Runtime Analysis of Success-Based Parameter Control Mechanisms for Evolutionary Algorithms on Multimodal Problems

Mario Alejandro Hevia Fajardo

The University of Sheffield
Faculty of Engineering
Department of Computer Science

This dissertation is submitted for the degree of

Doctor of Philosophy

April 2023

Declaration

Mario Alejandro Hevia Fajardo
April 2023

Acknowledgements

Completing a PhD is a long and challenging journey that requires determination, hard work, and support from many people. It is my pleasure to express my gratitude and appreciation to those who have helped me along the way.

I am fortunate to have had Dirk Sudholt as my supervisor throughout my PhD studies. I am grateful for his guidance, encouragement, and patience. His expertise, insights, and feedback were invaluable to my research, and I am grateful for the opportunity to work with him.

I would like to express my deep gratitude to Pietro Oliveto, whose insightful discussions were instrumental in shaping my ideas and research direction. His guidance and support were crucial to my success, and I am grateful for the time and effort he invested in my work.

I would also like to express my sincere appreciation to John Clark and Benjamin Doerr for taking the time to read my thesis and serving as my examiners. I am grateful for their dedication and commitment to ensuring that my thesis met the highest academic standards, and I am honored to have had them examine me on my work.

I am also grateful to Donya Yazdani and George T. Hall for their help and friendship throughout my studies. Their support and encouragement helped me overcome many obstacles, and their friendship made the journey more enjoyable.

I thank my fiancée Jagoda Karpowicz for her unwavering support throughout my PhD studies, especially during the long periods of isolation. Her encouragement, advice, and love were invaluable, and I am grateful for the time we spent together.

Last but not least, I would like to express my heartfelt gratitude to my family. Their love, support, and encouragement were the driving force behind my success, and I would not have been able to pursue my dreams without them. I am grateful for their sacrifices, their unwavering support, and their belief in me.

Each and every one of you played a crucial role, in one way or another, in the completion of this thesis.

Abstract

Evolutionary algorithms are simple general-purpose optimisers often used to solve complex engineering and design problems. They mimic the process of natural evolution: they use a population of possible solutions to a problem that evolves by mutating and recombining solutions, identifying increasingly better solutions over time. Evolutionary algorithms have been applied to a broad range of problems in various disciplines with remarkable success. However, the reasons behind their success are often elusive: their performance often depends crucially, and unpredictably, on their parameter settings. It is, furthermore, well known that there are no globally good parameters, that is, the correct parameters for one problem may differ substantially to the parameters needed for another, making it harder to translate previous successfully implemented parameters to new problems. Therefore, understanding how to properly select the parameters is an important but challenging task. This is commonly known as the parameter selection problem.

A promising solution to this problem is the use of automated dynamic parameter selection schemes (parameter control) that allow evolutionary algorithms to identify and continuously track optimal parameters throughout the course of evolution without human intervention. In recent years the study of parameter control mechanisms in evolutionary algorithms has emerged as a very fruitful research area. However, most existing runtime analyses focus on simple problems with benign characteristics, for which fixed parameter settings already run efficiently and only moderate performance gains were shown. The aim of this thesis is to understand how parameter control mechanisms can be used on more complex and challenging problems with many local optima (multimodal problems) to speed up optimisation.

We use advanced methods from the analysis of algorithms and probability theory to evaluate the performance of evolutionary algorithms, estimating the expected time until an algorithm finds satisfactory solutions for illustrative and relevant optimisation problems as a vital stepping stone towards designing more efficient evolutionary algorithms. We first analyse current parameter control mechanisms on multimodal problems to understand their strengths and weaknesses. Subsequently we use this knowledge to design parameter control mechanisms that mitigate the weaknesses of current mechanisms while maintaining their strengths. Finally, we show with theoretical and empirical analyses that these enhanced parameter control mechanisms are able to outperform the best fixed parameter settings on multimodal optimisation.

Contents

Nomenclature

Acronyms / Abbreviations

EA	Evolutionary Algorithm
GA	Genetic Algorithm
RLS	Randomised Local Search
RSH	Randomised Search Heuristics

Mathematical Symbols

λ	Offspring population size
$\ln(x)$	Natural logarithm of x
$\log(x)$	Logarithm base 2 of x
μ	Parent population size
e	Euler's number $e = \exp(1) = 2.7182\ldots$
n	Problem size, number of bits of a solution
p	Mutation probability or mutation rate
r	Mutation parameter

Sets

S	The set of search points in a given search space
$\mathbb{N}$	The set of the natural numbers
$\mathbb{R}$	The set of the real numbers
$\mathbb{N}_0$	The set of the natural numbers including 0

Chapter 1

Introduction

Optimisation is the selection of a *best option* from a set of available or feasible alternatives. More specifically, the optimisation problem is to systematically search for an input, in order to obtain a desired output from a model or function, while satisfying certain constraints. The term optimisation is also used to describe the minimisation or maximisation of a certain function $f : S \to \mathbb{R}$, where S is a given search space [145].

Optimisation is ubiquitous in science, industry and life [168]. Tasks such as product and process design, maximising profit of an organisation while minimising the economic risks and buying the best product with a limited budget are examples of optimisation problems. These and many other problems we encounter can be considered as optimisation tasks, hence it is one of the most important topics in engineering, mathematics, computer science and several other disciplines [77].

Given the wide range of optimisation problems, they are often subdivided into different classes according to their characteristics. It is difficult to provide a comprehensive list of all different optimisation classes. This thesis focuses on *pseudo-Boolean* optimisation problems of the form $f : \{0,1\}^n \to \mathbb{R}$, that is, optimisation problems where the search space S is defined over n binary variables. The study of pseudo-Boolean optimisation is extremely useful because pseudo-Boolean functions can be used to model a wide variety of problems, including constraint satisfaction problems (e. g. N-Queens, map coloring, SAT), graph theory problems (e. g. minimum vertex cover, minimum spanning tree) and many other combinatorial optimisation problems (e. g. knapsack problem, makespan scheduling problem).

1.1 Motivation

Throughout the years computer scientists have developed techniques and algorithms to solve optimisation problems. If we have an explicit representation of the underlying function f of the optimisation problem, including its derivatives, we can apply classical mathematical optimisation techniques. Even without such explicit representations, under the best of circumstances algorithms can be tailored towards a given optimisation problem to obtain an exact solution or a good approximation efficiently. However, this is a difficult task that requires good understanding of the problem and often there is a lack of resources, time or expertise to design such tailored algorithms. For such cases, a set of general-purpose iterative optimisation heuristics that use the help of stochastic decisions have been proposed and are widely used. These algorithms are called *Randomised Search Heuristics* (RSH). One of the most common types of RSH used for this purpose are *Evolutionary Algorithms* (EAs) [145].

