $\mathrm{H}(x, S^*) < \mathrm{H}(y, S^*) \leq \alpha$ implies $f(x) > f(y)$ for every $x, y \in \{0,1\}^n$ and S^* does not contain global optima.

The definition of α-difficulty for a neighborhood $N \supseteq N_1$ implies that all search points with Hamming distance less than α to S^* have worse fitness than every point in S^*. This immediately leads to lower bounds for the (1+1) MA once S^* has been reached.

Lemma 4.2.3. *Let S^* be α-difficult for some neighborhood that includes N_1. If the* (1+1) MA *using standard local search with a proper neighborhood N and mutation probability $p_m \leq (1-\varepsilon) \cdot \alpha/n$ for some $\varepsilon > 0$ reaches S^*, the remaining time until a global optimum is found is at least $2^{c\alpha}$ with probability $1 - 2^{-\Omega(\alpha)}$ for some constant $c > 0$.*

Proof. It is safe to assume that α grows with n since otherwise the theorem is trivial. Let r be the radius of N. Apart from individuals in S^*, the (1+1) MA only accepts an offspring if mutation creates a solution with Hamming distance at least $\alpha - r$ to its parent as otherwise local search runs back into S^* or the generation ends with an inferior solution. With mutation probability at most $(1-\varepsilon)\cdot\alpha/n$ the expected number of flipping bits in one mutation is at most $(1-\varepsilon)\cdot\alpha$. The probability that at least $\alpha - r$ bits flip is at most $2^{-\Omega(\alpha)}$ by Chernoff bounds. The probability that this happens at least once in $2^{c\alpha}$ steps, $c > 0$ a small enough constant, is still of order $2^{-\Omega(\alpha)}$. □

Simulated annealing can accept worse solutions with a certain probability that depends on the loss in fitness and the current temperature $T = T(t)$. Escaping a single local optimum is easy if the temperature is high enough such that all local moves have a good chance to be accepted. The reason is simple: if the current solution is close to the local optimum, there are more local moves leading away from it than moving closer to it. If the temperature is too low (or has been cooled down too fast), escaping a local optimum is much more difficult. We can make this precise for a scenario where the temperature leaves us with a noticeable bias towards search points with high fitness.

Lemma 4.2.4. *Let S^* be α-difficult for some neighborhood that contains N_1 and let $|f(x) - f(y)| \geq \Delta = \Delta(\alpha)$ if $y \in N_1(x)$ and x, y have Hamming distance at most α to S^*. If simulated annealing with temperature $T(t) \leq \Delta/(\ln(4n/\alpha))$ reaches a search point with Hamming distance at most $(1-\varepsilon)\alpha$ to S^* for some constant $0 < \varepsilon \leq 1$, the remaining optimization time is at least $2^{c\alpha}$ with probability $1 - 2^{-\Omega(\alpha)}$ for some constant $c > 0$.*

Proof. W. l. o. g. we can assume $\varepsilon \leq 1/2$. Observe $\mathrm{H}(y, x^*) \geq \mathrm{H}(y, S^*)$ for every y and every $x^* \in S^*$. This allows us to focus on a single search point $x^* \in S^*$ with minimal Hamming distance to the current search point x. If $\mathrm{H}(x, S^*) = k$ and $(1-\varepsilon)\alpha \leq k < \alpha$, the probability of increasing the Hamming distance to x^* (and hence S^*) by 1 is

$$p^+ \;\leq\; \frac{n-k}{n} \cdot e^{-\Delta/T} \;<\; e^{-\Delta/T} \;\leq\; \frac{\alpha}{4n}$$

due to the assumption on T. On the other hand, the probability of decreasing the Hamming distance to S^* by 1 is

$$p^- \;\geq\; \frac{k}{n} \;\geq\; \frac{(1-\varepsilon)\alpha}{n} \;\geq\; \frac{\alpha}{2n}$$

as all k steps moving closer to x^* are accepted. Hence, conditional on an exchange of the current search point, the probability of moving away from S^* is at most $1/3$ and the probability of getting closer is at least $2/3$.

The lower bound now follows from the Drift Theorem 2.3.10 applied to random variables $\alpha - \mathrm{H}(x, S^*)$ (the process stopping at value 0) with $a := 0$ and $b := \varepsilon\alpha$. As in previous applications of the Drift Theorem, we consider relevant steps, i. e., steps where the current distance is changed. When considering a search point with Hamming distance at most $(1-\varepsilon)\alpha$ to S^*, we start the drift analysis with $X_0 \geq b$. If $a < X_t < b$, we have $\mathrm{E}(X_{t+1} - X_t) \geq 1/3$ fulfilling the first condition of Theorem 2.3.10. The second condition holds trivially as only steps of length ± 1 are allowed. Hence Theorem 2.3.10 proves the claim. □

An α-difficult local optimum is challenging for memetic algorithms and simulated annealing as both have difficulties with large "valleys" in the fitness landscape. This similarity between evolutionary algorithms and simulated annealing has already been recognized by Jansen and Wegener (2007).

4.2.3 Mincut

Given an undirected graph $G = (V, E)$, the problem MINCUT is to partition all vertices into two non-empty subsets V_0, V_1 such that the number of edges between V_0 and V_1 is minimized. Such edges are called *cut edges*. We remark that specialized algorithms can solve the MINCUT problem in polynomial time, even in the case of weighted graphs (Stoer and Wagner, 1994).

Given an ordering of the vertices $V = \{v_1, \ldots, v_n\}$, we obtain a binary representation $x = x_1 \ldots x_n \in \{0,1\}^n$ for $n = |V|$ such that $v_i \in V_{x_i}$ for every i. If V_0 and V_1 are non-empty, the fitness function is encoded as follows. We choose the fitness as the number of non-cut edges, written as $\sum_{\{u,v\}\in E}(x_u x_v + (1-x_u)(1-x_v))$. However, if V_0 or V_1 is empty, i. e., $x \in \{0^n, 1^n\}$, the non-emptiness constraint is violated and we penalize such a solution by assigning a negative f-value.

$$f(x) := \begin{cases} \sum\limits_{\{u,v\}\in E} (x_u x_v + (1-x_u)(1-x_v)) & \text{if } x \notin \{0^n, 1^n\}, \\ -1 & \text{if } x \in \{0^n, 1^n\}. \end{cases}$$

Consider the following instance $G = (V, E)$ that consists of two cliques of size $n/2$, each (see Figure 4.2).

$$V = \{u_1, \ldots, u_{n/2}, v_1, \ldots, v_{n/2}\},$$
$$E = \{\{u_i, u_j\}, \{v_i, v_j\} \mid 1 \leq i < j \leq n/2\}.$$

Consider a partition $V = V_0 \cup V_1$ where w. l. o. g. $u_1 \in V_0$. Obviously, the optimal partition is $V_0 = \{u_1, \ldots, u_{n/2}\}$ with a cut of size 0. We claim that all partitions with

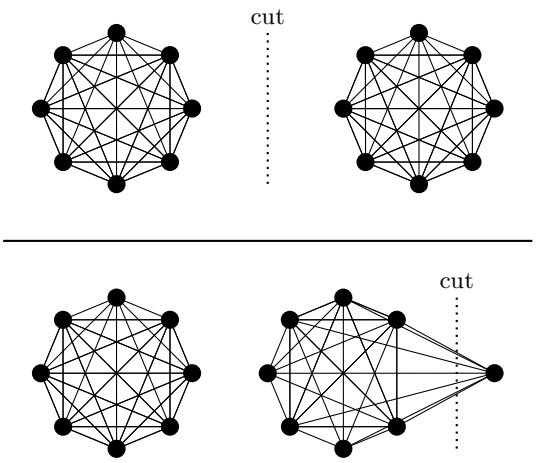

Figure 4.2: A global optimum (top) and a local optimum (bottom) for the MINCUT instance with $n = 16$.

$|V_0| = 1$ or $|V_0| = n - 1$ are difficult local optima with a cut size of $n/2$ as V_0 and V_1 are constrained to be non-empty. A sketch of a global and a local optimum is given in Figure 4.2.

Lemma 4.2.5. *The set S^* of partitions with $|V_0| = 1$ or $|V_0| = n - 1$ is α-difficult for $\alpha = \lfloor n/4 - 1 \rfloor$ w. r. t. N_1.*

Proof. Obviously S^* does not contain global optima. We need to show that for every x with $\mathrm{H}(x, S^*) \leq \alpha - 1$ and every $y \in N(x)$ with larger distance to S^*, $f(y) < f(x)$ holds. For $x \in S^*$ we have that every Hamming neighbor y is either infeasible or one side of the cut contains exactly 2 vertices. In both cases $f(y) < f(x)$ as demanded. For infeasible x every local move results in a solution in S^*, hence the condition on all y is trivial. If x is neither in S^* nor infeasible and $\mathrm{H}(x, S^*) = d \leq \alpha - 1$, then the smaller set of the partition contains $d + 1$ vertices. This implies that this side of the cut contains at most $d + 1 \leq n/4 - 1$ u-vertices and at most $n/4 - 1$ v-vertices. Every local move leading away from S^* has to add another u-vertex or another v-vertex to this side, which increases the cut size and hence decreases the fitness. □

During a typical run of the (1+1) EA or the (1+1) MA, it is equally likely for both cliques to move towards different sides of the cut or move towards the same side. We prove that the latter case leads to exponential optimization times and that this happens with probability close to 1/2.

Theorem 4.2.6. *Consider the* (1+1) MA *with standard local search using neighborhood N_1 and mutation probability $p_m \leq 1/5$. The optimization time for the* MINCUT *instance is at least 2^{cn} with probability $1/2 - O(1/\sqrt{n})$ for some constant $c > 0$.*

Proof. We already proved that all local optima are α-difficult for $\alpha = \lfloor n/4 - 1 \rfloor$. If n is large enough, we have $1/5 \leq (1-\varepsilon) \cdot \alpha/n$ for an appropriate $\varepsilon > 0$, hence the claim follows from Lemma 4.2.3 if we can prove that the algorithm reaches S^* with probability $1/2 - O(1/\sqrt{n})$.

Let S_0 contain all four (feasible and infeasible) solutions with no cut edge. Let $S_1 = N_1(S_0)$. We now argue that with high probability the algorithm evaluates a solution in S_1 before evaluating one from S_0. Consider the first generation where a solution in $S_0 \cup S_1$ is created. Let p_0 and p_1 denote the probabilities that this solution belongs to S_0 and S_1, respectively. We claim that $p_1 = \Omega(p_0 \cdot \sqrt{n})$. This implies that S_1 is found before S_0 with probability $1 - O(1/\sqrt{n})$.

Initialization creates a search point in S_0 with probability 2^{-n+2}. Assume that the algorithm does not start in $S_0 \cup S_1$. Consider a generation with current solution $x \notin (S_0 \cup S_1)$ and the next offspring creation. If $S_0 \cup S_1$ is reached during local search, the claim is trivial. Hence we focus on mutation only and fix a search point $z \in S_0$. If x has Hamming distance k to z, we have k solutions in S_1 with Hamming distance $k-1$ to x and $n-k$ solutions in S_1 with Hamming distance $k+1$. The probability of reaching a specific $y \in N_1(z)$ differs from the probability $\text{Prob}(z)$ of reaching z in just one bit position. More precisely, the probabilities differ by factors $p_m/(1-p_m)$ or $(1-p_m)/p_m$, dependent on whether this bit has to be flipped or not. Let $\text{Prob}(N_1(z))$ denote the probability of reaching $N_1(z)$, then

$$\begin{aligned}
\text{Prob}(N_1(z)) &\geq k \cdot \text{Prob}(z) \cdot \frac{1-p_m}{p_m} + (n-k) \cdot \text{Prob}(z) \cdot \frac{p_m}{1-p_m} \\
&= \text{Prob}(z) \left(\frac{k(1-p_m)^2 + (n-k)p_m^2}{p_m(1-p_m)} \right) \\
&= \text{Prob}(z) \left(\frac{k(1-2p_m) + np_m^2}{p_m(1-p_m)} \right).
\end{aligned}$$

We see that this term is increasing with k, hence in the worst case $k = 2$. Along with $p_m \leq 1/5$, we arrive at the bound

$$\text{Prob}(N_1(z)) \geq \text{Prob}(z) \left(\frac{1 + np_m^2}{p_m} \right).$$

In case $p_m \leq 1/\sqrt{n}$ the term in brackets is at least $1/p_m \geq \sqrt{n}$. If $p_m > 1/\sqrt{n}$, then this term is at least $np_m \geq \sqrt{n}$ as well. We conclude

$$\text{Prob}(N_1(z)) \geq \text{Prob}(z) \cdot \sqrt{n}.$$

Since this holds for all $z \in S_0$, $p_1 \geq p_0 \cdot \sqrt{n}$ follows and S_1 is found before S_0 with probability $1 - O(1/\sqrt{n})$.

As long as no local optimum is found, the fitness is indifferent to the question whether the majority of the u-vertices is in V_0 or in V_1. The same holds independently for the

v-vertices. Hence, if the first cut in S_1 is created, given that S_0 has not been found yet, all cuts in S_1 have the same probability of being found. Half of these cuts are local optima, hence the probability that a local optimum is found equals $1/2$. By the union bound, the probability of reaching a local optimum is at least $1/2 - O(1/\sqrt{n}) - 2^{-n+2} = 1/2 - O(1/\sqrt{n})$. □

Reusing ideas from the proof of Theorem 4.2.6, we show that also simulated annealing fails with probability close to $1/2$.

Theorem 4.2.7. *Simulated annealing with an arbitrary cooling schedule where $T(t)$ is monotone decreasing needs at least 2^{cn} steps for the* MINCUT *instance with probability $1/2 - 2^{-\Omega(n)}$ for some constant $c > 0$.*

Proof. We divide a run into two phases: the first phase ends when the temperature first drops to $n/12$ and then the second phase starts. Let T_1 be the number of generations in Phase 1 and let S_0 and S_1 be defined as in the proof of Theorem 4.2.6. We first prove that in Phase 1 no solution in S_0 will be evaluated in exponential time, with high probability.

Consider a search point x with $\mathrm{H}(x, S_0) = k \leq n/(4e^6)$. The probability of increasing the Hamming distance to S_0 is

$$p^+ \;\geq\; \frac{n-k}{n} \cdot e^{-n/(2T)} \;\geq\; \frac{1}{2} \cdot e^{-6}$$

as the worst fitness decrease equals $n/2$. The probability of decreasing the Hamming distance to S_0 equals

$$p^- \;=\; \frac{k}{n} \;\leq\; \frac{1}{4} \cdot e^{-6}.$$

Together, the conditional probability of decreasing the Hamming distance is bounded by $1/3$, provided that the Hamming distance is changed.

The probability of initializing simulated annealing with a search point x such that $\mathrm{H}(x, S_0) \leq n/(4e^6)$ is $2^{-\Omega(n)}$. Assuming to start at a larger distance and applying the Drift Theorem 2.3.10 to $\mathrm{H}(x, S_0)$ in the interval $[0, n/(4e^6)]$, the probability that S_0 is reached within the first $\min\{T_1, 2^{cn}\}$ steps is $2^{-\Omega(n)}$.

This concludes the proof if $T_1 \geq 2^{cn}$. Otherwise, we consider Phase 2 and assume that S_0 has not been reached in Phase 1. By the locality of the search operator S_1 is reached before S_0 and the probability that a local optimum is found equals $1/2$.

Given a fixed Hamming distance $k \leq n/12$ from S_0, the minimal fitness difference Δ between two neighbors, as defined in Lemma 4.2.4, is attained when only one clique is cut. In that case the cut clique has k vertices on one side of the partition and when adding a $(k+1)$-st vertex, the fitness decreases by $n/2 - 2k \geq n/3$. Applying Lemma 4.2.4 with $\alpha = n/12$ and $\Delta(\alpha) = n/3$ proves the claim for $T \leq \Delta/(\ln(4n/\alpha)) = 1/(3\ln(48))$ and hence for $T \leq n/12$. □

We have seen that the local optima of the MINCUT instance are extremely hard for standard evolutionary algorithms, memetic algorithms, and simulated annealing. In contrast to this, iterated VDS with the standard mutation probability $p_m = 1/n$ easily escapes from this local optimum. The following proof is surprisingly simple.

Theorem 4.2.8. *The expected number of generations of iterated VDS with mutation probability $p_m = 1/n$ on the* MINCUT *instance is $O(n)$.*

Proof. The first VDS reaches a global or local optimum. Assume that a local optimum is reached. With probability at least $1/(en)$ the following mutation creates an infeasible solution. In the following call of VDS, the first local move is to pull a single vertex to the empty side of the cut. W. l. o. g. this is a u-vertex. In the subsequent moves, VDS moves more u-vertices to the other side as these operations lead to a minimal fitness decrease. Once at least half of the u-clique has been moved, the fitness even increases when moving more u-vertices. Once all u-vertices have been moved, a global optimum is found. In case mutation failed to create an infeasible solution, VDS ends again with a local (or global) optimum. The expected waiting time for a successful mutation is $O(n)$. □

4.2.4 Knapsack

The Knapsack problem is a well-known NP-hard combinatorial problem. Suppose we are given a knapsack that can hold objects up to a specified weight limit W. Among a set of n objects with associated profit values $p_1, \ldots, p_n$ and weights $w_1, \ldots, w_n$, we have to select objects for the knapsack such that the total profit is maximized while respecting the weight limit.

As a fitness function, we take the profit of all chosen objects if the weight limit is respected. Otherwise, the fitness function gives hints to drop selected objects.

$$f(x) := \begin{cases} \sum_{i=1}^{n} x_i p_i & \text{if } \sum_{i=1}^{n} x_i w_i \leq W, \\ -\sum_{i=1}^{n} x_i & \text{if } \sum_{i=1}^{n} x_i w_i > W. \end{cases}$$

Consider the following KNAPSACK instance I for odd n and $N = (n+1)/2$.

$$\begin{aligned} 1 \leq i \leq N\colon\ & p_i = w_i = n \\ N < i \leq n\colon\ & p_i = w_i = n+1 \\ & W = N \cdot n \end{aligned}$$

For all objects profit equals weight. Hence, this instance also represents an instance of the subset sum problem, a restricted formulation of KNAPSACK.

Call the objects with weight $n+1$ *big* and the other ones *small*. The weight limit is chosen such that all N small objects exactly fit into the knapsack. This selection yields a total profit of $(n^2+n)/2$. On the other hand, if one big object is chosen, there is only

space for a total number of $N-1$ objects. If the current packing also contains at least one small object, the profit may be increased by dropping a small object and adding a big object, which increases the profit by 1. Thus, a packing of all $N-1$ big objects is locally optimal w. r. t. $N_1 \cup N_2$ with a profit of $(N-1)\cdot(n+1) = (n^2-1)/2$. Note that here considering the neighborhood N_1 it not very interesting as almost all packings with $N-1$ selected objects are local optima w. r. t. N_1.

We see that we have a non-optimal local optimum with all big objects and a unique global optimum with N small objects. Furthermore, when reaching the local optimum exchanging a big object for a small one decreases the fitness. Only after all big objects have been exchanged for small ones, an N-th small object may be added to yield a global optimum.

Theorem 4.2.9. *Consider the* (1+1) MA *with standard local search using a proper neighborhood and mutation probability $p_m \leq 1/2$. The optimization time for the* Knapsack *instance is at least 2^{cn} with probability $1-2^{-\Omega(n)}$ for some constant $c > 0$.*

Proof. Observe the following simple fact. As long as the (1+1) MA has not found the global optimum $\mathrm{OPT} = 1^N 0^{N-1}$ yet, it shows the same behavior on the function given by the Knapsack instance I as on every other function with the same fitness values for all $x \neq \mathrm{OPT}$. Hence, we may regard a modified instance I' with the same objects as in I, but where $W' = W - 1$, so that OPT receives the same fitness as all other infeasible solutions with the same number of chosen objects. The random time until OPT is sampled for I' equals the random time until OPT is sampled for I; only afterwards, a different behavior may occur. In order to account for local search sampling points within the radius r of the neighborhood, we estimate the optimization time by the time until a solution with Hamming distance at most r to OPT is sampled (either by mutation or during local search).

Let L_k be the set of packings with k selected objects. We also call L_k a level of packings. Let $A <_f B$ for $A, B \subseteq \{0,1\}^n$ if all search points in A have lower fitness than all search points in B. For the instance I' then

$$L_0 <_f L_1 <_f \cdots <_f L_{N-1} >_f L_N >_f L_{N+1} >_f \cdots >_f L_n. \tag{4.2}$$

The same also holds for another modified instance I'' containing n objects with profit and weight $n+1$ and weight limit $W'' = W'$, i. e., a variant of I' where all objects have the same size. We argue that the (1+1) MA behaves at least as bad on I'' as on I'.

Consider two random processes: one instance of the (1+1) MA on I' and another instance of the (1+1) MA on I''. In every generation, both processes perform a mutation which may be followed by a sequence of iterations of standard local search. For each such micro-steps, the number of 1-bits in the next point only depends on the number of 1-bits of the previous point. This holds if the number of 1-bits is changed during such a move. If there is no more local move available that would change the number of 1-bits,

local search stops on this level of 1-bits. When considering a generation of the (1+1) MA from a macro-perspective, if the offspring created by mutation and/or local search differs from its parent in terms of the number of 1-bits, the acceptance only depends on the number of 1-bits. In other words, the only differences between the processes occur when deciding between points with the same number of 1-bits. Hence, the probability that the process on I' has a point in L_i at time t equals the probability that the process on I'' has a point in L_i at the same time. More generally, for each point of time the probability distributions indicating the number of 1-bits in the current search point are identical for both processes.

Only when choosing between any two packings x_1, x_2 from L_k a different behavior may occur: if x_2 has more big objects than x_1, on I' x_2 will be preferred over x_1, while the situation is completely symmetric on I''. Note that x_2 has a larger Hamming distance from OPT than x_1 since the Hamming distance equals the number of selected big objects plus the number of non-selected small objects. Due to the assumption $p_m \leq 1/2$, the probability of creating a specific offspring decreases monotonically with increasing Hamming distance from the parent. Hence the remaining time until a point with Hamming distance at most r to OPT is sampled increases when replacing x_1 by x_2. Therefore, we are pessimistic (from the perspective of proving a negative result) when we accept a move where x_2 is replaced by x_1. This leads us to the symmetric instance I'' and justifies why it is pessimistic to consider I'' instead of I'. Note that local search on I'' will stop as soon as L_{N-1} is reached as there are no neighbors with larger fitness. Contrarily, local search on I' will stay on L_{N-1}, but move away from OPT on this level if possible, hence we are also pessimistic in this special case.

We focus on the symmetric instance I'', where we exploit the symmetry to prove the desired lower bound. The point OPT can only be found if mutation and/or local search create a search point in $Z = \{x \mid \mathrm{H}(x, \mathrm{OPT}) \leq r\}$. Let x_t be the t-th evaluated search point, regardless whether this point has been evaluated after mutation or during local search. The probability that during the first T evaluations a search point in Z is found equals $\sum_{t=1}^{T} \mathrm{Prob}(x_t \in Z)$. Note that we consider unconditional probabilities here; we ask whether the t-th evaluation will yield a point in Z, without conditioning on previous points such as x_{t-1}. Fix t, then by the law of total probability

$$\mathrm{Prob}(x_t \in Z) \;=\; \sum_{k=0}^{n} \mathrm{Prob}(x_t \in Z \mid x_t \in L_k) \cdot \mathrm{Prob}(x_t \in L_k).$$

Observe that due to the perfect symmetry of L_k, each point in L_k is equally likely to be x_t. Moreover, the size of Z is polynomially bounded while L_k has size $2^{\Omega(n)}$ for $k = n/2 \pm O(1)$. Hence, $\mathrm{Prob}(x_t \in Z \mid x_t \in L_k) = 2^{-\Omega(n)}$ and the probability of finding Z within the first T generations is at most

$$\sum_{t=1}^{T} \mathrm{Prob}(x_t \in Z) \;\leq\; \sum_{t=1}^{T} \sum_{k=0}^{n} 2^{-\Omega(n)} \cdot \mathrm{Prob}(x_t \in L_k) \;=\; T \cdot 2^{-\Omega(n)}.$$

Taking the union bound for $T := 2^{cn}$ steps, c a small enough positive constant, proves the claim. □

For KNAPSACK it makes sense to use a neighborhood like $N_1 \cup N_2$, where additions and removals of single objects are allowed as well as exchanges of two objects. In order to provide a fair comparison of Iterated VDS with simulated annealing, we also consider larger neighborhoods than the default neighborhood N_1 for simulated annealing. As the probability of changing the number of selected objects depends on the concrete fitness difference and hence on the fact whether small or big objects are moved, the arguments from the proof of Theorem 4.2.9 do not fully transfer to simulated annealing. Hence, we present a different proof.

Theorem 4.2.10. *The optimization time for simulated annealing on the* KNAPSACK *instance is at least* 2^{cn} *with probability* $1 - 2^{-\Omega(n)}$ *for every proper neighborhood, every cooling schedule, and some constant* $c > 0$.

Proof. The proof is an application of the Drift Theorem 2.3.10 to the random Hamming distance of the current search point to the global optimum OPT. This corresponds to the number of non-selected small objects plus the number of selected big objects. Consider a search point x with Hamming at most $(N-1)/3$ from OPT. The probability that initialization creates such a search point is $2^{-\Omega(n)}$, hence we assume that this does not happen.

Let L_i be the level of all search points x with $|x|_1 = i$. Apart from OPT itself, we have that on a fixed level the fitness increases with the Hamming distance to OPT. Formally, if $y_1, y_2 \in L_i$ and $0 < \mathrm{H}(y_1, \mathrm{OPT}) \leq \mathrm{H}(y_2, \mathrm{OPT})$, then $f(y_1) \leq f(y_2)$. If y_1 and y_2 both have the same Hamming distance k to the current search point, they have an equal probability of being created, but y_2 is more likely to be chosen by selection.

Consider the set $S_{k,i}(x) := N_k(x) \cap L_i$. We claim that for each k, i and every x the following holds. If y is chosen uniformly at random from $S_{k,i}$ and $\mathrm{H}(x, \mathrm{OPT}) \leq (N-1)/3$, then

$$\mathrm{E}(\mathrm{H}(y, \mathrm{OPT}) - \mathrm{H}(x, \mathrm{OPT}) \mid y \in S_{k,i}) \geq \frac{k}{3}. \tag{4.3}$$

Taking into account selection, this means that also the Hamming distance between old search point and new search point increases by at least $k/3$ in expectation. We ignore steps where the new solution is not accepted for our lower bound. In all accepted steps we have $k \geq 1$, hence also the expected distance increase without conditioning on $S_{k,i}$ is positive. This proves the first condition of Theorem 2.3.10 with respect to the distance interval $[0, (N-1)/3]$. The second condition follows from the bounded radius of the neighborhood, hence the claim follows from Theorem 2.3.10.

In order to prove Inequality (4.3), observe that the number of flipping bits is known as k and the number of final 1-bits is known as well. Hence, we already know that the number of flipping 0-bits is $(k + i - |x|_1)/2$ and the number of flipping 1-bits is

$(k - i + |x|_1)/2$. The only randomness left is the question how many of these bits belong to small objects and how many to big objects. Consider the flipping 1-bits, i. e., objects that are unchosen. As x has Hamming distance at most $(N-1)/3$ to OPT, the number of chosen big objects in x is at most $(N-1)/3$ and the number of chosen small objects in x is at least $N - (N-1)/3 \geq 2(N-1)/3$. In expectation, at least $(k - i + |x|_1)/3$ small objects are unchosen in y and at most $(k + i - |x|_1)/6$ big objects are unchosen, which contributes to the expected increase in Hamming distance a value of $(k-i+|x|_1)/6$. Symmetrically, the flipping 0-bits contribute a term of $(k+i-|x|_1)/6$ to the expected increase in Hamming distance, which proves Inequality (4.3) and completes the proof. □

Again, iterated VDS turns out to be effective. This even holds if mutation is never used, corresponding to the choice $p_m = 0$.

Theorem 4.2.11. *Iterated VDS using neighborhood $N_1 \cup N_2$ without mutation (i. e., $p_m = 0$) optimizes the* Knapsack *instance within 2 generations with probability 1.*

Proof. After initialization, VDS either runs into the local or the global optimum. Suppose that we have found the local optimum of $N-1$ big objects and consider the next call of VDS. The least decrease in fitness is to exchange a big object for a small one. This is repeated until all big objects have been replaced by small ones and then the last small object is added. □

4.2.5 Maxsat

Maxsat is another well-known and important combinatorial problem. Given n Boolean variables $x_1, \ldots, x_n$ a literal is either a variable or a negated variable. A clause is a disjunction of literals; for example $(x_1 \vee \overline{x_3} \vee x_4)$ is a clause with three literals. We say that a clause is satisfied w. r. t. an assignment x to the variables if the clause evaluates to true. Given a set C of clauses, the problem Maxsat asks for an assignment of the variables such that the number of satisfied clauses is maximized. This problem is known to be NP-hard even if all clauses only contain 2 literals.

A natural choice of the fitness function is to choose the number of satisfied clauses. This function has already been investigated by Droste, Jansen, and Wegener (2000) on the following instance.

$$\forall i \neq j \neq k \neq i \colon (x_i \vee \overline{x_j} \vee \overline{x_k}) \in C$$
$$(x_1), (x_2), \ldots, (x_n) \in C$$

An important observation is that this instance is symmetric in a sense that all variables are treated equally. Note that every clause has exactly one non-negated literal, hence the assignment 1^n satisfies every clause. On the other hand, most clauses contain two

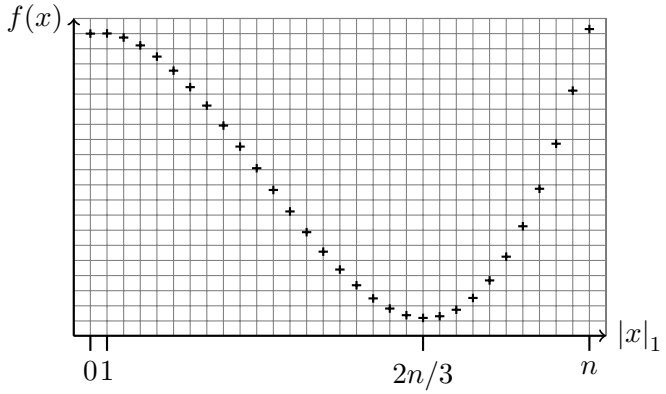

Figure 4.3: Sketch of the fitness landscape according to the MAXSAT instance with $n = 30$.

negated literals. This gives strong hints for a search heuristic to set variables to 0. Due to this deceptive property Papadimitriou (1994) first defined this instance as a worst-case example for the performance of a heuristic algorithm for MAXSAT.

Due to symmetry of the instance, we can formulate the fitness as a function of unitation, i. e., $f(x)$ only depends on the number of 1-bits in x. If $|x|_1 = i$, then i unit clauses (i. e., clauses with just one literal) are satisfied. Among the other $\binom{n}{3}$ clauses there are $n \cdot \binom{i}{2}$ clauses where the last two literals evaluate to false. Moreover, there are $(n-i)$ choices for the first variable such that the first literal also evaluates to false. Hence, $(n-i) \cdot \binom{i}{2}$ clauses of length 3 are unsatisfied. We conclude that the number of satisfied clauses and hence the fitness is given by the formula

$$f(x) = \binom{n}{3} - (n - |x|_1) \cdot \binom{|x|_1}{2} + |x|_1 .$$

A sketch of the function f is shown in Figure 4.3.

It is easy to see that $|x|_1 = n$ implies a global optimum with fitness $\binom{n}{3} + n$. The search point 0^n has fitness $\binom{n}{3}$ and all x with $|x|_1 = 1$ have fitness $\binom{n}{3} + 1$ as $\binom{1}{2} = 0$. Assuming $n \geq 6$ and n multiple of 3, we claim that the fitness decreases with $|x|_1$ in the interval $[1, 2n/3]$. Even stronger, the set $S^* = \{x \mid |x|_1 = 1\}$ is $(2n/3 - 1)$-difficult for the neighborhood N_1.

Lemma 4.2.12. *If $n \geq 6$ is a multiple of 3 then $S^* = \{x \mid |x|_1 = 1\}$ is $(2n/3-1)$-difficult for N_1.*

Proof. Let a_i denote the fitness of all x with $|x|_1 = i$. Note that $\mathrm{H}(x, S^*) = ||x|_1 - 1|$ as for $x = 0^n$ we have $\mathrm{H}(0^n, S^*) = 1$ and otherwise flipping $|x|_1 - 1$ 1-bits in x is necessary

and sufficient to reach a point in S^*. This means that proving $(2n/3-1)$-difficulty reduces to proving $a_0 < a_1 > a_2 > \cdots > a_{2n/3}$. We have $a_0 < a_1$ and claim $a_{i-1} > a_i$ for $i \geq 2$ and $i \leq 2n/3$.

$$\begin{aligned}
a_{i-1} - a_i &= (n-i) \cdot \binom{i}{2} - (n-i+1) \cdot \binom{i-1}{2} - 1 \\
&= (n-i) \cdot \frac{i(i-1)}{2} - (n-i+1) \cdot \frac{(i-1)(i-2)}{2} - 1 \\
&= (n-i) \cdot \frac{i(i-1)}{2} - (n-i) \cdot \frac{i(i-1)}{2} \cdot \frac{i-2}{i} - \frac{(i-1)(i-2)}{2} - 1 \\
&= (n-i) \cdot \frac{i(i-1)}{2} \cdot \frac{2}{i} - \frac{(i-1)(i-2)}{2} - 1 \\
&= (i-1)\left(n - \frac{3i}{2} + 1\right) - 1.
\end{aligned}$$

This expression is greater than 0 if $i = 2$ and $n \geq 4$ as then the term in parentheses is at least 2. For larger i, we exploit $i \leq 2n/3$ and arrive at the bound $(i-1) - 1 > 0$. □

Theorem 4.2.13. *Consider the* (1+1) MA *with standard local search using any proper neighborhood and mutation probability $p_m \leq 1/2$. The optimization time for the* MAXSAT *instance is at least 2^{cn} with probability $1 - 2^{-\Omega(n)}$ for some constant $c > 0$.*

Proof. Let r be the radius of the neighborhood used by local search. The probability that initialization creates a search point with at most $3n/5$ 1-bits is $1 - 2^{-\Omega(n)}$ by Chernoff bounds. Afterwards, a necessary condition to accept a search point whose number of 1-bits is larger than $3n/5$ is to create an offspring with at least $2n/3 - r$ 1-bits by mutation. Let x be a parent with $|x|_1 \leq 3n/5$, then the expected number of 1-bits in the mutant is $|x|_1 (1-p_m) + (n - |x|_1)p_m = p_m(n - 2|x|_1) + |x|_1$. If $|x|_1 \geq n/2$, then $p_m(n - 2|x|_1) + |x|_1 \leq |x|_1$. Otherwise, $n - 2|x|_1$ is non-negative and, along with $p_m \leq 1/2$, $p_m(n - 2|x|_1) + |x|_1 \leq 1/2 \cdot (n - 2|x|_1) + |x|_1 = n/2$. In both cases, the expected number of 1-bits in the offspring is bounded by $3n/5$. By Chernoff bounds, the probability of constructing an offspring with at least $2n/3 - r$ 1-bits is $2^{-\Omega(n)}$. Taking the union bound for 2^{cn} steps, $c > 0$ a small enough constant, proves the claim. □

Theorem 4.2.14. *The optimization time of simulated annealing on the* MAXSAT *instance is at least 2^{cn} with probability $1 - 2^{-\Omega(n)}$ for every cooling schedule and some constant $c > 0$.*

Proof. The probability that initialization creates a search point with at most $3n/5$ 1-bits is $1 - 2^{-\Omega(n)}$ by Chernoff bounds. Consider a search point x with $3n/5 \leq |x|_1 \leq 2n/3$. The probability that an offspring with less 1-bits is created equals $|x|_1 / n \geq 3/5$ and this offspring is accepted with probability 1. The probability that an offspring with more 1-bits is created equals $(n - |x|_1)/n \leq 2/5$. The probability of acceptance depends on

the fitness difference and the temperature, but the trivial bound 1 holds regardless of the temperature. The expected decrease in the number of 1-bits is therefore at least $3/5 - 2/5 = 1/5$, given $3n/5 \leq |x|_1 \leq 2n/3$. The Drift Theorem 2.3.10 applied to the $|x|_1$-value on the interval $[3n/5, 2n/3]$ then yields that the probability of reaching a search point with $2n/3$ 1-bits within 2^{cn} steps is still $2^{-\Omega(n)}$ for some constant $c > 0$, which proves the claim. □

Again, we ask ourselves what iterated VDS can do. Interestingly, iterated VDS without mutation is not effective for the MAXSAT instance.

Theorem 4.2.15. *Iterated VDS with neighborhood N_1 without mutation (i. e., $p_m = 0$) finds the global optimum of the* MAXSAT *instance only with probability $2^{-\Theta(n)}$.*

Proof. The lower bound on the success probability follows trivially from the fact that random initialization creates the global optimum with probability 2^{-n}.

For the upper bound, Chernoff bounds yield that the probability of starting with x such that $2 \leq |x|_1 < 2n/3$ is $1 - 2^{-\Omega(n)}$. In this case the fitness can only be increased by flipping a single 1-bit. Since this bit afterwards cannot flip back to 1, VDS returns a local optimum with a single 1-bit. Having reached such a local optimum, the least fitness decrease is obtained by flipping the unique 1-bit to 0. However, this implies that 1^n cannot be reached. As all other search points have worse fitness, VDS again returns a local optimum with a single 1-bit. □

However, with mutation the global optimum can be reached efficiently.

Theorem 4.2.16. *The expected number of generations of iterated VDS with mutation probability $p_m = 1/n$ on the* MAXSAT *instance is $O(n)$.*

Proof. The first VDS creates a local or global optimum. If a local optimum with a single 1-bit is reached, mutation creates 0^n with probability $1/n \cdot (1 - 1/n)^{n-1} \geq 1/(en)$ and the following VDS reaches 1^n with probability 1. The expected number of generations for this event is at most en. □

This is a first example where a hybrid algorithm outperforms its single components on a combinatorial problem.

4.2.6 Discussion and Conclusions

We have considered single instances for three combinatorial problems and shown that a memetic algorithm with VDS drastically outperforms many popular trajectory-based algorithms like the (1+1) EA, iterated local search, and simulated annealing. The list of combinatorial problems where VDS is effective is not complete. Similar analyses, using techniques from Section 4.2.2, can be performed, e. g., for GRAPH BISECTION, MAXIMUM CLIQUE, and VERTEX COVER. We have chosen MINCUT, KNAPSACK, and

MAXSAT since they represent typical and well-known problems from different classes of problems: cutting, packing, and constraint optimization.

Furthermore, these three problems pose different challenges for randomized search heuristics. The MINCUT instance yields a multimodal landscape with symmetric slopes. A search heuristic typically cannot tell in advance which hill might contain a global optimum. This secret is not revealed until the algorithm climbs to the top of the hill and then it may have to climb down a long distance. For MAXSAT the fitness landscape is deceptive, leading typical heuristics away from the global optimum. For the KNAPSACK instance we exploited that, from a macro-perspective, optimization is like searching for a needle in a haystack. All packings with the same number of objects have similar fitness, but only those without (or only few) big objects are promising. From a micro-perspective the instance is even worse since it gives deceptive hints towards big objects.

Iterated VDS is successful on these problem instances. This is partly due to the fact that VDS can cope with deceptive functions as it always encounters the bit-wise complement of the current search point. One may argue that iterated VDS is no more than a hill climber tailored towards deceptive functions, like for example a hill climber sampling around $\overline{x}$ in addition to the current population x. However, the MINCUT instance cannot be optimized by such a specialized strategy. Another argument is that for MINCUT and MAXSAT VDS after some time discovers a positive gradient towards the global optimum and then is able to reach it "on its own", without the tabu mechanism. Finally, VDS is robust w. r. t. modifications of the instance. In the KNAPSACK instance the global and the local optimum are complementary. However, if we add some new objects with low profit and large weight a simple algorithm for deceptive functions fails. We conclude that VDS is more powerful than an algorithm tailored towards deceptive functions.

However, we cannot conclude that memetic algorithms with VDS are, in general, superior to common trajectory-based algorithms. The perspective taken in this section is one-sided as we only presented instances where memetic algorithms with VDS perform well, compared to common search strategies. It may be possible to find instances where memetic algorithms with VDS perform badly. Moreover, theory should not be restricted to single instances. We therefore regard the presented analyses as appetizers on the usefulness of memetic algorithms in combinatorial optimization from a theoretical perspective. We are still in need of a complete lunch, that is, broader results for important classes of instances for combinatorial problems to bring forward the theoretical understanding of hybrid algorithms.

5 Swarm Intelligence

In nature several animals tend to live in large swarms like insect colonies, bird flocks or fish schools. The reason is that in the swarm each animal is more effective for evolution than single animals. For example, in a fish school chances of survival are higher for each individual in the swarm. While a fish swimming alone is an easy target for predators, in a fish school predators usually have difficulties to single out a fish to attack. Moreover, the school has a much better perception of its environment; after all, many heads are better than one. This enables the fish to quickly react to predator attacks and to find food and, of course, mating partners more easily.

Many social insects like ants, termites, bees, or wasps live in colonies or hives. They exhibit an astonishingly well-developed social behavior and are able to self-organize, even in the absence of a central leader like a queen. The secret of self-organization is the way the insects communicate. Honey bees communicate locations of food sources by the language of dance that is understood by all nearby honey bees. On the other hand, many insects use a form of indirect communication called *stigmergy*. Stigmergy works by leaving traces in the environment that can be understood by other insects. Termites use stigmergy to build complex nests by simple rules. A termite constructing a nest deposits material like a mudball and invests it with *pheromones*, a chemical that can be smelled by other termites. The smell of pheromones encourages other termites to deposit their material close to freshly deposited pheromones. This way, a group of termites can manage to synchronize so that they all work on the same spot.

Certain ant species use a similar strategy in their foraging behavior. While searching the environment for food, the ants deposit pheromones on the ground. Other ants are attracted by pheromones and tend to follow trails of previous ants. Despite its simplicity, this mechanism enables the ants to find shortest paths between the nest and a food source. When ants fan out to find food, it may happen that a lucky ant finds a short path to a new food source. It then takes some food with it and makes its way back to the nest. Since it is attracted by its own pheromone trail, it is likely that the ant follows its own path back to the nest, thereby leaving a second pheromone trail. If other ants happened to take a longer path to the food source, they arrive after the first ant and, when trying to make their way back to the nest, there is a good chance for them to be attracted by the short path, where already two pheromone trails have been laid. This reinforces the short path even more and makes it more attractive. Concerning the longer path, pheromones tend to evaporate after some time, so in the long run the long paths will be forgotten and almost all ants will take the short path.

Ant Colony Optimization

The fascinating collective behavior of swarms is a rich source of inspiration for the field of computational intelligence (see, e. g., Kennedy, Eberhart, and Shi, 2001). In particular, the ability of ants to find shortest paths has been transferred to shortest path problems in graphs (Dorigo and Stützle, 2004). The idea is that artificial ants traverse the graph from a start node (nest) to a target node (food). On each edge a certain amount of artificial pheromone is deposited. At each node each ant chooses which edge to take next. This choice is made probabilistically and according to the amount of pheromone placed on the edges. As in real ant colonies, the pheromones evaporate over time. The amount of evaporation is determined by the so-called *evaporation factor* ρ, $0 < \rho < 1$. In every pheromone update on every edge a ρ-fraction of the pheromone evaporates, i. e., if the edge contains pheromone τ, the remaining amount of pheromone is $(1 - \rho) \cdot \tau$ and then eventually new pheromone is added. Intuitively, a large evaporation factor implies that the impact of previously laid pheromones diminishes quickly and new pheromones have a large impact on the system. Small evaporation factors, on the other hand, imply that the system only adapts slowly to new pheromones.

In contrast to real ants, however, new pheromones are often not placed immediately after traversing an edge. In order to avoid rewarding cycles or paths leading to dead ends, pheromones are usually placed after the ant has found the target node and only for edges that are not part of a cycle on the ant's trail. Also, in such an artificial system the amount of pheromone placed may depend on the length of the constructed path, so that short paths are rewarded more than longer paths. Such an adaptation is often necessary as the movement of artificial ants is usually synchronized, in contrast to real ants in nature, where the order of ants arriving at a location is essential.

Optimization with artificial ants is known as *ant colony optimization* (ACO). ACO algorithms have been used for shortest path problems, but also for the well-known Traveling Salesman Problem (TSP) and routing problems. However, the use of artificial ants is not limited to graph problems. ACO algorithms can also be used to construct solutions for combinatorial problems, e. g., for pseudo-Boolean functions. Such a solution, i. e., a bit string of length n, can be obtained by letting an artificial ant traverse a so-called *construction graph* and mapping the path chosen by the ant to binary values.

Constructing Solutions by Artificial Ants

In Sections 5.1, 5.2, and 5.3, we will analyze various ACO algorithms for pseudo-Boolean functions, based on the so-called graph-based ant system (GBAS) presented by Gutjahr (2000). All algorithms rely on the same mechanism to construct new solutions by artificial ants. We formalize this procedure, adapted to our setting. For a more general description of ACO algorithms, we refer the reader to Dorigo and Stützle (2004).