Broadly speaking, evolutionary algorithms are RSH inspired by natural evolution, and they are studied by the research area of Evolutionary Computation. An evolutionary algorithm uses a population of candidate solutions and "evolves" the population by applying operators such as mutation, recombination and selection. Evolutionary algorithms have been applied to a wide range of problems. The popularity of these algorithms can be attributed to their ease of implementation and their effectiveness in problems with little a priori knowledge. Evolutionary algorithms can be used even in extreme cases where the optimisation problem is in a so-called black-box scenario, that is, the algorithm does not have direct access to its objective function, therefore treats the optimisation problem as a black-box where it can only probe the search space by evaluating candidate solutions [70].

Even though evolutionary algorithms can be applied to a wide range of problems, it is well known that the efficiency of the optimisation process may depend drastically on its parameters and the problem in hand [51, 140]. Therefore, an important aspect of using evolutionary algorithms is an understanding of how changes in its parameters affect their performance. An approach for parameter selection is to theoretically analyse the optimisation time (runtime analysis) of evolutionary algorithms to understand how different parameter settings affect their performance on different fitness landscapes. This approach tends to be highly specific for the problem and instance studied, nonetheless it has given us a better understanding of how to properly set the parameters of evolutionary algorithms. Moreover, it has shown that during the optimisation process the optimal parameter values may change, making any static parameter choice have sub-optimal performance [51].

In order to overcome these difficulties, several strategies to adapt the parameter values during the run have been proposed. These dynamic parameter setting strategies are called parameter control mechanisms; they aim to automatically find the optimal parameters at every stage of the optimisation process. In continuous optimisation, parameter control is indispensable to ensure convergence to the optimum, therefore, non-static parameter choices have been standard for several decades [45]. In contrast, in the discrete domain parameter control has only become more common in recent years. In particular, there has been an increasing interest in the theoretical study of parameter control mechanisms in order to understand how they work and when they perform better than static parameter settings. The growing interest can be attributed in part to theoretical studies demonstrating that fitness-dependent parameter control mechanisms can provably outperform the best static parameter settings [15, 21, 60, 67]. Despite the proven advantages, fitness-dependent mechanisms have an important drawback: to have an optimal performance they generally need to be tailored to a specific problem which needs a substantial knowledge of the problem in hand [51].

To overcome this constraint, several parameter control mechanisms that update the parameters using simple rules based on the success of the previous iteration have been proposed. These are called *success-based* parameter control mechanisms (also called *self-adjusting*). Theoretical studies have proven that in spite of their simplicity, these mechanisms are able to use *good* parameter values throughout the optimisation, obtaining the same or better performance than any static parameter choice. In fact, it has been shown in several theoretical studies that they can be faster than any static parameter choice on simple problems with only one (local or global) optimum (see the survey by Doerr and Doerr [51]).

The problem is that up until now, we do not have a good understanding of how self-adjusting parameter control mechanisms behave in complex problems with more than one local optimum (multimodal problems), even though they are widely used to solve them. Aggravating the situation, most real-world problems tend to be multimodal in nature.

The purpose of this thesis is to extend the fruitful research of success-based parameter control mechanisms for evolutionary algorithms towards the study of multimodal problems. In the literature when solving multimodal problems, sometimes the optimisation algorithms aim to locate multiple optimal (or near-optimal) solutions in a single run [155]. This thesis focus on optimisation algorithms that seek a single global optimum on a multimodal fitness landscape.

More specifically, the main aim of this thesis is to understand how success-based parameter control mechanisms for evolutionary algorithms can be efficiently implemented for multimodal optimisation. Multimodal optimisation is a challenging scenario for success-based parameter control mechanisms because once a local optimum is reached, the success of previous iterations does not give a good indication of what parameters are needed to escape the local optimum. Additionally, the information obtained from unsuccessful iterations may diverge the parameters towards too small or too large values that might not be the best choice to leave the local optimum.

A natural question that arises is whether current success-based parameter control mechanisms already have a good performance on multimodal problems. There is some empirical evidence that success-based parameter control mechanisms struggle on these problems (e.g. [74, 82, 99]), but there is little to none theoretical evidence. This thesis addresses this question by theoretical means shedding light on the strengths and weaknesses of current success-based parameter control mechanisms when applied on multimodal problems.

Another crucial question addressed by this thesis is how to mitigate the weaknesses of current success-based parameter control mechanisms while maintaining their strengths. We aim for "balanced" parameter control mechanisms that can hill-climb and deal with local optima efficiently. As mentioned before, current success-based parameter control mechanisms have a good performance on simple unimodal problems. It is especially important to maintain this behaviour because many multimodal problems have easy parts with similar characteristics as simple problems. Hence, any modifications to current success-based parameter control mechanisms need to take into account both easy and hard parts of the optimisation and not be over-tailored towards local optima or rugged fitness landscapes.

1.2 Thesis Outline

This thesis contains 7 chapters. The first 3 chapters give the necessary background of the subject including a literature review of the state of the art in parameter control mechanisms and the mathematical methods commonly used for their runtime analyses. Afterwards the next 3 chapters contain our main theoretical contributions. The last chapter contains a summary and conclusions of our work.

Chapter 2 introduces evolutionary algorithms and explains how the computational complexity of evolutionary algorithms is commonly analysed. In Section 2.1 evolutionary algorithms are introduced, explaining their main components and introducing the most commonly studied evolutionary algorithms in the theoretical field. Section 2.2 explains what is computational complexity in the context of evolutionary algorithms and details the most common approach to analysing the computational complexity of evolutionary algorithms. Section 2.3 covers common benchmark functions that are used in the analysis of evolutionary algorithms because they showcase common characteristics of real-world problems. In Section 2.4 we describe common methods used for the runtime analysis of evolutionary algorithms. The chapter ends with Section 2.5 where we give an overview of the parameter settings and how the theoretical field has helped to improve our understanding of the impact parameter settings have in the performance of evolutionary algorithms.

In Chapter 3 we present the state of the art in parameter control mechanisms. Given that this thesis is theory-driven the chapter focuses on theoretical studies, but also contains empirical studies because it is common for parameter control mechanisms implemented in empirical studies to be later analysed by theoretical means. The chapter follows five lines of studies that focus on how to adjust a specific parameter of an algorithm. Sections 3.1 and 3.2 showcase studies related to dynamic mutation rates and offspring population sizes for the $(1 + \lambda)$ EA. In Section 3.4 one of the most successful implementations of parameter control mechanisms is presented, that is, the self-adjusting $(1 + (\lambda, \lambda))$ GA [50]. This was the first success-based parameter control mechanism proven to reduce the optimisation time of an algorithm by more than a constant factor, compared to optimal static parameter settings. Section 3.3 reviews works analysing self-adaptive mutation rates. In Section 3.5 we describe runtime analyses of RLS_k with dynamic parameters, including recent studies of hyper-heuristics.

We start presenting our main contributions in Chapter 4. Here, we ask whether current parameter control mechanisms are able to find and maintain suitable parameter values on multimodal problems. To answer this question in Section 4.2 we analyse the well-known self-adjusting $(1 + (\lambda, \lambda))$ GA. We show that for the standard multimodal benchmark function Jump_k the self-adjusting $(1 + (\lambda, \lambda))$ GA is less efficient than other crossover-based algorithms using static parameter settings. In Section 4.3 we study variants proposed in the literature showing that restricting the possible parameter values that the control mechanism can use is beneficial and resetting the parameters can improve the performance significantly on Jump_k functions. In Section 4.4 we implement an experimental analysis of the algorithms studied via theoretical means in previous sections. We show that our theoretical results extend to common optimisation problems, that is, we confirm empirically that the original self-adjusting $(1 + (\lambda, \lambda))$ GA is inefficient on common multimodal problems and the variations studied can improve its performance for multimodal optimisation whilst not (or slightly) affecting its performance on simple problems.

Most success-based evolutionary algorithms that have been theoretically analysed are elitist algorithms. Hence, the performance of parameter control mechanisms in non-elitist algorithms is not well understood. Despite this, there are many applications of non-elitist evolutionary algorithms for which an improved theoretical understanding of parameter control mechanisms could bring performance improvements similar to the ones seen for elitist algorithms. To tackle this research gap in the literature, in Chapter 5 we analyse the behaviour of success-based rules for non-elitist algorithms. This chapter highlights the importance of selecting the correct success-based rules in non-elitist algorithms. We consider a self-adjusting version of the $(1, \lambda)$ EA that adjusts the offspring population size with the use of success-based rules. In Section 5.3 we demonstrate that for a sufficiently small constant success rate the algorithm achieves the optimal asymptotic runtime on OneMax for all unary unbiased black-box algorithms. However, we also show that if the success rate is a sufficiently large constant the algorithm takes exponential time to solve OneMax and other common benchmark functions. In contrast, in Section 5.4 we show that the non-elitist self-adjusting $(1, \lambda)$ EA is not affected by the choice of the success rate (from positive constants) if the problem in hand is everywhere hard, that is, improvements are always hard to find. We finish this chapter with an experimental analysis of the self-adjusting $(1, \lambda)$ EA in Section 5.5. The experiments performed in this section helps us enhance our understanding of the parameter control mechanism used, showing in detail how the behaviour of the mechanism is affected by its hyper-parameters and the problem in hand.

In Chapter 6 we explore how success-based rules for non-elitist algorithms behave on multimodal problems. Our understanding of the behaviour of simple non-elitist algorithms such as the $(1, \lambda)$ EA on multimodal problems is limited. Therefore, we start in Section 6.3 by studying the $(1, \lambda)$ EA with static parameters on Cliff, improving previous upper bounds and showing a lower bound for all static offspring population sizes. Afterwards, in Section 6.4 we compare these bounds with the performance of the $(1, \lambda)$ EA with self adjusting offspring population size and show a polynomial performance improvement compared to any static offspring population size. Finally, in Section 6.5 we extend our theoretical analysis with an empirical analysis on common multimodal problems, showing promising results for all problems tested.

We end this thesis with a summary and conclusion of our results in Chapter 7. We also discuss promising directions for future work.

1.3 Main Contributions

In this thesis, we substantially advance our understanding of how success-based parameter control mechanisms for evolutionary algorithms can be efficiently implemented for multimodal optimisation. Our main contributions are summarised as follows:

1. We provide a rigorous runtime analysis of the self-adjusting $(1 + (\lambda, \lambda))$ GA for general function classes by presenting a new general method to find upper bounds for the self-adjusting $(1 + (\lambda, \lambda))$ GA that is easy to use and enables a transfer of runtime bounds from the $(1 + 1)$ EA to the self-adjusting $(1 + (\lambda, \lambda))$ GA. With the help of our new general method we prove tight upper and lower bounds, up to lower-order terms, for the runtime of the self-adjusting $(1 + (\lambda, \lambda))$ GA on $\textsc{Jump}_k$.

2. We prove that the parameter control mechanism of the self-adjusting $(1 + (\lambda, \lambda))$ GA rapidly increases λ to its maximum value when encountering a local optima. This is ineffective for problems where large jumps are required. Capping λ at smaller values is beneficial for such problems. Finally, resetting λ to 1 allows the parameter to cycle through the parameter space. We show that resets are effective for all $\textsc{Jump}_k$ problems: the self-adjusting $(1 + (\lambda, \lambda))$ GA performs as well as the $(1 + 1)$ EA with the optimal mutation rate and evolutionary algorithms with heavy-tailed mutation, apart from a small polynomial overhead.

3. We show that the self-adjusting $(1 + (\lambda, \lambda))$ GA presents a bimodal parameter landscape with respect to λ on $\textsc{Jump}_k$. For appropriate problem size n and jump size k, the landscape features a local optimum in a wide basin of attraction and a global optimum in a narrow basin of attraction. To our knowledge this is the first proof of a bimodal parameter landscape for the runtime of an evolutionary algorithm on a multimodal problem.

4. We give a rigorous runtime analysis of the non-elitist self-adjusting $(1, \lambda)$ EA adjusting λ using a one-$(s+1)$-th success rule. We prove that, if the success rate is appropriately small, the self-adjusting $(1, \lambda)$ EA optimises $\textsc{OneMax}$ in $O(n)$ expected generations and $O(n \log n)$ expected evaluations, the best possible runtime for any unary unbiased black-box algorithm. A small success rate is crucial: we also show that if the success rate is too large, the algorithm has an exponential runtime on $\textsc{OneMax}$ and other functions with similar characteristics. Large success rates stagnate the optimisation on "easy" parts of the optimisation.

5. We demonstrate that the self-adjusting $(1, \lambda)$ EA is robust with respect to the choice of the success rate in the absence of easy slopes. We define a class of *everywhere hard* fitness functions, where for all search points the probability of finding an improvement is small. We present a simple and easy to use general upper bound on the expected number of evaluations using the fitness-level method that asymptotically matches the bound obtained for the $(1 + 1)$ EA. For the expected number of generations, we show an upper bound of $O(d + \log(1/p^+_{\min}))$ where d is the number of non-optimal fitness values and $p^+_{\min}$ is the smallest probability of finding an improvement from any non-optimal search point. As a byproduct of our analysis, we also show an upper bound for the expected number of evaluations of the elitist self-adjusting $(1 + \lambda)$ EA on arbitrary fitness functions.

6. We refine results from [106] for the optimisation time of the $(1, \lambda)$ EA on $\textsc{Cliff}$. We prove that the expected runtime is $\Omega(\lambda\xi^\lambda)$ and $O(\lambda\xi^\lambda \log n)$ for a base of $\xi \approx 6.196878$, for reasonable values of λ. For the best fixed λ, we show that the expected runtime is $O(n^\eta \log^2 n)$ for a constant $\eta \approx 3.976770136$, and that it grows faster than any polynomial of degree less than η.

7. We provide an example of significant performance improvements through parameter control for a multimodal problem. We enhance the self-adjusting $(1, \lambda)$ EA with a resetting mechanism and prove that it is able to optimise Cliff in $O(n)$ expected generations and $O(n \log n)$ expected evaluations. This is faster than any static parameter choice by a factor of $\Omega(n^{2.9767})$ and it is asymptotically the best possible runtime for any unary unbiased black-box algorithm.