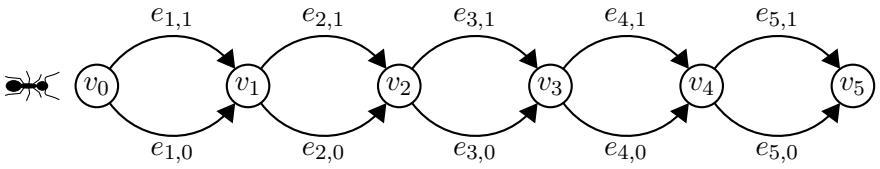

Figure 5.1: Construction graph for pseudo-Boolean optimization with $n = 5$ bits.

A natural construction graph for pseudo-Boolean optimization is known in the literature as *Chain* (Gutjahr, 2008). We use a simpler variant as described by Doerr and Johannsen (2007b), based on a directed multigraph $C = (V, E)$. In addition to a start node v_0, there is a node v_i for every bit i, $1 \leq i \leq n$. This node can be reached from v_{i-1} by two edges. The edge $e_{i,1}$ corresponds to setting bit i to 1, while $e_{i,0}$ corresponds to setting bit i to 0. The former edge is also called a *1-edge*, the latter is called *0-edge*. An example of a construction graph for $n = 5$ is shown in Figure 5.1.

In a solution construction process an artificial ant sequentially traverses the nodes $v_0, v_1, \ldots, v_n$. The decision which edge to take is made according to pheromones on the edges. Formally, we denote pheromones by a function $\tau\colon E \to \mathbb{R}_0^+$. From v_{i-1} the edge $e_{i,1}$ is then taken with probability $\tau(e_{i,1})/(\tau(e_{i,0}) + \tau(e_{i,1}))$. We give a formal description of this procedure for arbitrary construction graphs that returns a path P taken by the ant. In the case of our construction graph, we also identify P with a binary solution x as described above and denote the path by $P(x)$.

Operator 16 Construct(C, τ)

Let $P := \emptyset$, $v := v_0$, and mark v as visited.
repeat
  Let E_v be the set of edges leading to non-visited successors of v in C.
  if $E_v \neq \emptyset$ **then**
    Choose $e \in E_v$ with probability $\tau(e)/\sum_{e' | e' \in E_v} \tau(e')$.
    Let $e = (v, w)$, mark w as visited, set $v := w$, and append e to P.
until $E_v = \emptyset$.
return the constructed path P.

All ACO algorithms considered in this thesis start with an equal amount of pheromone on all edges: $\tau(e_{i,0}) = \tau(e_{i,1}) = 1/2$. Moreover, we ensure that $\tau(e_{i,0}) + \tau(e_{i,1}) = 1$ holds, i. e., pheromones for one bit always sum up to 1. This implies that the probability of taking a specific edge equals its pheromone value; in other words, pheromones and traversion probabilities coincide.

We remark that the solution construction in ACO algorithms is often enhanced by incorporating heuristic information for parts of the solution. The probability of choosing an edge then depends on both the pheromones and the heuristic information. We, however, prefer to stick to the black-box perspective, where no heuristic information is available. This also allows for a fair comparison between different search heuristics. Nevertheless, the impact of heuristic information remains an interesting topic for future work.

Given a solution x and a path $P(x)$ of edges that have been chosen in the creation of x, a pheromone update with respect to x is performed as follows. First, a ρ-fraction of all pheromones evaporates and a $(1-\rho)$-fraction remains. Next, some pheromone is added to edges that are part of the path $P(x)$ of x. However, with these simple rules we cannot exclude that pheromone on some edges may converge to 0, so that the probability of choosing this edge becomes nearly 0. In such a setting, the algorithm has practically no chance to revert a wrong decision. To prevent pheromones from dropping to arbitrarily small values, we follow the MAX-MIN ant system by Stützle and Hoos (2000) and restrict all pheromones to a bounded interval. The precise interval is chosen as $[1/n, 1-1/n]$. This choice is inspired by standard mutations in evolutionary computation where for every bit an evolutionary algorithm has a probability of $1/n$ of reverting a wrong decision.

Depending on whether an edge e is contained in the path $P(x)$ of the solution x, the pheromone values τ are updated to τ' as follows:

$$\begin{aligned} \tau'(e) &= \min\left\{(1-\rho)\cdot\tau(e)+\rho,\ 1-\frac{1}{n}\right\} && \text{if } e \in P(x) \text{ and} \\ \tau'(e) &= \max\left\{(1-\rho)\cdot\tau(e),\ \frac{1}{n}\right\} && \text{if } e \notin P(x). \end{aligned} \tag{5.1}$$

We verify the invariant $\tau(e_{i,0}) + \tau(e_{i,1}) = 1$. This holds obviously at initialization. Assume $\tau(e_{i,0}) + \tau(e_{i,1}) = 1$, then we claim $\tau'(e_{i,0}) + \tau'(e_{i,1}) = 1$. We perform a case distinction depending on which edges have touched their upper or lower pheromone bounds. First, we show that either pheromones are capped for both edges $e_{i,0}$ and $e_{i,1}$ or no pheromones are capped. W. l. o. g. the edge $e_{i,1}$ is rewarded, then pheromone for $e_{i,1}$ is capped at $1-1/n$ if and only if pheromone for $e_{i,0}$ is capped at $1/n$:

$$\begin{aligned} (1-\rho)\cdot\tau(e_{i,0}) &< \frac{1}{n} \\ \Leftrightarrow\ 1-(1-\rho)\cdot\tau(e_{i,0}) &> 1-\frac{1}{n} \\ \Leftrightarrow\ (1-\rho)\cdot(1-\tau(e_{i,0}))+\rho &> 1-\frac{1}{n} \\ \Leftrightarrow\ (1-\rho)\cdot\tau(e_{i,1})+\rho &> 1-\frac{1}{n}. \end{aligned}$$

In case the pheromones are not capped for both edges, we have

$$\tau'(e_{i,0}) + \tau'(e_{i,1}) \;=\; (1-\rho)\cdot(\tau(e_{i,0}) + \tau(e_{i,1})) + \rho \;=\; (1-\rho)\cdot 1 + \rho \;=\; 1.$$

If both pheromones are capped, clearly $\tau'(e_{i,0}) + \tau'(e_{i,1}) = 1/n + 1 - 1/n = 1$, which completes the proof of the invariant. Note that the pheromones on all 1-edges suffices to describe all pheromones. In the following, we will also refer to $p_i = \tau(e_{i,1})$ as *success probability* p_i for bit i, since for the functions investigated hereinafter setting specific bits to 1 can be considered a success.

The pheromone values or the success probabilities form a probabilistic model that somehow reflects a collective memory of many previous paths found by artificial ants. In general, this model is more complex than for evolutionary algorithms. One major difference is that typical evolutionary algorithms can be modelled as a Markov chain since their behavior only depends on the current population. Pheromones in ACO, however, may essentially depend on many past solutions. In an evolutionary algorithm with standard mutation, the probability of setting a specific bit to 1 is either $1/n$ or $1 - 1/n$. In an ACO algorithm, also values in between these two values are possible. The underlying domain is continuous and we cannot exclude that it is infinitely large. This makes an analysis much harder than for evolutionary algorithms and requires new proof methods. Fortunately, the state space for pheromone values is countably infinite, which spares us from dealing with uncountable spaces. Nevertheless, the analysis of ACO algorithms is more challenging than for evolutionary algorithms.

Our Contribution

We strive at following the path taken for the analysis of evolutionary algorithms. The analysis of evolutionary algorithms started for simply structured example functions with interesting properties. Those toy problems encouraged or demanded the development of new methods and tools and revealed important insights into the working principles of evolutionary algorithms. This then allowed the analysis of more sophisticated artificial problems and, finally, the analysis of problems from combinatorial optimization.

It therefore makes sense to start the investigation of ACO algorithms with simply structured example functions as well. The rigorous runtime analysis of ACO algorithms was started independently by Gutjahr (2008) and Neumann and Witt (2006) for (variations of) the function OneMax. Gutjahr (2008) analyzed a GBAS algorithm with a chain construction graph using a slightly different pheromone update rule on a class of generalized OneMax functions. For a fairly large value of ρ, he proved an upper bound of $O(n \log n)$ on the expected number of function evaluations. Neumann and Witt (2006) independently studied a simple ACO algorithm called 1-ANT that resembles and generalizes the (1+1) EA. The 1-ANT keeps track of the best ant solution found so far. If a generation creates a solution that is not worse than the best-so-far solution, a pheromone update with respect to the new solution is performed. Otherwise,

all pheromones remain unchanged. This resembles evolutionary algorithms with elitist selection where the population also remains unchanged if a non-accepted solution is created. Interestingly, Neumann and Witt (2006) proved that on the function OneMax, the evaporation factor ρ has a tremendous impact on the performance of the 1-ANT. If ρ is larger than a certain threshold, the 1-ANT behaves similar to the (1+1) EA. On the other hand, if ρ is below the threshold, the 1-ANT tends to stagnate. As the best-so-far OneMax-value increases more and more, the 1-ANT is forced to discover solutions with more and more 1-bits. However, the impact of the following pheromone update is so small that 1-bits are not rewarded enough and even rediscovering previously set 1-bits gets increasingly harder. This leads to stagnation and exponential runtimes.

In Section 5.1 we investigate the 1-ANT on two other example functions, LeadingOnes and BinVal, where bits have different priorities. The aim is to see whether stagnation also occurs for these functions and with the same threshold values for ρ. As the analysis methods of Neumann and Witt (2006) are tailored towards the symmetric function OneMax, we also develop new methods to deal with non-symmetric functions. The results show that the phase transition behavior observed for OneMax reappears for LeadingOnes and BinVal and, in contrast to OneMax, we are able to determine a tight bound for the location of the phase transition.

A lesson learned from the analysis of the 1-ANT is that performing an update only in case a new best-so-far solution is found may, in general, not be a good design choice. In fact, many ACO algorithms used in applications use so-called *best-so-far reinforcement* where in every generation the current best-so-far solution is reinforced. In other words, in every generation a pheromone update happens, using either the old or a newly generated best-so-far solution. Section 5.2 shows how these algorithms, variants of the MAX-MIN ant system (MMAS), can be analyzed for various example functions, including the class of unimodal functions and plateau functions. Thereby, we follow and extend previous work by Gutjahr and Sebastiani (2008), who first analyzed MMAS variants on OneMax, LeadingOnes, and functions with plateaus. Their results and our contributions show that the impact of ρ is by far not as drastic as for the 1-ANT. When decreasing ρ, the algorithms become more and more similar to random search and the runtime on simple functions grows with $1/\rho$, but there is no phase transition for polynomially small ρ as for the 1-ANT. We also demonstrate how (a restricted formulation of) the fitness-level method from Section 2.3.2 can be adapted to the analysis of ACO algorithms. Finally, we present lower bounds for ACO algorithms: a general lower bound for functions with unique optimum that grows with $1/\rho$ and an almost tight lower bound for LeadingOnes.

Another aspect of ACO algorithms is that in practice often local search is used to improve upon constructed ant solutions, similar to memetic evolutionary algorithms. In addition to the effects of introducing local search already discussed for evolutionary algorithms in Chapter 4, it is interesting to investigate the effect local search can have on the pheromones. In a typical ACO algorithm without local search, old and new best-so-far solutions are typically quite close to one another and the distribution of

constructed ant solutions follows the most recent best-so-far solutions. When introducing local search, old and new best-so-far solutions might be far apart. In Section 5.3 we discuss this effect in more detail and possible implications on search dynamics. To exemplify the impact on performance, we present an artificial function where an ant algorithm with local search drastically outperforms its variant without local search and a second function where the effect is reversed.

Particle Swarm Optimization

Particle swarm optimization represents another class of swarm algorithms inspired by the behavior of fish schools and bird flocks. Originally developed by Kennedy and Eberhart (1995), it has become a popular bio-inspired optimization principle in recent years. A comprehensive treatment is given in the book by Kennedy et al. (2001). A typical particle swarm optimizer (PSO) maintains a swarm of particles where each particle corresponds to a solution of the problem at hand. Most applications of PSO deal with continuous search spaces. There, each particle continuously moves through the space according to a certain velocity, a real vector that indicates both the direction and the speed of the particle. Each particle records the best previous position it has occupied up to now and the best previous position of all particles in its neighborhood. In every iteration, the velocity of a particle is stochastically updated in the direction of its own best position and the best-so-far position in its neighborhood. This means that, intuitively, every particle is attracted both by its memorized own best position and the best position it was able to observe among its neighbors. This kind of behavior is motivated both from animal swarms and from social-psychology theory as it combines cognitive and social effects to determine the behavior of each particle (Kennedy and Eberhart, 1995).

Kennedy and Eberhart (1997) presented a first binary particle swarm optimizer, called *Binary PSO*. As in classical PSO, velocities are used to determine the next position of a particle. However, as each bit may only obtain discrete values 0 and 1, velocities are used in a stochastic solution construction process. More precisely, the velocity value of a bit determines the probability to set this bit to 1 in the next solution construction. This closely resembles the solution construction of the binary ACO algorithms described above. In contrast to pheromones in ACO, however, velocities in PSO do not directly correspond to success probabilities, i. e., probabilities of setting bits to 1. Instead, a velocity value v is translated into a probability using the sigmoid function

$$s(v) := \frac{1}{1 + e^{-v}}$$

sketched in Figure 5.2. The success probability of a bit with velocity v is then $s(v)$. As the sigmoid function is strictly increasing, a larger velocity corresponds to a larger success probability. A velocity of 0 corresponds to a success probability of 1/2. Note

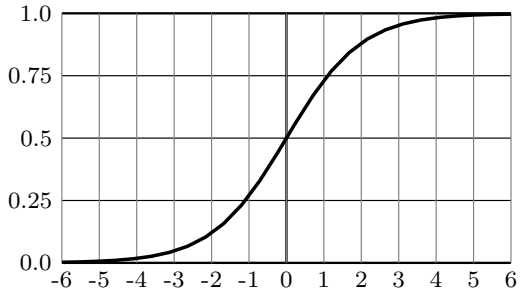

Figure 5.2: The sigmoid function $s(v) = \frac{1}{1+e^{-v}}$.

that as v increases from value 0 to larger values, the amount of increase of the success probability becomes smaller and smaller as the sigmoid function saturates. It is possible that the success probability attains values arbitrarily close to 0 or 1. One way to deal with this possible divergence of velocities is to restrict the velocities to a fixed interval $[-v_{\max}, v_{\max}]$. This solution has been proposed in the original formulation of the Binary PSO (Kennedy and Eberhart, 1997), with $v_{\max}$ defined as a constant value.

In Section 5.4 we present a first rigorous analysis of a binary particle swarm optimizer. As neighborhood we use a global neighborhood containing the whole swarm; a model known as *gbest* model. This implies that all particles are attracted by the same best-so-far position found by the swarm. Instead of choosing a fixed value for $v_{\max}$, we adapt $v_{\max}$ to asymptotically growing problem dimensions, such that the success probability is always in the interval $[1/n,\ 1 - 1/n]$. This corresponds to the choice of the pheromone bounds for the ACO algorithms described above. The adaptation is justified by proving exponential lower bounds for a setting where the velocity bound $v_{\max}$ is fixed while the problem size grows. For the adapted Binary PSO we then present a lower bound for all functions with unique global optimum. Upper bounds are derived by adapting the fitness-level method to PSO, similar to the analysis of MMAS ACO algorithms from Section 5.2. For the function OneMax, we perform a more detailed analysis of a simple Binary PSO variant. The so-called 1-PSO uses just one particle, hence the own best position and the global best position coincide. Despite its simplicity, the 1-PSO turns out to be surprisingly effective and competitive to evolutionary algorithms like the (1+1) EA.

5.1 The 1-ANT Ant Colony Optimization Algorithm

The theoretical runtime analysis for ant colony optimization lags far behind the results for evolutionary algorithms. Until 2006 only convergence results (e. g., Gutjahr, 2003) and results on the dynamics of models of ACO (e. g., Merkle and Middendorf, 2002)

were known. In a survey on theoretical studies of ACO by Dorigo and Blum (2005), researchers were encouraged to follow the approach taken for the analysis of EAs by starting a rigorous runtime analysis of simple ACO algorithms on OneMax. Soon after this appeal, first runtime analyses appeared in a work by Gutjahr (2008), and, independently, in a work by Neumann and Witt (2006). In the latter paper, a simple ACO algorithm called 1-ANT is defined based on the model of Gutjahr (2003) and the runtime w. r. t. the fitness function OneMax is bounded from above and below. It is shown that the evaporation factor ρ, the probably most important parameter in ACO algorithms, has a crucial impact on the runtime. More precisely, it is proved that there exists a threshold value for ρ below which no efficient optimization is possible. This threshold behavior shows that the 1-ANT is not robust w. r. t. the choice of ρ.

A closer look at the paper by Neumann and Witt (2006) reveals that the mathematical methods employed for the analysis of the 1-ANT differ heavily from those for the analysis of EAs. Even more conspicuously, it seems that the mathematical tools are tailored for the symmetric function OneMax. It is by no means clear whether a comprehensive runtime analysis of the 1-ANT can be conducted on more complicated problems. A recent analysis of the 1-ANT on the combinatorial minimum spanning tree problem by the same authors (Neumann and Witt, 2008) basically considers a special case of the 1-ANT with two pheromone values and fails to deliver statements on the choice of ρ.

The aim of this section is to put forward the analysis of the 1-ANT on example problems in a similar fashion to Neumann and Witt (2006). As Gutjahr (2007) has observed, such analyses are an important and emergent issue in the community of ACO. We choose the non-symmetric functions LeadingOnes and BinVal investigated by Droste, Jansen, and Wegener (2002a) and analyze the runtime of the 1-ANT on these functions w. r. t. n and the evaporation factor ρ.

The motivation for analyzing example problems is the same as for initial studies of evolutionary algorithms. Example functions are chosen that are simple enough to be attacked by rigorous arguments, have interesting properties, and yet reflect typical characteristics of practical problems. The simple function OneMax may be rediscovered in a practical problem when a specific target point has to be hit and the fitness function gives good hints towards this point. In the case of the functions LeadingOnes and BinVal, the bits have different priorities; some bits are more important than other bits. For BinVal this resembles a situation where the fitness gives good hints towards a target, but some local moves are rewarded more than others. The function LeadingOnes contains strong dependencies between specific bits, a characteristic often found in practical problems. Another interesting feature is that some bits are irrelevant to the fitness, until other bits have been set correctly. The investigations for LeadingOnes and BinVal in this section put forward the understanding of ACO algorithms in such situations and lead to the development of new proof techniques.

Our upcoming analyses for these functions show that a similar phase transition behavior can be observed as proven by Neumann and Witt (2006) for OneMax. If ρ is

asymptotically smaller than a threshold, no efficient optimization is possible; however, for values a little above the threshold, polynomial runtimes are very likely. Hence, our investigations again suggest that the 1-ANT is not robust w. r. t. the choice of ρ. The proofs contribute new methods for the runtime analysis of ACO algorithms and may serve as a basis for further theoretical studies.

The outline of this section is as follows. In Section 5.1.1, we provide the necessary definitions and recapitulate the previous results for OneMax. Section 5.1.2 proves general properties of the pheromone update mechanism that will be used in the following analyses. Sections 5.1.3 and 5.1.4 deal with the main results of the section, namely lower and upper bounds on the runtime of the 1-ANT on LeadingOnes, respectively. A generalization of the results to the function BinVal is discussed in Section 5.1.5. Our theoretical results are complemented by empirical results in Section 5.1.6 where we also consider an extension of the 1-ANT creating multiple ant solutions in every generation. We conclude in Section 5.1.7.

5.1.1 The Algorithm

We consider the 1-ANT algorithm defined by Neumann and Witt (2006). In its original formulation, the pheromone model was designed such that all pheromones in the whole construction graph sum up to 1 (instead of just the outgoing edges for every vertex). Also, a slightly different construction graph was used, where then the probabilities of choosing a 0-edge or a 1-edge differed from the respective pheromone values by a factor of $2n$. As still an absolute value of ρ was added to every rewarded edge, pheromones had to be normalized in every pheromone update. This pheromone model has also been used in the conference paper (Doerr, Neumann, Sudholt, and Witt, 2007b) and the extended journal version this section is based on. In this thesis, however, we use the simpler pheromone model, along with the simpler construction graph, both described in the introduction to this chapter. The simpler model has already been proposed by Doerr and Johannsen (2007b). The main advantage is that here pheromones on an edge exactly reflect the probability that this edge is taken, which avoids pheromone normalization and simplifies the calculations in comparison to Doerr et al. (2007b). There, many formulas contain a factor $2n$ as an artifact from the different scale. Moreover, the simpler model matches the pheromone model used for MMAS algorithms analyzed in the following sections. The price for changing the model is that ρ-values in one model translate to different ρ-values in the other model. This transformation can be done as follows. If ρ_{old} is an evaporation factor in the old model,

$$\rho_{\text{new}} = \frac{2n\rho_{\text{old}}}{1-\rho_{\text{old}}+2n\rho_{\text{old}}}$$

is an equivalent evaporation factor in the simplified model. We aim at a consistent pheromone model throughout this thesis, therefore we only consider the simpler model

in the following. The results for ρ-values reported in Neumann and Witt (2006) then have to be translated into the simplified model. Another minor disadvantage is that the experiments in Section 5.1.6 were done with the old pheromone model in mind. However, we think that the advantage of a consistent presentation significantly outweighs all mentioned drawbacks.

Recall that in our pheromone model for each bit the pheromones on its 0-edge and its 1-edge sum up to 1. In the initialization step of the 1-ANT, each edge gets a pheromone value of 1/2. An initial solution x^* is produced by the construction procedure described before. The pheromone values are then updated with respect to the constructed solution, according to the update formula from Equations (5.1). In each iteration following these initial steps, a new solution x is constructed and the pheromone values are updated if this solution is not inferior to the currently best solution x^*.

Algorithm 17 1-ANT

Set $\tau(e) := 1/2$ for all $e \in E$.
Construct a solution x^*.
Update pheromones w. r. t. x^*.
repeat
    Construct a solution x.
    if $f(x) \geq f(x^*)$ **then**
        Set $x^* := x$.
        Update pheromones w. r. t. x^*.

In the following, we use the term *generation* synonymously to *iteration*. Let $p_i = \text{Prob}(x_i = 1)$, $1 \leq i \leq n$, be the probability of setting the bit x_i to 1 in the next constructed solution. Neumann and Witt (2006) have shown a close connection between the 1-ANT and the (1+1) EA. The (1+1) EA sets a bit x_i in x to 1 with probability $1/n$ (if $x_i^* = 0$) or $1 - 1/n$ (if $x_i^* = 1$). The 1-ANT behaves like the (1+1) EA if $\rho \geq (n-2)/(n-1)$ (with respect to our pheromone model). The reason is that then the 1-ANT's pheromone values always attain their upper or lower bounds after the first update has occurred.

For the function OneMax the influence of ρ has been analyzed in greater detail. Neumann and Witt have shown that there is a phase transition from exponential to polynomial runtime as ρ grows. In particular, they have given an exponential lower bound for the case $\rho = O(n^{-\varepsilon})$ and a polynomial upper bound for $\rho = \Omega(1)$ where $\varepsilon > 0$ is a positive constant. In addition, Doerr and Johannsen (2007b) have proven that no polynomial expected runtimes are possible for $\rho = o(1/\log n)$. The main argument for the lower bound is that the value of the currently best solution and the expected value of the one constructed in the next iteration may differ. This makes it difficult even to rediscover the best-so-far solution, which leads to an exponential optimization time. In

contrast to this, the polynomial upper bound for large ρ relies on the observation that the function value of the last accepted solution determines the expected value of the next constructed solution almost exactly.

We consider the function LeadingOnes. Recall

$$\textsc{LeadingOnes}(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j,$$

i. e., the function value equals the number of leading ones in the considered bit string x. A non-optimal solution may always be improved by appending a single one to the leading ones. LeadingOnes differs from OneMax in the essential way that the assignment of the bits after the leading ones do not contribute to the function value. This implies that bits at the beginning of the bit string have a stronger influence on the function value than bits at the end. Because of this, the methods developed by Neumann and Witt (2006) cannot be used for analyzing the 1-ANT on LeadingOnes as these methods make particular use of the fact that all bits contribute equally to the function value. We will develop new methods to deal with the circumstance that different bits may have different priorities for the optimization process. A well-known linear function that relies on the different priorities is BinVal (introduced by Droste et al. (2002a)) defined as

$$\textsc{BinVal}(x) = \sum_{i=1}^{n} 2^{n-i} x_i,$$

which interprets a bit string as the binary representation of an integer. After having analyzed the 1-ANT on LeadingOnes, we will show how to adapt the developed methods for analyzing the 1-ANT on BinVal.

5.1.2 On the Pheromone Update Mechanism

To analyze the 1-ANT for pseudo-Boolean optimization, it is necessary to understand the development of pheromone values for single bits. The probability of setting a bit to 1 is called *success probability*. Recall that the success probability equals the pheromone value on the corresponding 1-edge. An analogous statement holds for the probability of setting the bit to 0 and the 0-edge. Finally, throughout the analysis it is crucial to note that the bits are processed independently by the construction procedure.

Consider an arbitrary but fixed bit x_i. If this bit is set to 1 in the next constructed solution, we speak of a success and a failure otherwise. Obviously, if a success occurs, the success probability in the next step is increased, unless the pheromone bounds have been hit in the previous step. The amount of increase depends on the previous pheromone value on the 1-edge (or, equivalently, the previous success probability). We introduce a notation for a success probability that has been increased t times.

Definition 5.1.1. Let p be the current success probability of a specific bit. Let $p^{(t)}$ be its success probability after $t \geq 0$ successes and no failures at the bit.

The following lemma describes how the rewarded success probability relates to the unrewarded one.

Lemma 5.1.2. *For every $t \geq 0$, unless $p^{(t)}$ is capped by pheromone bounds,*

$$p^{(t)} = 1 - (1-p) \cdot (1-\rho)^t.$$

Proof. We prove the claim by induction on t. The case $t = 0$ is obvious. For $t \geq 1$, using the induction hypothesis,

$$\begin{aligned} p^{(t)} &= (1-\rho)p^{(t-1)} + \rho \\ &= (1-\rho)(1 - (1-p) \cdot (1-\rho)^{t-1}) + \rho \\ &= 1 - \rho - (1-p) \cdot (1-\rho)^t + \rho \\ &= 1 - (1-p) \cdot (1-\rho)^t. \end{aligned}$$

□

By the preceding lemma, $p \geq q$ implies $p^{(t)} \geq q^{(t)}$ for all $t \geq 0$. Note that this also holds if the upper bound for the success probability is hit. This justifies in our forthcoming analyses the places where actual success probabilities are replaced with lower bounds on these probabilities.

5.1.3 A Lower Bound for LeadingOnes

We show that the 1-ANT is very inefficient on LeadingOnes if $\rho = o(1/\log n)$, i. e., if ρ is asymptotically smaller than $1/\log n$. The following theorem shows that then even polynomially many multistarts fail within polynomial time with overwhelming probability.

One of the main reasons for the failure is that with the small evaporation factor, the success probabilities at single bits can reach large enough values only slowly. In consequence, the 1-ANT is faster in finding good solutions than in storing this knowledge in the pheromone values.

Theorem 5.1.3. *With probability $1 - 2^{-\Omega(\min\{1/\rho,\, n\})}$, the runtime of the* 1-ANT *on* LeadingOnes *is at least $2^{c \cdot \min\{1/\rho,\, n\}}$ for some constant $c > 0$.*

Note that the statement of the theorem is only meaningful for $\rho = o(1)$. However, if we choose, e. g., $\rho = 1/\log^2 n$, the bound is already superpolynomially large.

Proof. Let $k = 1/(4\rho)$. Consider the state of the 1-ANT at the earliest time when one of the following two conditions is fulfilled.

(i) The fitness f_c of the current solution is at least $n/2$.

(ii) 1-ANT has performed k accepted steps.

We first convince ourselves that, in this situation, all success probabilities never left the interval $[1/4, 3/4]$, then, that with high probability we are not done yet, and finally, that the next accepted step takes the time claimed in the theorem.

Consider the success probability of a certain bit. During one pheromone update, the amount of new pheromone added is ρ. It follows that at most k accepted steps, independent of the particular accepted solutions, can increase the pheromone and the success probability by at most $k\rho$. Hence the success probability p after k such steps is bounded by $p \leq 1/2 + k\rho = 3/4$. By symmetry, the same holds for the failure probability $1 - p$. Hence $p \in [1/4, 3/4]$.

Now let us regard the last (accepted) step before the system reached the state fixed above. At the start of this step, we have all success probabilities in $[1/4, 3/4]$ and our current solution has fitness $f_0 < n/2$. Hence the probability that the 1-ANT finds the optimal solution in this single step is bounded by $(3/4)^{(n/2)}$—recall that we know already that this step will be accepted, hence the first f_0 bits of this solution are 1 with probability 1. Nevertheless, we see that with probability $1 - 2^{-\Omega(n)}$, we have not found the optimum yet.

Finally, let us estimate the time to obtain an accepted step from the state fixed above. We first estimate the current fitness f_c. If we did not perform k accepted steps, then clearly $f_c \geq n/2$. Hence let us assume that we actually did perform k accepted steps. Conditional on the fact that a step was accepted, the probability that this lead to a fitness increase is at least $1/4$ since the probability of a success at the leftmost zero-bit is at least $1/4$. Hence $\mathrm{E}(f_c) \geq k/4$, and Chernoff bounds imply $\mathrm{Prob}(f_c \geq k/8) = 1 - 2^{-\Omega(k)} = 1 - 2^{-\Omega(1/\rho)}$. Combining the two cases, we have

$$\mathrm{Prob}(f_c \geq \min\{n/2,\ k/8\}) \ = \ 1 - 2^{-\Omega(\min\{1/\rho,\, n\})}.$$

If $f_c \geq \min\{n/2, k/8\}$, then the probability that the next step is accepted is at most $(3/4)^{\min\{n/2,\, \lfloor k/8 \rfloor\}}$, i. e., at most $2^{-\Omega(\min\{1/\rho,\, n\})}$. Consequently, there is some small constant $c > 0$ such that a phase consisting of $2^{c \cdot \min\{1/\rho,\, n\}}$ steps does not produce another accepted step with probability $1 - 2^{-\Omega(\min\{1/\rho,\, n\})}$. □

5.1.4 An Upper Bound for LeadingOnes

In contrast to the situation from the last section, large values of ρ allow the 1-ANT to rediscover the leadings ones of previous solutions efficiently. In order to prove an upper bound for the 1-ANT, we make use of the observations from Section 5.1.2 and the following characteristics of LeadingOnes.

If k is the current best LeadingOnes-value, the success probabilities for the first k bits are increased in every pheromone update as an update is only performed when the

leading ones are rediscovered. We say that these bits are in *increasing state*. The bit at position $k+1$ is called an *essential bit* since it is essential for the fitness evaluation. All bits following the first 0-bit have been irrelevant to the fitness evaluation up to now. Since the pheromone update process is symmetric for 0-edges and 1-edges, the probability that an ant creates a 1 on such a bit equals the probability that the ant creates a 0. These bits are therefore said to be in *random state*.

Due to this randomness, an improvement may increase the current best LeadingOnes-value from k to a value larger than $k+1$ if the 1-ANT happens to create further bits with value 1. If the best LeadingOnes-value increases to $k+\ell$, the bits at positions $k+2,\ldots,k+\ell$ are called *free riders* (see Droste, Jansen, and Wegener, 2002a) if $\ell \geq 2$. In other words, free riders are 1-bits that we get "for free" in addition to the essential 1-bit at position $k+1$.

Free riders may help to increase the fitness, but on the other hand they may also prove as a burden for the 1-ANT. If an improvement creates many free riders, the 1-ANT is forced to rediscover all these free riders in the following steps in order to have another update. If ρ is small, this effect slows down the optimization. This is taken into account in the proof of the following upper bound.

Theorem 5.1.4. *The expected runtime of the* 1-ANT *on* LeadingOnes *is bounded from above by* $O(n^2 \cdot (6e)^{2/\rho})$.

Note that the bound is polynomial for $\rho = \Omega(1/\log n)$ and only $O(n^2)$ for $\rho = \Omega(1)$. For $\rho = o(1/\log n)$, it is superpolynomially large.

Proof. We show that the expected time until increasing the so far maximum LeadingOnes-value is always bounded by $O(n \cdot (6e)^{2/\rho})$ provided the optimum has not been reached. Multiplying the expected time for an improvement by the maximum number of improvements, n, will yield the theorem. In this proof, we focus on the main ideas. Some technical details are deferred to lemmas that will be stated and proved afterwards.

Suppose the currently best LeadingOnes-value equals $k < n$. For an improvement, it is necessary and sufficient to set the first $k+1$ bits in a newly constructed solution to 1. Since the essential bit at position $k+1$ was set to 0 in the last accepted solution, its success probability was decreased in the last pheromone update. Therefore, we estimate the success probability from below by $1/n$ for this bit. If we can prove that the first k bits are all set to 1 after an expected number of $O((6e)^{2/\rho})$ steps, we obtain the theorem as the 1-ANT processes all bits independently.

The first k bits are all in increasing state, but their success probabilities may differ significantly. To simplify the considerations, we remove the pheromone bounds for a bit as soon as it enters the increasing state. This implies that the success probabilities of these bits are allowed to exceed $1-1/n$. We argue that we then underestimate the expected optimization time by only a constant factor. The success probabilities with and without pheromone bounds differ by at most $1/n$. Hence, with at most $n-1$

bits in increasing state, the probability that solution constructions with and without pheromone bounds differ in at least one of these bits is at most $1 - (1 - 1/n)^{n-1} \leq 1 - 1/e$. If a solution construction with pheromone bounds creates value 0 for a bit in increasing state, this solution is rejected by the 1-ANT as otherwise the bit would not be in increasing state. Moreover, in this case the current best-so-far solution and all pheromone values remain untouched. Hence, compared to the setting without pheromone bounds for bits in increasing state, we have "idle" steps for the 1-ANT with pheromone bounds. Our modification disregards these idle steps, therefore we underestimate the expected optimization time by a factor of at most $1/(1 - 1/e)$.

As a result of our modification, the success probability of the i-th bit, $1 \leq i \leq k-1$, has been increased at least as often as the one of the $(i+1)$-st bit. However, the leading k bits we want to rediscover are still of two different types. Such a bit has either been essential in the past or it has been added to the leading bits as a free rider. This distinction is crucial since the premises for the unrewarded success probabilities are different:

1. The success probability of an essential bit x_i is decreased every time a best-so-far solution with $i - 1$ leading ones is created. At the time x_i enters increasing state, we therefore expect its unrewarded success probability to be biased towards the lower bound $1/n$.

2. A free rider x_i enters increasing state directly from random state, hence the unrewarded success probability p_i is random. More precisely, due to the 1-ANT's symmetric treatment of bits in random state, the expected unrewarded success probability equals $1/2$. The distribution of p_i is symmetric around the expectation, i. e., $\text{Prob}(p_i = x) = \text{Prob}(p_i = 1 - x)$ for $0 \leq x \leq 1$ (for a formal proof we refer to Lemma 5.2.9 in Section 5.2). We are dealing with a discrete distribution since the underlying state space is countably infinite.

In the following, when speaking of "rediscovering an essential bit (a free rider)", this means a success for a bit that has been an essential bit (a free rider) in the past, but now has reached the increasing state. We first concentrate on rediscovering all essential bits in increasing state. The unrewarded success probability of one such bit is bounded below by the general lower bound for success probabilities, $w^{(0)} := 1/n$. Fortunately the rewarded success probabilities can be much larger. Temporarily scanning the leading ones from right to left, the t-th essential bit in increasing state has been rewarded at least t times. Hence, its success probability is bounded below by the worst-case lower bound $w^{(t)}$, recalling the notation for rewarded success probabilities from Definition 5.1.1. As the 1-ANT sets bits to 1 independently, the probability for successes on all these bits is bounded by the product over $w^{(t)}$. The expected number of steps until this happens is bounded by the reciprocal since the considered random variable is geometrically distributed. We use the following estimation, the proof of which is deferred

to Lemma 5.1.5 following this proof:

$$\prod_{t=1}^{\infty} \frac{1}{w^{(t)}} \;\leq\; \prod_{t=1}^{\infty} \left(1 + \frac{1}{(1+\rho)^t - 1}\right) \;=\; O\Big((2e)^{2/\rho}\Big).$$

We are left with the task of rediscovering all free riders. We claim that this expected time is at most $O(3^{2/\rho})$, implying a total bound of $O((6e)^{2/\rho})$ for rediscovering all leading ones.

The consideration of free riders is more difficult than the arguments for essential bits. This is due to the fact that in an improvement the number of gained free riders is random. This motivates us to divide the k leading ones into blocks of random length. Denote by b the number of improving steps so far. All free riders gained during the i-th improvement, $1 \leq i \leq b$, form a block of free riders. Let us pessimistically assume that an unbounded number of free riders is possible, i. e., we prolong the bit string if necessary until the first 0-bit is created. We first estimate the success probability of a block, i. e., the probability of setting all bits of the block to 1, in isolation and argue later how to extend this analysis to multiple blocks. In the following, we will derive bounds on the expected time to rediscover a single block after the block has received t updates.

Consider the bit string of all m bits in random state. Let p_i be the success probability of the i-th bit in random state at the time an improvement takes place. Consider the new block of free riders created during this improvement and let $T^{(t)}$ denote the random time until this new block of free riders is rediscovered (or the global optimum is found), provided it has been rewarded t times. Our aim is to bound $\mathrm{E}\big(T^{(t)}\big)$. Therefore, we abbreviate $\mathbf{p} = (p_1, \ldots, p_m)$ and apply the law of total expectation

$$\mathrm{E}\Big(T^{(t)}\Big) = \mathrm{E}\Big(\mathrm{E}\Big(T^{(t)} \mid \mathbf{p}\Big)\Big)$$

as it is easier to estimate the inner, conditional expectation. Let L denote the random length of the new block of free riders. By the law of total probability

$$\mathrm{E}\Big(T^{(t)} \mid \mathbf{p}\Big) = \sum_{\ell=0}^{m} \mathrm{E}\Big(T^{(t)} \mid L = \ell \wedge \mathbf{p}\Big) \cdot \mathrm{Prob}(L = \ell \mid \mathbf{p}).$$

In order to have block length ℓ, the 1-ANT has to be successful on the first ℓ bits and to fail on the $(\ell+1)$-st bit. Therefore $\mathrm{Prob}(L = \ell \mid \mathbf{p}) = \prod_{i=1}^{\ell} p_i \cdot (1 - p_{\ell+1})$. The probability of rediscovering the first ℓ bits equals the product of the rewarded success probabilities $p_i^{(t)}$ since the 1-ANT processes bits independently. Since LeadingOnes is considered, no pheromone update takes place unless these bits are rediscovered. Hence, the waiting time is geometrically distributed with the mentioned product as parameter.

This yields

$$\begin{aligned}
\mathrm{E}\Big(T^{(t)} \mid \mathbf{p}\Big) &= \sum_{\ell=0}^{m-1} \prod_{i=1}^{\ell} p_i \cdot (1-p_{\ell+1}) \cdot \prod_{i=1}^{\ell} \frac{1}{p_i^{(t)}}. \\
&= \sum_{\ell=0}^{m-1} (1-p_{\ell+1}) \prod_{i=1}^{\ell} \frac{p_i}{p_i^{(t)}} \\
&= \sum_{\ell=0}^{m-1} (1-p_{\ell+1}) \prod_{i=1}^{\ell} \frac{p_i}{1-(1-p_i)\cdot(1-\rho)^t}, \qquad (5.2)
\end{aligned}$$

where the last equation follows from Lemma 5.1.2. Note that in case $L = m$ the global optimum has been found, hence we only deal with the case $\ell \leq m-1$.

To estimate the unconditional expectation $\mathrm{E}\big(T^{(t)}\big)$, we repeat that the p_i are independent random variables and their distributions satisfy the symmetry $\mathrm{Prob}(p_i = x) = \mathrm{Prob}(p_i = 1-x)$ for $0 \leq x \leq 1$, implying $\mathrm{E}(p_i = 1/2)$. Note, however, that the distributions are time-dependent. In the initial step, $\mathrm{Prob}(p_i = 1/2) = 1$, which need not be the case in later steps. We may ignore this time dependence by stating a bound which holds for all distributions of the above kind.

$$\begin{aligned}
\mathrm{E}\Big(T^{(t)}\Big) &= \mathrm{E}\Big(\mathrm{E}\Big(T^{(t)} \mid \mathbf{p}\Big)\Big) \\
&= \mathrm{E}\left(\sum_{\ell=0}^{\infty} (1-p_{\ell+1}) \prod_{i=1}^{\ell} \frac{p_i}{1-(1-p_i)\cdot(1-\rho)^t}\right) \\
&= \sum_{\ell=0}^{\infty} \mathrm{E}(1-p_{\ell+1}) \prod_{i=1}^{\ell} \mathrm{E}\left(\frac{p_i}{1-(1-p_i)\cdot(1-\rho)^t}\right) \\
&= \frac{1}{2} \sum_{\ell=0}^{\infty} \prod_{i=1}^{\ell} \mathrm{E}\left(\frac{p_i}{1-(1-p_i)\cdot(1-\rho)^t}\right).
\end{aligned}$$

For the third equation we have used the independence of the random variables and for the fourth equation we have used $\mathrm{E}(1-p_i) = 1/2$. The remaining expectation can be bounded by the following inequality, which is proven by Lemma 5.1.6 following this proof. The proof of Lemma 5.1.6 only relies on the symmetry of p_i and hence it holds regardless of the time dependence.

$$\mathrm{E}\left(\frac{p_i}{1-(1-p_i)\cdot(1-\rho)^t}\right) \leq \frac{1}{2-(1-\rho)^t}.$$

Together, we bound $\mathrm{E}\big(T^{(t)}\big)$ by

$$\frac{1}{2} \sum_{\ell=0}^{\infty} \prod_{i=1}^{\ell} \frac{1}{2-(1-\rho)^t} \leq \frac{1}{2}\left(1 + \frac{1}{1-(1-\rho)^t}\right) = 1 + \frac{1}{2}\frac{1}{(1-\rho)^{-t}-1},$$

where we have estimated the sum by an infinite series. Plugging in $(1-\rho) \leq 1/(1+\rho)$ yields

$$\mathrm{E}\Big(T^{(t)}\Big) \;\leq\; 1 + \frac{1}{2}\frac{1}{(1+\rho)^t - 1} \;:=\; \alpha(t).$$

We have successfully bounded the expected time to rediscover a single block of free riders. Although the distributions of success probabilities in later blocks differ from those in earlier blocks as is the case with the actual block lengths, our upper bound $\alpha(t)$ on $\mathrm{E}\big(T^{(t)}\big)$ holds regardless on the configuration of earlier blocks. This intuitively explains that the product over the corresponding $\alpha(t)$ for all blocks represents a bound for the expected time to rediscover all blocks of free riders in increasing state. A formal proof of this is given by Lemma 5.1.7 following this proof. Recall that the t-th newest block has been rewarded at least t times. Finally, the product of the $\alpha(t)$ can be bounded similarly to the product of inverse worst-case lower bounds from Lemma 5.1.5, it is proven by Lemma 5.1.8 at the end of this section. We obtain

$$\prod_{t=1}^{\infty} \alpha(t) \;\leq\; \prod_{t=1}^{\infty} \left(1 + \frac{1}{2}\frac{1}{(1+\rho)^t - 1}\right) \;=\; O\Big(3^{2/\rho}\Big)$$

which completes the proof. □

We still have to deliver a few technical lemmas that have been used in the proof of Theorem 5.1.4.