1.4 Underlying Publications

The contents of this thesis are based on the following publications. The authors are ordered alphabetically.

Chapter 4 is based on the following paper:

1. Mario Alejandro Hevia Fajardo and Dirk Sudholt. On the choice of the parameter control mechanism in the $(1 + (\lambda, \lambda))$ Genetic Algorithm. Submitted to: *ACM Transactions on Evolutionary Learning and Optimization*, 2022

   A preliminary version was published in:

   Mario Alejandro Hevia Fajardo and Dirk Sudholt. On the choice of the parameter control mechanism in the $(1 + (\lambda, \lambda))$ genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO' 20, page 832–840. ACM, 2020

Chapter 5 is based on the following papers:

2. Mario Alejandro Hevia Fajardo and Dirk Sudholt. Self-adjusting population sizes for non-elitist Evolutionary Algorithms. Submitted to: *Algorithmica*, 2022

   A preliminary version was published in:

   Mario Alejandro Hevia Fajardo and Dirk Sudholt. Self-adjusting population sizes for non-elitist Evolutionary Algorithms: Why success rates matter. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, page 1151–1159. ACM, 2021

3. Mario Alejandro Hevia Fajardo and Dirk Sudholt. Hard problems are easier for success-based parameter control. Submitted to: *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, ACM, 2022

Chapter 6 is based on the following paper:

4. Mario Alejandro Hevia Fajardo and Dirk Sudholt. Self-adjusting offspring population sizes outperform fixed parameters on the Cliff function. In *Proceedings of the 16th Workshop on Foundations of Genetic Algorithms*, FOGA '21, pages 5:1–5:15. ACM, 2021

Chapter 2

Background

Evolutionary computation as its name hints, is inspired by natural evolutionary processes. It mainly draws inspiration from Darwinian Evolution and genetics [77]. From Darwinian Evolution it takes the concept of natural selection, where individuals that are adapted to or fit the environmental conditions best, have a higher chance to create offspring. Another insight drawn from Darwin's theory is that, through random variations (mutation) new traits appear in the populations, which in turn help to create fitter individuals, driving evolution.

From genetics it uses the concept of genes. Each individual contains genes that describe or encode their behaviours and traits, therefore when mutations occur in such genes the individual changes its fitness to the environment. Also, during sexual reproduction a combination of the genes of both parents is made, generally creating a variation in the offspring's genes (and behaviours) different from each of the parents' genes.

The field of evolutionary computation uses these concepts to design automated optimisation algorithms called evolutionary algorithms.

2.1 Evolutionary Algorithms

All evolutionary algorithms follow the same general idea. Given an optimisation problem, a population of *individuals* (also called *search points* or *solutions*) is initialised. Using a fitness function, they assign fitness values to each of the individuals. Based on the fitness values or other indicators, evolutionary algorithms select a set of individuals (parents) to seed the next generation. An offspring population is created by applying variation operators to the parents, and later calculate their fitness values. Finally, the new generation is selected from the existing individuals, following certain criteria. This process is repeated until an optimal solution is found or another stopping criteria is met. Owing to the combination of variation operators and selection during the process, the population tends to increase in fitness.

A diagram from the scheme described before is shown in Figure 2.1. To design an evolutionary algorithm instance you need to define several components of this scheme, such as representation, initialisation, fitness function, population, parent selection, variation operators, survivor selection and stopping criteria.

Representation The first step to design an evolutionary algorithm is to define a mapping from a possible solution into a representation that the algorithm will use. The action of translating the solution into such representation is called encoding. An important caveat is that this mapping should be reversible, in order to be able to evaluate the new solutions created by the algorithm.

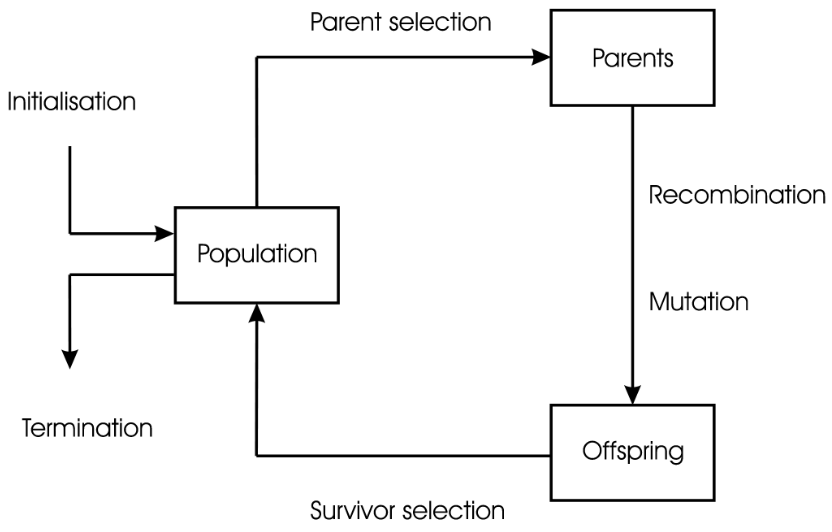

Figure 2.1: General scheme of an Evolutionary Algorithm. Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer, Introduction to Evolutionary Computing by A. E. Eiben, J. E. Smith © 2003

Initialisation The initialisation of an evolutionary algorithm in general is done by randomly sampling solutions from the search space. However, there is no restriction not to use previously known answers, as long as we take into account that this might affect the performance of the algorithm, guiding it to a local optimum.

Fitness Function The fitness function is a function that is used to represent the quality of a solution. It is used by the selection mechanisms to identify what solutions are helpful to the optimisation process. A common approach when using an evolutionary algorithm to solve an optimisation problem is to directly use the objective function as a fitness function.

Population The term population in an evolutionary algorithm refers to a multiset of individuals in a step of the optimisation process that an algorithm uses as solutions to the problem. A population can be defined by the amount of individuals, but it can also have more complex characteristics such as a spatial structure.

Parent Selection Parent selection is used in evolutionary algorithms to decide which individuals become parents of the next generation. The parent or parents selected create new individuals using the variation operators. An example is the use of random parent selection, which selects the parents uniformly at random.

Variation Operators Variation operators are used to create new individuals. Usually these operators make use of stochastic decisions. The most common operators are mutation and crossover (also called recombination). Mutation is a random modification of a single individual into a new one, while crossover combines two or more individuals by randomly selecting one or more parts from each parent to create offspring.

Survivor Selection The role of survivor selection is similar to parent selection, but in this case the selection decides which individuals will stay in the population.

Stopping Criteria The aim of evolutionary algorithms is to optimise a given problem, therefore a natural stopping criteria is when an individual with an *optimal* fitness is found. However, the fitness of an optimal solution is not always known for the given problem, and even if we have such information, there is no guarantee that the algorithm will find a solution in a reasonable time lapse. Other common stopping criteria are: maximum number of evaluations performed, maximum real time elapsed, stagnation of fitness values and population diversity. For the thoeretical analysis of evolutionary algorithms we can allow the algorithm run forever, since we analyse the expected time to find the optimum.

$(1 + 1)$ EA Now we will define one of the most simple evolutionary algorithms using the scheme explained before. This algorithm is used for the optimisation of pseudo-Boolean functions $f : \{0, 1\}^n \to \mathbb{R}$, therefore the *representation* of its individuals is also pseudo-Boolean. As most evolutionary algorithms it is *initialised* by sampling random solutions $x \in \{0, 1\}^n$ from the search space with n dimensions. The algorithm consists of a parent population of size $\mu = 1$, and in each iteration an offspring population is created with size $\lambda = 1$. The offspring population is created from the only parent through *standard bit mutation*, that is, flipping each bit independently at random with *mutation probability* $p = r/n$ (also called *mutation rate*) where r is the *mutation parameter.* The next generation is selected from both the offspring and the parent population with *elitist selection*, that is, always selecting the individual with best fitness. In case of ties the offspring is always chosen to favor exploration. The pseudocode of this algorithm is shown in Algorithm 1.

Algorithm 1: The (1 + 1) EA with mutation probability p.

```
1 Initialisation: Choose $x \in \{0,1\}^n$ uniformly at random.
2 Optimisation: for $t \in \{1, 2, \dots\}$ do
3     Mutation: Create $y \in \{0,1\}^n$ by flipping each bit in $x$ independently with
                probability $p$.
4     Selection: if $f(y) \geq f(x)$ then $x \leftarrow y$;
```

As mentioned before this is one of the most simple algorithms and also one of the most studied algorithms in the theoretical field.

$(\mu + \lambda)$ EA From the previous algorithm we can derive a more general version. Using the same process but allowing the population to be any positive integer value $\mu > 0$ and using random parent selection. Following the same process we can allow the offspring population size to be any positive integer value $\lambda > 0$. The resulting algorithm is shown in Algorithm 2. The theoretical study of this algorithm has an increased complexity, compared to the algorithms shown before, hence, it has been less studied. Other simpler versions are more commonly studied, such as the $(\mu + 1)$ EA and the $(1 + \lambda)$ EA where the offspring population size is $\lambda = 1$ and the parent population size is $\mu = 1$, respectively.

(μ, λ) EA There are other common variations in the literature from the $(\mu + \lambda)$ EA. The (μ, λ) EA differs in the type of survival selection used; the algorithm only uses the offspring population in the new generation, therefore $\lambda \geq \mu$ and in general it requires $\lambda \gg \mu$.

$(\mu + \lambda)$ GA Up until now all the evolutionary algorithms discussed only use mutation as a variation operator. The other variation from the algorithm is the use of recombination and mutation. To emphasise this they are called *Genetic Algorithms* (GAs). One of the most common crossover operators is *uniform crossover* where the offspring is formed selecting each bit independently at random from either parent with probability $1/2$. Similarly to the

Algorithm 2: The $(\mu + \lambda)$ EA with population size μ, offspring population size λ and mutation probability p.

```
1  Initialisation: Initialise $P_t$ with $\mu$ individuals $x \in \{0,1\}^n$ uniformly at random.
2  Optimisation: for $t \in \{1, 2, \dots\}$ do
3      $P'_t := \emptyset$
4      Mutation: for each $i \in \{1, \dots, \lambda\}$ do
5          Choose $x \in P_t$ uniformly at random.
6          Create $y \in \{0,1\}^n$ by copying $x$ and, independently for each bit, flip this bit
           with probability $p$.
7          $P'_t := P'_t \cup \{y\}$
8      Selection: Let $P_{t+1}$ contain $\mu$ individuals from $P_t \cup P'_t$ with maximal $f$-value
```

$(\mu + \lambda)$ EA there are simpler common versions such as the $(\mu + 1)$ GA. In general GAs need a population of at least two ($\mu \geq 2$) in order to be able to use recombination.

$(1 + (\lambda, \lambda))$ GA An exemption to the rule above is a genetic algorithm recently proposed by Doerr, Doerr, and Ebel [60], that only uses a parent population of size $\mu = 1$. Like other evolutionary algorithms it is initialised uniformly at random. Later it uses two phases per iteration. The first phase mutates the parent λ times using a mutation operator called mut_ℓ. The mutation operator first chooses ℓ different positions in $[n]$ uniformly at random and then it flips the values in those bits in the original bit string to create the mutated bit string. The variable ℓ is sampled from a binomial distribution $\mathcal{B}(n, p)$, where p denotes the mutation probability. In this algorithm ℓ is only sampled once per generation, making all offspring from this phase have the same distance to the parent. The second phase uses a biased uniform crossover λ times between the parent and the fittest mutated offspring. The biased uniform crossover operator called cross_c selects each bit independently at random from the fittest mutated offspring with probability c and from the parent with probability $1 - c$. In the end it performs an elitist selection only considering the parent and the λ offspring from the crossover phase. The pseudocode for this algorithm is shown in Algorithm 3.

Algorithm 3: The $(1 + (\lambda, \lambda))$ GA with offspring population size λ, mutation probability p, and crossover probability c.

```
1  Initialisation: Sample $x \in 0, 1^n$ uniformly at random and query $f(x)$;
2  Optimisation: for $t = 1, 2, \dots$ do
3      Sample $\ell$ from $\mathcal{B}(n, p)$;
4      Mutation phase: for $i = 1, \dots, \lambda$ do
5          Sample $x^{(i)} \leftarrow \text{mut}_\ell(x)$ and query $f(x^{(i)})$;
6      Choose $x' \in \{x^{(1)}, \dots, x^{(\lambda)}\}$ with $f(x') = \max\{f(x^{(1)}), \dots, f(x^{(\lambda)})\}$ u.a.r.;
7      Crossover phase: for $i = 1, \dots, \lambda$ do
8          Sample $y^{(i)} \leftarrow \text{cross}_c(x, x')$ and query $f(y^{(i)})$;
9      If $\{x', y^{(1)}, \dots, y^{(\lambda)}\} \backslash \{x\} \neq \emptyset$, choose $y \in \{x', y^{(1)}, \dots, y^{(\lambda)}\} \backslash \{x\}$ with
       $f(y) = \max\{f(x'), f(y^{(1)}), \dots, f(y^{(\lambda)})\}$ u.a.r.;
10     otherwise, set $y := x$;
11     Selection step: if $f(y) \geq f(x)$ then $x \leftarrow y$;
```

RLS$_k$ Random Local Search (RLS) is a RSH similar to the $(1 + 1)$ EA, therefore in this thesis we consider RLS as a part of evolutionary computation. RLS is commonly initialised with a solution $x \in \{0,1\}^n$ sampled uniformly at random from the solution space. It creates a new solution y by flipping a single bit chosen uniformly at random. The new solution replaces the current solution if $f(y) \geq f(x)$.