Lemma 5.1.5. *Let $w^{(0)} = 1/n$, then*

$$\prod_{t=1}^{\infty} \frac{1}{w^{(t)}} \;\leq\; \prod_{t=1}^{\infty} \left(1 + \frac{1}{(1+\rho)^t - 1}\right) \;=\; O\Big((2e)^{2/\rho}\Big).$$

Proof. By Lemma 5.1.2 and $(1-\rho) \leq \frac{1}{1+\rho}$,

$$\begin{aligned}
\prod_{t=1}^{\infty} \frac{1}{w^{(t)}} &= \prod_{t=1}^{\infty} \frac{1}{1 - (1-1/n)(1-\rho)^t} \\
&\leq \prod_{t=1}^{\infty} \frac{1}{1 - (1+\rho)^{-t}} \\
&= \prod_{t=1}^{\infty} \left(1 + \frac{1}{(1+\rho)^t - 1}\right).
\end{aligned}$$

For the second statement, we bound the first $\alpha = \lceil 1/\rho \rceil$ factors of the above product separately. By Bernoulli's inequality (Lemma A.15),

$$\begin{aligned}\prod_{t=1}^{\alpha}\left(1+\frac{1}{(1+\rho)^t-1}\right) &\leq \prod_{t=1}^{\alpha}\left(1+\frac{1}{\rho t}\right)\\ &\leq \prod_{t=1}^{\alpha}\left(1+\frac{\alpha}{t}\right) = \frac{(2\alpha)!}{(\alpha!)^2}\\ &= \frac{\Theta(\sqrt{\alpha})(2\alpha)^{2\alpha}}{\Theta(\alpha)\alpha^{2\alpha}} = O(4^\alpha)\end{aligned}$$

where the last estimations follow from Stirling's approximation (Lemma A.9) and $\alpha \geq 1$. The remaining terms are bounded as follows, using $(1+1/\alpha)^\alpha \geq 2$.

$$\begin{aligned}&\prod_{t=\alpha+1}^{\infty}\left(1+\frac{1}{(1+\rho)^t-1}\right)\\ \leq\ &\prod_{t=\alpha+1}^{\infty}\left(1+\frac{1}{(1+1/\alpha)^t-1}\right)\\ \leq\ &\prod_{t=\alpha+1}^{\infty}\left(1+\frac{1}{2^{t/\alpha}-1}\right)\\ \leq\ &\prod_{t=\alpha+1}^{\infty}\left(1+\frac{2}{2^{t/\alpha}}\right)\\ \leq\ &\prod_{t=\alpha+1}^{\infty}\exp\left(\frac{2}{2^{t/\alpha}}\right) = \exp\left(\sum_{t=\alpha+1}^{\infty}\frac{2}{2^{t/\alpha}}\right).\end{aligned} \tag{5.3}$$

Now,

$$\sum_{t=\alpha+1}^{\infty}\frac{2}{2^{t/\alpha}} \leq \sum_{t=\alpha}^{\infty}\frac{2}{2^{\lfloor t/\alpha \rfloor}}$$

and grouping α summands and performing an index transformation with $j = t\alpha$, we arrive at

$$\sum_{t=\alpha+1}^{\infty}\frac{2}{2^{t/\alpha}} \leq \sum_{j=1}^{\infty}\frac{2\alpha}{2^j} = 2\alpha.$$

Plugging this into (5.3) yields

$$\prod_{t=\alpha+1}^{\infty}\left(1+\frac{1}{(1+\rho)^t-1}\right) \leq e^{2\alpha}$$

and for the whole product the bound $O\left((4e^2)^\alpha\right) = O((2e)^{2/\rho})$. □

Lemma 5.1.6. *Consider a random variable $X \in [0,1]$ with finite support where the symmetry $\text{Prob}(X = x) = \text{Prob}(X = 1 - x)$ holds, then*

$$\text{E}\left(\frac{X}{1-(1-X)(1-\rho)^t}\right) \leq \frac{1}{2-(1-\rho)^t}.$$

Proof. We show that the expectation is maximized if $\text{Prob}(X = 1/2) = 1$. Let $\Omega(X)$ denote the (finite) support of X. By the symmetry of the distribution,

$$\begin{aligned}\text{E}\left(\frac{X}{1-(1-X)(1-\rho)^t}\right) &= \sum_{x\in\Omega(X)} \text{Prob}(X = x)\frac{x}{1-(1-x)(1-\rho)^t} \\ = \sum_{x\in\Omega(X)\cap[1/n,1/2)} &\text{Prob}(X = x)\left(\frac{x}{1-(1-x)(1-\rho)^t} + \frac{1-x}{1-x(1-\rho)^t}\right) \\ &+ \text{Prob}(x = 1/2)\cdot\frac{1/2}{1-(1-\rho)^t/2}.\end{aligned}$$

Since a finite set is considered, the term in parentheses attains a maximum at some point x^*. The expectation is maximized if the distribution gives maximal mass to x^* and $1 - x^*$, which means mass $1/2$ each if $x^* < 1/2$ and mass 1 if $x^* = 1 - x^* = 1/2$. It is easy to see that $\frac{x}{1-(1-x)(1-\rho)^t}$ is concave for positive x and $0 \leq (1-\rho)^t \leq 1$, hence the term in parentheses is non-decreasing, and we must choose $x^* = 1/2$ to maximize the expectation. This implies

$$\text{E}\left(\frac{X}{1-(1-X)(1-\rho)^t}\right) \leq \frac{1/2}{1-(1-1/2)(1-\rho)^t} = \frac{1}{2-(1-\rho)^t}.$$

□

We justify why blocks of free riders can be treated in isolation in the proof of Theorem 5.1.4. The product of bounds on waiting times for single blocks yields an upper bound on the waiting times for all blocks. We formulate the theorem also for BinVal as it will be reused in Section 5.1.5.

Lemma 5.1.7. *Consider the* 1-ANT *on either* LeadingOnes *or* BinVal. *Let $L_k \in \mathbb{N}_0$ be the (random) block length of the k-th block of "free-rider" bits gained during the k-th increase in the number of leading ones. Abbreviate $\ell[1,i] := \{L_1 = \ell_1 \wedge \cdots \wedge L_i = \ell_i\}$ for a given history of lengths for the first i blocks. Given $\ell[1,i]$, let $p_{i,j}^{\ell[1,i]}$ denote the concrete success probability of the j-th bit in the i-th block at a current point of time.*

Let $T_{\text{all}}(k)$ be the unconditional random number of further steps until the 1-ANT *for the first time rediscovers the first k blocks of free-rider bits. If there are upper bounds $\alpha_1, \ldots, \alpha_k$ such that*

$$\sum_{\ell_i=0}^{n} \text{Prob}(L_i = \ell_i \mid \ell[1, i-1]) \cdot \prod_{j=1}^{\ell_i} \frac{1}{p_{i,j}^{\ell[1,i]}} \;\leq\; \alpha_i$$

for all $1 \leq i \leq k$, then

$$\text{E}(T_{\text{all}}(k)) \;\leq\; \prod_{i=1}^{k} \alpha_i.$$

Proof. Note that

$$\text{Prob}(\ell[1,k]) \;=\; \prod_{i=1}^{k} \text{Prob}(L_i = \ell_i \mid \ell[1, i-1]). \tag{5.4}$$

Applying the law of total probability with respect to the k block lengths, we have

$$\text{E}(T_{\text{all}}(k)) \;=\; \sum_{\ell[1,k]} \text{Prob}(\ell[1,k]) \cdot \text{E}(T_{\text{all}}(k) \mid \ell[1,k]). \tag{5.5}$$

In the conditional expectation on the right-hand side, the lengths of all considered blocks are known and the concrete success probabilities are given as well. As all blocks cover disjoint portions of the bit string and the 1-ANT processes all bits independently, the probability of rediscovering all these bits is just the product of all success probabilities involved. Since LeadingOnes or BinVal is considered, no pheromone update takes place unless this event happens. Hence, the waiting time is geometrically distributed with the product of success probabilities as parameter, i. e.,

$$\text{E}(T_{\text{all}}(k) \mid \ell[1,k]) \;=\; \prod_{i=1}^{k} \prod_{j=1}^{\ell_i} \frac{1}{p_{i,j}^{\ell[1,i]}}.$$

Along with Equations (5.4) and (5.5),

$$\text{E}(T_{\text{all}}(k)) \;=\; \sum_{\ell[1,k]} \prod_{i=1}^{k} \text{Prob}(L_i = \ell_i \mid \ell[1, i-1]) \cdot \prod_{j=1}^{\ell_i} \frac{1}{p_{i,j}^{\ell[1,i]}}.$$

Abbreviate

$$F_i \;:=\; \text{Prob}(L_i = \ell_i \mid \ell[1, i-1]) \cdot \prod_{j=1}^{\ell_i} \frac{1}{p_{i,j}^{\ell[1,i]}}.$$

Pessimistically assuming that all blocks can have lengths between 0 and n, we have

$$\begin{aligned}\mathrm{E}(T_{\mathrm{all}}(k)) &= \sum_{\ell[1,k]} \prod_{i=1}^{k} F_i \leq \sum_{\ell_1=0}^{n} \sum_{\ell_2=0}^{n} \cdots \sum_{\ell_k=0}^{n} \prod_{i=0}^{k-1} F_i \\ &= \sum_{\ell_1=0}^{n} F_1 \cdot \sum_{\ell_2=0}^{n} F_2 \cdots \sum_{\ell_k=0}^{n} F_k \\ &\leq \prod_{i=1}^{k} \alpha_i\end{aligned}$$

according to the given bounds. □

Lemma 5.1.8.

$$\prod_{t=1}^{\infty} \left(1 + \frac{1}{2} \frac{1}{(1+\rho)^t - 1}\right) = O\left(3^{2/\rho}\right).$$

Proof. The proof is very similar to the proof of Lemma 5.1.5 as the calculations only differ in some constants. We now bound the first $\alpha = \lceil 1/(2\rho) \rceil$ factors separately. Using Bernoulli's inequality and Stirling's approximation,

$$\begin{aligned}\prod_{t=1}^{\alpha} \left(1 + \frac{1}{2} \frac{1}{(1+\rho)^t - 1}\right) &\leq \prod_{t=1}^{\alpha} \left(1 + \frac{1}{2\rho t}\right) \\ &\leq \prod_{t=1}^{\alpha} \left(1 + \frac{\alpha}{t}\right) = \frac{(2\alpha)!}{(\alpha!)^2} \\ &= \frac{\Theta(\sqrt{\alpha})(2\alpha)^{2\alpha}}{\Theta(\alpha)\alpha^{2\alpha}} = O(4^\alpha).\end{aligned}$$

The remaining terms are bounded as follows, using $(1 + 1/(2\alpha))^\alpha \geq 3/2$.

$$\begin{aligned}&\prod_{t=\alpha+1}^{\infty} \left(1 + \frac{1}{2} \frac{1}{(1+\rho)^t - 1}\right) \\ \leq &\prod_{t=\alpha+1}^{\infty} \left(1 + \frac{1}{2} \frac{1}{(1+1/(2\alpha))^t - 1}\right) \\ \leq &\prod_{t=\alpha+1}^{\infty} \left(1 + \frac{1}{2} \frac{1}{(3/2)^{t/\alpha} - 1}\right) \\ \leq &\prod_{t=\alpha+1}^{\infty} \left(1 + \frac{1}{2} \frac{3}{(3/2)^{t/\alpha}}\right) \\ \leq &\prod_{t=\alpha+1}^{\infty} \exp\left(\frac{3}{2} \frac{1}{(3/2)^{t/\alpha}}\right) = \exp\left(\frac{3}{2} \sum_{t=\alpha+1}^{\infty} \frac{1}{(3/2)^{t/\alpha}}\right). \qquad (5.6)\end{aligned}$$

Now,

$$\sum_{t=\alpha+1}^{\infty} \frac{1}{(3/2)^{t/\alpha}} \leq \sum_{t=\alpha}^{\infty} \frac{1}{(3/2)^{\lfloor t/\alpha \rfloor}}$$

and grouping α summands and performing an index transformation with $j = t\alpha$, we arrive at

$$\frac{3}{2} \sum_{t=\alpha+1}^{\infty} \frac{1}{(3/2)^{t/\alpha}} \leq \frac{3}{2} \cdot \alpha \sum_{j=1}^{\infty} (2/3)^j = 3\alpha.$$

Plugging this into (5.6) yields

$$\prod_{t=\alpha+1}^{\infty} \left(1 + \frac{1}{(1+\rho)^t - 1}\right) \leq e^{3\alpha}$$

and for the whole product the bound $O\big((4e^3)^\alpha\big) = O\big(3^{2/\rho}\big)$ as $(4e^3)^{1/4} < 3$. □

Theorems 5.1.3 and 5.1.4 reveal a phase transition in the behavior of the 1-ANT on LeadingOnes for $\rho \sim 1/\log n$. Below this threshold, no efficient optimization is possible since the effect of pheromone updates is more or less irrelevant. This is similar to the behavior observed on OneMax, where the threshold value (using the present notation) has been confined to a small region around $1/\log n$, more precisely between $\Omega(1/\log n)$ (Doerr and Johannsen, 2007b) and $O(1)$ (Neumann and Witt, 2006). This shows that the 1-ANT is not robust w. r. t. the choice of the parameter ρ on two well-known and simple example functions.

5.1.5 Generalization to BinVal

The example function BinVal (see Section 5.1.1) is a linear function, although, in some respect an extreme example. The coefficient 2^{n-i} of the i-th bit outweighs the sum of all smaller coefficients. This leads to the following relation to LeadingOnes: If LeadingOnes(x') > LeadingOnes(x) for $x, x' \in \{0,1\}^n$ then also BinVal(x') > BinVal(x). This allows us to treat the LeadingOnes-value of the current solution as a potential function while BinVal is optimized. It is sufficient to increase the potential at most n times to reach the optimal solution (albeit the number of different BinVal-values is 2^n). Similarly, with probability $1 - 2^{-\Omega(n)}$, it is necessary to increase the potential altogether by at least $n/2$ to reach the optimum since the initial LeadingOnes-value does not exceed $n/2$ with this probability.

When the (1+1) EA optimizes BinVal, the described approach allows us to immediately take over the upper bound $O(n^2)$ for the expected optimization time on LeadingOnes. This is not the best bound possible since $O(n \log n)$ can be shown by a direct

approach (Droste et al., 2002a). Such a direct approach seems difficult for the 1-ANT on BinVal. Therefore, we rather try to transfer our results from LeadingOnes to BinVal using the potential function.

With respect to the lower bound, we inspect the proof of Theorem 5.1.3. Instead of considering the real BinVal, we take the LeadingOnes-value of a bit string as a pseudo-fitness. The arguments on the pheromone values are still valid, and moreover, it is still necessary to create a solution with pseudo-fitness at least n to optimize BinVal, implying that the optimum is not found with probability $1 - 2^{-\Omega(n)}$ before the crucial point of time in the proof is reached. The Chernoff-bound-type arguments on the (pseudo-)fitness carry over, too. Moreover, for an accepted step, it is afterwards still necessary to create a search point with pseudo-fitness at least f_c. We have shown that $\rho = o(1/\log n)$ leads to superpolynomial runtimes also on BinVal.

Theorem 5.1.9. *With probability* $1 - 2^{-\Omega(\min\{1/\rho,\, n\})}$*, the runtime of the* 1-ANT *on* BinVal *is at least* $2^{c\cdot\min\{1/\rho,\, n\}}$ *for some constant* $c > 0$*.*

We can also adapt the proof of the upper bound on LeadingOnes for BinVal. As for LeadingOnes, the first leading ones are in increasing state since a failure on a leading one is not accepted by the 1-ANT. In addition, a sufficient condition for an improvement, provided we have k leading ones, is to have successes for the first $k+1$ bits. We can also speak of essential bits and free riders in the context of BinVal, where the distinction is again made whenever a bit enters increasing state. Using Lemma 5.1.5, the expected time to rediscover all essential bits is bounded by $O((2e)^{2/\rho})$.

The only but essential difference between LeadingOnes and BinVal is that BinVal is influenced by the configuration of the bits following the leftmost zero. Hence, the former "random state" is now biased towards high success probabilities and the probability of a bit being a "free rider" is likely to exceed 1/2. Recall $\mathrm{E}\big(T^{(t)}\big) = \mathrm{E}\big(\mathrm{E}\big(T^{(t)} \mid p_1, \ldots, p_m\big)\big)$. While the estimation of the inner conditional expectation from Equation (5.2) still holds true for BinVal, we cannot rely on the symmetry of the p_i-variables to estimate the outer expectation. However, we can make an estimation that does not rely on specific properties of the success probabilities $p_1, \ldots, p_m$. Instead, we show a bound that holds for all success probabilities.

Lemma 5.1.10. *For all* $m, t \geq 1$ *and all* $p_1, \ldots, p_m \in [0,1]$

$$\sum_{\ell=0}^{m-1}(1-p_{\ell+1})\prod_{i=1}^{\ell}\frac{p_i}{1-(1-p_i)(1-\rho)^t} \;\leq\; \frac{1}{1-(1-\rho)^t}.$$

Proof. For $0 \leq j \leq m-1$ define

$$S_j \;:=\; \sum_{\ell=j}^{m-1}(1-p_{\ell+1})\prod_{i=j+1}^{\ell}\frac{p_i}{1-(1-p_i)(1-\rho)^t}.$$

Observe that for $1 \leq j \leq m-1$ the S_j are due to the following recursive formulation:

$$S_{j-1} \;=\; (1-p_j) + \frac{p_j}{1-(1-p_j)(1-\rho)^t} \cdot S_j.$$

We prove $S_j \leq 1/(1-(1-\rho)^t)$ by induction on j. The case $j = 0$ proves the lemma. We start with $j = m-1$:

$$S_{m-1} \;=\; (1-p_m) \;\leq\; 1 \;\leq\; \frac{1}{1-(1-\rho)^t}.$$

Assume $S_j \leq 1/(1-(1-\rho)^t)$ and note that S_j does only depend on $p_{j+1}, \ldots, p_m$. This yields

$$\begin{aligned} S_{j-1} \;&=\; 1 - p_j + \frac{p_j}{1-(1-p_j)(1-\rho)^t} \cdot S_j \\ &\leq\; \max_{p_j} \left\{ 1 - p_j + \frac{p_j}{1-(1-p_j)(1-\rho)^t} \cdot \frac{1}{1-(1-\rho)^t} \right\} \end{aligned}$$

since S_{j-1} is monotone increasing with S_j. Call the term in brackets $r(p_j)$. We now argue that $r(p_j)$ is monotone increasing with p_j. Hence, the maximum is attained for $p_j = 1$ and it follows

$$S_{j-1} \;\leq\; 1 - 1 + \frac{1}{1-(1-1)(1-\rho)^t} \cdot \frac{1}{1-(1-\rho)^t} \;=\; \frac{1}{1-(1-\rho)^t}.$$

The derivative of $r(p_j)$ is

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}p_j} r(p_j) \;&=\; -1 + \frac{1-(1-p_j)(1-\rho)^t - p_j(1-\rho)^t}{(1-(1-p_j)(1-\rho)^t)^2} \cdot \frac{1}{1-(1-\rho)^t} \\ &=\; -1 + \frac{1-(1-\rho)^t}{(1-(1-p_j)(1-\rho)^t)^2} \cdot \frac{1}{1-(1-\rho)^t} \\ &=\; -1 + \frac{1}{(1-(1-p_j)(1-\rho)^t)^2} \end{aligned}$$

and this term is positive as $(1-(1-p_j)(1-\rho)^t)^2 \leq 1$. □

Lemma 5.1.10 bounds the expected time to rediscover a single block of "free-rider" bits after t updates by $\alpha(t) := 1/(1-(1-\rho)^t)$. Lemma 5.1.7 proves that the product over the $\alpha(t)$ is an upper bound on the expected time to rediscover all "free riders." We estimate

$$\prod_{t=1}^{\infty} \frac{1}{1-(1-\rho)^t} \;\leq\; \prod_{t=1}^{\infty} \left(1 + \frac{1}{(1+\rho)^t - 1}\right) \;=\; O\Big((2e)^{2/\rho}\Big)$$

where the first inequality follows from $(1-\rho) \leq 1/(1+\rho)$ and the second estimate follows from Lemma 5.1.5. This holds independently from the success probability of a bit at the time when it first becomes a "free rider." Together with the bound $O((2e)^{2/\rho})$ to rediscover the essential bits in increasing state, we have shown the following theorem.

Theorem 5.1.11. *The expected runtime of the* 1-ANT *on* BinVal *is bounded from above by* $O(n^2 \cdot (4e^2)^{2/\rho})$.

As for LeadingOnes, Theorems 5.1.9 and 5.1.11 show a phase transition behavior for $\rho \sim 1/\log n$. This strengthens the impression that the efficiency of the 1-ANT on pseudo-Boolean optimization problems is not robust w. r. t. the pheromone update mechanism and the choice of ρ.

5.1.6 Experiments

This section complements our theoretical investigations with experimental studies. The theoretical results are asymptotic ones. The aim of our experimental investigations is to show that the properties proven in the previous sections can also be observed for realistic input sizes. Later on, we also examine what happens if more than one ant is used in each iteration.

First, we examine how the experimental results for the 1-ANT complement our theoretical investigations. Our theoretical results show a phase transition in the runtime behavior of the 1-ANT on LeadingOnes and BinVal for $\rho \sim 1/\log n$. Experiments can reveal details about this phase transition that are not covered by our analyses, for example the exact location of the transition, including constant factors. Therefore, we investigate 18 values of ρ proportional to $1/\log n$ for the problem size $n = 64$. The precise choice of these values is due to the old pheromone model. With respect to ρ-values in the old model, values $\rho = c/(n \log n)$ have been chosen for $c \in \{0.15, 0.2, \ldots, 1.0\}$. After transformation to the simplified model, the new ρ-values are not perfectly equidistant; however, they are almost equidistant as the relationship between old and new ρ-values is almost linear on this part of the ρ-scale. In the following plots, we use the scale of the new pheromone model to maintain consistency throughout the thesis.

For the upper bound on LeadingOnes we argued that free riders slow down the optimization if ρ is small. For BinVal, we expect to have more free riders and we were only able to prove a worse upper bound than for LeadingOnes. It is therefore interesting to ask whether this intuition about free riders is true, i. e., whether the 1-ANT really performs worse on BinVal than on LeadingOnes.

The number of generations to obtain an optimal solution is shown in Figure 5.3 in dependence of the evaporation factor ρ. The results are averaged over 100 runs in each setting. Note that the number of generations is displayed on a logarithmic scale. We see that with decreasing values of ρ the runtime increases drastically for both functions and that the 1-ANT becomes inefficient for $\rho < 0.08/\log n$. It can also be observed

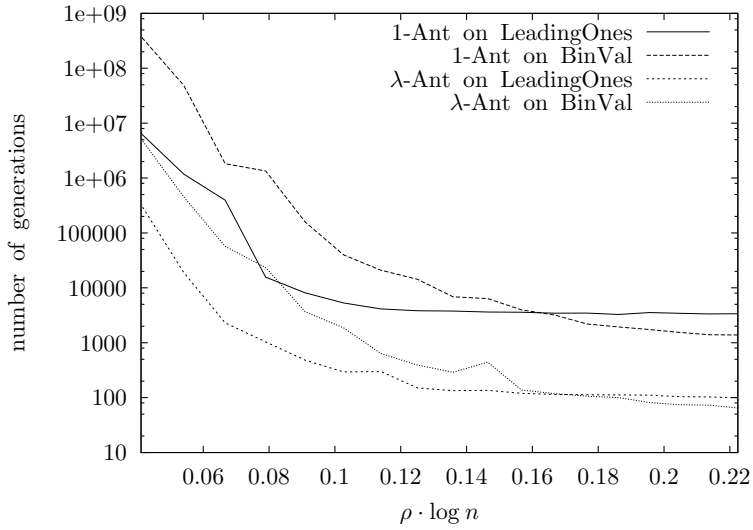

Figure 5.3: Number of generations for the 1-ANT and the λ-ANT to find the global optimum on LeadingOnes and BinVal, resp., for $\lambda = n = 64$, averaged over 100 runs for each value of ρ.

that the runtime of the 1-ANT on BinVal is larger than for LeadingOnes if ρ is small. This supports our argumentation that free riders can be a burden for the 1-ANT. On the other hand, for larger values of ρ the 1-ANT is more efficient on BinVal than on LeadingOnes. This effect can also be explained by theoretical arguments. With growing ρ the 1-ANT behaves more and more like the (1+1) EA and the (1+1) EA is more efficient on BinVal than on LeadingOnes. This difference is not reflected in our upper bound for BinVal as we focus on the runtime behavior for small ρ.

Another goal of our experimental studies is to examine situations related to our theoretical model where our rigorous analyses do not give a picture of the runtime behavior of ACO algorithms. A possible weakness of the 1-ANT is that it produces in each iteration just one single solution. Usually, many ants are used in one iteration for updating the pheromone values. Therefore, we also study a more realistic ACO algorithm that is closely related to the algorithm analyzed in the previous sections. Our algorithm called λ-ANT uses λ ants instead of a single one. The best solution found among the λ solutions is used for updating the pheromone values if this solution is at least as good as the best solution found so far during the run of the algorithm.

For our experimental studies, we choose $\lambda = n = 64$ such that the number of ants used in one iteration is sufficiently large to examine the effect of using more than one ant. As ACO algorithms can easily be parallelized, we report the number of generations

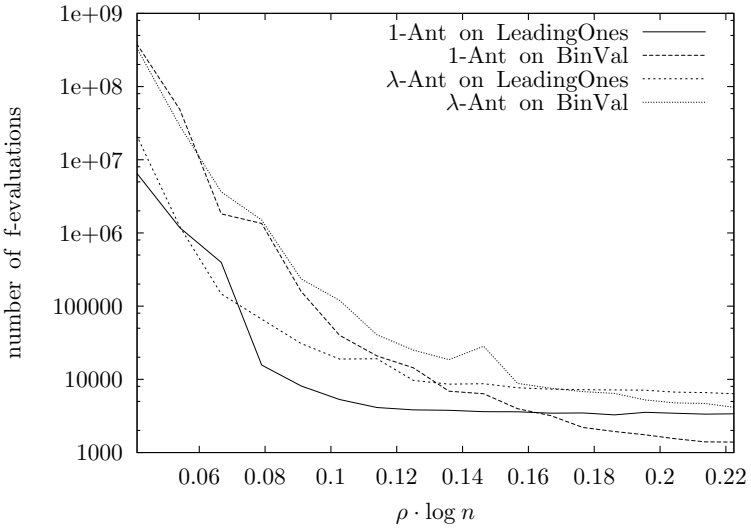

Figure 5.4: Number of fitness evaluations for the 1-ANT and the λ-ANT to find the global optimum on LeadingOnes and BinVal, resp., for $\lambda = n = 64$, averaged over 100 runs for each value of ρ.

(see Figure 5.3) as well as the number of fitness evaluations (see Figure 5.4) the λ-ANT needs to reach the optimal solution for LeadingOnes and BinVal. These results show that the number of generations can be reduced by using more than one ant in each iteration. This has already been observed in the context of evolutionary algorithms where the number of iterations can be reduced by producing λ offspring instead of a single one (see, e. g., Jansen et al., 2005).

As in the case of the investigations for evolutionary algorithms it is hard to reduce the number of fitness evaluations. Figure 5.4 shows larger average runtimes for the λ-ANT compared to the 1-ANT if ρ is large. For smaller ρ the curves partly overlap. Therefore, we use statistical tests for the hypothesis that the 1-ANT outperforms the λ-ANT. A non-parametric Mann-Whitney test shows a significant advantage for the 1-ANT for all observed values of ρ. The significance levels are $p < 0.05$ for LeadingOnes and $\rho = 0.041/\log n$, $p < 0.01$ for LeadingOnes and $\rho = 0.054/\log n$, and $p < 0.0001$ for all other ρ-values and all results on BinVal.

Due to these tests and insights from our theoretical investigations for the 1-ANT, we conjecture that using just one ant is the optimal choice for our model when considering the functions LeadingOnes and BinVal. However, a rigorous proof for this conjecture remains open.

5.1.7 Conclusions

We have investigated the runtime behavior of a simple ACO algorithm called 1-ANT. Our investigations show some general properties for the update scheme used in the 1-ANT. Based on these investigations, we have shown that there is a phase transition for $\rho \sim 1/\log n$ from exponential to polynomial runtimes for the functions LeadingOnes and BinVal. Our experimental investigations point out that this can even be observed for instances of moderate size as well as for the λ-ANT using a linear number of ants in each iteration. Moreover, the experiments indicate that using many ants cannot decrease the number of function evaluations. This result shows that ACO algorithms using a single ant are not only interesting from a theoretical point of view, but also have practical relevance.

5.2 MMAS Ant Colony Optimizers on Unimodal Functions and Plateaus

The analysis of the 1-ANT from Section 5.1 has shown that the 1-ANT is very sensitive w. r. t. the choice of the evaporation factor ρ. Decreasing the value of ρ only by a small amount may lead to a phase transition and turn a polynomial runtime into an exponential one. In contrast to this, a simple ACO algorithm called MMAS_{bs} has been investigated in a recent article by Gutjahr and Sebastiani (2008) where this phase transition does not occur.

The 1-ANT and MMAS_{bs} differ in their pheromone update mechanisms as well as in their selection strategies. While MMAS_{bs} reinforces the current best-so-far solution in each iteration, the 1-ANT only updates its pheromones in case the best-so-far solution is exchanged. Regarding the selection, MMAS_{bs} only accepts strict improvements while the 1-ANT also accepts solutions with equal fitness. In a recent survey, Gutjahr (2007, page 76) conjectures with reference to the above-mentioned phase transition observed for the 1-ANT:

> "The reason seems to lie in the different update rules for the best-so-far solution: exchanging x^* also in case of an equally good solution, as it is done in 1-ANT, obviously deteriorates the performance of the algorithm on OneMax. For LeadingOnes, the situation appears to be quite similar as for OneMax."

By an investigation of variants of MMAS_{bs}, we will disprove this conjecture that the different behavior of MMAS_{bs} and 1-ANT, i. e., the lack/presence of the phase transition, is due to their different selection strategies. Actually, we show that the different behavior results from the different pheromone update mechanisms.

The 1-ANT needs exponential time to optimize even very simple functions like OneMax, LeadingOnes, or BinVal in case the evaporation factor is set too low (see Neu-

mann and Witt (2006) and Section 5.1). Hence the update mechanism of MMAS_{bs} with pheromone updates in each iteration is to be preferred over the mechanism used by the 1-ANT. This motivates us to study MMAS variants with this more persistent update scheme. First, we consider the MMAS_{bs} algorithm by Gutjahr and Sebastiani (2008), in this thesis referred to as MMAS*, and show both improved and extended results. Due to the update mechanism of MMAS*, it is possible to use the fitness-level method (see Section 2.3.2) for the analysis of this algorithm. In Gutjahr and Sebastiani (2008), it is presented in a relatively general and formal way. Within this section, we state the method more intuitively by extending the presentation for evolutionary algorithms. After having clarified the fitness-level method in the context of ACO algorithms, we show that it is easy to obtain upper bounds on the simple pseudo-Boolean functions OneMax and LeadingOnes. These results for MMAS* have already been obtained in Gutjahr and Sebastiani (2008), but our simplified presentation of the fitness-level method yields really simple and short proofs which are easily transferable to other functions of similar structure. In particular, we prove general results on the important class of unimodal functions, which are not contained in Gutjahr and Sebastiani (2008).

Since the fitness-level method contains several pessimistic assumptions on the behavior of MMAS*, it only yields upper bounds. To judge the quality of the method, it is therefore crucial to complement the results by good lower bounds on the optimization time. A major contribution of this section and, at the same time, an extension to the analysis by Gutjahr and Sebastiani (2008), are the proofs of such lower bounds in the context of OneMax, and, in particular, LeadingOnes, where no lower bounds on the runtime of MMAS* were known before. These bounds show that upper bounds obtained before using the fitness-level method are almost tight. Still, there is room for improvement. For the LeadingOnes example, we present an improved upper bound which is tight with the lower bound for almost all values of ρ. This analysis indicates the limits of the general method. Moreover, the detailed study of MMAS* on LeadingOnes is interesting also from a methodological point of view. The random walk described by pheromone values for "unimportant" bits is investigated in detail and its hitting times for boundary values are bounded using a general study of martingale processes. This technique fosters our understanding of the stochastic process behind random pheromone values.

Afterwards, we examine the issue whether to accept equally good solutions or just strict improvements. Accepting equally fit solutions allows MMAS algorithms to explore plateaus of solutions with equal fitness. We compare the runtime behavior of MMAS* with a variant MMAS that accepts solutions of equal fitness on functions with plateaus of exponential and polynomial size, respectively. The analyses, which again rely on the above-mentioned martingale process behind random pheromone values, reveal that MMAS clearly outperforms MMAS* on both functions.

As our theoretical results are purely asymptotic, they are accompanied by empirical results. A comparison of MMAS and MMAS* on the unimodal functions OneMax and

LeadingOnes shows that these algorithms have similar performance. However, a closer look reveals that for certain values of ρ MMAS is faster than MMAS*. This difference is statistically significant. With regard to the above-mentioned conjecture in Gutjahr (2007), accepting equally good solutions does not deteriorate the performance of MMAS* on OneMax. Instead, the opposite is true for some values of ρ. Our theoretical results on plateau functions predict larger runtimes for MMAS*. Experiments for the investigated plateau functions show that the observed performance difference between MMAS and MMAS* is huge, even for small problem dimensions.

The remainder of this section is structured as follows. In Section 5.2.1, we introduce the above-mentioned algorithms that are the subject of our investigations. In Section 5.2.2, we analyze MMAS* and MMAS on unimodal functions. Section 5.2.3 then compares MMAS* with MMAS on plateau functions. Section 5.2.4 contains experimental supplements to a result from Section 5.2.3 and experiments comparing MMAS with MMAS* as mentioned above. We finish with some conclusions.

5.2.1 Algorithms

We consider the runtime behavior of several ACO algorithms that all use the construction procedure described in the introduction to this chapter. The first algorithm that has been investigated in terms of rigorous runtime analysis is the 1-ANT shown in Figure 5.5.

The 1-ANT only performs a pheromone update in case the best-so-far solution x^* is replaced. If ρ is large, such an update has a large effect on the pheromone values. In that case, the 1-ANT is able to reproduce the new best-so-far solution x^* quite efficiently and to discover better solutions in the region around x^*. However, if ρ is small, updates are rare and they only have a small effect on the pheromones. This makes it hard for the 1-ANT to focus on promising regions of the search space. The analyses from Neumann and Witt (2006) and Section 5.1 have revealed that the 1-ANT is unable to keep track of the best-so-far solution. As the 1-ANT discovers better and better best-so-far solutions, the hurdles for constructed solutions are set higher and higher. If the update strength is too small, the algorithm fails in storing the knowledge gained by new best-so-far solutions in the pheromones. The probability of reproducing the current best-so-far solution or finding a better one decreases and the 1-ANT slowly gets stuck. In fact, even for really simple functions like OneMax, the 1-ANT needs exponential time with overwhelming probability if ρ is small, e. g., less than $1/\log n$.

One aim of this section is to elaborate on the different behavior of the 1-ANT and the algorithm by Gutjahr and Sebastiani (2008) without phase transition behavior. The latter authors call their algorithm MMAS_{bs}, we refer to it as MMAS*. This algorithm is shown on the right-hand side in Figure 5.5. Compared to the 1-ANT, there are two differences. First, MMAS* only accepts strict improvements, i. e., new solutions x with $f(x) = f(x^*)$ are rejected by MMAS* while they are accepted by the 1-ANT. A second difference is that MMAS* reinforces the best-so-far solution in every generation,

1-ANT

Set $\tau(e) := 1/2$ for all $e \in E$.
Construct a solution x^*.
Update pheromones w. r. t. x^*.
repeat
  Construct a solution x.
  if $f(x) \geq f(x^*)$ **then**
    $x^* := x$.
    Update pheromones w. r. t. x^*.

MMAS

Set $\tau(e) := 1/2$ for all $e \in E$.
Construct a solution x^*.
Update pheromones w. r. t. x^*.
repeat
  Construct a solution x.
  if $f(x) \geq f(x^*)$ **then** $x^* := x$.
  Update pheromones w. r. t. x^*.

MMAS*

Set $\tau(e) := 1/2$ for all $e \in E$.
Construct a solution x^*.
Update pheromones w. r. t. x^*.
repeat
  Construct a solution x.
  if $f(x) > f(x^*)$ **then** $x^* := x$.
  Update pheromones w. r. t. x^*.

(1+1) EA

Choose x^* uniformly at random.
repeat
  Create x by flipping each bit
    in x^* with probability $1/n$.
  if $f(x) \geq f(x^*)$ **then** $x^* := x$.

(1+1) EA*

Choose x^* uniformly at random.
repeat
  Create x by flipping each bit
    in x^* with probability $1/n$.
  if $f(x) > f(x^*)$ **then** $x^* := x$.

Figure 5.5: The algorithms considered in this section. The starred variants on the right-hand side use a strict selection while the left-hand side algorithms also accept equally fit solutions. The 1-ANT only updates pheromones in case the best-so-far solution x^* is replaced. Contrarily, MMAS updates pheromones in every iteration. If ρ is so large that the pheromone borders are hit in a single pheromone update, both the 1-ANT and MMAS collapse to the (1+1) EA and MMAS* collapses to the (1+1) EA*.

regardless whether it has been replaced or not. Hence in generations without improvements the pheromones are changed towards the best-so-far solution in MMAS* while the pheromones remain unchanged in the 1-ANT.

As mentioned in the introduction, Gutjahr (2007) conjectures that the different behavior of MMAS_{bs} and 1-ANT depends on the former difference: the fact that MMAS_{bs} only accepts strict improvements. Therefore, we also investigate a variant of MMAS_{bs} that accepts equally fit solutions. We call this algorithm MMAS, it is displayed on the left-hand side of Figure 5.5. In Section 5.2.2, we will analyze the runtime behavior of MMAS* and MMAS on simple unimodal functions. These results are then compared to known results for the 1-ANT.

ACO algorithms like MMAS* are not far away from evolutionary algorithms. If the value of ρ is chosen large enough in MMAS*, the pheromone borders $1/n$ or $1 - 1/n$ are touched for every bit. In this case, MMAS* becomes the same as the algorithm called (1+1) EA*, which is known from the analysis of evolutionary algorithms (Jansen and Wegener, 2001).

As already pointed out by Jansen and Wegener (2001), the (1+1) EA* has difficulties with simple plateaus of constant fitness as no search points of the same fitness as the best so far are accepted. Accepting solutions with equal fitness enables the algorithm to explore plateaus by random walks. Therefore, it seems more natural to replace search points by new solutions that are at least as good. In the case of evolutionary algorithms, this leads to the (1+1) EA which differs from the (1+1) EA* only in step 2.b) of the algorithm. For the sake of completeness, both the (1+1) EA* and the (1+1) EA are also shown in Figure 5.5. By essentially the same arguments, we also expect MMAS to outperform MMAS* on plateau functions. The corresponding runtime analyses are presented in Section 5.2.3.

5.2.2 Unimodal Functions, OneMax, and LeadingOnes

In the following, we derive upper bounds on the expected optimization time of MMAS* and MMAS on unimodal functions, especially OneMax and LeadingOnes. These results can then be compared to previous results for the 1-ANT. In contrast to the 1-ANT, there is a similarity between MMAS* and evolutionary algorithms that can be exploited to obtain good upper bounds. Suppose that during a run there is a phase during which MMAS* never replaces the best-so-far solution x^* in step 4.b) of the algorithm. This implies that the best-so-far solution is reinforced again and again until all pheromone values have reached their upper or lower borders corresponding to the setting of the bits in x^*. We can say that x^* has been "frozen in pheromone." The probability of creating a 1 for every bit is now either $1/n$ or $1-1/n$. The distribution of constructed solutions equals the distribution of offspring of the (1+1) EA* and (1+1) EA with x^* as the current search point. We conclude that, as soon as all pheromone values touch their upper or lower borders, MMAS* behaves like the (1+1) EA* until a solution with larger

fitness is encountered. This similarity between ACO and EAs can be used to transfer the fitness-level method (also called the method of f-based partitions) from Section 2.3.2 to ACO. In particular, upper bounds for MMAS* will be obtained from bounds for the (1+1) EA by adding the so-called freezing time described in the following. This correspondence has already been observed by Gutjahr and Sebastiani (2008). However, as they strive for a higher level of generality, their presentation is rather formal.

We give a simplified and more intuitive presentation of the approach followed in Gutjahr and Sebastiani (2008), adapted to MMAS* with our choice of pheromone bounds. In particular, we review some known results from Gutjahr and Sebastiani (2008) and show that specialized results for MMAS* can be proven in an easy and intuitive way. Our presentation follows the presentation of the fitness-level method for evolutionary algorithms. This enables us to highlight the similarities between EAs and ACO and it reveals a way to directly transfer runtime bounds known for EAs to MMAS*. This is an important step towards a unified theory of EAs and ACO.

A Simplified Review of the Fitness-Level Method for ACO

Suppose the current best-so-far solution x^* is not changed and consider the random time t^* until all pheromones reach their borders corresponding to the bit values in x^*. We will refer to this random time as *freezing time*. Gutjahr and Sebastiani (2008) bound t^* from above by $-\ln(n-1)/\ln(1-\rho)$. This holds since a pheromone value which is only increased during t steps by Lemma 5.1.2 is at least $\min\{1-1/n, 1-(1-1/n)(1-\rho)^t\}$ after the iterations, pessimistically assuming the worst-case starting value $1/n$. We use $\ln(1-\rho) \leq -\rho$ for $0 \leq \rho \leq 1$ and arrive at the handy upper bound

$$t^* \leq \frac{\ln n}{\rho}. \tag{5.7}$$

Following Gutjahr and Sebastiani (2008), we now use the freezing time t^* to derive a general upper bound on the expected optimization time of MMAS* by making use of the following restricted formulation of the fitness-level method. Let $f_1 < f_2 < \cdots < f_m$ be an enumeration of all fitness values and let A_i, $1 \leq i \leq m$, contain all search points with fitness f_i. In particular, A_m contains only optimal search points. Now, let s_i, $1 \leq i \leq m-1$, be a lower bound on the probability of the (1+1) EA (or, in this case equivalently, the (1+1) EA*) to create an offspring in $A_{i+1} \cup \cdots \cup A_m$, provided the current population belongs to A_i. The expected waiting time until such an offspring is created is at most $1/s_i$ and then the set A_i is left for good. As every set A_i has to be left at most once, the expected optimization time for the (1+1) EA and the (1+1) EA* is bounded above by

$$\sum_{i=1}^{m-1} \frac{1}{s_i}. \tag{5.8}$$

Consider t^* steps of MMAS* and assume $x^* \in A_i$. Either the best-so-far fitness increases during this period or all pheromone values are frozen. In the latter case, the probability of creating a solution in $A_{i+1} \cup \cdots \cup A_m$ is at least s_i and the expected time until the best-so-far fitness increases is at most $t^* + 1/s_i$. We arrive at the following upper bound for MMAS*:

$$\sum_{i=1}^{m-1} \left(t^* + \frac{1}{s_i} \right).$$

This is a special case of Inequality (13) in Gutjahr and Sebastiani (2008). Using $t^* \leq (\ln n)/\rho$, we obtain the more concrete bound

$$\frac{m \ln n}{\rho} + \sum_{i=1}^{m-1} \frac{1}{s_i}. \tag{5.9}$$

The right-hand sum is the upper bound obtained for the (1+1) EA and (1+1) EA* from (5.8). Applying the fitness-level method to MMAS*, we obtain upper bounds that are only by an additive term $(m \ln n)/\rho$ larger than the corresponding bounds for (1+1) EA and (1+1) EA*. This additional term results from the (pessimistic) assumption that on all fitness levels MMAS* has to wait until all pheromones are frozen in order to find a better solution. We will see examples where for large ρ this bound is of the same order of growth as the bound for (1+1) EA and (1+1) EA*. However, if ρ is very small, the bound for MMAS* typically grows large. This reflects the long time needed for MMAS* to move away from the initial random search and to focus on promising regions of the search space.

In the article by Gutjahr and Sebastiani (2008), the proposed fitness-level method is applied in the context of the unimodal functions OneMax and LeadingOnes. We re-prove specialized results for MMAS*. Our purpose is to demonstrate both power and elegance of the fitness-level method. Moreover, the following proofs suggest how to apply the method to other problems.

In the runtime analysis of the 1-ANT on OneMax by Neumann and Witt (2006), it is shown that there exists a threshold value for ρ (in our notation basically $\rho = O(1/n^\varepsilon)$ for some small constant $\varepsilon > 0$) below which no polynomial runtime is possible. In Gutjahr and Sebastiani (2008) it is shown that such a phase transition does not occur with MMAS*. The following theorem has already been proven as Proposition 5.1 in Gutjahr and Sebastiani (2008) with a more general parametrization for the pheromone borders. We present a proof for MMAS* using our simplified presentation of the fitness-level method.

Theorem 5.2.1. *The expected optimization time of* MMAS* *on* OneMax *is bounded from above by* $O((n \log n)/\rho)$.

Proof. The proof is an application of the above-described fitness-level method with respect to the fitness-level sets $A_i = \{x \mid f(x) = i\}$, $0 \leq i \leq n$. On level A_i, a sufficient condition to increase the fitness is to flip a 0-bit and not to flip the other $n-1$ bits. For a specific 0-bit, this probability is $1/n \cdot (1-1/n)^{n-1} \geq 1/(en)$. As the events for all $n-i$ 0-bits are disjoint, $s_i \geq (n-i)/(en)$ and we obtain the bound

$$\sum_{i=0}^{n-1} \frac{en}{n-i} \;=\; en \sum_{i=1}^{n} \frac{1}{i} \;=\; O(n \log n).$$

Using (5.9), the upper bound $O((n \log n)/\rho)$ follows. □

Recall that the function LeadingOnes counts the number of leading ones in the considered bit string. A non-optimal solution may always be improved by appending a single one to the leading ones. Section 5.1 has shown for the 1-ANT on LeadingOnes that a similar phase transition behavior as on OneMax exists: for $\rho = o(1/\log n)$, the expected optimization time of the 1-ANT is superpolynomially large whereas it is polynomial for $\rho = \Omega(1/\log n)$ and even only $O(n^2)$ for $\rho = \Omega(1)$. Proposition 7.1 in Gutjahr and Sebastiani (2008) proves that this phase transition cannot occur with MMAS* on LeadingOnes. We again present a proof for MMAS* using our simplified presentation of the fitness-level method.

Theorem 5.2.2. *The expected optimization time of* MMAS* *on* LeadingOnes *is bounded from above by* $O(n^2 + (n \log n)/\rho)$.

Proof. For $0 \leq i < n$ the (1+1) EA adds an $(i+1)$-st bit to the i leading ones of the current solution with probability $s_i = (1-1/n)^i \cdot 1/n \geq 1/(en)$. Using the bound (5.9) results in the upper bound $((n+1)\ln n)/\rho + en^2 = O(n^2 + (n \log n)/\rho)$. □

In the following, we apply the method to arbitrary unimodal functions. We also extend the method to yield upper bounds for MMAS on unimodal functions. This extension is not immediate as MMAS may switch between solutions of equal fitness, which may prevent the pheromones from freezing. Moreover, we present lower bounds on the expected optimization time of MMAS* on all functions with unique optimum and an improved specialized lower bound for LeadingOnes. On one hand, these bounds allow us to conclude that the fitness-level method can provide almost tight upper bounds. On the other hand, as can be seen from a more detailed analysis in Section 5.2.2, the method still leaves room for improvement using specialized techniques.

Upper Bounds for Unimodal Functions

Unimodal functions are an important and well-studied class of fitness functions in the literature on evolutionary computation. Recall from Definition 2.2.1 that a function is unimodal if every non-optimal search point has a Hamming neighbor with strictly larger

fitness. Unimodal functions are often believed to be easy to optimize. This holds if the set of different fitness values is not too large. On the other hand, Droste et al. (2006) proved for classes of unimodal functions with many fitness values that *every* black-box algorithm needs exponential time on average.

We consider unimodal functions attaining d different fitness values for arbitrary $d \in \mathbb{N}$. Such a function is optimized by the (1+1) EA and (1+1) EA* in expected time $O(nd)$ (cf. Theorem 2.3.4). This bound is transferred to MMAS* by the following theorem.

Theorem 5.2.3. *The expected optimization time of* MMAS* *on a unimodal function attaining d different fitness values is $O((n + (\log n)/\rho)d)$.*

Proof. Because of the unimodality there is for each current search point x a better Hamming neighbor x' of x in a higher fitness-level set. The probability for the (1+1) EA (or, equivalently, MMAS* with all pheromone values at a border) to produce x' in the next step is at least $1/(en)$. By (5.9), this completes the proof. □

In order to freeze pheromones after t^* steps without an improvement, it is essential that equally good solutions are rejected. The fitness-level argumentation, including the bound from (5.9), cannot directly be transferred to MMAS as switching between equally fit solutions can prevent the system from freezing. Nevertheless, we are able to prove a similar upper bound on the optimization time of MMAS that is by a factor of n^2 worse than the bound for MMAS* in Theorem 5.2.3 if $\rho = O((\log n)/n)$. Despite the factor n^2, Theorem 5.2.4 yields a polynomial bound for MMAS if and only if Theorem 5.2.3 yields a polynomial bound for MMAS*.

Theorem 5.2.4. *The expected optimization time of* MMAS *on a unimodal function attaining d different fitness values is $O(((n^2 \log n)/\rho)d)$.*

Proof. We only need to show that the expected time for an improvement of the best-so-far solution is at most $O((n^2 \log n)/\rho)$. The probability that MMAS produces within $O((\log n)/\rho)$ steps a solution being at least as good as (not necessarily better than) the best-so-far solution x^* is $\Omega(1)$ since after at most $(\ln n)/\rho$ steps without exchanging x^* all pheromone values have touched their borders and then the probability of rediscovering x^* is $(1 - 1/n)^n = \Omega(1)$. We now show that the conditional probability of an improvement if x^* is replaced is $\Omega(1/n^2)$.

Let $x_1, \ldots, x_m$ be an enumeration of all solutions with fitness values equal to the best-so-far fitness value. Because of the unimodality, each x_i, $1 \leq i \leq m$, has some better Hamming neighbor y_i; however, the y_i need not be disjoint. Let X and Y denote the event to generate some x_i or some y_i, respectively, in the next step. In the worst case $y_1, \ldots, y_m$ are the only possible improvements, hence the theorem follows if we can show $\text{Prob}(Y \mid X \cup Y) \geq 1/n^2$, which is implied by $\text{Prob}(Y) \geq \text{Prob}(X)/(n^2 - 1)$.

If $p(x_i)$ is the probability of constructing x_i, we have $p(x_i)/p(y_i) \leq (1 - \frac{1}{n})/\frac{1}{n} = n - 1$ as the constructions only differ in one bit. Each y_i may appear up to n times in the

sequence $y_1, \ldots, y_m$, hence $\text{Prob}(Y) \geq \frac{1}{n} \sum_{i=1}^{m} p(y_i)$ and

$$\text{Prob}(X) \;=\; \sum_{i=1}^{m} p(x_i) \;\leq\; (n-1) \cdot \sum_{i=1}^{m} p(y_i) \;\leq\; n(n-1) \cdot \text{Prob}(Y).$$

Therefore, $\text{Prob}(Y) \geq \text{Prob}(X)/(n^2-1)$ follows. □

Theorems 5.2.3 and 5.2.4 show that the expected optimization times of both MMAS and MMAS* are polynomial for all unimodal functions as long as $d = \text{poly}(n)$ and $\rho = 1/\text{poly}(n)$. The result for MMAS has extensive implications.

Recall that MMAS only differs from MMAS* by accepting equally good solutions. The only difference between MMAS and the 1-ANT lies in the pheromone update mechanism: MMAS reinforces the best-so-far solution in every step, while the 1-ANT only performs a pheromone update in case the best-so-far solution is exchanged. The 1-ANT suffers from a phase transition behavior with exponential runtimes for simple unimodal functions if ρ is a small inverse polynomial. However, Theorems 5.2.3 and 5.2.4 prove that such a phase transition can occur neither with MMAS* nor with MMAS. Therefore, the phase transition behavior of the 1-ANT must result from the different pheromone update mechanism, contrary to the conjecture in Gutjahr (2007).

A General Lower Bound

The function OneMax is probably the simplest function with unique global optimum. The upper bound $O((n \log n)/\rho)$ from Theorem 5.2.1 is never better than $\Theta(n \log n)$, which describes the expected runtime of the (1+1) EA and (1+1) EA* on OneMax. At the moment, we are not able to show a matching lower bound $\Omega(n \log n)$ on the expected optimization time of MMAS*; however, we can show that the expected optimization time is growing with respect to $1/\rho$ as the upper bound suggests. We state our result in a more general framework: as known from the considerations by Droste et al. (2002a), the mutation probability $1/n$ of the (1+1) EA is optimal for many functions including OneMax. One argument is that the probability mass has to be quite concentrated around the best-so-far solution to allow the (1+1) EA to discover Hamming neighbors of the last accepted solution with good probability. Hence, the best-so-far solution needs to be "rediscovered" with good probability. Given a mutation probability of $\alpha(n)$, the probability of rediscovery equals $(1-\alpha(n))^n$, which converges to zero unless $\alpha(n) = O(1/n)$. The following lemma exploits the last observation for a general lower bound on the expected optimization time of both MMAS and MMAS*.

Theorem 5.2.5. *Let $f\colon \{0,1\}^n \to \mathbb{R}$ be a function with a unique global optimum. Choosing $\rho = 1/\text{poly}(n)$, the expected optimization time of* MMAS *and* MMAS* *on f is $\Omega((\log n)/\rho - \log n)$.*

Proof. W. l. o. g., 1^n is the unique optimum. If, for each bit, the success probability (defined as the probability of creating a 1) is bounded from above by $1 - 1/\sqrt{n}$ then the solution 1^n is created with only an exponentially small probability of at most $(1 - 1/\sqrt{n})^n \leq e^{-\sqrt{n}}$. Using the uniform initialization and Lemma 5.1.2, the success probability of a bit after t steps is bounded from above by $1 - (1 - \rho)^t/2$. Hence, all success probabilities are bounded as desired within $t := (1/\rho - 1) \cdot (\ln(n/4)/2) = \Omega((\log n)/\rho - \log n)$ steps since

$$1 - \frac{1}{2}(1-\rho)^t \;\leq\; 1 - \frac{e^{-(\ln n - \ln 4)/2}}{2} \;=\; 1 - \frac{1}{\sqrt{n}}.$$

Since $\rho = 1/\text{poly}(n)$ and, therefore $t = \text{poly}(n)$, the total probability of creating the optimum in t steps is still at most $te^{-\sqrt{n}} = e^{-\Omega(\sqrt{n})}$, implying the lower bound on the expected optimization time. □

Hence, the expected optimization time of MMAS* with $\rho = 1 - \Omega(1)$ on OneMax is bounded by $\Omega((\log n)/\rho)$. It is an open problem to show matching upper and lower bounds. We conjecture that the lower bound for OneMax is far from optimal and that $\Omega(n \log n + n/(\rho \log(2/\rho)))$ holds. A corresponding bound for LeadingOnes will be given in the following.

Improved Upper and Lower Bounds for LeadingOnes

We improve the bound of Theorem 5.2.2 to $O(n^2 + n/\rho)$ and beyond. Thereby, we show that it is not necessary for MMAS* to freeze the best-so-far solution in pheromone on every fitness level. Moreover, the forthcoming Theorem 5.2.6 applies to MMAS as well, where a straight application of the fitness-level method breaks down.

In the remainder of this section, it is crucial to distinguish two states for bits during the optimization process on LeadingOnes, as already done in Section 5.1. Suppose that the current best-so-far LeadingOnes-value equals k. We call the bits $1, \ldots, k$ *bits in increasing state* and the bits $k + 2, \ldots, n$ *bits in random state*. No special attention is paid to the 0-bit at position $k + 1$.

Our notions are justified as follows: since, due to the structure of LeadingOnes, all future best-so-far solutions must contain at least k leading ones, the pheromone values of the 1-edges (i. e., the success probabilities) corresponding to the first k bits are monotonically increasing in each iteration until the border $1 - 1/n$ is reached. The bits $k+2, \ldots, n$ have never had an impact on the LeadingOnes-value. Due to the symmetry of the construction procedure, it is intuitively clear that such a bit is unbiasedly set to 1 with probability 1/2 in the next constructed solution, hence the bit is purely random. A formal proof of this will be given later.

We state two improved upper bounds for LeadingOnes. The first bound is better than the bound from Theorem 5.2.2 by a factor of order $\log n$ if $\rho = O(1/n)$. The second bound is better than the first one by another factor of order $\log n$ if $\rho \leq n^{-(1+\Omega(1))}$.

Theorem 5.2.6. *The expected optimization time of* MMAS *and* MMAS* *on* LeadingOnes *is bounded by* $O(n^2 + n/\rho)$ *and* $O\left(n^2 \cdot (1/\rho)^\varepsilon + \frac{n/\rho}{\log(1/\rho)}\right)$ *for every constant* $\varepsilon > 0$.

Proof. Let ℓ be a positive real value defined later. Divide a run into phases. Phase i, $0 \le i \le n$, ends when the following two conditions are met:

1. the best-so-far LeadingOnes-value is at least i and
2. for $0 \le j \le i-1$ it holds that bit $i-j$ has been reinforced at least $(j+1)\ell$ times after entering the increasing state.

We now show that the expected time spent in Phase i, $1 \le i < n$, is bounded by $O(\ell + n \cdot e^{5/(\ell\rho)})$. This implies that the expected optimization time is bounded by $O(n\ell + n^2 \cdot e^{5/(\ell\rho)})$. Choosing $\ell := 5/\rho$ yields the first bound and $\ell := 5/(\varepsilon\rho\ln(1/\rho))$ yields the second bound.

Consider the first generation in Phase i for $1 \le i < n$. As Phase $i-1$ has been finished successfully, the first $i-1$ bits are in increasing state and bit $i-j$, $1 \le j \le i-1$, has been reinforced at least $j\ell$ times. This implies that the success probability for this bit is at least $\min\{1-1/n, 1-(1-\rho)^{j\ell}\} \ge (1-1/n)\cdot(1-(1-\rho)^{j\ell})$. The success probability for bit i is at least $1/n$, hence the probability of creating a solution with at least i leading ones is at least

$$\left(1-\frac{1}{n}\right)^{i-1} \cdot \frac{1}{n} \cdot \prod_{j=1}^{i-1}\left(1-(1-\rho)^{j\ell}\right) \;\ge\; \frac{1}{en} \cdot \prod_{j=1}^{\infty}\left(1-(1-\rho)^{j\ell}\right).$$

We estimate the $\prod$-term on the right-hand side and deal with the first factors separately. Define $m := \max\{j \in \mathbb{N}_0 \mid (1-(1-\rho)^{j\ell}) \le 1/2\}$ such that the factors with $j \le m$ are at most $1/2$. We estimate these terms from below by a linear function as follows. By induction, we show that $(1-(1-\rho)^k) \le 1/2$ implies $(1-(1-\rho)^k) \ge k\rho/2$ for $k \in \mathbb{N}_0$. This claim is obvious for $k = 0$. Assume that it holds for $k-1$, then

$$\begin{aligned} 1-(1-\rho)^k \;&=\; 1-(1-\rho)^{k-1}(1-\rho) \;=\; 1-(1-\rho)^{k-1}+\rho(1-\rho)^{k-1} \\ &\ge\; 1-(1-\rho)^{k-1}+\frac{\rho}{2} \;\ge\; \frac{k\rho}{2}. \end{aligned}$$

Hence the j-th factor, $j \le m$, is bounded from below by $j\ell\rho/2$. Solving the equation $1-(1-\rho)^x = 1/2$ for x yields the bound $m\ell \le -(\ln 2)/\ln(1-\rho) \le (\ln 2)/\rho$. Let $\alpha := (\ln 2)/(\ell\rho) \ge m$ and assume w. l. o. g. $\alpha \in \mathbb{N}$, then the product of the first m factors is at least

$$\begin{aligned} \prod_{j=1}^{m}\left(1-(1-\rho)^{j\ell}\right) \;&\ge\; \prod_{j=1}^{m}\frac{j\ell\rho}{2} \;\ge\; \prod_{j=1}^{\alpha}\frac{j\ell\rho}{2} \;=\; \alpha!\cdot\left(\frac{\ell\rho}{2}\right)^{\alpha} \\ &\ge\; \left(\frac{\alpha}{e}\right)^{\alpha}\cdot\left(\frac{\ell\rho}{2}\right)^{\alpha} \;=\; \left(\frac{\ln 2}{2e}\right)^{(\ln 2)/(\ell\rho)} \;\ge\; e^{-1/(\ell\rho)}. \end{aligned}$$

All factors with index $j > m$ are at least $1/2$. They can be estimated as follows. We exploit $1 + x \leq e^x$ for $x \in \mathbb{R}$ and $1 - x \geq e^{-2x}$ for $0 \leq x \leq 1/2$ (cf. Lemma A.12).

$$\prod_{j=m+1}^{\infty} \left(1 - (1-\rho)^{j\ell}\right) \geq \prod_{j=m+1}^{\infty} \exp\left(-2(1-\rho)^{j\ell}\right)$$
$$\geq \exp\left(-2\sum_{j=0}^{\infty}(1-\rho)^{j\ell}\right) = \exp\left(-\frac{2}{1-(1-\rho)^{\ell}}\right) \geq \exp\left(-\frac{2}{1-e^{-\ell\rho}}\right).$$

If $\ell\rho > 1$, the resulting bound is $\Omega(1)$. If $\ell\rho \leq 1$, then $1 - e^{-\ell\rho} \geq \ell\rho/2$ and we obtain the bound $\exp(-4/(\ell\rho))$. Together, the probability of creating a solution with at least i leading ones is $\Omega(1/n \cdot e^{-5/(\ell\rho)})$ and the expected waiting time for this event is $O(n \cdot e^{5/(\ell\rho)})$. This fulfills the first goal of Phase i. After an additional waiting time of ℓ steps the second goal is fulfilled as well as the first i bits are reinforced in every step and bit $i - j$, $1 \leq j \leq i - 1$, has already been reinforced at least $j\ell$ times before entering Phase i. Therefore, Phase i is completed in expected time $O(\ell + n \cdot e^{5/(\ell\rho)})$ as claimed. □

It is interesting that an almost tight lower bound can be derived. The following theorem shows that the expected optimization time of MMAS* on LeadingOnes is $\Omega(n^2 + n/(\rho\log(2/\rho)))$, hence never better than $\Omega(n^2)$. (We write $\log(2/\rho)$ in the lower bound instead of $\log(1/\rho)$ to make the bound $\Omega(n^2)$ for every constant ρ and to avoid division by 0.) Apart from this technical detail, the lower bound is tight with the upper bounds from Theorem 5.2.6 for $\rho = \Omega(1/n)$ and $\rho \leq n^{-(1+\Omega(1))}$, hence for almost all ρ. The proof is lengthy, however, for the case of large ρ, one essential idea is easy to grasp: already in the early stages of the optimization process, many, more precisely $\Omega(n)$, success probabilities reach their lower borders $1/n$, and the corresponding bits are set to 0. To "flip" such a bit, events of probability $1/n$ are necessary. This can be transformed into the lower bound $\Omega(n^2)$ on the expected optimization time.

Theorem 5.2.7. *Choosing $\rho = 1/\mathrm{poly}(n)$, the expected optimization time of* MMAS* *on* LeadingOnes *is bounded from below by $\Omega\left(n^2 + \frac{n/\rho}{\log(2/\rho)}\right)$.*

To prove the preceding theorem, we recall the distinction of bits in increasing and random state. We have already motivated why the latter bits are set to 1 with probability exactly $1/2$ even if the corresponding pheromone values on 1-edges might change from step to step. This is equivalent to a random, time-dependent success probability. The probability of setting the bit to 1 corresponds to the expected success probability.

Lemma 5.2.8. *Let the random variable P_t denote the probability of setting a bit in random state to 1 at time t. Then the bit is set to 1 at time t with probability $\mathrm{E}(P_t)$.*

Proof. We first observe that the underlying state space is countably infinite. This holds since the initial pheromone value is fixed to 1/2 and each time step allows at most two new, i. e., previously unvisited, pheromone values.

Conditioned on the fact that the random P_t equals p, the bit is set to 1 at time t with probability p. By the law of total probability, the unconditional probability of setting the bit to 1 equals

$$\sum_p p \cdot \text{Prob}(P_t = p) \;=\; \text{E}(P_t).$$

□

If we can prove that $\text{E}(P_t) = 1/2$ holds for all t where the bit is in random state, our analysis of the random-state bits is complete.

Lemma 5.2.9. *Let P_t be as defined in Lemma 5.2.8. Then for $0 \leq p \leq 1$ it holds $\text{Prob}(P_t = p) = \text{Prob}(P_t = 1 - p)$, hence $\text{E}(P_t) = 1/2$. This even holds if the bit is no longer in random state at time $t + 1$.*

Proof. We inductively show for all $t' \leq t$ the following stronger statement: for each vector of probabilities $(p_1, \ldots, p_{t'})$ it holds

$$\text{Prob}((P_1, \ldots, P_{t'}) = (p_1, \ldots, p_{t'})) \;=\; \text{Prob}((P_1, \ldots, P_{t'}) = (1 - p_1, \ldots, 1 - p_{t'}))$$

This symmetry will imply the lemma since

$$\begin{aligned} \text{E}(P_t) \;&=\; \sum_{0 \leq p \leq 1} p \cdot \text{Prob}(P_t = p) \\ &=\; \sum_{0 \leq p < 1/2} (p + (1 - p)) \cdot \text{Prob}(P_t = p) + \frac{1}{2} \cdot \text{Prob}(P_t = 1/2) \;=\; \frac{1}{2}. \end{aligned}$$

Due to the pheromone initialization, we have $P_0 = 1/2$ with probability 1. For the induction step, let p be arbitrary but fixed. Conditioned on $P_t = p$, there are two possible pheromone values on the 1-edges of the bit at time $t + 1$, or, equivalently, for the random P_{t+1}: either $d(p) := \max\{1/n, (1-\rho)p\}$ or $i(p) := \min\{1-1/n, (1-\rho)p+\rho\}$. Depending on whether the best-so-far solution is exchanged in the considered step, the probability of going from p to $d(p)$ (or $i(p)$) takes one of the three values 0, $1 - p'$ and 1 (or 0, p', 1). Actually, given that the best-so-far solution is not exchanged, the probability of increasing or decreasing the value is itself random and depends on the bit's setting in the last best-so-far solution. However, which of the three probabilities is used is independent of the random P_t even if the bit happens to leave the random state in the considered step. The event that a best-so-far solution with k ones is replaced is *equivalent* to the event that the first $k + 1$ bits are all set to 1, hence the setting of the bits $k + 2, \ldots, n$ is independent of this decision. Note that even in the step where

such a bit leaves the random state, it does not affect the decision whether to replace the best-so-far solution.

We are left with two cases. If an exchange of the best-so-far solution happens from time t to $t+1$ then, given that $P_t = p$, P_{t+1} takes the value $d(p)$ with probability $1-p$ and the value $i(p)$ with probability p. Conditioned on the event $P_t = 1-p$, P_{t+1} takes the value $d(1-p)$ with probability p and the value $i(1-p)$ with probability $1-p$. By the induction hypothesis, both events that we condition on are equiprobable. Moreover, $p \to 1-p$ is a bijection on the probability space for P_t. Noting that $d(1-p) = 1-i(p)$ and $i(1-p) = 1-d(p)$ and using the law of total probability, we have $\text{Prob}(P_{t+1} = i(p)) = \text{Prob}(P_{t+1} = 1-i(p))$, and, accordingly, $\text{Prob}(P_{t+1} = d(p)) = \text{Prob}(P_{t+1} = 1-d(p))$.

If no exchange of the best-so-far solution happens then P_t is increased or decreased with probability P_{t^*} and $1-P_{t^*}$, respectively, where $t^* < t$ is the time of the last exchange (or 0 for the initial step). We additionally consider an arbitrary fixed p^*. Conditioned on the event $(P_{t^*} = p^*) \wedge (P_t = p)$, we have $\text{Prob}(P_{t+1}) = i(p) = p^*$ and $\text{Prob}(P_{t+1} = d(p)) = 1-p^*$. Analogously, if $(P_{t^*} = 1-p^*) \wedge (P_t = 1-p)$, we have $\text{Prob}(P_{t+1} = i(1-p)) = 1-p^*$ and $\text{Prob}(P_{t+1} = d(1-p)) = p^*$. Using the induction hypothesis, both conditions are again equiprobable. Now $(p^*, p) \to (1-p^*, 1-p)$ is a bijection on the probability space for (P_{t^*}, P_t). By the law of total probability, we have, also in this case, $\text{Prob}(P_{t+1} = i(p)) = \text{Prob}(P_{t+1} = 1-i(p))$ and $\text{Prob}(P_{t+1} = d(p)) = \text{Prob}(P_{t+1} = 1-d(p))$. Altogether, the induction follows. □

We summarize what has been derived so far: the bits in random state are independently set to 1/2 in all constructed solutions until they leave the random state, where the independence follows directly from the construction procedure of MMAS*. This will allow us to treat the random bits in the same way as done in the analysis of the (1+1) EA on LeadingOnes by Droste et al. (2002a). To prove a lower bound on the expected optimization time, we have to take into account the "free rider" phenomenon, i. e., additional bits following the leftmost 0-bit that are collected "for free" in an improvement. To prove a lower bound, we have to control the number of free riders in improving steps. By the above analysis, the probability of i free riders is at most 2^{-i-1} per improvement since having i free riders is equivalent to setting bits $k+2, \ldots, k+i$ to 1 and setting bit $k+i+1$ to 0. Clearly, if $k+i > n$, having i free riders is even impossible. Altogether, the number of free riders in improving steps is now easy to control.

Lemma 5.2.10. *After the first $n/12$ improvements the* LeadingOnes*-value of the best-so-far solution is still less than $n/2$ with probability $1-2^{-\Omega(n)}$.*

Proof. With probability at least $1-2^{-n/4}$ the initial LeadingOnes-value is at most $n/4$. Working under this condition, we estimate the number of free riders in improving steps. Modeling the free rider decisions as at most $n/2$ independent trials as in Droste et al. (2002a), the number of free riders in k improvements is at most $2k$ with probability at

least $1 - e^{-k/16}$. Hence, the probability that $n/12$ improvements increase the LEADING-ONES-value by at least $n/4$ is $2^{-\Omega(n)}$. □

The main obstacle in the proof of Theorem 5.2.7 is related to the success probabilities of bits that leave the random state to become the leftmost zero-bit. For the lower bound $\Omega(n^2)$ given in the theorem, it would be nice to have many times, at best $\Omega(n)$ times, bits whose random success probabilities equal the lower border $1/n$ when they become the leftmost zero-bit. Then setting the bit to 1, as necessary for an improvement, would require $\Omega(n)$ steps in expectation, and $\Omega(n)$ such situations would take an expected number of $\Omega(n^2)$ steps. With strict selection a weaker condition is sufficient. For MMAS* a lower bound $\Omega(n^2)$ follows if $\Omega(n)$ bits reach success probabilities at the lower border at least once while being in random state.

Lemma 5.2.11. *Consider a point of time where the best-so-far* LEADINGONES*-value is less than $n/2$. If there are $\Omega(n)$ bits at the last $n/2$ positions whose success probabilities have reached their lower border $1/n$ at least once up to now then the remaining expected optimization time for* MMAS* *is $\Omega(n^2)$.*

Proof. The last $n/2$ bits are all in random state. Suppose a bit in random state has reached the lower border $1/n$ on the success probability. For each subsequent improvement, it can become necessary to set such a bit to 1, which has probability $1/n$ provided that the success probability is still at the border. However, the success probability of such a bit might increase again. Therefore, in each of the remaining improvements, we distinguish the events whether setting the bit to 1 is relevant for an improvement or not. If the bit is not relevant since it is still right of the leftmost zero after the improvement, its success probability does not change with probability $1 - 1/n$. Hence, by Chernoff bounds (Lemma A.6), it holds with probability $1 - 2^{-\Omega(n)}$ that after $O(n)$ improvements (note that there are at most n improvements) there are still $\Omega(n)$ success probabilities equal to $1/n$ left. We assume this to happen for some number cn, $c > 0$ a constant, of bits and call these bits *difficult*. Pessimistically ignoring all other bits, an improvement adds an expected number of at most $1 + 1/n < 2$ difficult bits to the leading ones of the best-so-far solution. The expected number of difficult bits gained in $cn/4$ improvements is therefore less than $cn/2$, and this number is less than cn with probability at least $1/2$. Hence, the remaining optimization time is $\Omega(n^2)$ with probability at least $1/2 - 2^{-\Omega(n)}$, and, therefore, also in expectation. □

In order to complete the proof of the lower bound $\Omega(n^2)$ we have to show that the preconditions of Lemma 5.2.11 are fulfilled with high probability and many bits in random state reach the lower border for their success probability. However, so far we only know that in expectation, at least half of the bits in random state have a success probability that is bounded from above by $1/2$. The probability, however, might still be very close to $1/2$, which is the actual value in the initial step.

Recall that in generations where the best-so-far solution x^* is not replaced the success probability at every bit is continuously increased or decreased as the algorithm continuously focuses on x^*. Intuitively, chances to reach upper or lower borders are high if some time elapses between improving steps. The following lemma bounds the average time between improvements from below. If all success probabilities are close to 1/2, then MMAS* behaves almost like random search. Hence, to obtain substantial progress in the optimization, it must take some time between two improvements for the pheromone values of the newly gained bits in increasing state to come up and reach their upper border $1-1/n$. In the following, we will bound the average time between improving steps by $\Omega(1/(\rho\log(2/\rho)))$, where the average is taken over $\Theta(\log(2/\rho))$ improvements. Thereby, we also prove a lower bound $\Omega(n/(\rho\log(2/\rho)))$ for the expected optimization time on LeadingOnes, corresponding to the second term in the bound from Theorem 5.2.7.

Lemma 5.2.12. *Let $\rho = 1/poly(n)$. Then there is some constant $c > 0$ such that after each step improving the* LeadingOnes*-value, the number of steps required by* MMAS* *for $c\log(2/\rho)$ further improvements is $\Omega(1/\rho)$ with probability $\Omega(1)$. Furthermore, the expected optimization time is $\Omega(n/(\rho\log(2/\rho)))$.*

Proof. W. l. o. g., we assume $1/\rho$ to be growing with n since otherwise the first statement of the theorem is trivial and the second one follows from Lemma 5.2.10.

We will show that a phase of $s := 1/\rho - 1$ steps following an arbitrary improving step contains an expected number of at most $O(\log(2/\rho))$ further improvements. This will imply the lemma for the following reasons. First, we apply Markov's inequality on the random number of improvements in s steps, which is therefore bounded by $O(\log(2/\rho))$ with probability at least 1/2. This is the first statement of the lemma. Second, we sum up the expected number of improvements in $c'n/\log(2/\rho)$ phases of length s, where $c' > 0$ is another constant. This yields a total number of $c'n/(\rho\log(2/\rho)) - c'n/\log(2/\rho) = \Omega(n/(\rho\log(2/\rho)))$ steps. If c' is small enough, the total expected number of improvements in this number of steps is less than $n/24$ and, by Markov's inequality, less than $n/12$ with probability at least 1/2. By Lemma 5.2.10, with probability $1 - 2^{-\Omega(n)}$, $n/12$ improvements are not enough to reach even a LeadingOnes-value of at least $n/2$. The sum of the failure probabilities is $1/2 + 2^{-\Omega(n)}$. Using the law of total probability, we obtain the second statement of the lemma.

We are left with the claim. Let k be the LeadingOnes-value at the beginning of the phase of length s. For some large enough variable γ to be chosen later, we concentrate on the block of $r := r(\gamma) := \gamma\log(2/\rho)$ bits at positions $k+2, \ldots, k+2+r-1$, i. e., following the leftmost 0-bit at the beginning of the phase. If all r bits are in increasing state by the end of the phase, the phase is called successful. Clearly, if the phase is unsuccessful, it contains, due to strict selection, at most r improvements, which is the event whose probability has to be bounded from above.

In the beginning of the phase, all bits of the block are in random state. We apply Lemma 5.2.9 to bound the number of block bits whose success probability is at most 1/2

in the beginning of a phase. The probability is at least 1/2 for a single bit, and by Chernoff bounds, with a failure probability of most $2^{-r/12}$, less than $r/4$ success probabilities are at most 1/2. We assume $r/4$ bits with this property and estimate the probability of setting all these bits to 1 simultaneously in at least one improving step by the end of the phase (which is necessary for the phase to be successful). The success probability of a bit with initial pheromone value 1/2 is still at most $1-(1-\rho)^t/2$ if it has been only in increasing state for t steps. The total number of iterations in the phase is $1/\rho - 1$. Hence, by the end of the phase, all considered success probabilities are at most $1-(1-\rho)^{1/\rho-1}/2 \leq 1-e^{-1}/2 < 0.82$. The probability of a single improving step setting the $r/4$ bits to 1 is therefore at most $(0.82)^{r/4} \leq 2^{-r/14}$. By the union bound, the probability of this happening within s steps is at most $s2^{-r/14} = 2^{\log(2/\rho)-\gamma\log(2/\rho)} \leq 2^{-r/15}$ if γ is large enough. More precisely, there is a constant γ_0 such that the statement holds for all $\gamma \geq \gamma_0$. Additionally taking into account the above failure probability, the probability of the phase being successful is at most $2^{-r/12} + 2^{-r/15} \leq 2^{-r/20}$ if γ is large enough.

Using the last observation, we finally bound the expected number of improvements per phase. Let I denote the random number of improvements. Then we have $\mathrm{E}(I) = \sum_{v\geq 1} \mathrm{Prob}(I \geq v) \leq \sum_{i=0}^{\infty} r_0 \mathrm{Prob}(I \geq ir_0)$, where now $r_0 := \gamma_0 \log(2/\rho)$ with γ_0 as in the last paragraph. In the definition of r, we can choose γ arbitrarily large. Applying the result of the last paragraph for $\gamma := i\gamma_0$, we obtain that $\mathrm{Prob}(I \geq i\gamma_0 \log(2/\rho))$ is at most $2^{-i\gamma_0 \log(2/\rho)/20}$ for all $i \geq 1$. Since $2^{-\gamma_0 \log(2/\rho)/20} = 1 - \Omega(1)$, we have $\sum_{i=1}^{\infty} 2^{-i\gamma_0 \log(2/\rho)/20} = O(1)$, and, altogether, $\mathrm{E}(I) \leq r\mathrm{Prob}(I \geq 0) + r \cdot O(1) = O(r) = O(\log(2/\rho))$. □

We remark that Lemma 5.2.12 does not contain a statement on the expected time between two improvements. Rather, an amortized consideration over $\Omega(\log(2/\rho))$ improvements is given. The reason is that the very first improvements are easy to obtain even for random search.

Now we are left with the proof that $\Omega(n)$ bits in random state reach their lower border on the success probabilities during the first $n/12$ improvements. To this end, we consider a bit in random state and again denote by P_t its random success probability at time t. All the following estimations hold until the bit leaves the random state.

We are interested in the stopping time $T_{\min} := \min\{t \geq 0 \mid P_t = 1/n\}$. Before this time, the stochastic process describing the P_t-value equals the same stochastic process with the lower border $1/n$ on the success probability removed (i. e., $P_{t+1} = \min\{1-1/n, (1-\rho)P_t + \rho\}$ if P_t is increased and $P_{t+1} = (1-\rho)P_t$ otherwise. In the following, we only consider this modified process, however, still denote it by P_t, $t \geq 0$.

The P_t-process is not necessarily Markovian. Recalling the reasoning in the proof of Lemma 5.2.9, P_{t+1} is obtained either by making a binary decision with probability P_t or by increasing or decreasing the value according to the result of an earlier decision when the best-so-far solution is not exchanged while going from time t to time $t+1$. Let

the random times T_i, $i \geq 0$, describe the points of time where the best-so-far solution is exchanged; in particular, $T_0 = 0$ denotes the initialization of the best-so-far solution. Since the bit describing the P-process is assumed to stay in random state, the lengths $L_i := T_{i+1} - T_i$ of the time spans between exchanges are independent of the P_t-values. We consider the process $M_i := P_{T_i}$, $i \geq 0$, which is just a macroscopic view of the P-process at the exchange points. Since exactly the behavior at the exchange points is relevant for the decision whether to increase or decrease the pheromone, the M-process is Markovian. Furthermore, it can be characterized as follows.

Lemma 5.2.13. *The M_i form a supermartingale, i. e.,* $\mathrm{E}(M_{i+1} \mid M_0, \ldots, M_i) \leq M_i$.

Proof. Since the process is Markovian, it suffices to condition on M_i. Given $M_i = p$, the value is increased L_i times with probability p and otherwise decreased L_i times. Hence, by the definition of the M-process,

$$\begin{aligned}
&\mathrm{E}(M_{i+1} \mid M_i = p) \\
&\quad = \; (1-p) \cdot \left(M_i \cdot (1-\rho)^{L_i}\right) + p \cdot \min\left\{1 - \frac{1}{n}, 1 - (1-M_i) \cdot (1-\rho)^{L_i}\right\} \\
&\quad \leq \; (1-p) \cdot p \cdot (1-\rho)^{L_i} + p - (1-p) \cdot p \cdot (1-\rho)^{L_i} \;=\; p \;=\; M_i.
\end{aligned}$$

□

We are still looking for a bound on $T_{\min}$. This will be obtained from the stopping time $T_{\text{M-min}} := \min\{i \geq 0 \mid M_i \leq 1/n\}$ by incorporating our knowledge on the average time between two improvements from Lemma 5.2.12. The following lemma, which is adapted from Lemma 3.1.8, states bounds on stopping times for martingale processes in a quite general setting. In contrast to Lemma 3.1.8, we consider a real state space instead of $\{0, \ldots, n\}$. Moreover, the conditions are relaxed in two ways. First, we only require a lower bound on the *amortized* variance in ℓ steps for some period ℓ. Second, it is sufficient to bound the one-sided variance.

Lemma 5.2.14. *Consider a stochastic process $\{X_t\}_{t\geq 0}$ on a bounded subset of $\mathbb{R}_0^+$. Let $\mathfrak{F}_t$ denote $X_0, \ldots, X_t$, $E_{t+1} := \mathrm{E}(X_{t+1} \mid \mathfrak{F}_t)$ and*

$$\Delta_{t+1} = \mathrm{E}\big((X_{t+1} - E_{t+1})^2 \cdot \mathbb{I}_{X_{t+1} < E_{t+1}} \mid \mathfrak{F}_t\big)$$

with $\mathbb{I}_F$ being the indicator function of event F.

Given $\alpha \in \mathbb{R}$, define $T := \min\{t \colon |X_t - X_0| \geq \alpha \mid X_0\}$. If

1. *$\{X_t\}_{t\geq 0}$ is a supermartingale (i. e. $\mathrm{E}(X_{t+1} \mid \mathfrak{F}_t) \leq X_t$) and*
2. *there exist $\delta > 0$ and $\ell \geq 1$ such that $\sum_{k=t}^{t+\ell} \Delta_k/\ell \geq \delta$ for all $1 \leq t \leq T - \ell$*

then

- $\mathrm{E}(T) \leq \ell + (2X_0 + \alpha) \cdot (\alpha/\delta)$ *and*
- $\mathrm{Prob}(X_T < X_0) \geq \alpha/(\alpha + \mathrm{E}(X_0 - X_T \mid X_T < X_0))$.

Note that the second condition of the lemma is implied if we have a lower bound on the single-step one-sided variance according to $\Delta_t \geq \delta$ for all $1 \leq t \leq T$. We state the more general condition since in the forthcoming analysis, Lemma 5.2.12 yields only a statement on the amortized one-sided variance in $\Theta(\log(2/\rho))$ steps.

Proof of Lemma 5.2.14. We first replace $\{X_t\}_{t\geq 0}$ by a simpler process $\{Y_t\}_{t\geq 0}$ defined by

$$Y_t := \min\{X_0 + \alpha, X_t\}.$$

Hence, the Y-process is capped at $X_0 + \alpha$ when the X-process reaches a point at least $X_0 + \alpha$ for the first time and is identical to the X-process before. By definition, the stopping time T in terms of the Y-process does not change. Second, $\{Y_t\}_{t\geq 0}$ clearly is a supermartingale, too. Third, by the definition of Δ_{t+1}, the second condition of the lemma holds for $\{Y_t\}_{t\geq 0}$ as well. We consider

$$V(t+1) := \operatorname{Var}(Y_{t+1} \mid \mathfrak{F}_t) = \operatorname{E}\big((Y_{t+1} - \operatorname{E}(Y_{t+1} \mid \mathfrak{F}_t))^2 \mid \mathfrak{F}_t\big)$$

and obtain

$$V(t+1) \geq \Delta_{t+1}.$$

for all $t \leq T - 1$.

We define a third process $\{Z_t\}_{t\geq 0}$ by $Z_t := -(Y_t)^2 + \sum_{k=1}^{t} V(k)$ and consider

$$\operatorname{E}(Z_{t+1} \mid \mathfrak{F}_t) = -\operatorname{E}\big((Y_{t+1})^2 \mid \mathfrak{F}_t\big) + \sum_{k=1}^{t+1} \operatorname{E}(V(k) \mid \mathfrak{F}_t).$$

Regarding the $\sum$-term, the summand for $k = t + 1$ by definition of $V(k)$ equals

$$\operatorname{E}(V(k) \mid \mathfrak{F}_t) = V(k)$$

and for $k \leq t$ we have

$$\operatorname{E}(V(k) \mid \mathfrak{F}_t) = V(k)$$

since the right-hand side is $\mathfrak{F}_t$-measurable. Secondly, by the formula $\operatorname{E}\big(A^2\big) = (\operatorname{E}(A))^2 + \operatorname{E}\big((A - \operatorname{E}(A))^2\big)$ we have

$$\begin{aligned} \operatorname{E}\big((Y_{t+1})^2 \mid \mathfrak{F}_t\big) &= (\operatorname{E}(Y_{t+1} \mid \mathfrak{F}_t))^2 + \operatorname{E}\big((Y_{t+1} - \operatorname{E}(Y_{t+1} \mid \mathfrak{F}_t))^2 \mid \mathfrak{F}_t\big) \\ &\leq (Y_t)^2 + V(t+1), \end{aligned}$$

where the last inequality follows from $\{Y_t\}_{t\geq 0}$ being a supermartingale on the space $\mathbb{R}_0^+$. Together,

$$\begin{aligned}\mathrm{E}(Z_{t+1} \mid \mathfrak{F}_t) \;&\geq\; -(Y_t)^2 - V(t+1) + \sum_{k=1}^{t+1} V(k) \\ &=\; -(Y_t)^2 + \sum_{k=1}^{t} V(k) \\ &=\; Z_t.\end{aligned}$$

Thus, $\{Z_t\}_{t\geq 0}$ is a submartingale with respect to $\{Y_t\}_{t\geq 0}$.

$\sum_{k=t}^{t+\ell} \Delta_k/\ell \geq \delta$ implies, along with the supermartingale property, that we have non-zero probability that $Y_{t+1} < Y_t$ at least every ℓ steps, hence $\mathrm{E}(T) < \infty$ follows. Since X_t is bounded and positive by assumption, also $|Y_t|$ and $V(t)$ are bounded, hence $\mathrm{E}(|Z_{t+1} - Z_t| \mid \mathfrak{F}_t)$ is bounded as well and we can apply the martingale stopping theorem (Theorem 2.3.14) for submartingales. This yields $\mathrm{E}(Z_T) \geq \mathrm{E}(Z_0) = (Y_0)^2 = (X_0)^2$ on one hand and, along with $\sum_{k=1}^{T} V(k) \geq \sum_{k=1}^{T} \Delta_k \geq \sum_{k=1}^{\lfloor T/\ell \rfloor \cdot \ell} \Delta_k \geq \lfloor T/\ell \rfloor \ell\delta \geq (T-\ell)\delta$,

$$\mathrm{E}(Z_T) \;=\; \mathrm{E}\big(Y_T^2\big) - \mathrm{E}\left(\sum_{k=1}^{T} V(k)\right) \;\leq\; \mathrm{E}\big(Y_T^2\big) - \delta \cdot \mathrm{E}(T-\ell)$$

on the other hand. This implies

$$\mathrm{E}(T) \;\leq\; \frac{\mathrm{E}\big(Y_T^2\big) - (X_0)^2}{\delta} + \ell \;\leq\; \frac{(X_0+\alpha)^2 - (X_0)^2}{\delta} + \ell \;=\; (2X_0+\alpha)(\alpha/\delta) + \ell,$$

the first statement of the lemma.

The second statement $\mathrm{Prob}(X_T < X_0) \geq \alpha/(\alpha + \mathrm{E}(X_0 - X_T \mid X_T < X_0))$ follows, using the martingale stopping theorem for supermartingales, from

$$\begin{aligned}0 \;\geq\; \mathrm{E}(X_T) - X_0 \;=\; &-\mathrm{Prob}(X_T < X_0) \cdot \mathrm{E}(X_0 - X_T \mid X_T < X_0) \\ &+ (1 - \mathrm{Prob}(X_T < X_0)) \cdot \mathrm{E}(X_T - X_0 \mid X_T > X_0)\end{aligned}$$

by rearranging terms and estimating $\mathrm{E}(X_T - X_0 \mid X_T > X_0) \geq \alpha$. □

We now study the M-process to bound $T_{\text{M-min}}$. Note that $M_0 = P_0 = 1/2$ since our consideration starts with the initialization of MMAS*.

Lemma 5.2.15. *For every constant $\varepsilon > 0$ we have $T_{M\text{-}min} = O(n^{\varepsilon} \log^2(2/\rho))$ with probability $\Omega(1)$.*

Proof. To apply Lemma 5.2.14, we need a bound on the variance of the M-process. However, this variance heavily depends on the current M-value since the process is not homogeneous with respect to place. The closer the value is to one of its bounds 0 and $1 - 1/n$, the smaller the expected displacement from the current position becomes. We therefore split the run of the process into different phases.

Phase i, $i \geq 1$, starts at the first point of time (counting the steps of the M-process) where the current M-value is in the interval $(\ell_i, u_i] := (1/n^{(i+1)\gamma}, 1/n^{i\gamma}]$, where $\gamma = \varepsilon/2$. Phase 0 is different and starts when the value is in the interval $(\ell_0, u_0] := (1/n^{\gamma}, 1/2]$ for the first time. Let $\alpha_i := u_i - \ell_i$, $i \geq 0$ denote the length of the interval. We additionally introduce an infinitely long error phase, which is entered from Phase i when the current M-value exceeds $u_i + \alpha_i$. Note that, due to $M_0 = 1/2$, we start with a non-error phase of positive index. Obviously, if we can prove that Phase $1/\gamma$ starts after $O(n^{\varepsilon} \log^2(2/\rho))$ steps with probability $\Omega(1)$, the lemma follows.