Even though RLS only flips 1 bit per iteration, it has been shown that, for the minimum spanning tree problem, iterations with 2-bit flips are necessary [144]. In other studies even more bit flips are considered, therefore here we use a generalised version RLS$_k$ with k-bit flips shown in Algorithm 4. The RLS$_k$ algorithm has also been called $(1+1)$ EA [51], but here we decided to use the name of RLS$_k$ to emphasise that it only creates new solutions in a bounded distance from the current solution.

Algorithm 4: The RLS$_k$ with mutation strength k.

```
1 Initialisation: Choose $x \in \{0,1\}^n$ uniformly at random.
2 optimisation: for $t \in \{1,2,\dots\}$ do
3     Mutation: Create $y \in \{0,1\}^n$ by flipping $k$ distinct bits in $x$ chosen uniformly
               at random.
4     Selection: if $f(y) \geq f(x)$ then $x \leftarrow y$;
```

2.2 Computational Complexity of EAs

When analysing the computational complexity of an algorithm, we aim to predict the computational effort that the algorithm requires to solve an optimisation problem. In the case of evolutionary algorithms the number of fitness function evaluations T (or T^{eval}) required by the algorithm to query for the first time an optimal solution is commonly used as a proxy for the computational effort to solve an optimisation problem [107]. Here we denote T as *optimisation time*, *sequential optimisation time*, *running time* or *runtime*. In addition, we often accompany the optimisation time with the number of generations T^{gen} also denoted as *parallel optimisation time*. The former reflects the total computational effort, whereas the latter is more relevant when the execution and the generation of offspring can be parallelised efficiently.

Given that evolutionary algorithms are randomised algorithms, the optimisation time of an algorithm is not always the same, even when using the same input. Hence, the field of evolutionary computation uses the expectation of the random number of fitness function evaluations, i. e. $\mathrm{E}(T)$ [107]. Sometimes the expected optimisation time can be deceiving, because the algorithm can have a constant probability of deviating from $\mathrm{E}(T)$. For this reason, often alongside the expected optimisation time the *success probability* ($\Pr(T \leq t)$) is given, i. e. the probability that given t number of fitness function evaluations the algorithm have found the optimum.

Similar to the analyses of deterministic algorithms, in the study of evolutionary algorithms, bounds on the (expected) optimisation time are usually given on their growth rate with the use of asymptotic notation, also known as Landau notation or "Big-O" notation [107]. This notation is commonly used to classify algorithms according to how their runtime grow as the problem size n grows, typically omitting constant factors and lower order terms. Additionally it can be used to show how the probability of an event or the runtime of an algorithm grows according to some characteristic of the problem or parameter setting used in the algorithm.

Definition 2.2.1 (Asymptotic notation)**.** For any two functions $f, g : \mathbb{N}_0 \to \mathbb{R}$. We write:

- $f(n) = O(g(n))$ if and only if there exist constants $c > 0$ and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.
- $f(n) = \Omega(g(n))$ if and only if $g(n) = O(f(n))$.

- $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
- $f(n) = o(g(n))$ if and only if $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$.
- $f(n) = \omega(g(n))$ if and only if $g(n) = o(f(n))$.

In the following we give a list of classes of functions that we will encounter in this work when analysing the running time of an algorithm.

Definition 2.2.2 (Function classes)**.** Let $c > 0$ be a constant and $f : \mathbb{N}_0 \to \mathbb{R}$. Then f is:
- logarithmic if $f = O(\log n)$
- polylogarithmic (polylog (n)) if $f = O((\log n)^c)$
- polynomial (poly (n)) if $f = O(n^c)$
- superpolynomial if $f = \omega(n^c)$
- exponential if $f = 2^{\Omega(n^c)}$

When dealing with probability of events we are particularly interested in typical events, that is, events that have a probability that tends to 1 as n tends to infinity. We say that an event A happens *with high probability* (w. h. p.) if $1-\Pr(A) = O(n^{-c})$ and *with overwhelming probability* (w. o. p.) if $1 - \Pr(A) = 2^{-\Omega(n^c)}$ for some constant $c > 0$.

2.3 Benchmark Functions

Throughout the history of the theoretical analysis of evolutionary algorithms several test functions have been proposed in order to compare the suitability of algorithms for different applications. These benchmark functions capture abstract ideas or characteristics from real-world applications inside their *fitness landscape* and are designed to aid the theoretical analysis of such characteristics. Most of these characteristics are described using the concept of distance between search points. A common distance metric in pseudo-Boolean functions is the *Hamming distance*, which corresponds to the number of bits that are different between two bit strings.

Definition 2.3.1 (Hamming distance)**.** For two bit strings $x, y \in \{0,1\}^n$ with bit values $x = x_1, \dots, x_n$ and $y = y_1, \dots, y_n$, then the Hamming distance between x and y is

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

For the purpose of this thesis there are two main function classes: *unimodal* and *multimodal* functions. A function f is considered a unimodal function if all non-optimal search points x have at least one search point y with Hamming distance $\mathrm{H}(x,y) = 1$ (Hamming neighbors) where $f(y) > f(x)$ and considered multimodal otherwise [70]. In other words, all local optima in unimodal functions are also global optima. We are more interested in multimodal benchmark functions because they capture a larger variety of features than unimodal functions.

In the context of evolutionary computation, the features of a fitness function (*fitness landscape*) can be described using a metaphor of natural landscapes. In the following we informally describe common features in a fitness landscape using this metaphor:
- *Peak* - Search point where all of its Hamming neighbors have lower fitness value, that is, a local optimum.
- *Basin of attraction* - Region of the search space where the fitness points towards a peak.
- *Hill* - Search region comprised by a peak and its basin of attraction, that is, Hamming neighbors with similar fitness values that guide towards the local optimum.

- *Valley* - Search region surrounded by points with higher fitness values.
- *Plateau* - Search region where all of its search points have the same fitness value.
- *Ridge* - Path of search points of high fitness sorrounded by search points of lower fitness.

We will now present the benchmark functions used in this thesis for theoretical analyses. If known we give as reference the asymptotic runtime for the $(1+1)$ EA with mutation probability $p = 1/n$ and the optimal static mutation probability.

2.3.1 Unimodal Functions

We first present unimodal functions because they are often used as a base to construct the more complicated multimodal functions. In the following and in the remaining of this thesis, we denote the number of 1-bits and 0-bits in a bit string x as $|x|_1$ and $|x|_0$ respectively.

Definition 2.3.2 (OneMax)**.** Let $x \in \{0,1\}^n$ for all $n \in \mathbb{N}$, then OneMax: $\{0,1\}^n \to \mathbb{N}$ is

$$\textsc{OneMax}(x) := |x|_1 = \sum_{i=1}^{n} x_i.$$

The optimum of the function is a bit string with only 1-bits (1^n), and the fitness function counts the number of ones in the bit string.

It was designed as a simple benchmark function, in particular is known as the easiest function with a unique optimum for the $(1+1)$ EA [58]. It has been studied profoundly and traditionally it is one of the first benchmark problems studied on any evolutionary algorithm. For the $(1+1)$ EA the optimal mutation probability is $p = 1/n$ and the expected runtime is $en \ln n - \Theta(n)$ [57].