The proof strategy is now to show that with probability $\Omega(1)$, the M-process passes through phases with increasing index without falling into the error phase. Depending on the size of ρ, some phases might be empty. We therefore estimate the number of steps until Phase at least $i+1$ begins when Phase i has started. Let X_{0_i} be the M-value at the start of Phase i. In terms of Lemma 5.2.14, we are interested in the first point of time T_{i+1} where distance α_i from X_{0_i} has been reached and, in particular, in the probability $\text{Prob}\big(X_{T_{i+1}} \leq X_{0_i}\big)$. Since for $E^- := \text{E}\big(X_{0_i} - X_{T_{i+1}} \mid X_{T_{i+1}} < X_{0_i}\big)$ we have $E^- \leq X_{0_i} \leq u_i \leq 2\alpha_i$ (for n large enough), it follows $\alpha/(E^- + \alpha) \geq 1/3$. Hence, by the lemma, it holds $X_{T_{i+1}} \leq X_{0_i} - \alpha_i$ with probability at least 1/3. The latter is sufficient to start Phase at least $i + 1$ without falling into the error phase. Assuming $X_{T_{i+1}} \leq X_{0_i} - \alpha_i$ and repeating this argument at most $1/\gamma$ times, we conclude that Phase $1/\gamma$ starts after an expected number of at most $E^* := \sum_{i=1}^{1/\gamma} \text{E}(T_i)$ steps with probability at least $(1/3)^{1/\gamma}$. Additionally using Markov's inequality, the total number of steps to start Phase $1/\gamma$ is at most $2E^*$ with probability at least $(1/3)^{1/\gamma+1} = \Omega(1)$ since γ is constant. Hence, it suffices to prove a bound on E^* using Lemma 5.2.14. This will be done by bounding Δ-values and, thereby, $\text{E}(T_i)$, separately for all phases. Each Δ-value can be bounded from below by identifying a possible decrease of the current state and estimating the probability and the decrease from below.

Let us pessimistically ignore that Phase i may be left in favor of a higher-index phase before time T_{i+1}. As the M-value is always at most 1/2, the M-process has probability at least 1/2 of decreasing its value. Recall that the M-process records the P-value of a random-state bit only at exchanges of the best-so-far solution. By Lemma 5.2.12, a sequence of $\ell = \Theta(\log(2/\rho))$ steps of the M-process takes $\Omega(1/\rho)$ steps of MMAS* with probability at least $\Omega(1)$. If all these steps of the M-process were decreasing, this would decrease the value at the beginning of the sequence by a factor of $f := (1-\rho)^{\Omega(1/\rho)} = 1 - \Omega(1)$ using the pheromone update formula. Actually, some steps of the sequence will probably be increasing. However, according to Chernoff bounds, since each single

step is decreasing with probability $\Omega(1)$, we have $\Omega(\ell)$ decreasing steps within ℓ steps with probability $\Omega(1)$. Given a current value of v before the first decreasing step, the sum of the amounts of decrease in decreasing steps (not counting increasing steps in between since we are aiming at a bound of the one-sided variance Δ) is then at least $v - vf = \Omega(1) \cdot v$, hence at least $v \cdot \Omega(1/\ell)$ on average over ℓ steps. Again using the definition of the phases, we have $v \geq \ell_i$, hence this decrease is at least

$$\frac{1}{n^{(i+1)\gamma}} \cdot \Omega\left(\frac{1}{\ell}\right) \;=\; \Omega\left(\frac{1}{n^{(i+1)\gamma}\log(2/\rho)}\right).$$

In order to compute Δ, the last value still has to be multiplied by the probability $\Omega(1)$ of the event and taken to the square. Applying Jensen's inequality, the average of the squares is at least as large as the square of the average. Altogether, the average Δ-value in $\Theta(\log(2/\rho))$ steps before time T_{i+1} is at least $\delta_i = \Omega(1/(n^{2(i+1)\gamma}\log^2(2/\rho)))$.

We apply Lemma 5.2.14. Estimating $(2X_{0_i} + \alpha_i) \leq 5\alpha_i$, we get $\mathrm{E}(T_i) \leq 5\alpha_i^2/\delta_i + \ell = O(n^{2(i+1)\gamma - 2i\gamma}\log^2(2/\rho) + \log(2/\rho)) = O(n^{2\gamma}\log^2(2/\rho))$. By the definition of γ, we have shown $\mathrm{E}(T_i) = O(n^{\varepsilon}\log^2(2/\rho))$, and, using $1/\gamma = O(1)$, also $E^* = O(n^{\varepsilon}\log^2(2/\rho))$. This completes the proof. □

Now all arguments for the proof of Theorem 5.2.7 are ready.

Proof of Theorem 5.2.7. A lower bound $\Omega(n/(\rho\log(2/\rho)))$ follows from Lemma 5.2.12, hence we only need to show a lower bound $\Omega(n^2)$. Consider a time phase containing the first $n/12$ improvements. Lemma 5.2.10 implies that at the end of the phase the current LeadingOnes-value is still less than $n/2$ with probability $1 - 2^{-\Omega(n)}$, which we assume to happen.

Consider the last $n/2$ bits in random state and note that every improvement corresponds to one step of the M-process. Choosing $0 < \varepsilon < 1$, along with $\rho = 1/\mathrm{poly}(n)$ and n large enough, the time bound for the M-process from Lemma 5.2.15 is $cn^{\varepsilon}\log^2(2/\rho) \leq n/12$ for some constant $c > 0$. This implies that during the phase any of the last $n/2$ bits independently reaches the lower border for its success proability at least once with probability $\Omega(1)$. By Chernoff bounds, we have $\Omega(n)$ such bits with probability $1 - 2^{-\Omega(n)}$. Assuming this to happen, Lemma 5.2.11 is in force and the outstanding lower bound $\Omega(n^2)$ follows by the law of total probability. □

In the lengthy proof, we had to carefully look at the random bits and to study the structure of the optimization process. It seems to be even harder to prove a corresponding lower bound for MMAS since accepting equally good solutions implies that more than n exchanges of the best-so-far solution can happen. Also additional ideas are required to transfer the proof of Theorem 5.2.7 and to obtain an improved lower bound for MMAS* on OneMax. Nevertheless, our analysis for LeadingOnes is important since it contains important proof techniques for lower bounds on the runtime of MMAS algorithms. Moreover, it rigorously shows that the upper bounds derived by the fitness-level

method described in Section 5.2.2 can be very good and almost asymptotically tight. Still, they leave room for improvement using specialized techniques as presented in Theorem 5.2.6 since it is not necessary for the process to freeze the best-so-far solution in pheromone for every fitness level.

5.2.3 Plateau Functions

The general upper bounds from Theorems 5.2.3 and 5.2.4 for unimodal functions yield a gap of only polynomial size between MMAS and MMAS*. In addition, we have proven the same upper bounds on LeadingOnes for both MMAS and MMAS*. This may give the impression that MMAS and MMAS* behave similarly on all functions. However, this only holds for functions with a certain gradient towards better solutions. On plateaus MMAS and MMAS* can have a totally different behavior.

We reconsider the function Needle where only one single solution has objective value 1 and the remaining ones get value 0. In its general form, the function is defined as

$$\textsc{Needle}(x) := \begin{cases} 1 & \text{if } x = x_{\text{OPT}}, \\ 0 & \text{otherwise,} \end{cases}$$

where x_{OPT} is the unique global optimum. Gutjahr and Sebastiani (2008) compare MMAS* and (1+1) EA* w. r. t. their runtime behavior. For suitable values of ρ that are exponentially small in n, MMAS* has expected optimization time $O(c^n)$, $c \geq 2$ an appropriate constant, and beats the (1+1) EA*. The reason is that MMAS* behaves nearly as random search on the search space while the initial solution of the (1+1) EA* has Hamming distance n to the optimal one with probability 2^{-n}. To obtain from such a solution an optimal one, all n bits have to flip, which has expected waiting time n^n, leading in summary to an expected optimization time $\Omega((n/2)^n)$. In the following, we show a similar result for MMAS* if ρ decreases only polynomially with the problem dimension n.

Theorem 5.2.16. *Choosing $\rho = 1/\text{poly}(n)$, the optimization time of* MMAS* *on* Needle *is at least $(n/6)^n$ with probability $1 - e^{-\Omega(n)}$.*

Proof. Let x be the first solution constructed by MMAS* and denote by x_{OPT} the optimal one. As it is chosen uniformly at random from the search space, the expected number of positions where x and x_{OPT} differ is $n/2$ and there are at least $n/3$ such positions with probability $1 - e^{-\Omega(n)}$ using Chernoff bounds. At these positions of x the "wrong" edges of the construction graph are reinforced as long as the optimal solution has not been obtained. This implies that the probability of obtaining the optimal solution in the next step is at most $2^{-n/3}$. After at most $t^* \leq (\ln n)/\rho$ (see Inequality (5.7)) iterations, the pheromone values of x have touched their borders provided x_{OPT} has not been obtained. The probability of having obtained x_{OPT} within a phase of t^* steps is

at most $t^* \cdot 2^{-n/3} = e^{-\Omega(n)}$. Hence, the probability of producing a solution that touches its pheromone borders and differs from x_{OPT} in at least $n/3$ positions before producing x_{OPT} is $1 - e^{-\Omega(n)}$. In this case, the expected number of steps to produce x_{OPT} is $(n/3)^n$ and the probability of having reached this goal within $(n/6)^n$ steps is at most 2^{-n}. □

The probability of choosing an initial solution x that differs from x_{OPT} by n positions is 2^{-n}, and in this case, after all n bits have reached their corresponding pheromone borders, the probability of creating x_{OPT} equals n^{-n}. Using the ideas of Theorem 5.2.16, the following corollary can be proved which asymptotically matches the lower bound for the (1+1) EA* given in Gutjahr and Sebastiani (2008).

Corollary 5.2.17. *Choosing $\rho = 1/\text{poly}(n)$, the expected optimization time of* MMAS* *on* NEEDLE *is $\Omega((n/2)^n)$.*

It is well known that the (1+1) EA that accepts each new solution has expected optimization time $\Theta(2^n)$ on NEEDLE (see Garnier, Kallel, and Schoenauer, 1999; Wegener and Witt, 2005) even though it samples with high probability in the Hamming neighborhood of the latest solution. On the other hand, MMAS* will have a much larger optimization time unless ρ is superpolynomially small (Theorem 5.2.16). Our aim is to show in the forthcoming Theorem 5.2.20 that MMAS is more efficient than MMAS* and almost competitive to the (1+1) EA.

As in Section 5.2.2, the heart of our analysis will be the random process describing the pheromone values if the fitness function does not give any information. The random walk on the success probabilities of a bit in random state, which was captured by the M-process, reappears. The situation now is easier: as long as the needle has not been found, MMAS exchanges the best-so-far solution in each step. Hence, in terms of the M-process (which however has no lower bound on success probabilities) we would have the special case of $L_i = 1$ for $i \geq 0$. Our setting collapses to a Markovian random walk P_t, $t \geq 0$, on the state space $[1/n, 1 - 1/n]$ that, according to the pheromone update formula, increases or decreases its current state with probability P_t and $1 - P_t$, respectively. We call this random walk the *P-process*. Note that we assume each step to exchange the solution, hence the process does not freeze after the needle has been found. Until the first hitting time of the needle, the P-process and the random success probability at a bit coincide.

Many analyses conducted in Section 5.2.2 carry over in simplified shape. In particular, Lemma 5.2.9 is in force. In conjunction with Lemma 5.2.8, it implies the unconditional success probability at time t to be purely random, hence the underlying bit is set to 1 with probability 1/2 at each point of time. We will however have to inspect success probabilities at later points of time, assuming that the needle has not been found yet. Assuming w. l. o. g. $x_{\text{OPT}} = 1^n$, this condition biases the current success probability towards decreasing values.

In the following, we will basically redo the random walk analysis from Lemma 5.2.15 to bound the time until a random success probability, starting at an arbitrary point in $[1, n, 1-1/n]$, exceeds $1/2$. Intuitively, then the expected success probability will stay at least $1/2$ for all following points of time. This is made precise as follows.

Lemma 5.2.18. *If $P_t \geq 1/2$ then for every $t' \geq t$, $\mathrm{E}(P_{t'} \mid P_t) \geq 1/2$.*

Proof. We inductively show a stronger statement. If at some time t the so-called super-symmetry

$$\mathrm{Prob}(P_t \geq p) \;\geq\; \mathrm{Prob}(P_t \leq 1-p) \;\text{ for all } p \geq 1/2$$

holds, then super-symmetry holds at all times $t' \geq t$. Using the same arguments as in the proof of Lemma 5.2.9, super-symmetry at time t implies $\mathrm{E}(P_t) \geq 1/2$.

To prove the statement, assume super-symmetry to hold at time t. For the induction step, we exploit the following monotonicity of the formula $i(p) = (1-\rho)p + \rho = 1 - (1-\rho)(1-p)$ for the pheromone increase: $p' \geq p \;\Rightarrow\; i(p') \geq i(p)$ (see Lemma 5.1.2). Moreover, the probability of increasing a pheromone value (i. e., success probability) is obviously monotone in the probability itself.

We exploit these observations to iteratively replace the distribution behind the random variable P_t by a simpler distribution whose successor P_{t+1} is easier to estimate; actually, we will apply Lemma 5.2.9 in the end. Let $\mu_t(p) := \mathrm{Prob}(P_t = p)$ be the probability mass of point p at time t. Given two points $p_1 > p_2$, we set $\mu'_t(p_1) := \mu_t(p_1) - a$ for some $0 \leq a \leq \mu_t(p_1)$, $\mu'_t(p_2) := \mu_t(p_2) + a$, and $\mu'_t(p) := \mu_t(p)$ for all $p \notin \{p_1, p_2\}$. In other words, we conduct a so-called *probability shift* to decrease the mass of a point corresponding to a higher success probability by some amount and increase the mass of a lower point by the same amount. Let P'_t be the random variable corresponding to μ'_t. As a consequence, $\mathrm{Prob}(P_t \geq p) \geq \mathrm{Prob}(P'_t \geq p)$ for all $0 \leq p \leq 1$. Now, due to the above-mentioned monotonicities, $\mathrm{Prob}(P_{t+1} \geq p) \geq \mathrm{Prob}\big(P'_{t+1} \geq p\big)$ and $\mathrm{Prob}(P_{t+1} \leq 1-p) \leq \mathrm{Prob}\big(P'_{t+1} \leq 1-p\big)$, i. e., also the distributions at the following step stochastically dominate each other as needed to show super-symmetry at time $t+1$.

Starting from P_t, we carry out a sequence of probability shifts as follows. Let $\ell_0 := -\infty$ and let $\ell_1 < \cdots < \ell_r < 1/2$ be all points in the lower half of the pheromone scale with positive mass, sorted in ascending order. Note that there are only finitely many such points after finitely many steps. Let μ denote the distribution which is iteratively modified. We start from $\mu := \mu_t$. Then, for $1 \leq i \leq r$, we shift probability $\mathrm{Prob}(1-\ell_{i-1} > P_t \geq 1-\ell_i) \geq \mu(\ell_i)$ from all points in the interval $(1-\ell_{i-1}, 1-\ell_i]$ to the point $1-\ell_i$. If $\mu(1-\ell_i) > \mu(\ell_i)$ after this shift, we additionally shift the difference $\mu(1-\ell_i) - \mu(\ell_i)$ from $1-\ell_i$ to $1-\ell_{i+1}$, where $\ell_{r+1} := 1/2$. Inductively over i, it follows from the super-symmetry at time t that we do not shift more probability than allowed. Moreover, by the procedure, super-symmetry is still valid for the distribution P'_t obtained after the k iterations since even the perfect symmetry $\mathrm{Prob}(P'_t = p) = \mathrm{Prob}(P'_t = 1-p)$ holds then. By Lemma 5.2.9, its successor P'_{t+1} is perfectly symmetrical, too. Hence,

the actual distribution P_{t+1} stochastically dominates a symmetrical distribution, and, therefore, has super-symmetry. This completes the induction from t to $t+1$. □

We now describe the analysis of the random walk as it will be needed in the proof of the theorem. Still concentrating on the random success probabilities P_t, $t \geq 0$, at a single bit, let η_a denote the first point of time $t \geq 0$ such that $P_t \geq 1/2$ given that for the initial pheromone value it holds $P_0 = a < 1/2$ (different from the initialization in the MMAS algorithm).

Lemma 5.2.19. *For every $a < 1/2$, $\mathrm{E}(\eta_a) = O(n^2/\rho^2)$.*

Proof. We start with a useful transformation. Since we would like to reuse Lemma 5.2.14, which was previously applied to bound the first hitting time of the state $1/n$ for the M-process, we swap the meaning of ones and zeros and consider the process $X_t := 1 - P_t$, $t \geq 0$, instead. We now are interested in the first hitting time for the states $\{x \mid x \leq 1/2\}$ of the process X_t given $X_0 = 1 - a > 1/2$. To this end, we are allowed to remove the lower border $1/n$ on the success probabilities. In consequence, the X-process equals the M-process given that $L_i = 1$ for all i.

By Lemma 5.2.13, the X-process is a Markovian supermartingale. Hence, we can apply the general Lemma 5.2.14. To this end, we need a lower bound on Δ_{t+1}, which is obtained by identifying a single decreasing step and the amount and probability of decrease. We can assume $X_t > 1/2$.

For $X_{t+1} \leq X_t$ to happen, an event of probability $1 - X_t \geq 1/n$ is sufficient (note that the upper bounds on success probabilities have not been removed). In this case, $X_{t+1} = (1-\rho)X_t$. Moreover, due to $X_t \geq 1/2$, we have $X_{t+1} \leq X_t - \rho/2$ then. Altogether, $\Delta_{t+1} \geq ((1-X_t)(\rho/2))^2 \geq \rho^2/(4n^2)$. We invoke Lemma 5.2.14 with $\alpha = 1/2$, $\delta := \rho^2/(4n^2)$, and $\ell = 1$. As $X_0 \leq 1$, for the resulting stopping time T we have $\mathrm{E}(T) = O(n^2/\rho^2)$. Moreover, $X_0 > 1/2$, $X_T \leq 1$, and $|X_T - X_0| \geq 1/2$ implies $X_T \leq 1/2$. Hence $P_T \geq 1/2$ and $\mathrm{E}(\eta_a) = O(n^2/\rho^2)$ as claimed. □

The following theorem shows that the expected optimization time of MMAS on Needle is at most by a polynomial factor larger than the one of the (1+1) EA unless ρ is superpolynomially small.

Theorem 5.2.20. *The expected optimization time of* MMAS *on* Needle *is bounded from above by $O((n^2 \log n)/\rho^2 \cdot 2^n)$.*

Proof. By the symmetry of the construction procedure and uniform initialization, we w. l. o. g. assume that the needle x_{OPT} equals the all-ones string 1^n. As in Wegener and Witt (2005), we study the process on the constant function $f(x) = 0$. The first hitting times for the needle are the same on Needle and the constant function, while the constant function is easier to study as MMAS accepts each new search point forever for this function.

The proof idea is to study a kind of "mixing time" $t(n)$ after which each bit is independently set to 1 with a probability of at least 1/2 regardless of its initial success probability (recall that this means the probability of setting the bit to 1). Since bits are treated independently, this implies that the probability of creating the needle is at least 2^{-n} in some step after at most $t(n)$ iterations. We successively consider independent phases of (random) length $t(n)$ until the needle is sampled. The number of phases required follows a geometric distribution with parameter at least 2^{-n}, hence, the expected number of phases required to sample the needle is at most 2^n. By the linearity of expectation, the expected time until 2^n phases have elapsed is bounded by $\mathrm{E}(t(n)) \cdot 2^n$. The theorem follows if we can show that $\mathrm{E}(t(n)) = O((n^2 \log n)/\rho^2)$.

We recall the above-described random walk on the success probabilities. Consider the independent success probabilities of the n bits for any initial distribution. We call a success probability *good* at a certain time t if is has been bounded from below by 1/2 at least once in the t steps after initialization and *bad* otherwise. We are interested in the time T^* until all n success probabilities have become good. For each single success probability, the expected time until becoming good is $O(n^2/\rho^2)$ according to Lemma 5.2.19. Due to Markov's inequality, the time is $O(n^2/\rho^2)$ with probability at least 1/2. Repeating $2 \log n$ independent such phases, this implies that after $O(n^2/\rho^2 \cdot \log n)$ steps, each success probability is bad with probability at most $1/(2n)$. Hence, by the union bound, the probability is only at most 1/2 that there is a bad success probability left after this number of steps. Repeating this argument an expected number of at most 2 times, $\mathrm{E}(T^*) = O((n^2 \log n)/\rho^2)$ follows. By definition, all success probabilities have been at least 1/2 at least once after T^* steps. Using Lemma 5.2.18, we obtain that T^* is an upper bound on $t(n)$, and the theorem follows. □

The function Needle requires an exponential optimization time for each algorithm that has been considered. Often plateaus are much smaller, and randomized search heuristics have a good chance to leave them within a polynomial number of steps. Gutjahr and Sebastiani (2008) consider the function NH-OneMax that consists of the Needle-function on $k = \log n$ bits and the function OneMax on $n - k$ bits, which can only be optimized if the needle has been found on the needle part. The function is defined as

$$\text{NH-OneMax}(x) = \left(\prod_{i=1}^{k} x_i\right)\left(\sum_{i=k+1}^{n} x_i\right).$$

Taking into account the logarithmic size of the Needle-function of NH-OneMax, MMAS* with polylogarithmically small ρ cannot optimize the needle part within an expected polynomial number of steps. The proof ideas are similar to those used in the proof of Theorem 5.2.16. After initialization the expected Hamming distance of the needle part to the needle is $(\log n)/2$, and it is at least $(\log n)/3$ with probability $1 - o(1)$. Working under this condition, this means that the probability of sampling

the needle is at most $2^{-(\log n)/3} = n^{-1/3}$ in each step before the needle is found. As $\rho = 1/\text{polylog}(n)$ holds, the lower pheromone borders of the $(\log n)/3$ "wrong" bits from the initial solution are reached in at most $t^* \leq (\ln n)/\rho = \text{polylog}(n)$ steps. Hence, the needle is found before this situation has been reached with probability at most $\text{polylog}(n)/n^{1/3} = o(1)$. Afterwards, the probability of sampling the needle is at most $n^{-(\log n)/3} = 2^{-\Omega(\log^2 n)}$. This proves the following superpolynomial lower bound.

Theorem 5.2.21. *If $\rho = 1/\text{polylog}(n)$, the expected optimization time of* MMAS* *on* NH-OneMax *is $2^{\Omega(\log^2 n)}$.*

Also the proof of Theorem 5.2.20 carries over to a major extent. The random walk arguments leading to $\mathrm{E}(t(n)) = O((n^2 \log n)/\rho^2)$ still hold since random bits are considered independently and the borders for the pheromone values have not been changed. What has been changed is the size of the needle. As now the needle part only consists of $\log n$ bits, the probability of creating it is at least $2^{-\log n} = 1/n$ after $t(n)$ steps. Hence, MMAS can find the needle after an expected number of $O((n^2 \log n)/\rho^2 \cdot n)$ steps. After this goal has been achieved, the unimodal function OneMax, which contains at most $n+1$ different fitness values, has to be optimized. We conclude from Theorem 5.2.4, the general bound on unimodal functions, that MMAS optimizes OneMax in an expected number of $O((n^3 \log n)/\rho)$ steps. Putting the two bounds together, the following result has been proved.

Theorem 5.2.22. *The expected optimization time of* MMAS *on* NH-OneMax *is at most $O((n^3 \log n)/\rho^2)$.*

This bound is polynomial if $\rho = 1/\text{poly}(n)$, in contrast to the superpolynomial bound for MMAS* from Theorem 5.2.21. Hence, MMAS is superior to MMAS* on NH-OneMax as well.

5.2.4 Experiments

We supplement our theoretical investigations by experiments. The time bounds presented in the previous sections are asymptotic, hence they do not reveal whether certain effects predicted for large n also occur for small problem dimensions. Moreover, experimental data can reveal details that are not captured by our theorems.

The Performance of MMAS on Needle

First, we take a closer look at MMAS on Needle for several values of ρ. Theorem 5.2.20 gives an upper bound of order $O((n^2 \log n)/\rho^2 \cdot 2^n)$, thus growing with $1/\rho$. However, we conjecture that for some values of ρ the upper bound is not tight. If ρ is extremely large such that MMAS equals the (1+1) EA, the best-so-far solution performs a random walk through the search space and the expected time until the needle is found is about

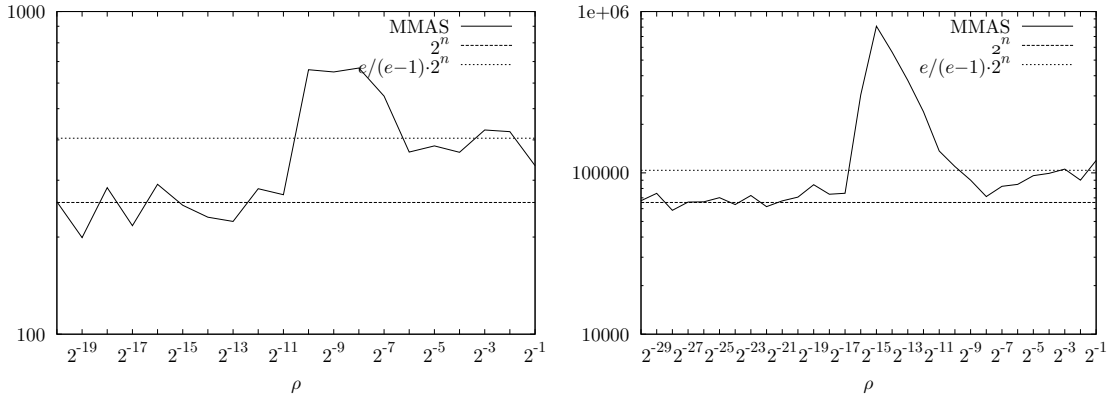

Figure 5.6: Average runtime for MMAS on NEEDLE with $n = 8$ (left) and $n = 16$ (right) and exponentially decreasing values of ρ. The data is averaged over 100 runs for each setting.

$e/(e-1) \cdot 2^n \approx 1.58 \cdot 2^n$ due to Garnier et al. (1999). In addition, if ρ is extremely small, MMAS degenerates to almost random search and the expected time to find the needle is then close to 2^n.

We are tempted to believe that the optimization time is always of order 2^n and hence independent of ρ. The upper bound $O((n^2 \log n)/\rho^2 \cdot 2^n)$ would then be far too pessimistic for small values of ρ. This is true for extremely small ρ, but we do not know how MMAS behaves with intermediate ρ-values. Therefore, we perform experiments for NEEDLE with $n \in \{8, 16\}$ and exponentially decreasing values for ρ, $\rho = 2^{-1}, 2^{-2}, 2^{-3}, \ldots$ and measure the average runtime of MMAS during 100 independent runs for each setting. The results are shown in Figure 5.6.

Consider the plot for $n = 16$. First, observe that the average runtime slightly decreases when decreasing ρ from 2^{-1} to 2^{-8}. A possible explanation is that the (1+1) EA tends to resample the current solution, which happens with probability approximately $1/e$. Hence, the expected waiting time until a new solution is sampled is about $1/(1-1/e) = e/(e-1)$, which explains the bound of about $e/(e-1) \cdot 2^n$ by Garnier et al. (1999). This bound also holds for MMAS with very large ρ. When decreasing ρ, the probability of resampling the current solution decreases and so does the average runtime.

When further decreasing ρ, the average runtime increases very fast. This can be explained since the "mixing time" (cf. proof of Theorem 5.2.20) increases with a function of $1/\rho$. Due to the slow mixing, MMAS tends to spend a lot of time sampling the same region of the search space over and over again. If we are unlucky, MMAS focuses on regions that are far away from the needle where many (say $\Omega(n)$) pheromones hit the "wrong" borders. In this case the probability of generating the needle is far below 2^{-n},

yielding an overly large average optimization time. This observation shows that it is necessary for the upper bound from Theorem 5.2.20 to grow with some function of $1/\rho$ for these ρ-values.

Lastly, at some point on the ρ-scale, MMAS suddenly turns into almost random search. Here, ρ is so small that typically the needle is found by random search before the pheromones get the chance to move towards a border. A comparison with the lower horizontal line indicates that the average runtime is highly concentrated around 2^n.

For $n = 8$, MMAS shows a similar behavior, although the effects are less pronounced. We conclude that, surprisingly, MMAS shows a phase transition behavior for exponentially decreasing values of ρ (in contrast to the phase transition behavior of the 1-ANT for only polynomially decreasing ρ). Although MMAS is competitive with the (1+1) EA for very large and very small values of ρ, it is much more inefficient for intermediate values of ρ.

Comparing MMAS and MMAS* on Unimodal Functions

We now turn to a comparison of MMAS and MMAS*. Regarding unimodal functions, the upper bound for MMAS from Theorem 5.2.4 can be by a factor of n^2 larger than the upper bound for MMAS* from Theorem 5.2.3. We believe that the larger bound for MMAS is too pessimistic for many unimodal functions. In this bound, we had to account for situations where pheromones in MMAS cannot freeze to a single solution on the current fitness level. However, the only situation where many bits will not freeze properly occurs when the current best-so-far solution consistently switches between different solutions with the same fitness and large Hamming distance. Such a situation appears to be very atypical for most unimodal functions. We expect MMAS to either freeze towards a small set of solutions that are close in Hamming space or to find an improvement beforehand. This behavior is very close to MMAS*. Another reason why we believe that MMAS and MMAS* behave similarly is that the upper bounds for LeadingOnes from Theorem 5.2.6 are equal for the two algorithms.

We perform experiments for the two unimodal functions investigated in Section 5.2.2, OneMax and LeadingOnes. The experimental setting is taken over from the previous subsection. That is, we use exponentially decreasing values for ρ to cover the range from almost pure random search to an algorithm close to the (1+1) EA. Preliminary experiments indicated a small but noticeable difference between MMAS and MMAS* for a specific range of ρ-values. Therefore, we repeated the experiments with an increased number of 1000 runs per setting. The results are shown in Figure 5.7.

The observed average runtimes for MMAS and MMAS* are nearly equal for almost all values of ρ, except for a range of ρ-values, roughly around $\rho = 2^{-9}$, where MMAS is more efficient than MMAS*. We applied a statistical test to see for each ρ-value whether the differences between MMAS and MMAS* are significant. A non-parametric Mann-Whitney test revealed highly significant results ($p \leq 0.001$) in favor of MMAS for

OneMax with $n = 8$ and $\rho \in \{2^{-9}, \ldots, 2^{-5}\}$, OneMax with $n = 16$, $\rho \in \{2^{-14}, \ldots, 2^{-5}\}$ and $\rho = 2^{-16}$, and finally for LeadingOnes with $n = 16$ and $\rho \in \{2^{-13}, \ldots, 2^{-7}\}$.

The differences on OneMax are surprising in the light of the conjecture by Gutjahr (2007) that accepting equally good solutions deteriorates the performance of MMAS* on OneMax. Our statistical tests confirm that for some ρ-values the opposite holds and that accepting equally good solutions can lead to a significant speed-up.

It seems that for OneMax jumping between equally fit solutions is beneficial. For a possible explanation we identify bits with their corresponding 1-edges. Imagine two bits, one with low pheromone and one with high pheromone. If only the low-pheromone bit is reinforced, the increase in pheromone is larger compared to the case where only the high-pheromone bit is reinforced. Also the probability of setting both bits to 1 simultaneously increases more if only the low-pheromone bit is reinforced. The difference is even stronger if the high-pheromone bit already hit the upper pheromone border. This observation might explain why on OneMax it is better to switch between different bits to reinforce than to reinforce the same bits over and over again.

Comparing MMAS and MMAS* on Plateaus

Now we compare the performance of MMAS* and MMAS on the plateau functions Needle and NH-OneMax, averaged over 100 runs. The results from Section 5.2.3 predict much larger runtimes for MMAS*.

First consider Needle. After the random freezing time t^*, all pheromones in MMAS* are frozen until the end of the run. If k bits are frozen towards the "wrong" bit value, the probability of finding the needle with the next constructed solution is exactly $(1/n)^k \cdot (1 - 1/n)^{n-k}$. This probability is extremely small. For, say, $n = 16$ and $k = 12$, it is less than $(1/16)^{12} = 2^{-64}$ and the expected time needed of finding the needle is larger than 2^{64}. As these values are far too large to allow a complete simulation, the optimization times of MMAS* with $n = 16$ are estimated by adding the expected time to find the needle to the observed freezing time. (In rare cases where the needle was found before all pheromones were frozen, the real optimization time was recorded.) The resulting values still contain randomness as k is a random variable. On the other hand the estimations yield a reduction of variance, which gives us an even better picture of the average performance than a complete simulation. The optimization times for $n = 8$ were obtained by real simulations, though.

Figure 5.7 shows the resulting average optimization times. The data for MMAS is taken over from Figure 5.6, but the large peaks from Figure 5.6 now clearly pale in comparison with the average runtime of MMAS*. One can see that MMAS and MMAS* behave almost identically for very small values of ρ, since then both algorithms degenerate to pure random search. Focussing on the remaining ρ-values, there is no doubt that MMAS outperforms MMAS*. For $n = 8$, MMAS* is by a factor of more than 100 slower

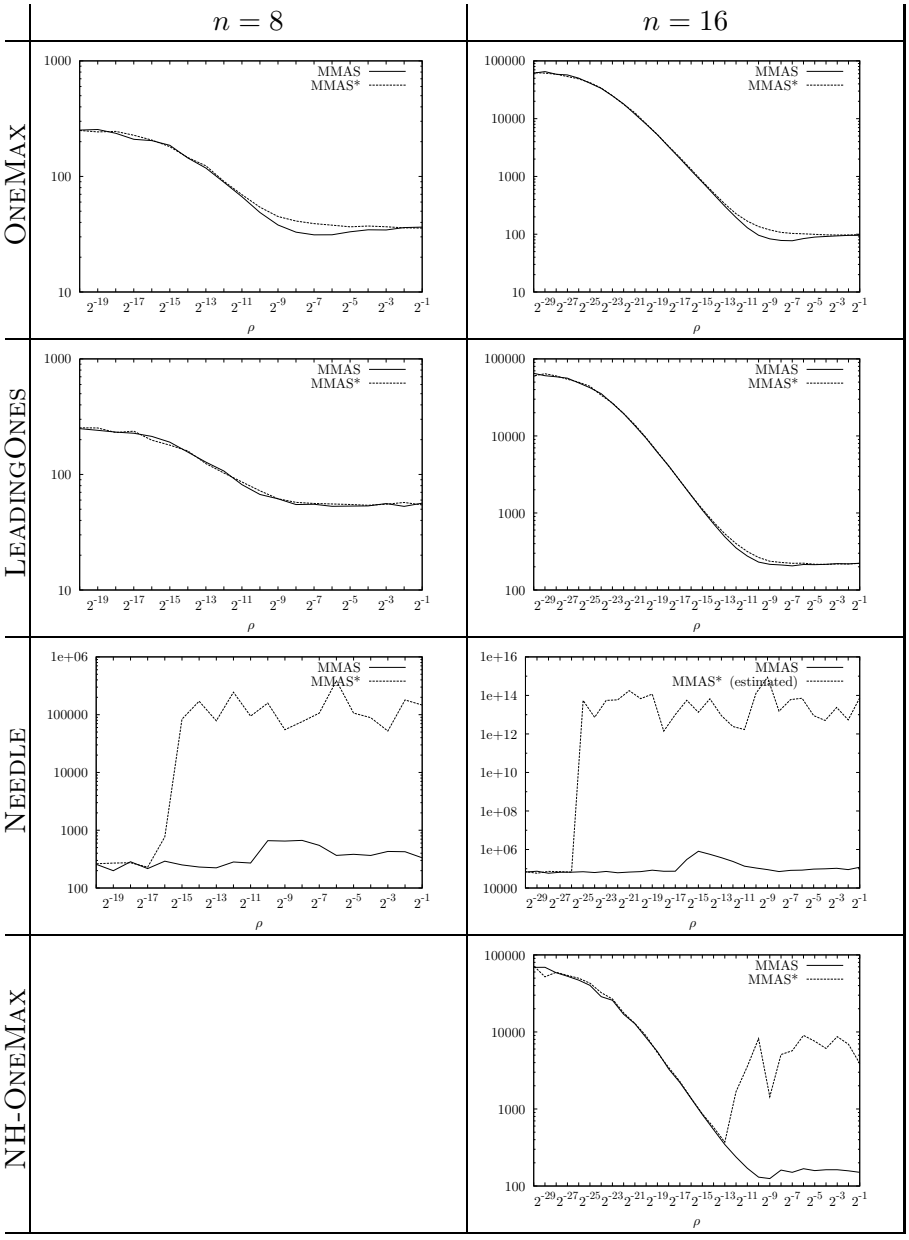

Figure 5.7: Average runtime for MMAS and MMAS* on OneMax, LeadingOnes, Needle, and NH-OneMax. The runtime of MMAS* on Needle with $n = 16$ is estimated by the (conditional) expected remaining optimization time once all pheromones are frozen. The case of NH-OneMax with $n = 8$ is omitted due to the small size of the needle.

than MMAS. For $n = 16$, the differences are even more drastic: the average optimization time of MMAS* is often larger than 10^{13} while MMAS never exceeds 10^6.

For NEEDLE the best strategy is to choose ρ so small that MMAS and MMAS* degenerate to random search. This is different for NH-ONEMAX where, due to the underlying ONEMAX-problem, we can expect MMAS and MMAS* to outperform pure random search with a sufficiently large value of ρ. We investigate NH-ONEMAX with $n = 16$, i. e., a needle consisting of 4 bits. The size of the needle may look small, but finding needles is more challenging than for NEEDLE. The problem dimension is much larger than the size of the needle, and so the pheromone borders are more extreme compared to NEEDLE when considering equally sized needles. As a consequence, it is harder to find the needle in case some bits have pheromones at the "wrong" borders.

Figure 5.7 shows the average optimization times for MMAS and MMAS*. The results show that it is necessary to increase the value of ρ to deal with the ONEMAX-part of the function until $\rho \geq 2^{-13}$. Larger values of ρ increase the optimization time of MMAS* as this algorithm is not able to perform a random walk on the NEEDLE-part of the function. In contrast to this, MMAS also performs well for large values of ρ as the search for the needle is achieved by replacing equally good solutions.

5.2.5 Conclusions

In this section, we have compared the 1-ANT with an MMAS variant investigated by Gutjahr and Sebastiani (2008). The 1-ANT is extremely inefficient on ONEMAX and LEADINGONES if the evaporation factor is small and exhibits a phase transition from polynomial to exponential runtime, in sharp contrast to MMAS*. Gutjahr (2007) conjectured that this difference is due to the different selection mechanisms, i. e., the question whether to accept solutions of equal fitness. Therefore, we investigated a variant of MMAS*, called MMAS, that accepts equally fit solutions, so MMAS and the 1-ANT only differ in their pheromone update mechanisms. Our analyses from Section 5.2.2 revealed that on ONEMAX and LEADINGONES there is no phase transition for MMAS* and MMAS. Hence, the phase transition of the 1-ANT must result from the pheromone update mechanism.

One conclusion drawn from this comparison is that MMAS is more effective for simple problems than the 1-ANT. Moreover, MMAS better reflects ACO algorithms used in applications. Future work should therefore focus on the analysis of MMAS variants. Nevertheless, the analysis of the 1-ANT was an important step as it provided a detailed insight into the effect of pheromone updates in ACO and stimulated further studies (Attiratanasunthron and Fakcharoenphol, 2008; Doerr and Johannsen, 2007b; Doerr, Johannsen, and Tang, 2008).

Gutjahr and Sebastiani (2008) have shown how to transfer the fitness-level method, a powerful and well-known tool from the analysis of evolutionary algorithms, to the analysis of ACO algorithms. Here, we have presented these previous results in a simplified

	MMAS*	MMAS
General lower bound	$\Omega\left(\frac{\log n}{\rho} - \log n\right)$	$\Omega\left(\frac{\log n}{\rho} - \log n\right)$
OneMax	$O\left(\frac{n \log n}{\rho}\right)$	$O\left(\frac{n^3 \log n}{\rho}\right)$
LeadingOnes	$O\left(n^2 + \frac{n}{\rho}\right)$ $O\left(n^2 \cdot \left(\frac{1}{\rho}\right)^{\varepsilon} + \frac{n/\rho}{\log(1/\rho)}\right)$ $\Omega\left(n^2 + \frac{n/\rho}{\log(2/\rho)}\right)$	$O\left(n^2 + \frac{n}{\rho}\right)$ $O\left(n^2 \cdot \left(\frac{1}{\rho}\right)^{\varepsilon} + \frac{n/\rho}{\log(1/\rho)}\right)$
Unimodal functions	$O\left(\left(n + \frac{\log n}{\rho}\right) d\right)$	$O\left(\frac{n^2 \log n}{\rho} \cdot d\right)$
Needle	$\Omega\left(\left(\frac{n}{2}\right)^n\right)$	$O\left(\frac{n^2 \log n}{\rho^2} \cdot 2^n\right)$
NH-OneMax	$2^{\Omega(\log^2 n)}$	$O\left(\frac{n^3 \log n}{\rho^2}\right)$

Table 5.1: Bounds on the expected optimization times for MMAS* and MMAS for every constant $\varepsilon > 0$ and d the number of fitness values. The general lower bound holds for all functions with unique global optimum. The bound for MMAS on OneMax follows from Theorem 5.2.4. The lower bounds require $\rho = 1/\text{polylog}(n)$ for MMAS* on NH-OneMax and otherwise $\rho = 1/\text{poly}(n)$.

framework and extended them to the broad class of all unimodal functions. Moreover, the upper bounds obtainable by this method have been improved using specialized techniques for the LeadingOnes function, and the first lower bounds for MMAS* on OneMax and LeadingOnes have been given. The analyses cover random walks of pheromone values, which were not well understood before. In particular, we have estimated first hitting times for pheromone borders in a general framework for random processes that are not homogeneous w. r. t. place. As an important insight, it has been shown that every pheromone value quickly reaches one of the borders even if the corresponding bit does not have any significant contribution to the fitness. The random-walk analysis is considered valuable also from a methodological point of view.

Another goal of this section was to elaborate on the difference between MMAS and MMAS*. Table 5.1 summarizes the considered bounds on the expected optimization

times. Regarding unimodal functions, our theoretical bounds and experiments for OneMax and LeadingOnes showed similar performance for both algorithms. However, for a range of evaporation factors, surprisingly, MMAS empirically outperforms MMAS*. The observed differences were confirmed by statistical tests. Unlike MMAS*, accepting equally fit solutions allows MMAS to explore plateaus of equal fitness by a random walk of the best-so-far solution. For well-known plateau functions such as Needle, replacing equally fit solutions is essential. Both our asymptotical theoretical results and additional experiments clearly show that the performance gap between MMAS and MMAS* on plateau functions is huge.

5.3 Hybridization of Ant Colony Optimization with Local Search

Often ACO is combined with local search methods (Dorigo and Stützle, 2004; Hoos and Stützle, 2004; Levine and Ducatelle, 2004). Experimental investigations show that the combination of ACO with a local search procedure improves the performance significantly. On the other hand, there are examples where local search cannot help to improve the search process or even mislead the search process (Balaprakash, Birattari, Stützle, and Dorigo, 2006). Therefore, it is interesting to figure out how the incorporation of local search into ACO algorithms can significantly influence the optimization process.

Our aim is to point out situations where the effect of local search becomes visible in a way that can be tackled by rigorous arguments. Therefore we present functions where MMAS variants with and without local search show a strongly different runtime behavior. On one function, MMAS with local search outperforms MMAS without local search, while on a different function the effect is reversed. The differences shown in this section are so drastic that the question of whether to use local search or not decides between polynomial and exponential runtimes.

The outline of this section is as follows. In Section 5.3.1, we define MMAS variants with and without local search. Section 5.3.2 discusses different effects that a combination of ACO and local search can have. In Section 5.3.3, we present a rigorous analysis showing the benefits of such a combination. Contrarily, in Section 5.3.4 we investigate a different function and prove the opposite effect. We finish with some conclusions.

5.3.1 Algorithms

We consider the runtime behavior of two ACO algorithms. MMAS* has already been defined in Section 5.2. We enhance MMAS* with local search and call the result MMAS-LS*. In the following, local search(x) is a procedure that, starting from x, repeatedly replaces the current solution by a Hamming neighbor with strictly larger fitness

until a local optimum is found. We do not specify a pivot rule, hence we implicitly deal with a class of algorithms.

Algorithm 18 MMAS-LS*

Set $\tau(e) := 1/2$ for all $e \in E$.
Construct a solution x^*.
Set $x^* :=$ local search(x^*).
Update pheromones w. r. t. x^*.
repeat
    Construct a solution x.
    Set $z :=$ local search(x).
    if $f(z) > f(x^*)$ **then** $x^* := z$.
    Update pheromones w. r. t. x^*.

The fitness functions considered in the following only have a linear number of fitness values, hence the number of iterations in one local search call is bounded by $O(n)$. Depending on the pivot rule, the number of fitness evaluations needed to find a better Hamming neighbor may vary; however, it is trivially bounded by n. Hence, the number of function evaluations is at most by a factor $O(n^2)$ larger than the number of generations.

We consider as performance measure the number of generations, also referred to as optimization time. This yields an advantage for MMAS-LS* w. r. t. fitness evaluations. However, the upcoming performance gaps are between polynomial and exponential values, and an advantage of order n^2 is negligible.

5.3.2 The Effect of Hybridizing ACO with Local Search

The effect of using local search with ACO algorithms is manifold. Firstly, local search can help to find good solutions more quickly as it increases the "greediness" within the algorithm. Similar to memetic evolutionary algorithms, local search can also be used to discover the real "potential" of a solution as it can turn a bad looking solution into a good local optimum. Moreover, the pivot rule used in local search may guide the algorithm towards certain regions of the search space. For example, first ascent pays more attention to the first bits in the bit string, which may induce a search bias. However, we will not deal with this effect here. In particular, our functions are designed such that the pivot rule is not essential.