Definition 2.3.3 (ZeroMax)**.** Let $x \in \{0,1\}^n$ for all $n \in \mathbb{N}$, then ZeroMax: $\{0,1\}^n \to \mathbb{N}$ is

$$\textsc{ZeroMax}(x) := |x|_0 = \sum_{i=1}^{n} (1 - x_i).$$

This fitness function counts the number of zeroes in the bit string. Hence, its global optimum is a bit string with only zeroes.

Most evolutionary algorithms use variation and selection operators that are unbiased towards bit-values their behaviour and expected runtime is the same as in OneMax, this includes the $(1+1)$ EA.

Definition 2.3.4 (LeadingOnes)**.** Let $x \in \{0,1\}^n$ for all $n \in \mathbb{N}$, then LeadingOnes: $\{0,1\}^n \to \mathbb{N}$ is

$$\textsc{LeadingOnes}(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j.$$

The LeadingOnes function has the same optimum as OneMax but the fitness function is different. To calculate the fitness of an individual, the number of consecutive 1-bits are counted starting from the left and we stop whenever a 0-bit is found.

For the $(1+1)$ EA the expected runtime with mutation probability $p = 1/n$ is approximately $0.85914n^2$ [21]. The optimal mutation probability is $p \approx 1.5936/n$ and with this parameter choice the expected runtime is approximately $0.77201n^2$ [21].

Definition 2.3.5 (Ridge)**.** Let $x \in \{0,1\}^n$ for all $n \in \mathbb{N}$, then Ridge: $\{0,1\}^n \to \mathbb{N}$ is

$$\textsc{Ridge}(x) := \begin{cases} n + |x|_1 & \text{if } x = 1^i 0^{n-i}, i \in \{0, 1, \ldots, n\}, \\ |x|_0 & \text{otherwise.} \end{cases}$$

The function directs the search towards the beginning of a ridge that starts at 0^n. Once a search point on the ridge is found the search is directed along the ridge towards the global optimum 1^n.

The expected optimisation time of the $(1+1)$ EA with $p = 1/n$ on RIDGE is $\Theta(n^2)$ [107].

Definition 2.3.6 (ONEMAXBLOCKS). Let $x \in \{0,1\}^n$ for all $n \in \mathbb{N}$, then ONEMAXBLOCKS: $\{0,1\}^n \to \mathbb{N}$ is

$$\text{ONEMAXBLOCKS}(x) := \sum_{j=1}^{\lfloor n/k \rfloor} \left(\prod_{i=1}^{(j-1)k} x_i \right) \cdot \sum_{i=(j-1)k+1}^{jk} x_i.$$

This function is comprised of *blocks* of k bits. A block is *complete* if it only contains 1-bits and *incomplete* otherwise. The function returns the number of 1-bits in the longest prefix of completed blocks plus the number of 1-bits in the first incomplete block. Evolutionary algorithms typically optimise this function by optimising each ONEMAX-like block of size k from left to right until the global optimum 1^n is reached.

This is a new benchmark function proposed in this thesis, therefore there are no previous runtime analyses for the $(1+1)$ EA on ONEMAXBLOCKS.

2.3.2 Multimodal Functions

Multimodal functions tend to be more complex and difficult to optimise. We start by defining one of the most simple multimodal functions.

Definition 2.3.7 (TWOMAX). Let $x \in \{0,1\}^n$ for all $n \in \mathbb{N}$, then TWOMAX: $\{0,1\}^n \to \mathbb{N}$ is

$$\text{TWOMAX}(x) := \max\left\{|x|_1, |x|_0\right\}.$$

The TWOMAX function can be considered as a bimodal version of ONEMAX. The fitness landscape is comprised of two symmetrical slopes resembling ONEMAX and ZEROMAX that guide the search towards the global optima 1^n and 0^n respectively.

This function is often studied in the context of diversity of populations, hence analyses tend to calculate the expected time for the algorithm to have found both optima. In this context the $(1+1)$ EA optimises TWOMAX in $\Omega(n^n)$ expected function evaluations [84]. In our context we are only interested in the expected time for an algorithm to find either one of the optima, therefore it is not hard to see that the $(1+1)$ EA would typically behave as on either ONEMAX or ZEROMAX depending on the initialisation.

Definition 2.3.8 (TRAP). Let $x \in \{0,1\}^n$ for all $n \in \mathbb{N}$, then TRAP: $\{0,1\}^n \to \mathbb{N}$ is

$$\text{TRAP}(x) := \begin{cases} |x|_1 & \text{if } |x|_1 \neq 0 \\ n+1 & \text{otherwise} \end{cases}$$

The TRAP function has a similar behaviour as ONEMAX, it counts the number of 1-bits in the bit string, but it contains the global optimum in the maximum Hamming distance from the ONEMAX optimum, that is, in the 0^n bit string.

This function has a deceiving fitness landscape that guides hill-climbing algorithms towards its local optimum, making it increasingly harder for them to find the global optimum. It has been proven to be an example of the worst running time possible for the $(1+1)$ EA with an expected $\Theta(n^n)$ evaluations needed in order to find the optimum [76].

Definition 2.3.9 (JUMP$_k$). Let $x \in \{0,1\}^n$ for all $n \in \mathbb{N}$, then JUMP$_k$: $\{0,1\}^n \to \mathbb{N}$ is

$$\text{JUMP}_k(x) := \begin{cases} n - |x|_1 & \text{if } n-k < |x|_1 < n \\ k + |x|_1 & \text{otherwise} \end{cases}$$

The JUMP_k function has a construction similar to the TRAP function, but allows to adjust the difficulty with the parameter k. The function increases its fitness value with the first $n-k$ 1-bits in the bit string, leading to a set of local optima. After reaching a local optimum, increasing the number of 1-bits leads to a valley that contains the lowest fitness values in all the function. The fitness value in the valley decreases when adding 1-bits, with the minimum fitness neighboring the global optimum. The optimum is the bit string 1^n.

The expected optimisation time of the $(1+1)$ EA with $p = 1/n$ is $\Theta(n^k + n \log n)$ [76], while for the optimal mutation probability of $p = k/n$ the expected optimisation time is $O\left(\frac{n^k}{k^k}\left(\frac{n}{n-k}\right)^{n-k}\right)$ [62].

The JUMP_k function has been of great importance in the analysis of evolutionary algorithms and has been extensively used to study other random search heuristics such as estimation of distribution algorithms [49, 97], memetic algorithms [179], hyper-heuristics [137], artificial immune systems [32, 34], ant colony optimisers [19] and voting algorithms [165].