There is, however, another effect that we want to investigate more closely. The pheromone values induce a sampling distribution over the search space. On a typical fitness landscape, once the best-so-far solution has reached a certain quality, sampling new solutions with a high variance becomes inefficient and the current best-so-far solution x^* is maintained for some time. The analyses from Section 5.2 have shown that then

the pheromones quickly reach the upper and lower bounds corresponding to x^*. This means that the algorithm turns to sampling close to x^*. In other words, MMAS variants typically reach a situation where the "center of gravity" of the sampling distribution follows the current best-so-far solution and the variance of the sampling distribution is low.

When introducing local search into an MMAS algorithm, this may not be true. Local search is able to find local optima that are far away from the current best-so-far solution. In this case the "center of gravity" of the sampling distribution is far away from the best-so-far solution.

Assume there is a path of Hamming neighbors with increasing fitness leading to a local optimum. Assume further that all points close to the path have lower fitness. Then for MMAS* it is likely that the sampling distribution closely follows the path. The path of increasing fitness need not be straight. In fact, it can make large bends through the search space until a local optimum is reached. On the other hand, MMAS-LS*, when starting with the same setting, will reach the local optimum within a single iteration of local search. Then the local optimum becomes the new best-so-far solution x^*, while the sampling distribution is still concentrated around the starting point. In the following generations, as long as the best-so-far solution is not exchanged, the pheromone values on all bits synchronously move towards their respective bounds in x^*. This implies for the sampling distribution that the "center of gravity" takes a (sort of) direct route towards the local optimum, irrespective of the bent path taken by local search. An illustration is given in Figure 5.8.

Consequences are that different parts of the search space are sampled by MMAS* and MMAS-LS*, respectively. Moreover, with MMAS* the variance in the solution construction is always quite low as the sampling distribution is concentrated on certain points on the path. But when the best-so-far solution with local search suddenly moves a long distance, the variance in the solution construction may be very high as the bits differing between the starting point and x^* may have pheromones close to $1/2$. These bits are assigned almost randomly. This strongly resembles a uniform crossover operation from evolutionary computation. There, every bit in the offspring receives a bit value from a parent that is chosen uniformly at random and anew for each bit, which implies that bits differing in the two parents are assigned randomly. MMAS-LS* in this setting therefore simulates a uniform crossover of the starting point and the local optimum x^*.

Our aim in the following is to create functions where MMAS* and MMAS-LS* have a different runtime behavior. Moreover, we want the performance difference to be drastic in order to show how deep the impact of local search can possibly be. To this end, we exploit that the sampling distributions can follow different routes through the search space. For one function we place a target region with many global optima on the straight line between starting point and local optimum and turn the local optimum into a trap that is hard to overcome. In such a setting, we expect MMAS-LS* to drastically outperform MMAS*. These ideas are made precise in Section 5.3.3. On the other hand,

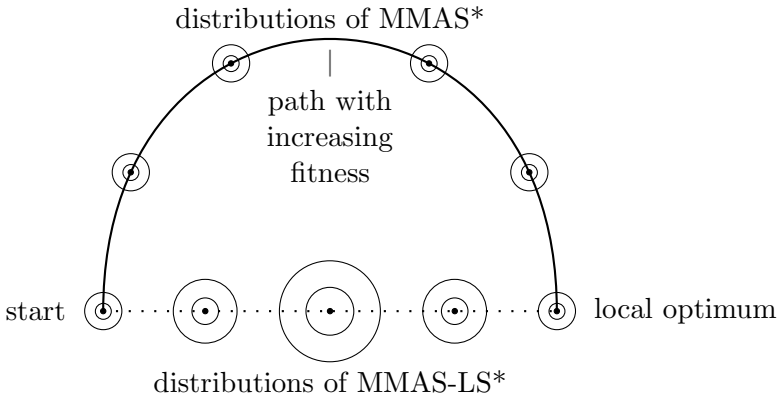

Figure 5.8: A sketch of the search space showing the behavior of MMAS* and MMAS-LS*. The dots and circles indicate the sampling distributions of MMAS* and MMAS-LS*, resp., at different points of time. While the distribution of MMAS* tends to follow the fitness-increasing path from left to right, the distribution of MMAS-LS* takes a direct route towards the local optimum.

if the region of global optima is made a region of traps and the global optimum is very close to the local optimum, MMAS* has a clear advantage over MMAS-LS*. Another function following this idea is defined and analyzed in Section 5.3.4.

5.3.3 A Function where Local Search is Beneficial

We now formally define a function where local search is beneficial according to the ideas from Section 5.3.2. It is named SP-Target (short path with target). The path with increasing fitness is given by the set $\mathrm{SP} = \{1^i 0^{n-i} \mid 0 \leq i \leq n\}$. The path ends with the local optimum 1^n. A large target area containing all global optima is specified by $\mathrm{OPT} = \{x \mid |x|_1 \geq (3/4) \cdot n \wedge \mathrm{H}(x, \mathrm{SP}) \geq n/(\gamma \log n)\}$, where $\mathrm{H}(x, \mathrm{SP})$ denotes the Hamming distance of x to the closest search point of SP and $\gamma \geq 1$ is a constant to be chosen later. For all remaining search points, the function SP-Target gives hints to reach 0^n, the start of the path SP.

$$\text{SP-Target}(x) := \begin{cases} |x|_0 & \text{if } x \notin (\mathrm{SP} \cup \mathrm{OPT}), \\ n + i & \text{if } x = 1^i 0^{n-i} \in \mathrm{SP}, \\ 3n & \text{if } x \in \mathrm{OPT}\,. \end{cases}$$

The function SP-Target is sketched in Figure 5.9. Note that we have actually defined a class of functions dependent on γ. All following results will hold for arbitrary constant $\gamma \geq 1$ unless stated otherwise.

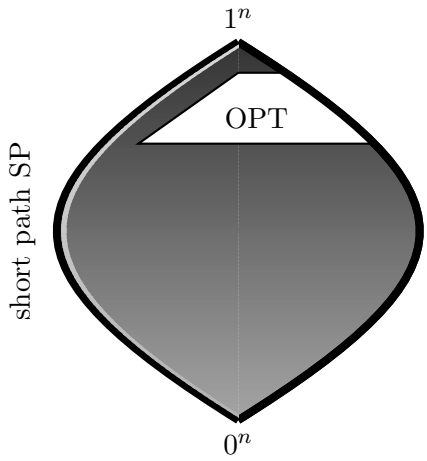

Figure 5.9: Illustration of the Boolean hypercube and the function SP-Target. The vertical position of a search point is determined by the number of 1-bits. Its horizontal position is determined by the position of 1-bits in the bit string. The objective value is indicated by the brightness; dark areas indicate low values and light areas indicate high values.

The following theorem shows that MMAS* without local search is not successful. We restrict ourselves to polynomially large $1/\rho$ here and also in the following as otherwise the ACO component would be too close to random search.

Theorem 5.3.1. *Choosing* $\rho = 1/\text{poly}(n)$*, the optimization time of* MMAS* *on* SP-Target *is at least* $2^{cn^{2/9}}$ *with probability* $1 - 2^{\Omega(n^{2/9})}$ *for some constant* $c > 0$*.*

To prove the preceding theorem, we have to take into account situations where the pheromone values of MMAS* have not yet reached their bounds and the construction procedure samples with high variance. This is the case in particular after initialization. The following lemma will be used to check the probability of finding the optimum in the early steps of MMAS* on SP-Target.

Lemma 5.3.2. *If the best-so-far solution of* MMAS* *has never had more than* $2n/3$ 1*-bits, the probability of creating a solution with at least* $3n/4$ 1*-bits is* $2^{-\Omega(n)}$ *in each generation.*

Proof. The proof is an application of Chernoff bounds w. r. t. the number of ones in the solutions created by MMAS*. Let the potential $P_t := p_1 + \cdots + p_n$ at time t denote the

current sum of the probabilities of sampling ones over all bits, which, by definition of the construction procedure, equals the expected number of ones in the next constructed solution. Observe that $P_t \leq 2n/3$ implies by Chernoff bounds that the probability of creating a solution with at least $3n/4$ 1-bits is $2^{-\Omega(n)}$. We now show: if all best-so-far solutions up to time t have at most $2n/3$ ones, then $P_i \leq 2n/3$ for $0 \leq i \leq t$. This will prove the lemma.

For the last claim, we denote by k the number of ones in the best-so-far solution according to which pheromones are updated. Due to the pheromone update mechanism, the new potential P_{i+1} is obtained from P_i and k according to $P_{i+1} = (1-\rho)P_i + k\rho$. Hence, if $P_i \leq 2n/3$ and $k \leq 2n/3$ then also $P_{i+1} \leq 2n/3$. The claim follows by induction since $P_0 = n/2 \leq 2n/3$. □

Proof of Theorem 5.3.1. We distinguish two phases in the run according to the best-so-far solution x^*. Phase 1 holds as long as $x^* \notin \mathrm{SP}$ and $|x^*|_1 \leq 2n/3$, and Phase 2 applies as long as $x^* \in \mathrm{SP}$. Our aim is to show that a typical run passes through the two phases in their order with a failure probability of $2^{-\Omega(n^{2/9})}$. The probability of finishing the second phase will be bounded by $2^{-\Omega(n^{2/9})}$ for each step of the phase. This implies the theorem as, by the union bound, the total probability in $2^{cn^{2/9}}$ generations, $c > 0$ a small constant, is still $2^{-\Omega(n^{2/9})}$.

Consider the first (and best-so-far) solution x^* created by MMAS*. By Chernoff bounds, $n/3 \leq |x^*|_1 \leq 2n/3$ with probability $1 - 2^{-\Omega(n)}$. There is only a single solution in SP for each value of $|x^*|_1$. By the symmetry of the construction procedure, we conclude $\mathrm{Prob}(x^* \in \mathrm{SP} \mid |x^*|_1 = k) = 1/\binom{n}{k}$. The last expression is $2^{-\Omega(n)}$ for $n/3 \leq k \leq 2n/3$. Hence, with probability $1 - 2^{-\Omega(n)}$, there is a non-empty Phase 1. By Lemma 5.3.2, the probability that a specific generation in Phase 1 creates an optimum is $2^{-\Omega(n)}$. Otherwise, the behavior is as for MMAS* on the function $|x|_0$. Using $\rho = 1/\mathrm{poly}(n)$ and the analyses for the symmetric function $|x|_1$ from Section 5.2.2, the expected time until the first phase is finished is polynomial. By Lemma 2.3.6, the total failure probability in Phase 1 is bounded by the product of its expected length and the failure probability in a single generation. Therefore, the total failure probability for the first phase is still of order $2^{-\Omega(n)}$.

In Phase 2 we have $x^* \in \mathrm{SP}$. The goal is now to show that a solution from SP with high probability can only be created if the sampling distribution is sufficiently concentrated around solutions in SP. This in turn makes creating solutions of high Hamming distance from SP, including OPT, very unlikely.

We make this idea precise and consider a point $1^i0^{n-i} \in \mathrm{SP}$. This search point consists of a prefix of i ones and a suffix of $n-i$ zeros. For a newly constructed solution x we define $P(i) := p_1 + \cdots + p_i$ as the expected number of ones in the prefix and $S(i) := (1-p_{i+1}) + \cdots + (1-p_n)$ as the expected number of zeros in the suffix. The number of ones in the prefix plus the number of zeros in the suffix yields the number of bits equaling in 1^i0^{n-i} and x, i. e., $n - \mathrm{H}\left(1^i0^{n-i}, x\right)$. We call $P(i)$ ($S(i)$) *insufficient* if

and only if $P(i) \leq i - i^{2/3}$ ($S(i) \leq (n-i) - (n-i)^{2/3}$) holds. We now show that with insufficiencies it is very unlikely to create 1^i0^{n-i}. As this holds for all i, we conclude that if SP is reached after a certain number of generations, the pheromones do not have insufficiencies, with high probability.

Let $s(i)$ denote the probability of constructing the solution 1^i0^{n-i}. We distinguish three cases and apply Chernoff bounds to prove the following implications:

Case 1: $i < n^{2/3}$. Then insufficient $S(i)$ implies $s(i) = 2^{-\Omega(n^{1/3})}$.

Case 2: $i > n - n^{2/3}$. Then insufficient $P(i)$ implies $s(i) = 2^{-\Omega(n^{1/3})}$.

Case 3: $n^{2/3} \leq i \leq n - n^{2/3}$. Then insufficient $P(i)$ and insufficient $S(i)$ each imply $s(i) = 2^{-\Omega(n^{2/9})}$.

We assume that the described insufficiencies do not occur whenever a best-so-far solution $x^* = 1^i0^{n-i}$ in Phase 2 is accepted. The failure probability is $2^{-\Omega(n^{2/9})}$ for each new best-so-far solution x^*. Generations in between two exchanges of x^* cannot create insufficiencies as $P(i)$ and $S(i)$ can only increase as long as x^* is maintained. Hence, we do not have insufficiencies in Phase 2 for at least $2^{\Omega(n^{2/9})}$ generations with probability at least $1 - 2^{-\Omega(n^{2/9})}$.

Being in Phase 2 without insufficiencies, we show depending on the three cases for the current $x^* = 1^i0^{n-i}$ that creating an optimal solution has probability $2^{-\Omega(n^{2/9})}$. In the first case, the expected number of zeros in the suffix of x is at least $(n-i) - (n-i)^{2/3}$. By Chernoff bounds, the random number of zeros is at least $(n-i) - 2(n-i)^{2/3}$ with probability at least $1 - 2^{-\Omega(n^{1/3})}$. Along with $i < n^{2/3}$, it follows that then the solution has Hamming distance at most $3n^{2/3}$ from SP. By the definition of SP-Target, this is not enough to reach OPT. The second case is treated analogously. In the third case, the probability of obtaining less than $i - 2i^{2/3}$ ones in the prefix or less than $(n-i) - 2(n-i)^{2/3}$ zeros in the suffix is altogether bounded by $2^{-\Omega(n^{2/9})}$. Then the solution has Hamming distance at most $4n^{2/3}$ from SP, which is also not enough to reach the optimum. This finishes the analysis of the second phase, and, therefore, proves the theorem. □

The following theorem proves the benefits of local search. Recall that the number of fitness evaluations is at most by a factor of $O(n^2)$ larger than the stated optimization time.

Theorem 5.3.3. *Choosing* $1/\mathrm{poly}(n) \leq \rho \leq 1/16$*, the optimization time of* MMAS-LS* *on* SP-Target *is* $O(1/\rho)$ *with probability* $1 - 2^{-\Omega(n)}$*. If* $\gamma \geq 1$ *is chosen large enough but constant, the expected optimization time is also* $O(1/\rho)$*.*

Proof. Note that every call of local search ends either with 1^n or with a global optimum. If the initial local search creates $x^* = 1^n$, all pheromone values increase simultaneously and uniformly from their initial value $1/2$ towards their upper bound $1 - 1/n$. We divide a run into two phases. The first phase ends when either all pheromones become larger than $27/32$ or when a global optimum has been found. The second phase ends when

a global optimum has been found, hence it is empty if the first phase ended with an optimum.

We bound the length of the first phase by the first point of time t^* where all pheromone values exceed 27/32. By Lemma 5.1.2 after t steps the pheromone values are at least $\min\{1-1/n,\ 1-(1/2)(1-\rho)^t\}$. Solving the equation

$$1-\left(\frac{1}{2}\right)(1-\rho)^t \ = \ 27/32 \iff (1-\rho)^t \ = \ 5/16$$

yields the upper bound

$$t^* \ \leq \ \left\lceil \frac{\ln(5/16)}{\ln(1-\rho)} \right\rceil \ \leq \ \frac{\ln(16/5)}{\rho}+1 \ = \ O(1/\rho).$$

The assumption $\rho \leq 1/16$ implies that at the last step in the first phase the pheromone value at every bit is within the interval $[25/32, 27/32]$, pessimistically assuming that a global optimum has not been found before (neither by a constructed ant solution, nor by local search). The next constructed search point x then fulfills the following two properties with probability $1-O(2^{-n/2400})$:

1. $\frac{3n}{4} \leq |x|_1 \leq \frac{7n}{8}$,
2. $\mathrm{H}(x, \mathrm{SP}) \geq n/(\gamma \log n)$.

Using Chernoff bounds with $\delta := 1/25$, the failure probability for the first event is at most $2e^{-(25n/32)(\delta^2/3)} = 2e^{-n/2400}$. To bound the failure probability of the second event, given the first event, we exploit that all pheromone values are equal. Therefore, if we know that $|x|_1 = k$ then x is uniform over all search points with k ones. Since the number of search points with k ones is monotone decreasing for $3n/4 \leq k \leq 7n/8$, we only consider search points with $k = 7n/8$ ones as a worst case. The number of such search points is $\binom{n}{n/8}$, and the number of search points of Hamming distance at most $m := n/(\gamma \log n)$ from SP is at most $m \cdot \binom{n}{m}$. Altogether, the probability of $\mathrm{H}(x, \mathrm{SP}) \leq m$, given that $3n/4 \leq |x|_1 \leq 7n/8$, is bounded from above by

$$\frac{m\binom{n}{m}}{\binom{n}{n/8}} \ \leq \ \frac{m\left(\frac{en}{m}\right)^m}{\left(\frac{n}{n/8}\right)^{n/8}} \ \leq \ m \cdot 2^{o(n)} \cdot 8^{-n/8}.$$

The last expression is even $O(2^{-n/8})$. Altogether, the sum of the failure probabilities is $O(2^{-n/2400})$ as suggested, and the first statement follows.

For the second statement we estimate the time in the second phase, provided that the first phase has been unsuccessful. Using the bound $(\ln n)/\rho$ on the expected freezing time from Section 5.2.2 and $\rho = 1/\mathrm{poly}(n)$, the time to reach the pheromone bound is

$O((\log n)/\rho) = \text{poly}(n)$, or an optimum is created anyway. With all pheromones at the upper bound, the solution construction process equals a standard mutation of 1^n, i. e., flipping each bit in 1^n independently with probability $1/n$. Flipping the first m bits results in a global optimum as 0^m1^{n-m} has Hamming distance at least m to 1^i0^{n-i} for every i. The probability of creating 0^m1^{n-m} in a standard mutation is at least

$$\left(\frac{1}{n}\right)^{n/(\gamma \log n)} \left(1 - \frac{1}{n}\right)^{n - n/(\gamma \log n)} \geq e^{-1} \cdot 2^{-n/\gamma}.$$

This means that the expected time in the second phase is $O(\text{poly}(n)2^{n/\gamma})$. Using that the first phase is unsuccessful only with probability $O(2^{-n/2400})$ and applying the law of total probability, the expected optimization time altogether is $O(1/\rho) + O(2^{-n/2400}) \cdot O(\text{poly}(n)2^{n/\gamma})$. The latter is $O(1)$ for $\gamma > 2400$, which proves the second statement. □

5.3.4 A Function where Local Search is Detrimental

Similarly to the function SP-Target, we design another function SP-Trap (short path with trap) where local search is detrimental, using ideas from Section 5.3.2. We take over the path with increasing fitness, $\text{SP} = \{1^i0^{n-i} \mid 0 \leq i \leq n\}$, but in contrast to SP-Target, the former region of global optima now becomes a trap, $\text{TRAP} = \{x \mid |x|_1 \geq (3/4) \cdot n \wedge \text{H}(x, \text{SP}) \geq n/\log n\}$. The unique global optimum is placed within distance 2 from the local optimum: $\text{OPT} = \{0^21^{n-2}\}$. This ensures that local search climbing the path SP cannot reach the global optimum. All remaining search points give hints to reach the start of the path.

$$\text{SP-Trap}(x) := \begin{cases} |x|_0 & \text{if } x \notin (\text{SP} \cup \text{TRAP} \cup \text{OPT}), \\ n + i & \text{if } x = 1^i0^{n-i} \in \text{SP}, \\ 3n & \text{if } x \in \text{TRAP}, \\ 4n & \text{if } x \in \text{OPT}. \end{cases}$$

The function SP-Trap is sketched in Figure 5.10.

In the remainder of this section, we prove that MMAS* is efficient on SP-Trap while MMAS-LS* fails dramatically. Tuning the definition of SP-Trap, we could also extend the following theorem by a polynomial bound on the expected optimization time. We refrain from such modifications to illustrate the main effects.

Theorem 5.3.4. *Choosing $\rho = 1/\text{poly}(n)$, the optimization time of* MMAS* *on* SP-Trap *is $O((n \log n)/\rho + n^3)$ with probability $1 - 2^{-\Omega(n^{2/9})}$.*

Proof. By the argumentation from the proof of Theorem 5.3.1, the probability that a solution in TRAP is produced within $O(n^3)$ generations is at most $2^{-\Omega(n^{2/9})}$.

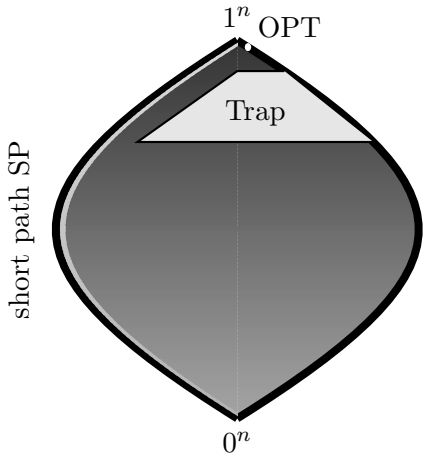

Figure 5.10: Illustration of the Boolean hypercube and the function SP-Trap. The vertical position of a search point is determined by the number of 1-bits. Its horizontal position is determined by the position of 1-bits in the bit string. The objective value is indicated by the brightness; dark areas indicate low values and light areas indicate high values.

Under the assumption that TRAP is never reached until the global optimum is found, MMAS* behaves equally on SP-Trap and a modified function where $x \in \text{TRAP}$ receives fitness $|x|_0$. We apply the fitness-level method described in Section 5.2.2 to estimate the expected optimization time on the latter, easier function. The number of fitness levels is $O(n)$. On every fitness level, the number of generations until either all pheromones are frozen or the current best-so-far solution has improved is bounded by $(\ln n)/\rho$ with probability 1. We pessimistically assume that an improvement can only happen once all pheromones have been frozen. Then the optimization time is bounded by $O((n \log n)/\rho)$ plus the sum of waiting times for improvements on all fitness levels. Showing that the latter quantity is bounded by $O(n^3)$ with probability $1 - 2^{-\Omega(n)}$ completes the proof.

After freezing, the solution construction process equals a standard mutation of the best-so-far solution x^*. The probability for an improvement from $x^* = 1^n$ is at least $1/(en^2)$. For all other $x^* \notin \text{TRAP}$, there is always a better Hamming neighbor, hence the probability for an improvement is at least $1/(en)$. Together, the expected waiting times for improvements on all fitness levels sum up to $en^2 + O(n) \cdot en = O(n^2)$. By Markov's inequality the probability of waiting more than cn^2 steps is at most $1/2$ for a suitable constant $c > 0$. Hence, the probability that more than n independent phases of length cn^2 are needed is bounded by $2^{-\Omega(n)}$. Therefore, the bound $O(n^3)$ holds with probability $1 - 2^{-\Omega(n)}$. □

Theorem 5.3.5. *Choosing* $1/\text{poly}(n) \leq \rho \leq 1/16$*, the optimization time of* MMAS-LS* *on* SP-Trap *is at least* 2^{cn} *with probability* $1 - 2^{-\Omega(n)}$ *for some constant* $c > 0$*.*

Proof. We follow the lines of the proof of Theorem 5.3.3. As long as $\text{OPT} = 0^2 1^{n-2}$ is not created, the behavior of MMAS-LS* on SP-Trap and SP-Target is identical. Reconsider the first phase described in the proof of Theorem 5.3.3 (with the former OPT replaced by TRAP) and denote by $P := p_1 + \cdots + p_n$ the sum of probabilities of sampling ones over all bits. Throughout the phase, $P \leq 27n/32$, hence the probability of sampling at least $n - 2$ ones, which is necessary to reach OPT, is $2^{-\Omega(n)}$ according to Chernoff bounds.

With probability $1 - 2^{-\Omega(n)}$, the first best-so-far solution 1^n is replaced by some $x^{**} \in \text{TRAP}$ where $|x^{**}|_1 \leq 7n/8$ when the first phase is ended. Due to strict selection, x^{**} then can only be replaced if OPT is created. The latter has probability $2^{-\Omega(n)}$ for the following reasons: the P-value is at most $27n/32 \leq 7n/8$ when x^{**} is accepted. Hence, following the argumentation from the proof of Lemma 5.3.2, the P-value will not exceed $7n/8$ unless x^{**} is replaced. With a P-value of at most $7n/8$, creating OPT has probability $2^{-\Omega(n)}$ and the claim follows for a suitable constant $c > 0$. □

5.3.5 Conclusions

We have investigated the combination of ACO and local search from a theoretical point of view and pointed out how this combination can influence the search process. In particular, we have rigorously shown that the combination of both methods can outperform ACO algorithms not using local search procedures. Furthermore, we have proven that the combination of ACO and local search may mislead the search process.

Our results represent a first step in the runtime analysis of hybridizations of ACO algorithms. The runtime analysis of ACO has grown from the analysis of simple test functions like OneMax and LeadingOnes to more complex artificial functions. The proofs also show that the method of typical runs (cf. Section 2.3.3) can be applied for ACO algorithms. We expect other techniques to be applicable for ACO algorithms as well. Moreover, future work should focus on applying the techniques developed for the analysis of ACO algorithms to problems from combinatorial optimization.

5.4 Analysis of a Binary Particle Swarm Optimizer

We now turn from ACO to PSO and investigate the runtime of the Binary Particle Swarm Optimizer (Binary PSO) introduced by Kennedy and Eberhart (1997) for optimizing pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$. Recall from the introduction to this chapter that the Binary PSO maintains a swarm of particles searching for good solutions. Each particle consists of a current position from $\{0,1\}^n$, an own best position and a velocity vector used in a probabilistic process to update its current position. The velocities for a

particle are then updated in the direction of its own best position and the best position found by the swarm.

In the original formulation all velocities are clamped to a fixed interval $[-v_{\max}, v_{\max}]$ to prevent divergence of the system. It is recommended to choose $v_{\max}$ as a constant value. We prove in Section 5.4.1 that the effect on the performance is disastrous if the velocity bound $v_{\max}$ is fixed while the problem size grows. Instead, we present a formulation of the Binary PSO with $v_{\max}$ adjusted towards growing problem dimensions. The new choice of $v_{\max}$ leads to provably efficient optimization times without alternative approaches for velocity control as described, e. g., by Bratton and Kennedy (2007).

In Section 5.4.2 we present lower bounds on the runtime of the Binary PSO. Section 5.4.3 shows how fitness-level arguments can be used for the analysis of the Binary PSO. We exemplarily apply this technique to the class of unimodal functions. In Section 5.4.4, we consider a specific variant of the Binary PSO in more detail. The 1-PSO works with a swarm consisting of only one particle. Despite its simplicity, the 1-PSO turns out to be surprisingly efficient. A thorough analysis on the function OneMax in Section 5.4.4 shows that the 1-PSO is competitive to the (1+1) EA. This theoretical result is supplemented in Section 5.4.5 by an experimental comparison of these two algorithms on OneMax. We conclude in Section 5.4.6.

5.4.1 The Binary PSO

We consider the Binary PSO by Kennedy and Eberhart (1997) for the optimization of a pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$. The Binary PSO algorithm maintains μ triples $(x^{(i)}, x^{(i)*}, v^{(i)})$, $1 \leq i \leq \mu$, denoted as particles. Each particle i consists of its current position $x^{(i)} \in \{0,1\}^n$, its own best position $x^{(i)*} \in \{0,1\}^n$ and its velocity $v^{(i)} \in \mathbb{R}^n$. Note that the velocity is from a continuous domain. In PSO terminology, the three components of a particle are often called vectors. Using the language of optimization, we will refer to particle positions synonymously as solutions.

The movement for each particle is influenced by the best particle in its neighborhood. Hence, depending on the neighborhood structure, different particles may be guided by different good solutions. In this section, however, we only consider a global neighborhood consisting of the whole swarm. This model is known as *gbest* model and it implies that all particles are influenced by a single global best position, denoted as x^{**}. For simplicity, we hereinafter use the term *global best* and the symbol x^{**} both for the global best position and the particle whose own best position determined the global best position.

The velocities are updated as follows. The velocity vector is changed towards the particle's own best and towards the global best x^{**}. Using the language of social-psychology, the first component is often called cognitive component and the latter is often called social component. The impact of these two components is determined by so-called *learning factors* $c_1, c_2 \in \mathbb{R}_0^+$ representing parameters of the system. The factor

c_1 is the learning factor for the cognitive component and c_2 is the one for the social component. A common choice for the learning factors is to set $c_1 = c_2 = 2$.

We give a precise definition for the Binary PSO with a swarm size of μ and learning factors c_1, c_2. By lower indices we address the n components of the three parts of the particle.

Algorithm 19 Binary PSO

Initialize velocities with 0^n and all solutions with $\perp$.
repeat
  Choose $r_1 \in (U[0, c_1])^n$ and $r_2 \in (U[0, c_2])^n$.
  for $j := 1$ to μ **do** // *update current*
    for $i := 1$ to n **do** *and best positions*
      Set $x_i^{(j)} := 1$ with probability $s(v_i^{(j)})$, otherwise set $x_i^{(j)} := 0$.
    if $f(x^{(j)}) > f(x^{(j)*})$ or $x^{(j)*} = \perp$ **then** $x^{(j)*} := x^{(j)}$.
    if $f(x^{(j)*}) > f(x^{**})$ or $x^{**} = \perp$ **then** $x^{**} := x^{(j)*}$.
  for $j := 1$ to μ **do** // *update velocities*
    Set $v^{(j)} := v^{(j)} + r_1 \otimes (x^{(j)*} - x^{(j)}) + r_2 \otimes (x^{**} - x^{(j)})$.
    for $i := 1$ to n **do** // *restrict velocities*
      Set $v_i^{(j)} := \max\{v_i^{(j)}, -v_{\max}\}$. *to* $[-v_{\max}, v_{\max}]$
      Set $v_i^{(j)} := \min\{v_i^{(j)}, v_{\max}\}$.

The algorithm starts with an initialization step, where all velocities are set to all-zeros vectors and all solutions, including own best and global best, are undefined, represented by the symbol $\perp$. The subsequent loop chooses anew in each iteration random vectors $r_1 \in (U[0, c_1])^n$ and $r_2 \in (U[0, c_2])^n$ with each component drawn independently and uniformly from the given interval. These values are then used as weights for the cognitive and the social component, respectively. Using the language of evolutionary algorithms, we refer to iterations synonymously as generations.

In the next step of the loop, the velocity is probabilistically translated into a new position for the particle, i. e., a new solution. As proposed in the original formulation, we use the sigmoid function

$$s(v) := \frac{1}{1 + e^{-v}}$$

(see Figure 5.2 for a sketch). Positive velocity components bias the corresponding bit towards 1-values while negative velocities favor 0-values. At velocity 0^n, each bit is completely random, hence the first created solution is uniformly distributed over $\{0, 1\}^n$.

Afterwards, the own best and global best are exchanged if the newly constructed solution is better. Note that the selection is strict, i. e., a best solution is only exchanged in case the new solution has strictly larger fitness.

In the following step the Binary PSO performs some vector arithmetic to update the velocity vectors probabilistically in the direction of the particle's own best and the global best. The operator $\otimes$ denotes componentwise vector multiplication. To ensure convergence of the heuristic, every velocity vector is bounded componentwise by minimum and maximum values, i. e., to an interval $[-v_{\max}, v_{\max}]$. This reflects the common choice of a maximum velocity as studied by Shi and Eberhart (1998). For practical purposes, often velocities in the interval $[-4, 4]$ are proposed. Since we will however conduct an asymptotic analysis, we allow the maximum velocity to grow with the problem dimension n and confine the components to logarithmic values by letting $v_{\max} := \ln(n-1)$. We will justify this choice later.

As performance measures we consider the number of generations as well as the number of function evaluations. We will deal with different parametrizations of the Binary PSO, differing in the swarm size μ and the learning factors c_1 and c_2. In particular, we deal with a remarkably simple yet effective algorithm, the so-called 1-PSO using just one particle. With just one particle, own best and global best coincide. Therefore, it makes sense to turn off the social component by setting $c_2 = 0$. The cognitive learning factor is set to the default value $c_1 = 2$. Note that the same algorithm is described by the choice $c_1 = 0$ and $c_2 = 2$. Dropping the upper index in the notation, the 1-PSO can be stated as follows.

Algorithm 20 1-PSO

Initialize $v := 0^n$ and $x^{**} := \perp$.
repeat
  Choose $r \in (U[0,2])^n$.
  for $i := 1$ to n **do** // *update current and best position*
    Set $x_i := 1$ with probability $s(v_i)$, otherwise set $x_i := 0$.
  if $f(x) > f(x^{**})$ or $x^{**} = \perp$ **then** $x^{**} := x$.
  Set $v := v + r \otimes (x^{**} - x)$. // *update velocities*
  for $i := 1$ to n **do** // *restrict velocities to* $[-v_{\max}, v_{\max}]$
    Set $v_i := \max\{v_i, -v_{\max}\}$.
    Set $v_i := \min\{v_i, v_{\max}\}$.

Note that during the velocity update in expectation just $x^{**} - x$ is added to v.

There are several good reasons to investigate the 1-PSO. One is that in the Binary PSO without social component, i. e., with $c_2 = 0$, all particles behave like independent instances of the 1-PSO. Moreover, by analyzing the 1-PSO we gain insight into the

probabilistic model underlying the Binary PSO. This then helps to analyze more complex PSO variants. Finally, the investigation of the 1-PSO is interesting on its own as the 1-PSO turns out to be surprisingly effective.

Lower Bound for Constant Velocity Range

The value $v_{\max}$ is often set to a constant value. This makes sense when dealing with problems of bounded size. However, one should be aware of the fact that for growing problem sizes a fixed value for $v_{\max}$ leads to an extreme decline in performance. The following lower bound shows that if all velocities are restricted to constant values, then the Binary PSO is too close to random search and the algorithm fails badly, even given exponential time and a large number of global optima.

Theorem 5.4.1. *Consider the Binary PSO with arbitrary values for μ, c_1, and c_2, where $v_{\max}$ is redefined to a constant value. Then there is a constant $\alpha = \alpha(v_{\max})$ such that the following holds. If f contains at most $2^{\alpha n}$ global optima, the probability that the Binary PSO finds a global optimum on f within $2^{\alpha n}$ function evaluations is $2^{-\alpha n}$.*

Proof. Choose α such that $s(v_{\max}) = 2^{-3\alpha}$ and note that α is a positive constant if $v_{\max}$ is constant. We estimate the probability of constructing any specific solution x. Since the Binary PSO treats 0- and 1-bits symmetrically, we can w. l. o. g. assume that x is the all-ones string 1^n. Then, even if all velocities are at $v_{\max}$, the probability of constructing x is still bounded by $(s(v_{\max}))^n = 2^{-3\alpha n}$. By the union bound, the probability of constructing any global optimum out of at most $2^{\alpha n}$ ones is bounded by $2^{\alpha n} \cdot 2^{-3\alpha n} = 2^{-2\alpha n}$. By the same argument, the probability that this happens at least once in $2^{\alpha n}$ solution constructions is at most $2^{\alpha n} \cdot 2^{-2\alpha n} = 2^{-\alpha n}$. □

For the common choice $v_{\max} := 4$, the constant α is approximately 0.00873. As $2^{0.00873 \cdot n}$ is small for small n, the bad runtime behavior can only be observed if the problem size is large enough. This certainly isn't the case for $n = 100$ where $2^{\alpha n} < 2$. However, for a problem size of $n = 10000$, the claimed bound has grown to $2^{\alpha n} > 10^{26}$ and we would not expect to live long enough to see the Binary PSO find an optimum.

Theorem 5.4.1 rules out a constant default value for $v_{\max}$ that works well for all problem sizes. We therefore propose to let $v_{\max}$ scale with the problem size. More precise, we set $v_{\max} = \ln(n-1)$. As $s(-v_{\max}) = 1/n$ and $s(v_{\max}) = 1 - 1/n$, the probability of setting a bit to 1 is always in the interval $[1/n, 1-1/n]$. This corresponds to the pheromone bounds in the ACO algorithms from the preceding sections. It is inspired by standard mutation operators in evolutionary computation, where incorrectly set bits have a probability of $1/n$ of being corrected. We will see in the following that this choice leads to a surprisingly good runtime behavior.

5.4.2 A Lower Bound for the Binary PSO

An important step towards runtime bounds for the Binary PSO is to understand the dynamics of the probabilistic model underlying PSO, that is, the behavior of the velocity vector. Consider a single bit that is set to 1 both in the own best and in the global best. Then, as long as these solutions are not exchanged, its velocity value v is guided towards the upper bound $v_{\max}$. An important observation is that the velocity is only increased in case the bit is set to 0 in the next constructed solution. The probability that this happens is given by

$$1 - s(v) \;=\; 1 - \frac{1}{1+e^{-v}} \;=\; \frac{1}{1+e^{v}},$$

and we see that this probability decreases rapidly with growing v. Hence, the closer the velocity is to the bound $v_{\max}$, the harder it is to get closer. A symmetric argument holds for velocities that are guided towards $-v_{\max}$.

Remark 5.4.2. Note that the mechanism to increase velocities in the Binary PSO differs from the mechanism to increase pheromones in MMAS. In MMAS, the best-so-far solution is automatically reinforced in every generation. If a bit value is fixed to 1, the corresponding success probability is increased with every generation, hence there is a deterministic upper bound for the time until the pheromone bound is hit. In the Binary PSO, the success probability is only increased if an appropriate bit value is sampled, hence the time until the velocity bound is hit is random. If value 1 is a "good" bit value present in own best and global best, then a "bad" bit value 0 has to be sampled to increase the success probability. This may appear counterintuitive at first glance, but it reflects the paradigm of particles moving from their current position towards good solutions.

As long as the v-values are not too close to the velocity bounds $-v_{\max}$ and $v_{\max}$, the search of the Binary PSO is too random for it to find single optima with high probability. We can make this idea precise by the following, general lower bound, which holds for all practical choices of the learning factors c_1 and c_2 and a polynomial swarm size μ. The maximum change of a velocity per generation will be abbreviated by $\bar{c} := c_1 + c_2$ hereinafter.

Theorem 5.4.3. *Let f be a function with a unique global optimum, and let $\mu = \mathrm{poly}(n)$ and $\bar{c} = O(1)$. Then the expected number of generations of the Binary PSO on f is $\Omega(n/\log n)$.*

Proof. W. l. o. g. the global optimum is 1^n. Let $t := \alpha n/\ln n$ for a small constant $\alpha > 0$, which is chosen later. We show that the probability of not creating 1^n within t generations is $1 - o(1)$, which implies the claim of the theorem.

We consider an arbitrary bit in an arbitrary particle. The event of creating a one at this bit is called *success*. Let a bit be called *weak* if its success probability has been at

most $p := 1 - \frac{e \ln(\mu n)}{n}$ up to and including the current generation. Let a set of bits be called weak if it contains only weak bits. We will show that with probability $1 - 2^{-\Omega(n)}$ after t generations of the 1-PSO each particle still contains a weak subset of bits of size at least n/e. The probability of setting all bits of such a weak subset to 1 simultaneously is bounded from above by $p^{n/e} \leq 1/(\mu n)$ for each particle. Note that this event is necessary to create 1^n in a particle. Thus, the probability of finding the optimum within t generations creating μ new solutions each is still less than $t\mu/(\mu n) = O(1/\log n) = o(1)$, which will prove the theorem.

We still have to show that with probability $\Omega(1)$, after t generations, there is a weak subset of size at least n/e in each particle. One step can increase the velocity by at most $\bar{c}$. Note that $p = 1 - O((\ln n)/n)$ since $\mu = \text{poly}(n)$. To reach success probability at least p, the current velocity must be between $s^{-1}(p) - \bar{c} = \ln(p/(1-p)) - \bar{c}$ and $s^{-1}(p)$ at least once. Pessimistically assuming the first value as current velocity, the probability of not increasing it in a single step is at least

$$\frac{1}{1 + e^{-\ln(p/(1-p)) + \bar{c}}} \;=\; 1 - \frac{e^{\bar{c}}(1-p)}{p + e^{\bar{c}}(1-p)} \;\geq\; 1 - 2e^{\bar{c}}(1-p)$$

if n is large enough for $p \geq 1/2$ to hold. The last expression equals $1 - 2e^{\bar{c}+1} \ln(\mu n)/n$ by definition of p. Hence, along with $\bar{c} = O(1)$ and again $\mu = \text{poly}(n)$, the probability of not increasing the velocity within t steps is at least

$$\left(1 - \frac{2e^{\bar{c}+1} \ln(\mu n)}{n}\right)^t \;=\; \left(1 - \frac{O(\ln n)}{n}\right)^{\alpha n / \ln n} \;\geq\; 2e^{-1}$$

if α is chosen small enough. This means that each bit in each particle independently has a probability of at least $2e^{-1}$ of being weak at generation t. Using Chernoff bounds, the probability of not having a weak set of size at least n/e in a specific particle is at most $e^{-\Omega(n)}$. As $\mu = \text{poly}(n)$, the probability that there exists a particle without weak subset at generation t is still $\mu e^{-\Omega(n)} = e^{-\Omega(n)}$. □

5.4.3 Upper Bounds for the Binary PSO

In this section we derive an upper bound for the Binary PSO. The lower bound from Theorem 5.4.3 relied on the fact that a velocity that is guided towards $v_{\max}$ does not reach this value in short time and so the Binary PSO cannot find a single target efficiently. On the other hand, if we consider a longer period of time, velocities may reach the bounds $-v_{\max}$ or $v_{\max}$ they are guided to. In this case we say that the velocity has been "frozen" as the only chance to alter the velocity again is to have an improvement of the own best or the global best. The random time until a bit is frozen is called *freezing time*.

Observe that freezing is only possible for bits where own best and global best coincide or in case one of the learning factors c_1, c_2 is 0. Freezing does not happen if, e. g.,

$c_1 = c_2 > 0$ and own best and global best differ for a bit i. W. l. o. g., the own best has bit value 1 and $x_i^{**} = 0$, then the new velocity v_i' for this bit equals

$$v_i' \;=\; v_i + r_{1,i}(1 - x_i) + r_{2,i}(0 - x_i) \;=\; v_i + r_{1,i} - (r_{1,i} + r_{2,i}) \cdot x_i,$$

where $r_{1,i}$ ($r_{2,i}$) denotes the i-th entry of the random vector r_1 (r_2). This means that if $v_i \gg 0$, there is a good chance that $x_i = 1$ and then $v_i' = v_i - r_{2,i}$. Contrarily, if $v_i \ll 0$, it is likely that $x_i = 0$ and $v_i' = v_i + r_{1,i}$. In both cases the velocity v_i is driven towards velocity value 0. As a side remark, note that if all these bits reach velocities close to 0, the solution construction process is very similar to a uniform crossover between own best and global best solution. So, it makes sense to define the freezing time only for bits whose velocity can freeze at $-v_{\max}$ or $v_{\max}$. We summarize our sufficient conditions for freezing by the following definition.

Definition 5.4.4. A bit $x_i^{(j)}$ of a particle $x^{(j)}$ in the Binary PSO is called *well guided* with respect to a given time interval if at least one of the following conditions is met.

- The corresponding bit in the global best, x_i^{**}, remains fixed in the time interval and $c_1 = 0$.
- The corresponding bit in the own best, $x_i^{(j)*}$, remains fixed in the time interval and $c_2 = 0$.
- Both x_i^{**} and $x_i^{(j)*}$ remain fixed in the time interval and $x_i^{**} = x_i^{(j)*}$.

In order to arrive at an estimation of the expected freezing time, we investigate the random process of velocities moving towards a bound. Assume we are interested in the random hitting time until a velocity has increased towards a value of at least a, for some fixed real value a. Intuitively, we expect a monotonicity to hold: larger initial velocities are always better than smaller ones, in terms of the expected hitting time. However, such a monotonicity does not hold in general. The reason is that the probability of increasing a velocity decreases with the velocity itself and velocities being very close to a can actually be inferior to values that are a little further away.

We give a simple example for the 1-PSO. Consider two bits with current velocity values v_1 and v_2, guided towards $v_{\max}$ with $v_{\max} > 3$. Assume that $v_1 = 2$ and $v_2 = 2.5$ and let V_1', V_2' denote the random velocities after the next step. We compare the probabilities that the random new velocities reach a value of at least 3 in the next step. As the current velocity for v_2 is larger, we expect chances for this bit to be higher. In order to increase the velocity, we have to create value 0 at the bit. The amount of increase is then uniform over $[0, 2]$. Hence we have

$$\mathrm{Prob}\big(V_1' \geq 3\big) \;=\; (1 - s(v_1)) \cdot \frac{1}{2} \;=\; s(-2) \cdot \frac{1}{2} \approx 0.0596.$$

On the other hand,

$$\mathrm{Prob}\big(V_2' \geq 3\big) \;=\; (1 - s(v_2)) \cdot \frac{3}{4} \;=\; s(-2.5) \cdot \frac{3}{4} \approx 0.0569$$

and $\mathrm{Prob}(V_2' \geq 3) < \mathrm{Prob}(V_1' \geq 3)$, in contrast to our expectations. In this example, the larger velocity only seems to have a clear advantage over the smaller velocity. In fact, monotonicity does not hold as the probability of increasing a current velocity of v equals $1 - s(v)$, i. e., decreases rapidly with v.

Nevertheless, our observation does not imply in general that small velocities are superior to large velocities. This only holds at a microscopic level. If, on the other hand, $v_2 \geq v_1 + \bar{c}$ is presupposed in our example (recall that $\bar{c} = c_1 + c_2$ denotes the maximum increase of velocity in a generation), then $\mathrm{Prob}(V_2' \geq a) \geq \mathrm{Prob}(V_1' \geq a)$ holds for every value of a. The reason is that the increase of a velocity is bounded from above by $\bar{c}$, so V_1' cannot "overtake" V_2' in this case. Unfortunately, the corresponding inequality $V_2' \geq V_1' + \bar{c}$ does not necessarily carry over to the next step. Therefore, we take a slightly different viewpoint. If the current velocity V is random and follows a probability distribution on the interval $[-v_{\max}, v_{\max}]$, we study the process $V - \bar{c}$ obtained by decreasing all velocities by the maximum change of velocity, namely $\bar{c}$. Values $v \in [-v_{\max}, v_{\max}]$ with positive probability mass are mapped to $v - \bar{c} \in [-v_{\max} - \bar{c}, v_{\max} - \bar{c}]$. If the modified process decides to move from $v - \bar{c}$ to some higher velocity, it does so with at least the same probability as the original process since its velocity is lower. Although the modified process might take a littler longer to reach the desired value a than the original, it has an advantage that is maintained for the following time steps.

In the following, we introduce a relaxed concept of domination reflecting the preceding considerations. One instance of the velocity-increasing process is allowed to be by up to $\bar{c}$ behind another instance and has an advantage from this perspective. For notational convenience, we add $\bar{c}$ to the velocity of the dominated process rather than subtracting $\bar{c}$ from the dominating one.

Definition 5.4.5. Consider the random velocities of two bits that are well guided, w. l. o. g., to $+v_{\max}$. Denote by $V_1^{(t)}$ and $V_2^{(t)}$ their random velocity values after t generations have elapsed. We say that $V_1^{(t)}$ $\bar{c}$-dominates $V_2^{(t)}$, written as $V_1^{(t)} \succeq_{\bar{c}} V_2^{(t)}$, if for all $a \in \mathbb{R}$ it holds

$$\mathrm{Prob}\Big(V_1^{(t)} \geq a\Big) \;\geq\; \mathrm{Prob}\Big(V_2^{(t)} \geq a + \bar{c}\Big).$$

If one velocity $\bar{c}$-dominates another one at some time, this property will be preserved in the following.

Lemma 5.4.6. *For all $t \geq 0$: $V_1^{(t)} \succeq_{\bar{c}} V_2^{(t)} \;\Rightarrow\; V_1^{(t+1)} \succeq_{\bar{c}} V_2^{(t+1)}$ if the bit is well guided in the time interval $[t, t+1]$.*

Proof. We assume that the distributions of $V_1^{(t)}$ and $V_2^{(t)}$ are continuous. If they are discrete or have discrete components, the following arguments can be proven in the same manner with a slight change of notation.

Suppose that we are at time t and that the current velocity x is sampled according to $V_1^{(t)}$. For any $a \in \mathbb{R}$, we are interested in the event $V_1^{(t+1)} \geq a$. If $x \geq a$, this happens with probability 1 since, by our assumptions, velocities are not decreased. If $x < a - \bar{c}$, the event is excluded since a velocity is increased by at most $\bar{c}$ in a step. Finally, if $x \in [a - \bar{c}, a)$, the Binary PSO has to sample a 0-bit, which happens with probability $s(-x)$, and choose $r_1 + r_2 \geq a - x$ afterwards, where (slightly abusing notation) $r_1 \in U[0, c_1]$ and $r_2 \in U[0, c_2]$. Let $G(s) := \text{Prob}(r_1 + r_2 \geq s)$ denote the distribution function for the sum of r_1 and r_2. Writing $f_1^{(t)}(x)$ for the density function of $V_1^{(t)}$ at x, our considerations yield

$$\text{Prob}\Big(V_1^{(t+1)} \geq a\Big) \;=\; \text{Prob}\Big(V_1^{(t)} \geq a\Big) + \int_{a-\bar{c}}^{a} f_1^{(t)}(x) \cdot s(-x) \cdot G(x - a + \bar{c})\,\mathrm{d}x \quad (5.10)$$

for every $a \in \mathbb{R}$. Analogously, we have

$$\text{Prob}\Big(V_2^{(t+1)} \geq a + \bar{c}\Big) \;=\; \text{Prob}\Big(V_2^{(t)} \geq a + \bar{c}\Big) + \int_{a}^{a+\bar{c}} f_2^{(t)}(x) \cdot s(-x) \cdot G(x - a)\,\mathrm{d}x \quad (5.11)$$

Using an index transformation, we rewrite (5.10) as

$$\text{Prob}\Big(V_1^{(t+1)} \geq a\Big) \;=\; \text{Prob}\Big(V_1^{(t)} \geq a\Big) + \int_{a}^{a+\bar{c}} f_1^{(t)}(x - \bar{c}) \cdot s(-x + \bar{c}) \cdot G(x - a)\,\mathrm{d}x \quad (5.12)$$

as we will concentrate on a comparison of the integrals.

Abbreviating $\Delta^{(t)}(a) := \text{Prob}\Big(V_1^{(t)} \geq a\Big) - \text{Prob}\Big(V_2^{(t)} \geq a + \bar{c}\Big)$, our aim is to show that $\Delta^{(t)}(a) \geq 0$ implies $\Delta^{(t+1)}(a) \geq 0$. We obtain a pessimistic estimation of $\Delta^{(t+1)}(a)$ if we estimate $\text{Prob}\Big(V_2^{(t+1)} \geq a + \bar{c}\Big)$ from above and $\text{Prob}\Big(V_1^{(t+1)} \geq a\Big)$ from below. Since $s(-x + \bar{c}) \geq s(-a) \geq s(-x)$ for $x \in [a, a + \bar{c}]$, we obtain appropriate estimations by replacing all sigmoid terms in (5.12) and (5.11) by $s(-a)$. Hence,

$$\begin{aligned}
\Delta^{(t+1)}(a) \;&=\; \text{Prob}\Big(V_1^{(t+1)} \geq a\Big) - \text{Prob}\Big(V_2^{(t+1)} \geq a + \bar{c}\Big) \\
&\geq\; \Delta^{(t)}(a) + \int_{a}^{a+\bar{c}} (f_1^{(t)}(x - \bar{c}) - f_2^{(t)}(x)) \cdot s(-a) \cdot G(x - a)\,\mathrm{d}x \\
&=\; \Delta^{(t)}(a) + \int_{a}^{a+\bar{c}} g(x) \cdot s(-a) \cdot G(x - a)\,\mathrm{d}x, \qquad (5.13)
\end{aligned}$$

where $g(x) := f_1^{(t)}(x - \bar{c}) - f_2^{(t)}(x)$. Recall that our aim is to bound (5.13) by at least 0. To this end, it is helpful to express also $\Delta^{(t)}(a)$ in terms of the density functions. After

the same index transformation as for (5.12), we obtain

$$\Delta^{(t)}(a) \;=\; \int_{a+\bar{c}}^{v_{\max}+\bar{c}} (f_1^{(t)}(x-\bar{c}) - f_2^{(t)}(x))\,\mathrm{d}x \;=\; \int_{a+\bar{c}}^{v_{\max}+\bar{c}} g(x)\,\mathrm{d}x.$$

Now (5.13) can be written with unified integrals as

$$\begin{aligned}\Delta^{(t+1)}(a) \;&=\; \int_{a+\bar{c}}^{v_{\max}+\bar{c}} g(x)\,\mathrm{d}x + \int_{a}^{a+\bar{c}} g(x)\cdot s(-a)\cdot G(x-a)\,\mathrm{d}x \\ &=\; \int_{a}^{v_{\max}+\bar{c}} g(x)\cdot w(x)\,\mathrm{d}x, \end{aligned} \tag{5.14}$$

where the weight function $w(x)$ is defined by $w(x) := s(-a)\cdot G(x-a)$ for $x \in [a, a+\bar{c}]$ and $w(x) := 1$ for $x \geq a+\bar{c}$. The aim is still to bound the right-hand side in (5.14) by at least 0. This is done by considering portions of the integral and bounding these terms from below. Note that $0 \leq w(x) \leq 1$ for $x \geq a$ and that $w(x)$ increases monotonically in x.

The function $g(x)$ may be positive or negative on certain portions of the integral. In the unweighted case, without $w(x)$, we have for every $b \in \mathbb{R}$

$$\int_{b}^{v_{\max}+\bar{c}} g(x)\,\mathrm{d}x \;=\; \Delta^{(t)}(b-\bar{c}) \;\geq\; 0 \tag{5.15}$$

by our assumptions for time t. This implies that $g(x)$ must be positive for x sufficiently close to $v_{\max}+\bar{c}$. Hence, if there is a root of $g(x)$ in the interval $[a, v_{\max}+\bar{c}]$, then $g(x) \geq 0$ if x is greater than the largest root. If even $g(x) \geq 0$ within the whole interval $[a, v_{\max}+\bar{c}]$ then

$$\int_{a}^{v_{\max}+\bar{c}} g(x)\cdot w(x)\,\mathrm{d}x \;\geq\; \int_{a}^{v_{\max}+\bar{c}} g(x)\,\mathrm{d}x \;\geq\; 0.$$

If $g(x)$ changes signs in the interval, we denote by $r^{**} < r^*$ the largest two roots of $g(x)$ in the interval (defining $r^{**} = a$ if there is only a single root) and recall that $g(x) \geq 0$ for $x \geq r^*$. Here it is crucial that $w(x)$ increases, implying that

$$\begin{aligned}\int_{r^{**}}^{v_{\max}+\bar{c}} g(x)\cdot w(x)\,\mathrm{d}x \;&=\; \underbrace{\int_{r^{**}}^{r^*} g(x)\cdot w(x)\,\mathrm{d}x}_{\leq 0} + \underbrace{\int_{r^*}^{v_{\max}+\bar{c}} g(x)\cdot w(x)\,\mathrm{d}x}_{\geq 0} \\ &\geq\; \int_{r^{**}}^{r^*} g(x)\cdot w(r^*)\,\mathrm{d}x + \int_{r^*}^{v_{\max}+\bar{c}} g(x)\cdot w(r^*)\,\mathrm{d}x \\ &=\; w(r^*)\cdot \int_{r^{**}}^{v_{\max}+\bar{c}} g(x)\,\mathrm{d}x \;\geq\; 0, \end{aligned}$$

the last inequality following from (5.15). If there are further roots (say, r^{***} and r^{****}), this argument can be iterated to yield $\int_{r^{****}}^{r^{**}} g(x) \cdot w(x)\,\mathrm{d}x \geq 0$, which inductively proves $\int_a^{v_{\max}+\bar{c}} g(x) \cdot w(x)\,\mathrm{d}x \geq 0$ for every number of roots. □

We are aiming at an upper bound on freezing times, i. e., the first point of time where a velocity value has reached the bound corresponding to the global best, which, w. l. o. g., is $+v_{\max}$. The concept of domination allows us to take the following worst-case perspective for the velocity value after a number of steps t. We shift the velocity scale up by $\bar{c}$ and arrive at a simplified Markov process $\tilde{v}_t$, $t \geq 0$, called *$\tilde{v}$-process*, defined on $[-v_{\max}+\bar{c}, v_{\max}+\bar{c}]$, as follows. Initially, $\tilde{v}_0 := -v_{\max}+\bar{c}$. For $t \geq 0$, the random state $\tilde{v}_{t+1}$ is obtained as follows:

$$\tilde{v}_{t+1} := \begin{cases} \min\{\tilde{v}_t + r_1 + r_2, v_{\max}+\bar{c}\} & \text{with probability } \frac{1}{1+e^{\tilde{v}_t}} \\ \tilde{v}_t & \text{otherwise,} \end{cases}$$

where $r_1 \in U[0, c_1]$ and $r_2 \in U[0, c_2]$ are drawn independently anew in each step. The reason for shifting the velocity scale is that we are looking for velocities that $\bar{c}$-dominate the $\tilde{v}$-process. Hence, a velocity of at least v_t in the current process at time t will be related to a velocity of at least $v_t + \bar{c}$ in the $\tilde{v}$-process. Note that an increase of the first velocity value happens with probability $1/(1+e^{v_t})$, whereas this probability is only $1/(1+e^{v_t+\bar{c}})$ with the second velocity. However, the ratio of the two probabilities can be bounded according to $(1+e^{v_t})/(1+e^{v_t+\bar{c}}) \geq e^{-\bar{c}}$, i. e., independently of v_t. From an opposite perspective, a (random) velocity $\tilde{v}_t$ of the $\tilde{v}_t$-process corresponds one-to-one to a (random) probability $\tilde{P}_t := 1/(1+e^{-\tilde{v}_t})$ of setting the considered bit to 1 (called success) at time t. At velocity $\tilde{v}_t - \bar{c}$, the actual process has a success probability of only $P_t := 1/(1+e^{-\tilde{v}_t-\bar{c}})$.

We justify why the $\tilde{v}$-process is really a worst case if we are interested in first hitting times for the upper velocity bound $v_{\max}$.

Lemma 5.4.7. *Consider a bit of a particle that is well guided for t generations, w. l. o. g., to $+v_{\max}$, and let v_t and P_t denote its velocity and success probability, respectively, after t generations have elapsed. Then v_t $\bar{c}$-dominates the $\tilde{v}$-process. Moreover, for every $p \geq 0$, $\mathrm{Prob}(P_t \geq 1 - e^{\bar{c}}p) \geq \mathrm{Prob}\Big(\tilde{P}_t \geq 1-p\Big)$, where $\tilde{P}_t$ denotes the success probability of the $\tilde{v}$-process.*

Proof. We prove the first claim by induction. It is obvious that with every initial value $v_0 \in [-v_{\max}, v_{\max}]$ of the real velocity v_0 $\bar{c}$-dominates $\tilde{v}_0 := -v_{\max}+\bar{c}$, the initial value of the $\tilde{v}$-process. Due to Lemma 5.4.6, this property is preserved for all following time steps where the bit is still well guided.

The second property follows from the fact $(1+e^{v_t})/(1+e^{v_t+\bar{c}}) \geq e^{-\bar{c}}$ by means of $1 - 1/(1+e^x) = 1/(1+e^{-x})$. □

Having established the $\tilde{v}$-process as a worst-case perspective for the real velocity process, we can now move towards analyzing first hitting times for the real process. The proof of Lemma 5.4.9 will use Chernoff and Hoeffding bounds. For convenience, we state formulations that have been tailored toward our applications. The first bound is taken over from Lemma A.6, the second one follows from Lemma A.7.

Lemma 5.4.8 (Chernoff and Hoeffding bounds)**.** *Let $X_1, \ldots, X_n$ be independent random variables and denote $X := X_1 + \cdots + X_n$. If $X_i \in \{0,1\}$ for $1 \leq i \leq n$ then for $0 < \delta \leq 1$*

$$\text{Prob}(X \leq (1-\delta)\text{E}(X)) \leq e^{-\text{E}(X)\delta^2/2}.$$

If $0 \leq X_i \leq b$ for $1 \leq i \leq n$, then for $\delta > 0$

$$\text{Prob}(X \leq (1-\delta)\text{E}(X)) \;\leq\; e^{-2\text{E}(X)^2\delta^2/(nb^2)}.$$

The following lemma will be used to bound the expected freezing time for specific bits. We state a more general result that also reveals insight into the distribution of random velocities on their way to the velocity bounds. In fact, the lemma captures the random success probabilities induced by these velocities very precisely, depending on the number of generations t the bit has been given to "evolve" its distribution towards the velocity bound. This will be important later on in Section 5.4.4, when performing a more detailed analysis on the specific function OneMax.

Formally, Lemma 5.4.9 makes a statement about the random probability P_t of setting a bit to 1, after it has been guided towards the velocity bound $v_{\max}$ for t time steps. The length of the considered time span t can be chosen almost arbitrarily, subject to a mild lower bound. For fixed t, the lemma presents lower bounds for the probability that P_t exceeds a family of probability thresholds. The thresholds are specified by a second value i; they decrease from high thresholds to lower thresholds as i increases. The probability that P_t exceeds the i-th threshold is at least $1 - e^{-i}$. This means that the probability of not overcoming the i-th threshold decreases exponentially with i. Intuitively, this accounts for the fact that low velocities are increased far more easily than high velocities. Failing to overcome low thresholds is, therefore, far more unlikely than failing on higher ones.

Lemma 5.4.9. *Consider a well-guided bit of a particle in the Binary PSO. After a number t of elapsed generations, let P_t denote the probability of setting the bit according to the bit value it is guided to, w. l. o. g. value 1.*

Choosing $c^* := \min\{1, \bar{c}/2\}$, $t \geq (16 \ln n)/c^*$ and n large enough, it holds for every real-valued $i \in [1, c^*t/192]$

$$\text{Prob}\left(P_t \geq 1 - \frac{96ie^{\bar{c}}}{c^* \cdot t}\right) \;\geq\; 1 - e^{-i}$$

or the bit's velocity is capped by $v_{\max}$ anyway.

More generally, this statement holds if P_t denotes the success probability of a bit whose velocity $\bar{c}$-dominates the $\tilde{v}$-process at time t.

Note that if we were only interested in an upper bound on the expected freezing time, it would suffice to state the lemma only for one threshold, namely $P_t = 1 - 1/n$, which corresponds to a velocity at $v_{\max}$.

Proof. Instead of the actual velocity v_t after t steps, we consider the $\tilde{v}$-process. According to Lemma 5.4.7, this provides a pessimistic estimation of the actual (random) velocity value after t steps in terms of $\bar{c}$-domination and justifies the more general claim in the last sentence of the lemma. Moreover, the required success probability $1 - 96ie^{\bar{c}}/(c^*t)$ at the bit in the worst case corresponds to a success probability $1 - 96i/(c^*t)$ of the $\tilde{v}$-process. We show the following claim: with probability at least $1 - e^{-i}$, it holds for the success probability $\tilde{P}_t$ of the $\tilde{v}$-process after t steps that $\tilde{P}_t \geq 1 - 96i/(c^*t)$, or $\tilde{v}_t$ has reached $v_{\max} + \bar{c}$ anyway; the latter case will pessimistically be ignored hereinafter.

Recall that the probability of increasing the $\tilde{v}$-value decreases monotonically with the $\tilde{v}$-value. It is therefore bounded from below by $96i/(c^*t)$ before the velocity has reached the desired value. Note that for smaller velocities this probability is much higher as it increases rapidly with increasing distance from the bound. The progress of the velocity from $\tilde{v}_0 = -v_{\max} + \bar{c}$ to the value corresponding to success probability $1 - 96i/(c^*t)$ is divided into two big epochs. Throughout the first epoch, the $\tilde{v}$-value is at most 0, which, by definition of the $\tilde{v}$-process, implies that each generation increases the velocity with a probability of at least 1/2, i. e., independent of t and i. In the second epoch, we have to be more careful. An increase of the velocity by a constant additive amount derates the probability of a further increase by a constant factor. Hence, to obtain a good bound on the probability of an increase throughout the second epoch, we divide the progress into phases with exponentially decreasing probabilities.

We start with the first epoch. Our aim is to show that it ends after at most $t/2$ steps with high probability. For this to happen, the following two events together are sufficient:

1. In the first $t/2$ steps, there are at least $t/8$ increases, or a positive $\tilde{v}$-value is reached.
2. The total increase in $t/8$ increases is at least $c^*t/16$, or a positive $\tilde{v}$-value is reached.

Since $t \geq (16 \ln n)/c^*$ is assumed in this lemma, the proposed total increase is at least $v_{\max}$, i. e., sufficient to leave the velocity range $[-v_{\max} + \bar{c}, 0]$.

To estimate the joint probability of the two events, we use the Chernoff-Hoeffding bounds from Lemma 5.4.8. As we are in the first epoch, an increase happens with probability at least 1/2. Using $\mathrm{E}(X) \geq t/4$, $\delta := 1/2$ and the first bound of the lemma, the failure probability for the first event is at most $e^{-t/32}$, which is at most e^{-6i} since $i \leq c^*t/192 \leq t/192$. For the second event, we apply the second bound of Lemma 5.4.8.

Here we observe that the random amount of increase is uniform over $[0, \bar{c}]$ with an expectation of $\bar{c}/2$. By pessimistically minimizing $\bar{c}/2$ with 1 and making the distribution uniform over $[0, 2c^*]$, the new average amount of increase c^* yields a pessimistic estimation. Hence, the second bound of the lemma can be applied using $\delta = 1/2$, $\mathrm{E}(X) \geq c^*t/8$ and $b \leq 2c^*$. We obtain a failure probability for the second event of at most $e^{-t/64} \leq e^{-3i}$, altogether a failure probability of at most $e^{-3i} + e^{-6i}$ for the first epoch.

Assuming a positive velocity, i. e., a success probability greater than $1/2$ and a probability less than $1/2$ for an increase of the velocity, we have entered the second epoch. The number of phases that it is subdivided into equals $\lceil \log(c^*t/(192i)) \rceil$. Note that some phases may be empty and phases are traversed in decreasing order. Phase k, $1 \leq k \leq \lceil \log(c^*t/(192i)) \rceil$, is defined such that throughout the phase, the current success probability (i. e., the probability of *not* increasing the velocity) is within the interval $\left[1 - \frac{96i2^k}{c^*t},\ 1 - \frac{96i2^{k-1}}{c^*t}\right] \cap [1/2, 1]$; hence, the complementary probabilities exhibit an exponential decay. We use the inverse sigmoid function $s^{-1}(z) = \ln(z/(1-z))$ to map a success probability z of the $\tilde{v}$-process back to a velocity. This allows us to bound the length of the above-defined intervals on the velocity scale. In case that $1 - \frac{96i2^k}{c^*t} \geq 1/2$, the length is at most

$$\ln\left(\frac{1 - \frac{96i2^{k-1}}{c^*t}}{\frac{96i2^{k-1}}{c^*t}}\right) - \ln\left(\frac{1 - \frac{96i2^k}{c^*t}}{\frac{96i2^k}{c^*t}}\right) = \ln\left(\frac{2^k}{2^{k-1}} \cdot \frac{1 - \frac{96i2^{k-1}}{c^*t}}{1 - \frac{96i2^k}{c^*t}}\right) \leq \ln\left(\frac{2 \cdot 2^k}{2^{k-1}}\right) = \ln 4.$$

In the other case, which is equivalent to $\frac{96i2^k}{c^*t} > 1/2$, the interval ranges from 0 to $s^{-1}\left(1 - \frac{96i \cdot 2^{k-1}}{c^*t}\right)$. We have $\frac{96i2^{k-1}}{c^*t} = \frac{1}{2} \cdot \frac{96i2^k}{c^*t} \geq \frac{1}{4}$. Hence, $s^{-1}\left(1 - \frac{96i \cdot 2^{k-1}}{c^*t}\right) \leq \ln((1-1/4)/(1/4)) \leq \ln 4$, implying that the length of an interval on the velocity scale is bounded by $\ln 4$ in both cases. We will next show that, with high probability, we spend at most $\frac{tk}{2^{k+1}}$ steps in a phase. If this holds for all phases, the total time spent in all phases of the second epoch is less than $t \sum_{k=1}^{\infty} \frac{k}{2^{k+1}} = \frac{t}{2}$. Together with the first epoch, the number of steps does not exceed t.

In a similar (but more detailed) manner as for the first epoch, the following set of conditions is sufficient to raise the velocity during the second epoch to the desired value. Note that Condition 2. suffices to leave any considered interval of success probabilities.

For $k = \left\lceil \log(\frac{c^*t}{192i}) \right\rceil$ down to 1 it holds

1. If the current success probability is in the interval

$$\left[1 - \frac{96i2^k}{c^*t},\ 1 - \frac{96i2^{k-1}}{c^*t}\right] \cap \left[\frac{1}{2}, 1\right],$$

   then $\frac{tk}{2^{k+1}}$ steps contain at least $12ik/c^*$ increases.
2. The total increase in $12ik/c^*$ increases is at least $\ln 4$.

We may finish the considerations prematurely if the desired $\tilde{v}$-value is reached within less steps or less total increase.

It remains to estimate the probabilities of these events. This is done similarly as for the first epoch by applying Lemma 5.4.8. For the first event we estimate $\mathrm{E}(X) \geq 24ik/c^*$ and use $\delta = 1/2$. For the second event, we use $\mathrm{E}(X) \geq 12ik$ and $\delta = 3/4$ since, clearly, $12ik/4 > \ln 4$. Hence the failure probabilities for the two events can be bounded by e^{-3ik} and $e^{-27ik/8}$, respectively. Together with the first epoch, the sum of all failure probabilities is at most

$$e^{-3i} + e^{-6i} + \sum_{k=1}^{\infty}(e^{-3ik} + e^{-27ik/8}) \;=\; e^{-3i} + e^{-6i} + \frac{1}{e^{3i}-1} + \frac{1}{e^{27i/8}-1} \;\leq\; e^{-i},$$

which proves the lemma. □

Finally, we can compute the expected time until bits freeze towards velocity bounds.

Lemma 5.4.10. *Consider a selection of $k \geq 1$ well guided bits in the swarm of the Binary PSO with $\bar{c} = \Theta(1)$. Assuming that well-guidance is preserved, the expected time until all k bits freeze to the velocity bounds they are guided to is bounded by $O(n \ln(2k))$.*

Note that the factor 2 in $\ln(2k)$ cannot be dropped as otherwise for $k = 1$ we would have $\ln k = 0$. However, if $k > 1$, then $\ln(2k) = O(\ln k)$ and the bound simplifies to $O(n \ln k)$.

Proof. Let $c^* := \min\{1, \bar{c}/2\}$. Fix one of the selected bits and invoke Lemma 5.4.9 with $i := \ln(2k)$ and $t := 96ie^{\bar{c}}n/c^*$. Let P_t denote the probability that this bit is set to the bit value it is guided to, then

$$\mathrm{Prob}\left(P_t \geq 1 - \frac{96ie^{\bar{c}}}{c^*t}\right) \;=\; \mathrm{Prob}\left(P_t = 1 - \frac{1}{n}\right) \;\geq\; 1 - e^{-\ln(2k)} \;=\; 1 - \frac{1}{2k}.$$

The probability that this bit does not freeze within t steps is at most $1/(2k)$. Taking the union bound for k bits, the probability that any of these bits does not freeze within t steps is at most $1/2$. In this case, we repeat the argumentation with a new period of t steps. The expected number of periods until all bits are frozen is at most 2, hence we obtain the upper bound $2t = O(n \ln(2k))$. □

The result on the expected freezing time allows the derivation of upper bounds for the Binary PSO in a similar way as for evolutionary algorithms. For the sake of simplicity, we first describe this idea for the 1-PSO. Due to the strict selection in the 1-PSO, the global best is only exchanged in case a better solution is discovered. This means that after some time either the global best has improved or all velocities in the global best are frozen. In the latter case, since $v_{\max} = \ln(n-1)$, the probability of creating a 1

for every bit x_i^{**} is now either $s(-v_{\max}) = 1/n$ or $s(v_{\max}) = 1 - 1/n$. The distribution of constructed solutions now equals the distribution of offspring for the (1+1) EA with the global best x^{**} as the current search point. Recall that we refer to the (1+1) EA* as the (1+1) EA with strict selection. If for the 1-PSO all velocity values take their upper or lower bounds, the 1-PSO behaves like the (1+1) EA* until a solution with larger fitness is encountered. We arrive at upper bounds using a restricted fitness-level method, similarly to the case of MMAS* in Section 5.2.

Let $f_1 < f_2 < \cdots < f_m$ be an enumeration of all fitness values of f and let A_i, $1 \leq i \leq m$, contain all solutions with fitness f_i. We also say that A_i is the i-th fitness level. Let s_i, $1 \leq i \leq m-1$, be a lower bound on the probability of the (1+1) EA (or, equivalently, the (1+1) EA*) to create an offspring in $A_{i+1} \cup \cdots \cup A_m$, provided the current population belongs to A_i. Recall that summing up expected waiting times until all fitness levels are left yields the upper bound

$$\sum_{i=1}^{m-1} \frac{1}{s_i} \tag{5.16}$$

for the (1+1) EA and the (1+1) EA*. A similar bound holds for the 1-PSO and, more generally, for the Binary PSO.

Theorem 5.4.11. *Let A_i form the i-th fitness level of f and let s_i be the minimum probability for the* (1+1) EA *to leave A_i towards $A_{i+1} \cup \cdots \cup A_m$. If $\bar{c} = \Theta(1)$, the expected number of generations for the Binary PSO to optimize f is bounded above by*

$$O(mn \log n) + \sum_{i=1}^{m-1} \frac{1}{s_i}.$$

The right-hand sum is the upper bound obtained for the (1+1) EA and (1+1) EA* from (5.16). Compared to the (1+1) EA, the bound on the number of generations only increases by an additive term $O(mn \log n)$.

Also note that the upper bound does not depend on the swarm size μ. However, when taking the number of function evaluations as performance measure instead of the number of generations, the bound has to be multiplied by μ.

Proof of Theorem 5.4.11. Assume that the global best is located on the i-th fitness level. As long as no improvement happens, for this particle own best and global best coincide. Hence, all bits in the particle are well guided. By Lemma 5.4.10 the expected time until all n bits in x^{**} are frozen or an improvement happens in the meantime is $O(n \log(2n)) = O(n \log n)$.

In case all bits in x^{**} are frozen, there is a probability of at least s_i that the new position for x^{**} is an improvement. The expected waiting time for an improvement is thus bounded from above by $1/s_i$ and, together, the expected time on the i-th fitness

level is bounded by $O(n \log n) + 1/s_i$. Summing up expected waiting times for the first $m - 1$ fitness levels proves the claim. □

In the proof of Theorem 5.4.11 we only relied on the global best to find an improvement. It may happen, though, that other particles find improvements somewhere else or that other particles are attracted by the global best. The latter case can lead to a better estimate of the probability of an improvement. If there are several particles whose velocities have been frozen at borders corresponding to the global best, several particles search in the region around x^{**} in parallel. We make this precise for the Binary PSO using only the social component, i. e., $c_1 = 0$. In this extreme case, all particles follow the global best and, as long as the global best is not exchanged, all bits are well guided. After freezing, the swarm reduces the expected time to find an improvement by a factor of order μ. This behavior resembles a (1+λ) EA that creates $\lambda = \mu$ offspring in each generation.

Theorem 5.4.12. *Let A_i form the i-th fitness level of f and let s_i be the minimum probability for the* (1+1) EA *to leave A_i towards $A_{i+1} \cup \cdots \cup A_m$. If $\mu = \text{poly}(n)$, $c_1 = 0$, and $c_2 = \Theta(1)$, the expected number of generations for the Binary PSO to optimize f is bounded above by*

$$O\left(mn \log n + \frac{1}{\mu} \cdot \sum_{i=1}^{m-1} \frac{1}{s_i}\right).$$

Proof. We only need to prove that the expected number of generations to increase the fitness from the i-th fitness level is at most $O(n \log n + 1/(\mu s_i))$.

Invoking Lemma 5.4.10 for all μn bits in the swarm, the expected time until all bits in the swarm are frozen towards the bounds in x^{**} (or an improvement happened anyway) is $O(n \log(2\mu n)) = O(n \log n)$ as $\log(2\mu n) = O(\log n)$ follows from $\mu = \text{poly}(n)$.

Once all bits are frozen to the corresponding bounds of x^{**}, all particles behave equally until the next improvement. This implies that the Binary PSO performs μ trials in each generation to create a solution with higher fitness and the probability for an improvement in one trial is bounded below by s_i. The probability that the Binary PSO does not find an improvement within a period of $1/s_i$ trials is bounded by

$$1 - (1 - s_i)^{1/s_i} \;\geq\; 1 - e^{-1} \;=\; \frac{e-1}{e}.$$

The expected number of periods is therefore bounded by $e/(e-1)$. The number of generations needed to have a period of $1/s_i$ trials equals $\lceil 1/(\mu s_i) \rceil$. Hence the expected number of generations to increase the fitness is bounded by

$$\frac{e}{e-1} \cdot \left(\frac{1}{\mu s_i} + 1\right) \;=\; O\left(\frac{1}{\mu s_i} + 1\right).$$

Adding the expected freezing time for the swarm yields the claimed bound $O(n \log n + 1/(\mu s_i))$ for fitness level i. □

The additive term $O(mn \log n)$ in the bounds from Theorems 5.4.11 and 5.4.12 results from the (pessimistic) assumption that on all fitness levels, the Binary PSO has to wait until all velocities are frozen in order to find a better solution. Nevertheless, fitness-level arguments represent a powerful tool that can easily be applied to various problems. We exemplarily present an application for unimodal functions.

Recall from Definition 2.2.1 that a function f is unimodal if every non-optimal search point has a Hamming neighbor with strictly larger fitness. The probability for the (1+1) EA* to create a specific Hamming neighbor as offspring equals $1/n \cdot (1-1/n)^{n-1} \geq 1/(en)$. We conclude $s_i \geq 1/(en)$ for every non-optimal fitness level. Theorem 5.4.11 yields the following bound.

Corollary 5.4.13. *Let f be a unimodal function with m different function values. Then the expected number of generations for the Binary PSO to optimize f is bounded by*

$$O\left(mn \log n + \sum_{i=1}^{m-1} en\right) \;=\; O(mn \log n + m \cdot en) \;=\; O(mn \log n).$$

5.4.4 The 1-PSO on OneMax

The function OneMax is one of the best known test functions for randomized search heuristics. For several classes of heuristics, initial theoretical studies focused on the analysis of OneMax. Probably the first rigorous runtime analysis of an evolutionary algorithm was done on this function by Mühlenbein (1992). Droste (2006) presented a first rigorous runtime analysis of an estimation-of-distribution algorithm on OneMax and other linear functions. Finally, Neumann and Witt (2006) and Gutjahr (2008) independently presented the first rigorous runtime analyses of an ant colony optimization algorithm on OneMax. It makes sense to continue the tradition and to consider the 1-PSO on OneMax in more detail.

Corollary 5.4.13 yields a bound $O(n^2 \log n)$ on the expected optimization time of the 1-PSO on OneMax. Compared to the bound $\Theta(n \log n)$ that holds for the (1+1) EA in this setting, this seems relatively rough. To improve the $O(n^2 \log n)$ bound, we have to show that it is not necessary for the 1-PSO to spend $\Theta(n \log n)$ steps on each fitness level until all bits have been frozen to the velocity bounds of the global best.

In the following, we will improve the optimization time bound of the 1-PSO on OneMax to $O(n \log n)$. This implies that the 1-PSO has the same asymptotic upper runtime bound as the (1+1) EA. For the proof, we will basically show that $O(\log n)$ steps adjusting the velocity entries are enough on each fitness level for the 1-PSO to attain almost the same probability of improving the current fitness as the (1+1) EA.

Consider a bit that has been well guided towards the velocity bound $v_{\max}$ for t steps. Then the expected velocity value increases with t. This means that a bit whose entry in the global best has been fixed to 1 for a long time has a good chance to contribute to the OneMax-value of new solutions. We introduce a handy notation for such a bit.

Definition 5.4.14. A bit is called *t-strong* for $t \in \mathbb{N}_0$, if its (random) velocity $\bar{c}$-dominates the $\tilde{v}$-process at time t.

As a consequence from Lemma 5.4.6, we obtain the following statement.

Lemma 5.4.15. *Consider a bit that is currently t-strong. If the bit is well guided towards value 1 throughout the next s steps, the bit will be $(t+s)$-strong afterwards.*

Proof. Let v_0 denote the current velocity at the bit and let v_s denote its velocity after s steps. By the definition of strongness and $\bar{c}$-dominance, $\text{Prob}(v_0 \geq a) \geq \text{Prob}(\tilde{v}_t \geq a + \bar{c})$ for all $a \in \mathbb{R}$. By Lemma 5.4.6 and the assumption that the x^{**}-entry stays fixed, $\bar{c}$-dominance is maintained for all following steps. After s steps, the current velocity v_s $\bar{c}$-dominates $\tilde{v}_{t+s}$, hence, by definition, the bit is $(t+s)$-strong. □

We take a fresh look at Lemma 5.4.9 and note that its last paragraph just repeats the definition of a t-strong bit. Let us recall the arguments leading to the lemma. The random velocity v_t of a t-strong bit gives us a random probability P_t of setting the considered bit to 1, i. e., a success. If we just plug in a single value for i into the lemma, we know that P_t respects the given bound on the success probability with probability at least $1 - e^{-i}$. However, if the bound is not respected, no immediate conclusion except the trivial bound $P_t \geq 1/n$ can be drawn.

Still, there is always a well-defined probability of a success at a t-strong bit. This probability equals the expectation of the random P_t since $\text{E}(P_t) = \int p \cdot f_{P_t}(p)\,\text{d}p$ (or, if P_t has a discrete distribution, $\text{E}(P_t) = \sum p \cdot \text{Prob}(P_t = p)$). Since the distribution of P_t has already been characterized quite precisely in Lemma 5.4.9, it is not too difficult to derive good lower bounds on $\text{E}(P_t)$, the expected success probability, using the lemma. Note that $\bar{c} = c_1 + c_2 = 2$ and $c^* = \min\{1, \bar{c}/2\} = 1$ hold as the 1-PSO is considered.

Lemma 5.4.16. *The expected success probability of a t-strong bit, $t \geq 384 \ln n$, is at least $1 - O(1/t + 1/n)$.*

Proof. Note that the bound to be proven is never better than $1 - O(1/n)$. Hence, in the following, we ignore the upper bound $1 - 1/n$ on success probabilities and allow such a probability to become arbitrarily close to 1; the possible error introduced is $O(1/n)$. In order to apply Lemma 5.4.9, we consider a random variable X with support $\{1 - 96e^2 i/t \mid 1 \leq i \leq t/192,\ i \in \mathbb{N}\}$, i. e., the support matches the bounds on P_t given in the lemma. We define a distribution for X according to

$$\begin{aligned} \text{Prob}\big(X = 1 - 96e^2/t\big) &= 1 - e^{-1}, \\ \text{Prob}\big(X = 1 - 96e^2 i/t\big) &= e^{-i+1} - e^{-i} \text{ for } 2 \leq i \leq t/192, \\ \text{and } \text{Prob}(X = 1/n) &= e^{-t/192}. \end{aligned}$$

Obviously, all probabilities sum up to 1.

Using Lemma 5.4.9, it follows that the success probability of the t-strong bit stochastically dominates X. It is therefore enough to bound the expected success probability according to X from below. Using the law of total probability, this expectation is at least

$$\begin{aligned}
&(1-e^{-1}) \cdot \left(1 - \frac{96e^2}{t}\right) + \sum_{i=2}^{t/192} \left(e^{-i+1} - e^{-i}\right) \cdot \left(1 - \frac{96e^2 i}{t}\right) \\
&\geq\ (1-e^{-1}) \cdot \left(1 - \frac{96e^2}{t}\right) + (e^{-1} - e^{-t/192}) - \sum_{i=2}^{t/192} e^{-i+1} \cdot \frac{96e^2 i}{t} \\
&\geq\ (1-e^{-t/192}) \left(1 - \frac{96e^2}{t}\right) - O\left(\frac{1}{t}\right) \ =\ 1 - O\left(\frac{1}{t}\right).
\end{aligned}$$

Subtracting $O(1/n)$ for the possible error, we obtain the claim of the lemma. □

The expected success probability we have just bounded is the actual probability of having a success at a t-strong bit. Hence, the lemma will be useful when we study OneMax and are interested in reproducing 1-bits. However, we need an additional statement. Given that a bit is t-strong, we are interested in the first point of time where the bit is actually set to 1. Assuming that the bit remains well-guided towards 1, the expected time until we have a success at such a bit decreases as t increases. Hence, we pessimistically assume that the bit is always exactly t-strong and wait for a success at the bit. More precisely, we are looking for the expected waiting time. If the success probability P_t was deterministic, say $P_t = p$ with probability 1, we could derive the expected waiting time by inspecting a geometrically distributed random variable, which would imply that the expectation would be $1/p$. However, since we are dealing with random probabilities, the expected waiting time is not necessarily the reciprocal of the expected success probability. We need more arguments, as described by the following lemma. Interestingly, since P_t follows a quite concentrated distribution, the following asymptotic upper bound is still the reciprocal of the bound derived in Lemma 5.4.16.

Lemma 5.4.17. *The expected time for a success at a t-strong bit, $t \geq 384 \ln n$, is bounded by $1 + O(1/t + 1/n)$.*

Proof. As in the proof of Lemma 5.4.16, we ignore the upper bound $1 - 1/n$ on success probabilities. Since the expected time for a success at success probability $1 - 1/n$ equals $1/(1 - 1/n) = 1 + O(1/n)$, the asymptotic upper bound of the lemma is not affected.

We reconsider the random variable X defined in the proof of Lemma 5.4.16. Since $t \geq 384 \ln n$, the last line of its definition implies $\text{Prob}(X = 1/n) \leq 1/n^2$. Since the success probability of the t-strong bit stochastically dominates X, it is enough to bound the expected time for a success according to X from above. Given that X has the value p,

the waiting time for a success follows a geometric distribution with expectation $1/p$. We can bound the reciprocals of the single success probabilities according to $1/(1-96e^2 i/t) \leq 1 + 192e^2 i/t$ if n is not too small. By the law of total probability, the unconditional expected waiting time is at most

$$\begin{aligned}
&(1-e^{-1}) \cdot \left(1 + \frac{192e^2}{t}\right) + \sum_{i=2}^{t/192} \left(\left(e^{-i+1} - e^{-i}\right) \cdot \left(1 + \frac{192e^2 i}{t}\right)\right) + \frac{1}{n^2} \cdot n \\
&\leq \left(1 + \frac{192e^2}{t}\right) + \sum_{i=2}^{t/192} \left(\left(e^{-i+1} - e^{-i}\right) \frac{192e^2 i}{t}\right) + \frac{1}{n} \\
&\leq \left(1 + \frac{192e^2}{t}\right) + \sum_{i=2}^{t/192} \left(e^{-i+1} \frac{192e^2 i}{t}\right) + \frac{1}{n} \;=\; 1 + O\left(\frac{1}{t} + \frac{1}{n}\right).
\end{aligned}$$

□

During an improvement of the OneMax-value, some bits in x^{**} may be turned from 1 to 0, but a larger number of bits is turned from 0 to 1. We refer to the latter as new 1-bits being gained during this improvement. In the following analysis, we will consider random velocities of 1-bits gained while optimizing OneMax. Freshly gained 1-bits tend to be weaker than older ones. Sorting the bits from weak to strong, this is reflected by the following layering.

Definition 5.4.18. We say that an ordered sequence of bits $1, \ldots, k$ forms an *m-layer*, $m \in \mathbb{N}$, if and only if bit j, $1 \leq j \leq k$, is jm-strong. A set of k bits forms an m-layer if and only if it can be arranged as an m-layer.

For the following theorem, we will consider layers where the j-th bit is basically $\Theta(j \ln n)$-strong. Defining that a set of bits is successful if all have successes simultaneously, we show the following lemma. Note that we assume x^{**} to be fixed to 1 for these bits.

Lemma 5.4.19. *Let $k \leq n$ bits form a $(384 \ln n)$-layer. Then the expected time until all have a success simultaneously is bounded by $O(1)$.*

Proof. We ignore the fact that velocities may increase while waiting for a success and pessimistically assume that the velocities remain fixed. However, the velocities and, therefore, the success probabilities at the k bits are still random variables. Let us consider such a random variable for some fixed bit. While waiting for a success, we consider the same random variable in all time steps. Hence, if we know the value of the variable at the beginning of our considerations, the variable will take the same value in all following time steps.

We claim that multiplying the expected success times for the single bits as bounded in Lemma 5.4.17 yields an upper bound on the expected time until all bits have successes simultaneously. Bits are independently set to 1 by the 1-PSO with the same (random) success probability over time; hence, conditioning on all outcomes of the k success probabilities, the waiting time follows a geometric distribution. Applying the law of total probability in the same way as in the proof of Lemma 5.4.17 proves the claim.

According to our assumption, the j-th bit, $1 \leq j \leq k$, is $384j \ln n$-strong. Due to Lemma 5.4.17, its expected success time is bounded by $1 + \frac{\kappa}{384j \ln n} + \frac{\kappa}{n}$ for some large constant κ. Taking the product over all j, we obtain

$$\prod_{j=1}^{k} \left(1 + \frac{\kappa}{384j \ln n} + \frac{\kappa}{n}\right) \leq e^{\sum_{j=1}^{n} \frac{\kappa}{384j \ln n} + \frac{\kappa}{n}},$$

which is $O(1)$ since $\sum_{j=1}^{n} 1/j = O(\ln n)$. □

Now we can state the improved bound for OneMax.

Theorem 5.4.20. *The expected optimization time of the* 1-PSO *on* OneMax *is bounded by* $O(n \log n)$.

Proof. The basic proof idea is to keep track of the velocities of the newly gained 1-bits after improvements of the global best x^{**}. We wait on average $O(\log n)$ steps after an improvement and show that after that, the probability of improving is at least in the same order as for the (1+1) EA.

A difficulty with these arguments is that 1-bits in x^{**} may be set to 0 if the global best is exchanged. The current OneMax-value may only increase, but specific 1-bits may be lost if enough new 1-bits are gained somewhere else. We say that the lost 1-bits are reset. Resets may disturb the velocity increase on 1-bits as strong 1-bits may be replaced by weaker 1-bits.

In order to simplify the argumentation, we first describe an analysis for an idealized setting and then argue how to extend the arguments to the real setting. Assume in the following that the 1-PSO does not accept resets of 1-bits, i. e., an improvement of the OneMax-value is only accepted in case all 1-bits are set to 1 in the new global best.

We now divide a run of the 1-PSO into phases. Phase i for $0 \leq i \leq n-1$ starts with the end of the previous phase (Phase 0 starts after initialization) and it ends when the following two conditions are met:

1. The best-so-far OneMax-value is at least $i + 1$.
2. At least $i + 1$ 1-bits form an m-layer for $m := 384 \ln n$.

Note that the second condition will be fulfilled throughout the run as all 1-bits are maintained forever in our idealized setting and hence their velocities are well guided and monotone increasing over time.

We claim that the expected time spent in Phase i is bounded above by $m+O(n/(n-i))$ for each $0 \leq i \leq n-1$. Note that phases may be empty. Moreover, when finishing Phase $n-1$ the global optimum has been found. Hence, the expected time to find a global optimum is bounded by

$$\sum_{i=0}^{n-1} \left(m + O\left(\frac{n}{n-i} \right) \right) \;=\; O(n \log n) + O(n) \cdot \sum_{i=1}^{n} \frac{1}{i} \;=\; O(n \log n).$$

Consider the 1-PSO at the time it enters Phase i and assume pessimistically that the best-so-far OneMax-value is still i. As Phase $i-1$ has been completed, i 1-bits form an m-layer. According to Lemma 5.4.19, all these bits are set to 1 simultaneously after an expected number of $O(1)$ steps. Independently of these bits, the 1-PSO turns at least one of the $n-i$ 0-bits to 1 with a probability bounded below as follows. The probability of creating a 0-value at any 0-bit is at most $s(-v_{\max}) = 1 - 1/n$. The probability that not all $n-i$ 0-bits result in 0-values can be estimated, using $1 + x \leq \exp(x) \leq 1 + x/2$ for $-1 \leq x \leq 0$ (see Lemma A.12), by

$$1 - \left(1 - \frac{1}{n}\right)^{n-i} \;\geq\; 1 - \exp\left(-\frac{n-i}{n}\right) \;\geq\; \frac{n-i}{2n}.$$

The expected waiting time for this event is at most $2n/(n-i)$. As 1-bits and 0-bits are treated independently, we can multiply expectations using the justification from the proof of Lemma 5.4.19. Altogether, the expected time until constructing a solution with OneMax-value at least $i+1$ has been bounded from above by $O(n/(n-i))$.

Once the best-so-far OneMax-value has increased to at least $i+1$, the velocities on $i+1$ 1-bits are well guided and monotone increasing. At the beginning of the phase, i 1-bits form an m-layer. In order to obtain an m-layer of $i+1$ bits, we can consider any other 1-bit and wait for it to become m-strong. As a consequence from Lemma 5.4.15, if x^{**} remains fixed for these $i+1$ bits, they form an m-layer of size $i+1$ after m steps. Together, the claimed bound $m + O(n/(n-i))$ follows for the expected time in Phase i. This also finishes the analysis for the idealized setting without resets.

A reset of a bit can destroy the velocity layers as a strong 1-bit might be exchanged by a weak 1-bit. In the worst case, such a new 1-bit is only 0-strong. If an improvement resets d bits, an m-layer of i bits may shrink to an m-layer of $i-d$ 1-bits. By an amortized analysis, we wait for the velocities to recover so that we end up with an m-layer of i bits again.

Consider an improvement in a setting where k bits form an m-layer. A t-strong bit is reset with probability at most $O(1/t + 1/n)$ according to Lemma 5.4.16. The expected number of bits among these k layered bits reset during this improvement is therefore bounded from above by

$$\sum_{j=1}^{k} \left(\frac{O(1)}{384j \ln n} + \frac{O(1)}{n} \right) \;=\; O(1).$$

Hence, an improvement prolongs the time spent in the current phase in expectation by $O(\ln n)$. Note that we can repeat the argumentation if another improvement occurs in the meantime since we only consider reset probabilities for all bits in a layer. As we can only have n improvements, we obtain an additional term $O(n \log n)$ in our runtime bound, which proves the time bound $O(n \log n)$ for the real setting. □

5.4.5 Experiments for OneMax

The expected runtime of the (1+1) EA on OneMax is known to be of order $\Theta(n \log n)$. Theorems 5.4.3 and 5.4.20 show that the expected runtime of the 1-PSO is between $\Omega(n/\log n)$ and $O(n \log n)$. We conjecture that the runtime of the 1-PSO is of order $\Theta(n \log n)$ as well, but proving a corresponding lower bound seems difficult. We therefore perform experiments to get an impression of the actual runtimes for various problem dimensions and to compare the 1-PSO with the (1+1) EA.

A drawback of the 1-PSO might be that the velocities of 1-bits increase only slowly, which may yield an increased waiting time to rediscover previously gained 1-bits. On the other hand, the 1-PSO has the advantage that the velocities of 0-bits tend to $-v_{\max}$ only slowly as well, and so an incorrect decision may be reverted more easily. In contrast to the 1-PSO, the (1+1) EA only has a probability of $1/n$ to revert an incorrectly set bit. Since it typically starts with a linear number of such bits, the lower bound $\Omega(n \log n)$ follows, as shown in Theorem 2.3.5. To conclude, the slow adaptation of velocities can have effects in opposite directions; we cannot tell a priori which effect will dominate the runtime and decide which algorithm performs better.

We therefore perform experiments for both the (1+1) EA and the 1-PSO with problem dimensions $n = 100, 200, \ldots, 2000$. For each setting 1000 independent runs are performed and the number of generations until the optimum is found is measured. Figure 5.11 shows box-and-whisker plots of the resulting runtimes. The box ranges from the first to the third quartile, the horizontal line within the box denotes the median, and the whiskers indicate minimum and maximum values.

One can observe that the median for the 1-PSO is consistently larger than the median for the (1+1) EA. Even stronger, the boxes and the minimum values for the 1-PSO are consistently higher than those for the (1+1) EA. Another observation is that the size of the boxes is quite small, which implies that the runtime has a low variance. The variance is similar for both algorithms.

As this study focuses on expected runtimes, we investigate the average runtimes in more detail. Figure 5.12 shows the average runtimes for the (1+1) EA and the 1-PSO. A regression analysis was done using gnuplot 4.2 with the standard settings. Both data sets were fitted towards functions $c \cdot n \log n$ for a constant c, log denoting the logarithm to the base 2. For the (1+1) EA the function $1.70891n \log n$ resulted in the best fit and for the 1-PSO the function $1.97469n \log n$ was best. Comparing the two constants indicates that the expected runtime for the 1-PSO is about 15% larger than for the (1+1) EA.

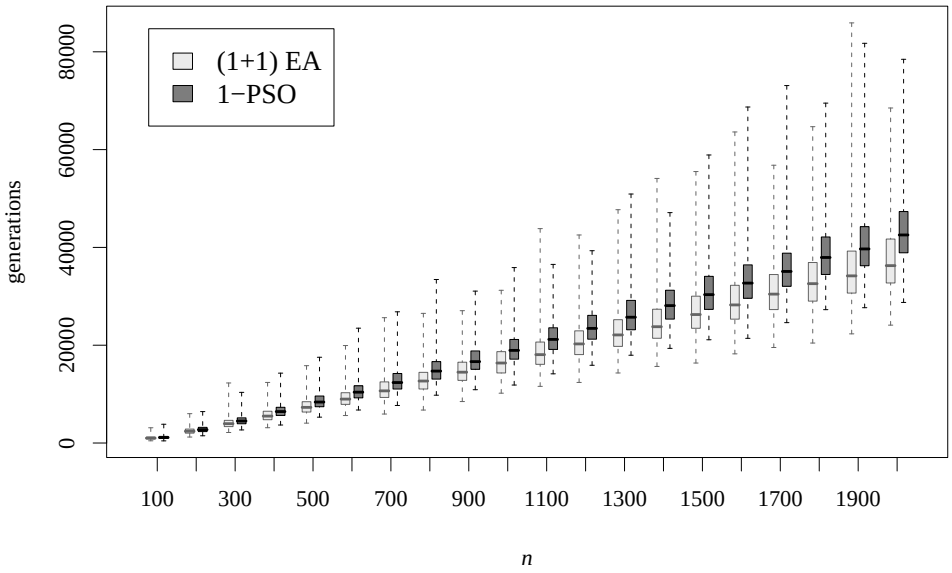

Figure 5.11: Box-and-whisker plot of the runtimes of the (1+1) EA (light gray) and the 1-PSO (dark gray) for $n = 100, 200, \ldots, 2000$ and 1000 runs for each setting. The whiskers show minimum and maximum values.

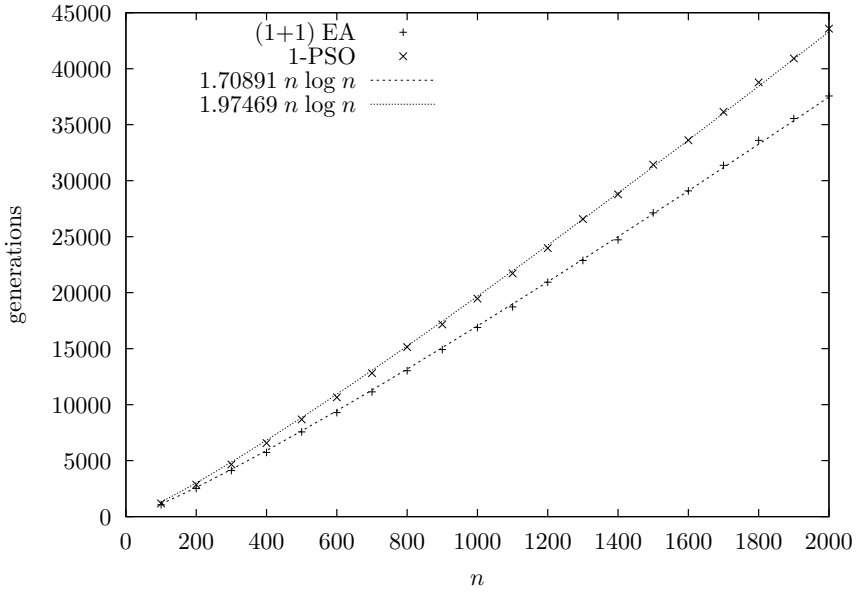

Figure 5.12: Average runtimes for the (1+1) EA and the 1-PSO and the fitted functions $1.70891n \log n$ for the (1+1) EA and $1.97469n \log n$ for the 1-PSO.

Finally, to see whether the observed differences in runtime are statistically significant, we applied non-parametric Mann-Whitney tests to each sample of 1000 runs. For each problem dimension n, a two-sided test resulted in highly significant differences with $p \ll 0.0001$. A one-sided test confirmed that the differences were in favor of the (1+1) EA in all cases.

5.4.6 Conclusions

We have considered the runtime behavior of the Binary Particle Swarm Optimizer by Kennedy and Eberhart. Thereby, we adapted the choice of the maximum velocity $v_{\max}$ to growing problem sizes and justified why this adaptation is essential when dealing with different problem sizes. For the resulting Binary PSO we have proved a lower bound $\Omega(n/\log n)$ on the expected number of generations for every function where the global optimum is unique. This bound holds for almost every choice of the swarm size and the learning factors for the cognitive and the social component of PSO.

Moreover, we were able to transfer a restricted fitness-level method from the analysis of evolutionary algorithms to PSO. This can be used as a tool to prove upper bounds for various parametrizations of the Binary PSO and various functions. An exemplary application to the class of all unimodal functions showed that the Binary PSO is effective on these functions.

A more detailed analysis on the function OneMax revealed the bound $O(n \log n)$ for the 1-PSO and hence the same upper bound as known for the (1+1) EA. Experimental results suggest that the expected runtimes of both the 1-PSO and the (1+1) EA have the same order of growth, but the 1-PSO is by roughly 15% slower than the (1+1) EA. The observed differences were confirmed by statistical tests.

6 Conclusions and Outlook

Looking back at the topics addressed in this thesis, many questions have been answered, but still many questions remain open. In Chapter 3 we have dealt with extensions of simple evolutionary algorithms. In the black-box scenario, if no knowledge on the problem is available, it makes sense to use unbiased search operators as there is no reason to guide the search into predetermined regions of the search space. Nevertheless, many evolutionary algorithms used in practice intentionally or unintentionally exhibit some search bias. In Section 3.1 we have investigated an evolutionary algorithm with a mutation operator biased towards bit strings with either few or many 1-bits. Our investigations have led to insights in which situations such a search bias can be useful. If good solutions have few or many 1-bits and the fitness function gives appropriate hints towards these regions, the biased mutation operator leads to significant speed-ups. On the other hand, we have seen an example where the search bias misleads the search, resulting in an exponential performance gap (see Theorem 3.1.16). In many other situations, the search bias is overridden by a strong fitness bias. Future work could deal with other settings where a search bias is present. This also includes non-binary search spaces (e. g., permutations), where common operators are less well understood. Further theoretical investigations can increase our understanding of operator bias on search dynamics, which is essential to design more robust or more adapted evolutionary algorithms.

Section 3.2 has dealt with another important design aspect in evolutionary computation: population dynamics and the preservation of diversity. We have considered a bimodal function where a large diversity in the population is needed to find the optimum efficiently with high probability. The fitness landscape contains two large symmetric hills, one with a local and the other one with a unique global optimum. In a common population-based evolutionary algorithm, the competition between groups of individuals climbing different hills typically leads to the extinction of one such group before both hills can be explored. This means that the global optimum is only reached in polynomial time with probability close to 1/2. We have seen that some diversity mechanisms from the literature do not perform much better in our setting. Contrarily, deterministic crowding, where every offspring directly competes with his parents, and fitness sharing, a mechanism to derate the fitness of clustered individuals, perform very well on the considered function. This again increased our understanding of the effects certain components of evolutionary algorithms have on their performance.

Chapter 4 covered hybridizations of evolutionary algorithms with local search, known as memetic algorithms. We have presented the first rigorous analyses of these algorithms, thereby highlighting difficulties in their design as well as their strong potential, compared to standard evolutionary algorithms.

Designing such a hybrid is not trivial as some parameters in the interplay of the evolutionary part and local search must be chosen appropriately. For a simple memetic algorithm we have considered the local search depth and the local search frequency to balance the computational effort dedicated to the evolutionary and the local search. Section 4.1 presented families of artificial functions where the choice of these parameters is essential. Only slight changes to the parametrization can turn a polynomial runtime into a superpolynomial or even exponential one. Moreover, for almost every parametrization there is a function where this parametrization is drastically outperformed by other parametrizations. Although the constructed problems are clearly artificial, they rule out a priori design guidelines for the parametrization that work well for all problems.

In Section 4.2 we have abandoned this pessimistic point of view towards a more optimistic perspective. We have considered the classical combinatorial optimization problems Mincut, Knapsack, and Maxsat. For single instances of these problems, we have shown that a memetic algorithm with a particular kind of local search, variable-depth search, performs surprisingly well in a setting where many common trajectory-based algorithms fail. The shown negative results apply to the (1+1) EA, a memetic algorithm using a "greedy" kind of local search, and simulated annealing. Moreover, the negative results hold for very broad classes of parametrizations. This implies that the bad performance is not due to a "wrong" parameter choice, but due to fundamental differences in the search dynamics. Furthermore, the analysis of the Maxsat problem shows that a hybrid algorithm can significantly outperform its single components.

Despite these initial studies on memetic algorithms, this research area is still in its infancy. In order to gain a broader understanding of the capabilities and limitations of these hybrid algorithms, thorough analyses on broader classes of combinatorial problems are needed.

Finally, Chapter 5 contained some of the first rigorous analyses for swarm intelligence algorithms: ant colony optimization (ACO) and particle swarm optimization (PSO). The simple ACO algorithm 1-ANT, derived as a simple ACO algorithm for theoretical studies, turned out to be very sensitive to the choice of the evaporation factor ρ. It took some time until the reason became obvious: as opposed to common ACO algorithms, the 1-ANT only performs a pheromone update in case an improvement of the best-so-far fitness happens. If ρ is too small, this may lead to stagnation even on simple functions such as OneMax, LeadingOnes, and BinVal (see Section 5.1).

Research has then focused on more "typical" ACO algorithms. Section 5.2 dealt with the analysis of two variants of the MAX-MIN ant system (MMAS). Building upon previous work by Gutjahr and Sebastiani (2008), we have presented an elegant way of transferring the fitness-level method from evolutionary algorithms to ACO. This method

can easily be applied to yield upper bounds for various functions. We also developed lower bounds for all functions with unique global optimum and a good lower bound for LeadingOnes. Moreover, we have shown that an MMAS variant accepting equally good solutions outperforms its variant with strict selection on functions with plateaus. On these functions it is essential that the best-so-far solution can perform a random walk on the plateau. The mentioned analyses have led to important insights and to the development of new proof techniques, especially for pheromone values of bits that do not directly contribute to the fitness.

Another interesting topic with regard to Chapter 4 is the hybridization of ACO with local search. As shown in Section 5.3, the use of local search can affect the sampling distribution in a way which is not possible for memetic evolutionary algorithms. Sudden changes of the best-so-far solution may guide the algorithm to sample in regions of the search space that were never explored before, which is very unexpected behavior for a trajectory-based algorithm. This effect has been demonstrated on artificial functions designed such that the use of local search may lead to exponential performance gaps in both directions. These studies also show that the analysis of ACO algorithms has made significant progress and that nowadays analyses on more complex problems are possible.

An interesting topic for future work is to apply the analysis methods developed throughout this work to the analysis of combinatorial graph problems like shortest paths, the TSP, or spanning tree problems. For these problems it is possible to use the input graph as construction graphs, which more naturally matches the ACO paradigm. The mentioned problems also allow to asses the use of heuristic information in a sensible manner—another important component in ACO. A first such analysis on minimum spanning trees covering these two issues has been presented by Neumann and Witt (2008), however, for a strongly simplified pheromone model. As many new insights and methods have been developed since then, we hope that the analysis of more complex ACO systems will be possible in the near future.

Regarding particle swarm optimization, Section 5.4 contains a first runtime analysis of such an algorithm. In contrast to many applications in continuous spaces, a discrete PSO variant has been considered. In the Binary PSO velocities are used to construct new solutions, similar to pheromones in ACO. Also, the same bounds have been chosen, so that the probability of creating value 1 at each bit is always in the interval $[1/n,\ 1-1/n]$. We have justified this adaptation of the original Binary PSO as the latter needs exponential time on *every* non-trivial function if the velocity bounds remain fixed while the problem size grows. Similar to MMAS variants, we have also seen for this PSO algorithm how the fitness-level method can be used for the analysis of the Binary PSO. A more detailed analysis of a simple Binary PSO variant on OneMax has delivered useful insights into the probabilistic model underlying the Binary PSO and a runtime bound $O(n \log n)$, hence the same upper bound as for the (1+1) EA.

As particle swarm optimization has originally been defined for continuous optimization, one topic for future research is to consider PSO in these search spaces. There, PSO

does not necessarily converge to the global optimum. Therefore, additional mutation operators have been proposed in the literature and one globally convergent PSO algorithm has been proposed. Initial runtime analyses by Witt (2009) have revealed interesting similarities to a (1+1) evolution strategy, the continuous variant of the (1+1) EA. Moreover, it would be interesting to see a natural problem where the swarm dynamics in ACO or PSO are essential in order to find the optimum efficiently.

All these open questions lead to the conclusion that the computational complexity of evolutionary algorithms, hybridizations, and swarm intelligence remains an interesting, challenging, and fruitful research area.

Appendix A

Mathematical Tools

This appendix contains some mathematical tools used throughout this thesis. The following statements assume a probability space, including a sample space Ω and a probability function Prob. The first statements can be found in Mitzenmacher and Upfal (2005) along with a general introduction into probability theory.

Lemma A.1 (Expectation of a geometric random variable). Let X be a geometric random variable with parameter p, i. e.,

$$\text{Prob}(X = k) \;=\; (1-p)^{k-1} \cdot p.$$

Then $\text{E}(X) = 1/p$.

Lemma A.2 (Law of total probability). For an event A and mutually disjoint events $B_1, \ldots, B_k$ such that $\bigcup_i^k B_i = \Omega$,

$$\text{Prob}(A) \;=\; \sum_{i=1}^{k} \text{Prob}(A \cap B_i) \;=\; \sum_{i=1}^{k} \text{Prob}(A \mid B_i) \cdot \text{Prob}(B_i).$$

If additionally Ω is finite,

$$\text{E}(A) \;=\; \sum_{i=1}^{k} \text{E}(A \mid B_i) \cdot \text{Prob}(B_i).$$

Lemma A.3 (Law of total expectation). For random variables $X, Y \colon \Omega \to \mathbb{R}$,

$$\text{E}(X) \;=\; \text{E}(\text{E}(X \mid Y)).$$

Lemma A.4 (Union bound). For a finite or countably infinite sequence $A_1, A_2, A_3, \ldots$ of events

$$\text{Prob}\left(\bigcup_{i \geq 1} A_i\right) \;\leq\; \sum_{i \geq 1} \text{Prob}(A_i).$$

Lemma A.5 (Markov's inequality). For a random variable $X\colon \Omega \to \mathbb{R}$ that assumes only non-negative values and $t \in \mathbb{R}^+$,

$$\mathrm{Prob}(X \geq t \cdot \mathrm{E}(X)) \leq \frac{1}{t}.$$

Lemma A.6 (Chernoff bounds). Let $X = X_1 + \cdots + X_n$ be the sum of independent random variables with $X_i \in \{0, 1\}$ for all $1 \leq i \leq n$. Then

$$\begin{aligned}
\mathrm{Prob}(X \geq (1+\delta)\mathrm{E}(X)) &< \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mathrm{E}(X)} && \text{for } \delta > 0 \\
\mathrm{Prob}(X \leq (1-\delta)\mathrm{E}(X)) &\leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mathrm{E}(X)} && \text{for } 0 < \delta < 1 \\
\mathrm{Prob}(X \geq (1+\delta)\mathrm{E}(X)) &\leq e^{-\mathrm{E}(X)\delta^2/3} && \text{for } 0 < \delta < 1 \\
\mathrm{Prob}(X \leq (1-\delta)\mathrm{E}(X)) &\leq e^{-\mathrm{E}(X)\delta^2/2} && \text{for } 0 < \delta < 1.
\end{aligned}$$

These bounds also hold if $\mathrm{E}(X)$ is consistently replaced by a pessimistic bound on $\mathrm{E}(X)$ (i. e., an upper bound for the first and the third inequality and a lower bound for the second and the fourth one), see Scheideler (2000, page 52f.).

We state two Hoeffding bounds. The first bounds is adapted from (Scheideler, 2000, Theorem 3.37). The second bound follows from the first one by symmetry of X_i and $-X_i$.

Lemma A.7 (Hoeffding bounds). Let $X = X_1 + \cdots + X_n$ be the sum of independent random variables with $a_i \leq X_i \leq b_i$ for all $1 \leq i \leq n$. Then for $\delta > 0$

$$\begin{aligned}
\mathrm{Prob}(X \geq \mathrm{E}(X) + \delta) &\leq e^{-2\delta^2 / \sum_{i=1}^{n} (b_i - a_i)^2} \\
\mathrm{Prob}(X \leq \mathrm{E}(X) - \delta) &\leq e^{-2\delta^2 / \sum_{i=1}^{n} (b_i - a_i)^2}.
\end{aligned}$$

The following tail inequality (Scheideler, 2000, Theorem 3.67) is an extension of the Azuma-Hoeffding inequality.

Lemma A.8 (Method of Bounded Martingale Differences). Let $X_1, \ldots, X_n$ be arbitrary (not necessarily independent) random variables and let f be a function such that for every $1 \leq i \leq n$ there is a $c_i \geq 0$ with

$$|\mathrm{E}(f \mid X_1, \ldots, X_i) - \mathrm{E}(f \mid X_1, \ldots, X_{i+1})| \leq c_i.$$

Then for $\delta > 0$

$$\begin{aligned}
\mathrm{Prob}(f \geq \mathrm{E}(f) + \delta) &\leq e^{-\delta^2 / \left(2 \sum_{i=1}^{n} c_i^2\right)} \\
\mathrm{Prob}(f \leq \mathrm{E}(f) - \delta) &\leq e^{-\delta^2 / \left(2 \sum_{i=1}^{n} c_i^2\right)}.
\end{aligned}$$

The following variant for Stirling's approximation can be found, e. g., in Mitzenmacher and Upfal (2005, page 246).

Lemma A.9 (Stirling's approximation). For $n \in \mathbb{N}$,

$$\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \;\leq\; n! \;\leq\; \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \cdot e^{1/(12n)}.$$

In particular,

$$n! \;\geq\; \left(\frac{n}{e}\right)^n.$$

The following estimations are well known and easy to prove; the last inequality follows from Stirling's approximation.

Lemma A.10 (Binomial coefficients). For $k, n \in \mathbb{N}$ with $k \leq n$,

$$\frac{n^k}{k^k} \;\leq\; \binom{n}{k} \;\leq\; \frac{n^k}{k!} \;\leq\; \left(\frac{ne}{k}\right)^k.$$

The next estimation is contained in the proof of Lemma 2.10 in Mitzenmacher and Upfal (2005).

Lemma A.11 (Harmonic numbers). For $n \in \mathbb{N}$,

$$H_n := \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \;\leq\; \ln(n) + 1.$$

We present three bounds for the exponential function. The first bound is contained in Motwani and Raghavan (1995, Proposition B.3). The second one holds for the two boundary values $x \in \{-1, 0\}$ and for all values in between since $1 + x/2$ is linear and e^x is convex. The third bound follows from the first one as $e^{-x} \geq 1 - x \Leftrightarrow e^x \leq 1/(1-x)$ for $x < 1$.

Lemma A.12 (Bounds for the exponential function).

$$\begin{aligned} 1 + x \;&\leq\; e^x && \text{for } x \in \mathbb{R}, \\ e^x \;&\leq\; 1 + \frac{x}{2} && \text{for } -1 \leq x \leq 0, \\ e^x \;&\leq\; \frac{1}{1-x} && \text{for } x < 1. \end{aligned}$$

The following estimations are useful when dealing with standard mutations. Proofs are given by Motwani and Raghavan (1995, Proposition B.3).

Lemma A.13. For $n \in \mathbb{N}$,

$$\left(1 - \frac{1}{n}\right)^n \;\leq\; \frac{1}{e} \;\leq\; \left(1 - \frac{1}{n}\right)^{n-1}.$$

The last statements are well known and can be proved by simple inductions.

Lemma A.14 (Geometric series). For $|x| < 1$,

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \qquad \text{and} \qquad \sum_{i=1}^{\infty} x^i = \frac{x}{1-x}.$$

Lemma A.15 (Bernoulli's inequality). For $r \in \mathbb{N}_0$ and $x \in \mathbb{R}, x \geq -1$,

$$(1+x)^r \geq 1 + rx.$$

Bibliography

Aickelin, U., Burke, E. K., and Li, J. (2007). An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. *Journal of the Operational Research Society*, 58:1574–1585.

Attiratanasunthron, N. and Fakcharoenphol, J. (2008). A running time analysis of an ant colony optimization algorithm for shortest paths in directed acyclic graphs. *Information Processing Letters*, 105(3):88–92.

Balaprakash, P., Birattari, M., Stützle, T., and Dorigo, M. (2006). Incremental local search in ant colony optimization: Why it fails for the quadratic assignment problem. In: *Proceedings of ANTS Workshop (ANTS '06)*, pages 156–166.

Boughaci, D. and Drias, H. (2005). Efficient and experimental meta-heuristics for MAX-SAT problems. In: *Workshop on Experimental/Efficient Algorithms*, volume 3503 of *LNCS*, pages 501–512.

Bratton, D. and Kennedy, J. (2007). Defining a standard for particle swarm optimization. In: *Swarm Intelligence Symposium (SIS 2007)*, pages 120–127. IEEE Press.

Briest, P., Brockhoff, D., Degener, S., Englert, M., Gunia, C., Heering, O., Jansen, T., Leifhelm, M., Plociennik, K., Röglin, H., Schweer, A., Sudholt, D., Tannenbaum, S., and Wegener, I. (2004a). Evolutionäre Algorithmen zwischen experimenteller und theoretischer Analyse – Endbericht der Projektgruppe 427. Technical report, Universität Dortmund.

Briest, P., Brockhoff, D., Degener, S., Englert, M., Gunia, C., Heering, O., Jansen, T., Leifhelm, M., Plociennik, K., Röglin, H., Schweer, A., Sudholt, D., Tannenbaum, S., and Wegener, I. (2004b). FrEAK – Free Evolutionary Algorithm Kit. Available from `http://sourceforge.net/projects/freak427/`.

Brockhoff, D., Friedrich, T., Hebbinghaus, N., Klein, C., Neumann, F., and Zitzler, E. (2007). Do additional objectives make a problem harder? In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '07)*, pages 765–772. ACM Press.

Chaiyaratana, N., Piroonratana, T., and Sangkawelert, N. (2007). Effects of diversity control in single-objective and multi-objective genetic algorithms. *Journal of Heuristics*, 13(1):1–34.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. The MIT Press, 2nd edition.

Cotta, C. (2005). Memetic algorithms with partial Lamarckism for the shortest common supersequence problem. In: *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, volume 3562 of *LNCS*, pages 84–91.

Dawkins, R. (1976). *The Selfish Gene*. Oxford University Press, New York.

Doerr, B., Hebbinghaus, N., and Neumann, F. (2007a). Speeding up evolutionary algorithms through unsymmetric mutation operators. *Evolutionary Computation*, 15:401–410.

Doerr, B. and Johannsen, D. (2007a). Adjacency list matchings—an ideal genotype for cycle covers. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '07)*, pages 1203–1210. ACM Press.

Doerr, B. and Johannsen, D. (2007b). Refined runtime analysis of a basic ant colony optimization algorithm. In: *Proceedings of the Congress of Evolutionary Computation (CEC '07)*, pages 501–507. IEEE Press.

Doerr, B., Johannsen, D., and Tang, C. H. (2008). How single ant ACO systems optimize pseudo-Boolean functions. In: *Parallel Problem Solving from Nature (PPSN X)*, pages 378–388. Springer.

Doerr, B., Neumann, F., Sudholt, D., and Witt, C. (2007b). On the runtime analysis of the 1-ANT ACO algorithm. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '07)*, pages 33–40. ACM Press.

Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344:243–278.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. MIT Press.

Droste, S. (2006). A rigorous analysis of the compact genetic algorithm for linear functions. *Natural Computing*, 5(3):257–283.

Droste, S., Jansen, T., Tinnefeld, K., and Wegener, I. (2003). A new framework for the valuation of algorithms for black-box optimization. In: *Foundations of Genetic Algorithms 7 (FOGA)*, pages 253–270. Morgan Kaufmann, San Francisco, CA.

Droste, S., Jansen, T., and Wegener, I. (2000). A natural and simple function which is hard for all evolutionary algorithms. In: *Proceedings of the 26th annual Conference of the IEEE Industrial Electronics Society (IECON '00)*, pages 2704–2709. IEEE Press.

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

Droste, S., Jansen, T., and Wegener, I. (2002b). Optimization with randomized search heuristics—the (A)NFL theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 287(1):131–144.

Droste, S., Jansen, T., and Wegener, I. (2006). Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39(4):525–544.

Droste, S. and Wiesmann, D. (2003). On the design of problem-specific evolutionary algorithms. In: *Advances in Evolutionary Computing: Theory and Applications*, pages 153–173. Springer-Verlag New York, Inc.

Englert, M., Röglin, H., and Vöcking, B. (2007). Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. In: *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1295–1304.

Fischer, S. (2004). A polynomial upper bound for a mutation-based algorithm on the two-dimensional Ising model. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '04)*, pages 1100–1112. Springer.

Fischer, S. and Wegener, I. (2005). The one-dimensional Ising model: Mutation versus recombination. *Theoretical Computer Science*, 344(2–3):208–225.

Friedrich, T., Hebbinghaus, N., and Neumann, F. (2007). Rigorous analyses of simple diversity mechanisms. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '07)*, pages 1219–1225. ACM Press.

Friedrich, T., Oliveto, P. S., Sudholt, D., and Witt, C. (2008). Theoretical analysis of diversity mechanisms for global exploration. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '08)*, pages 945–952. ACM Press.

Garnier, J., Kallel, L., and Schoenauer, M. (1999). Rigorous hitting times for binary mutations. *Evolutionary Computation*, 7(2):173–203.

Giel, O. and Wegener, I. (2003). Evolutionary algorithms and the maximum matching problem. In: *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS '03)*, pages 415–426. Springer.

Goldberg, D. E., Van Hoyweghen, C., and Naudts, B. (2002). From twomax to the Ising model: Easy and hard symmetrical problems. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '02)*, pages 626–633. Morgan Kaufmann.

Gutjahr, W. J. (2000). A graph-based ant system and its convergence. *Future Generation Computer Systems*, 16:873–888.

Gutjahr, W. J. (2003). A generalized convergence result for the graph-based ant system metaheuristic. *Probability in the Engineering and Informational Sciences*, 17:545–569.

Gutjahr, W. J. (2007). Mathematical runtime analysis of ACO algorithms: Survey on an emerging issue. *Swarm Intelligence*, 1:59–79.

Gutjahr, W. J. (2008). First steps to the runtime complexity analysis of ant colony optimization. *Computers and Operations Research*, 35(9):2711–2727.

Gutjahr, W. J. and Sebastiani, G. (2008). Runtime analysis of ant colony optimization with best-so-far reinforcement. *Methodology and Computing in Applied Probability*, 10:409–433.

Hajek, B. (1982). Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 14:502–525.

Hart, W., Krasnogor, N., and Smith, J. E. (2004a). Editorial introduction. *Evolutionary Computation: Special Issue on Memetic Algorithms*, 12(3):v–vi.

Hart, W. E. (1994). *Adaptive Global Optimization with Local Search*. Ph.D. thesis, University of California, San Diego, CA.

Hart, W. E. (2003). Locally-adaptive and memetic evolutionary pattern search algorithms. *Evolutionary Computation*, 11(1):29–51.

Hart, W. E., Krasnogor, N., and Smith, J. E., editors (2004b). *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*. Springer.

He, J. and Yao, X. (2004). A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search: Foundations & Applications*. Elsevier/Morgan Kaufmann.

Hutter, M. and Legg, S. (2006). Fitness uniform optimization. *IEEE Transactions on Evolutionary Computation*, 10:568–589.

Igel, C. and Toussaint, M. (2004). A no-free-lunch theorem for non-uniform distributions of target functions. *Journal of Mathematical Modelling and Algorithms*, 3(4):313–322.

Ishibuchi, H., Yoshida, T., and Murata, T. (2003). Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation*, 7(2):204–223.

Jägersküpper, J. (2007). Algorithmic analysis of a basic evolutionary algorithm for continuous optimization. *Theoretical Computer Science*, 379(3):329–347.

Jansen, T. (Winter term 2006/2007). Evolutionäre Algorithmen. Lecture notes to a lecture at the Universität Dortmund, available from `http://ls2-www.cs.tu-dortmund.de/lehre/winter200607/evolA/`.

Jansen, T., De Jong, K. A., and Wegener, I. (2005). On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13:413–440.

Jansen, T. and Wegener, I. (2001). Evolutionary algorithms—how to cope with plateaus of constant fitness and when to reject strings of the same fitness. *IEEE Transactions on Evolutionary Computation*, 5:589–599.

Jansen, T. and Wegener, I. (2005). Real royal road functions—where crossover provably is essential. *Discrete Applied Mathematics*, 149:111–125.

Jansen, T. and Wegener, I. (2007). A comparison of simulated annealing with a simple evolutionary algorithm on pseudo-Boolean functions of unitation. *Theoretical Computer Science*, 386(1-2):73–93.

Jerrum, M. and Sorkin, G. B. (1998). The Metropolis algorithm for graph bisection. *Discrete Applied Mathematics*, 82(1–3):155–175.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In: *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948. IEEE Press.

Kennedy, J. and Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. In: *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics (WMSCI)*, pages 4104–4109.

Kennedy, J., Eberhart, R. C., and Shi, Y. (2001). *Swarm Intelligence*. Morgan Kaufmann.

Kernighan, B. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Tech Journal*, 49(2):291–307.

Krasnogor, N. and Smith, J. (2005). A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488.

Krasnogor, N. and Smith, J. (2008). Memetic algorithms: The polynomial local search complexity theory perspective. *Journal of Mathematical Modelling and Algorithms*, 7:3–24.

Land, M. W. S. (1998). *Evolutionary Algorithms with Local Search for Combinatorial Optimization*. Ph.D. thesis, University of California, San Diego, CA.

Levine, J. and Ducatelle, F. (2004). Ant colony optimisation and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(7):705–716.

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

Lourenço, H. R., Martin, O., and Stützle, T. (2002). Iterated local search. In: *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA.

Mahfoud, S. W. (1997). Niching methods. In: T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C6.1:1–4. Institute of Physics Publishing and Oxford University Press, Bristol, New York.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18:50–60.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30.

Merkle, D. and Middendorf, M. (2002). Modelling the dynamics of Ant Colony Optimization algorithms. *Evolutionary Computation*, 10(3):235–262.

Merz, P. (2004). Advanced fitness landscape analysis and the performance of memetic algorithms. *Evolutionary Computation*, 12(3):303–326.

Michiels, W., Aarts, E., and Korst, J. (2007). *Theoretical Aspects of Local Search*. Springer.

Mitzenmacher, M. and Upfal, E. (2005). *Probability and Computing*. Cambridge University Press.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & OR*, 24(11):1097–1100.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, CA.

Moscato, P. (1999). Memetic algorithms: a short introduction. In: D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 219–234. McGraw-Hill.

Motwani, R. and Raghavan, P. (1995). *Randomized Algorithms*. Cambridge University Press.

Mühlenbein, H. (1992). How genetic algorithms really work: Mutation and hillclimbing. In: *Parallel Problem Solving from Nature (PPSN II)*, pages 15–26.

Neri, F., Toivanen, J., Cascella, G. L., and Ong, Y.-S. (2007). An adaptive multimeme algorithm for designing HIV multidrug therapies. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(2):264–278.

Neumann, F. (2004). Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. In: *Proceedings of the Congress of Evolutionary Computation (CEC '04)*, pages 904–910. IEEE Press.

Neumann, F. and Reichel, J. (2008). Approximating minimum multicuts by evolutionary multi-objective algorithms. In: *Parallel Problem Solving from Nature (PPSN X)*, volume 5199 of *LNCS*, pages 72–81. Springer.

Neumann, F. and Wegener, I. (2004). Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '04)*, volume 3102 of *LNCS*, pages 713–724. Springer.

Neumann, F. and Wegener, I. (2006). Minimum spanning trees made easier via multi-objective optimization. *Natural Computing*, 5(3):305–319.

Neumann, F. and Wegener, I. (2007). Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378(1):32–40.

Neumann, F. and Witt, C. (2006). Runtime analysis of a simple ant colony optimization algorithm. In: *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC '06)*, volume 4288 of *LNCS*, pages 618–627. Springer.

Neumann, F. and Witt, C. (2008). Ant Colony Optimization and the minimum spanning tree problem. In: *Proceedings of Learning and Intelligent Optimization (LION '07)*, volume 5313 of *LNCS*, pages 153–166. Springer.

Oliveto, P., He, J., and Yao, X. (2008). Population-based evolutionary algorithms for the vertex cover problem. In: *Proceedings of the Congress of Evolutionary Computation (CEC '08)*.

Oliveto, P. S., He, J., and Yao, X. (2007). Computational complexity analysis of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4(3):281–293.

Oliveto, P. S. and Witt, C. (2008). Simplified drift analysis for proving lower bounds in evolutionary computation. In: *Parallel Problem Solving from Nature (PPSN X)*, volume 5199 of *LNCS*, pages 82–91. Springer.

Ong, Y.-S., Lim, M.-H., Zhu, N., and Wong, K.-W. (2006). Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(1):141–152.

Papadimitriou, C. (1994). *Computational Complexity*. Addison Wesley.

Pelikan, M. and Goldberg, D. E. (2000). Genetic algorithms, clustering, and the breaking of symmetry. In: *Parallel Problem Solving from Nature (PPSN VI)*, pages 385–394. Springer.

Quick, R. J., Rayward-Smith, V. J., and Smith, G. D. (1998). Fitness distance correlation and ridge functions. In: *Parallel Problem Solving from Nature (PPSN V)*, pages 77–86. Springer.

Raidl, G. R., Koller, G., and Julstrom, B. A. (2006). Biased mutation operators for subgraph-selection problems. *IEEE Transactions on Evolutionary Computation*, 10(2):145–156.

Rechenberg, I. (1973). *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart.

Reichel, J. and Skutella, M. (2007). Evolutionary algorithms and matroid optimization problems. In: *Proceedings of the Genetic and Evolutionary Compution Conference (GECCO '07)*, pages 947–954. ACM Press.

Rudolph, G. (1997a). *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovač.

Rudolph, G. (1997b). How mutation and selection solve long-path problems in polynomial expected time. *Evolutionary Computation*, 4(2):195–205.

Scharnow, J., Tinnefeld, K., and Wegener, I. (2004). The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, 3(4):349–366.

Scheideler, C. (2000). *Probabilistic Methods for Coordination Problems.* HNI-Verlagsschriftenreihe 78, University of Paderborn. Habilitation Thesis, available at `http://www14.in.tum.de/personen/scheideler/index.html.en`.

Schwefel, H.-P. (1975). *Evolutionsstrategie und numerische Optimierung*. Dr.-Ing. Thesis, Technical University of Berlin, Department of Process Engineering.

Shi, Y. and Eberhart, R. C. (1998). Parameter selection in particle swarm optimization. In: *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 591–600.

Sinha, A., Chen, Y., and Goldberg, D. E. (2004). Designing efficient genetic and evolutionary algorithm hybrids. In: W. E. Hart, N. Krasnogor, and J. E. Smith, editors, *Recent Advances in Memetic Algorithms*, pages 259–288. Springer.

Stoer, M. and Wagner, F. (1994). A simple min cut algorithm. In: *Proceedings of the Second Annual European Symposium on Algorithms (ESA '94)*, pages 141–147.

Storch, T. and Wegener, I. (2004). Real royal road functions for constant population size. *Theoretical Computer Science*, 320:123–134.

Stützle, T. and Hoos, H. H. (2000). MAX-MIN ant system. *Journal of Future Generation Computer Systems*, 16:889–914.

Sudholt, D. (2005). Crossover is provably essential for the Ising model on trees. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '05)*, pages 1161–1167. ACM Press.

Sudholt, D. (2006a). Local search in evolutionary algorithms: the impact of the local search frequency. In: *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC '06)*, volume 4288 of *LNCS*, pages 359–368. Springer.

Sudholt, D. (2006b). On the analysis of the (1+1) memetic algorithm. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '06)*, pages 493–500. ACM Press.

Ursem, R. K. (2002). Diversity-guided evolutionary algorithms. In: *Parallel Problem Solving from Nature (PPSN-VII)*, pages 462–471. Springer.

Wald, A. (1944). On cumulative sums of random variables. *The Annals of Mathematical Statistics*, 15(3):283–296.

Watson, J.-P., Howe, A. E., and Whitley, L. D. (2003). An analysis of iterated local search for job-shop scheduling. In: *Fifth Metaheuristics International Conference (MIC '03)*.

Wegener, I. (2002). Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In: R. Sarker, X. Yao, and M. Mohammadian, editors, *Evolutionary Optimization*, pages 349–369. Kluwer.

Wegener, I. (2005a). *Complexity Theory—Exploring the Limits of Efficient Algorithms*. Springer.

Wegener, I. (2005b). Simulated annealing beats Metropolis in combinatorial optimization. In: *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP '05)*, volume 3580 of *LNCS*, pages 589–601.

Wegener, I. (Summer term 2002). Evolutionäre Algorithmen. Lecture notes to a lecture at the Universität Dortmund, available from `http://ls2-www.cs.tu-dortmund.de/lehre/sommer2002/evol-alg/`.

Wegener, I. and Witt, C. (2005). On the optimization of monotone polynomials by simple randomized search heuristics. *Combinatorics, Probability and Computing*, 14:225–247.

Williams, D. (1991). *Probability with Martingales*. Cambridge University Press.

Witt, C. (2005). Worst-case and average-case approximations by simple randomized search heuristics. In: *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS '05)*, volume 3404 of *LNCS*, pages 44–56. Springer.

Witt, C. (2006). Runtime analysis of the (μ+1) EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14(1):65–86.

Witt, C. (2008). Population size versus runtime of a simple evolutionary algorithm. *Theoretical Computer Science*, 403(1):104–120.

Witt, C. (2009). Why standard particle swarm optimisers elude a theoretical runtime analysis. In: *Foundations of Genetic Algorithms 10 (FOGA '09)*. ACM Press. To appear.

Index