Definition 2.3.10 (CLIFF_d)**.** Let $x \in \{0,1\}^n$ for all $n \in \mathbb{N}$, then $\text{CLIFF}_d \colon \{0,1\}^n \to \mathbb{N}$ is

$$\text{CLIFF}_d(x) := \begin{cases} |x|_1 & \text{if } |x|_1 \leq n-d \\ |x|_1 - d + 1/2 & \text{otherwise} \end{cases}$$

The CLIFF_d is comprised of two slopes similar to ONEMAX. Following Lissovoi et al. [137], we say that all search points x with $|x|_1 \leq n-d$ form the *first slope* and all other search points form the *second slope*. The first slope is a slope that guides the search towards a local optimum with fitness $n-d$, which is the *cliff* of the function. The second slope has size d and starts when $|x|_1 = n-d+1$, with a fitness $n-2d+3/2$. The slope increases in a ONEMAX fashion ending at the global optimum with fitness $n-d+1/2$.

The CLIFF_d function was first proposed by Jägersküpper and Storch [106] with $d = n/3$. We write CLIFF to refer to CLIFF_d with the default value $d = n/3$. CLIFF was later generalised by Paixão, Pérez Heredia, Sudholt, and Trubenová [153]. Our definition, taken from [153], differs from [106] in that the cliff is located at $n-d$ 1-bits instead of $n-d-1$. We choose the definition from [153] because it resembles the definition of the JUMP_k class functions.

This function is similar to the JUMP_k function, but once you jump down the *cliff*, the function has a slope guiding evolutionary algorithms towards the global optimum, which favours evolutionary algorithms with non-elitist selection. The $(1+1)$ EA with $p = 1/n$, have similar behaviour as on the JUMP_k function, in fact, the expected optimisation time is also $\Theta(n^d)$ for any $2 \leq d \leq n/2$ [153].

The CLIFF function was used as a benchmark in several works, including studies of the Strong Selection Weak Mutation (SSWM) model of evolution [153], artificial immune systems [36] and hyper-heuristics [137].

2.4 Runtime Analysis of Evolutionary Algorithms

The theoretical analysis of evolutionary algorithms is a relatively young research area that has developed a range of powerful methods. These methods have helped researchers find more efficient parameter choices and design different variation and selection operators. In this section we describe some of the commonly used methods that will be employed in our upcoming analyses. We start with well-known identities, taken from [16, 48, 119] that are helpful in the analysis of evolutionary algorithms. Later, we continue with tools from probability theory based on the collection made by Doerr [48], and finish with other mathematical techniques used in the runtime analysis of evolutionary algorithms that have appeared in the book chapters [125, 131, 150].

2.4.1 Useful Estimates

Common expressions that arise in the study of evolutionary algorithms, can be difficult to work with, making convenient to estimate their value with more simple expressions. The first group of identities shown here use the exponential function to bound other more complex expressions.

Lemma 2.4.1.

(a) For all $x \in \mathbb{R}$, $1 + x \le e^x$.

(b) For all $r \ge 1$ and $0 \le x \le r$, $\left(1 - \frac{x}{r}\right)^r \le e^{-x} \le \left(1 - \frac{x}{r}\right)^{r-x}$.

(c) For all $|x| \le r$ and $r > 1$, $e^x\left(1 - \frac{x^2}{r}\right) \le \left(1 + \frac{x}{r}\right)^r$.

(d) For all $0 \le x \le 1$ and $r \in \mathbb{N}$, $\frac{xr}{1+xr} \le 1 - (1 - x)^r$.

Similar to the estimates of the exponential functions, the Bernoulli's inequality, and its derivatives are extremely useful.

Lemma 2.4.2.

(a) For all $x \ge -1$ and $r \in \mathbb{R} \setminus (0, 1)$, $1 + rx \le (1 + x)^r$.

(b) For all $x \in [-1, 0]$ and $r \in \mathbb{N}$, $(1 + x)^r \le \frac{1}{1-rx}$.

Lemma 2.4.2 (a) is well known as Bernoulli's inequality. We give a short proof for Lemma 2.4.2 (b) for the sake of completeness. A simpler proof without the use of the Bernoulli's inequality is shown by Rowe and Sudholt [166, Lemma 8].

Proof of Lemma 2.4.2 (b). For the case where $x = -1$ and $x = 0$ it is trivial to show that $(1 + x)^r \le \frac{1}{1-rx}$ holds. Let $\phi(x) = \frac{-x}{1+x}$, where $x \ne -1$, then $\phi(\phi(x)) = x$. Moreover, for all $x \in (0, \infty)$ we have $\phi(x) \in (-1, 0)$ and for all $x \in (-1, 0)$ we have $\phi(x) \in (0, \infty)$. Let $x = \phi(y)$, then $y = \phi(x)$ and

$$(1 + x)^r = \left(1 - \frac{y}{1 + y}\right)^r = \left(\frac{1}{1 + y}\right)^r = \frac{1}{(1 + y)^r}.$$

Using Lemma 2.4.2 (a) with $y \in (0, \infty)$ we get,

$$\frac{1}{(1 + y)^r} \le \frac{1}{1 + ry} = \frac{1}{1 - \frac{rx}{1+x}} \le \frac{1}{1 - rx}.$$

Hence $(1 + x)^r \le \frac{1}{1-rx}$ for all $x \in (-1, 0)$. □

We often encounter the binomial coefficients while studying evolutionary algorithms.

Definition 2.4.3 (*Binomial Coefficients*)**.** For $0 \le k \le n$, the binomial coefficients are defined by

$$\binom{n}{k} := \frac{n!}{k!(n - k)!}$$

It is sometimes useful to estimate the binomial coefficient. Some useful inequalities are shown in Lemma 2.4.4.

Lemma 2.4.4. *For all $n \in \mathbb{N}$ and $0 \le k \le n$,*

$$\binom{n}{k} \le 2^n \tag{2.1}$$

$$\frac{n^k}{k^k} \le \binom{n}{k} \le \frac{n^k}{k!} \le \left(\frac{en}{k}\right)^k \tag{2.2}$$

At last, frequently in the analysis of evolutionary algorithms, we come across the *harmonic number* H_n.

Definition 2.4.5 (Harmonic number H_n)**.** For all $n \in \mathbb{N}$ the n-th harmonic number is defined by $H_n := \sum_{k=1}^{n} \frac{1}{k}$.

We can approximate H_n as,

$$\ln n \leq H_n \leq 1 + \ln n. \tag{2.3}$$

2.4.2 Tools from Probability Theory

Evolutionary algorithms are a type of randomised algorithms. Therefore, the theory of evolutionary computation shares many tools with classical randomised algorithms and probability theory. Here we will introduce some of these tools.

The expectation is a key characteristic for any random variable. An important property of the expectation is its linearity.

Lemma 2.4.6 (Linearity of expectation)**.** *Let $X_1, \ldots, X_n$ be arbitrary random variables and $a_1, \ldots, a_n \in \mathbb{R}$. Then*

$$\mathrm{E}\left(\sum_{i=1}^{n} a_i X_i\right) = \sum_{i=1}^{n} a_i \mathrm{E}(X_i)$$

This property can be used to simplify the calculation of complicated random variables. For example, we can use this fact to calculate the expected value of a binomial random variable, with probability $\Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$. We just need to divide the variable into n binary random variables with probability p and calculate $\mathrm{E}(X) = \mathrm{E}(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} \mathrm{E}(X_i) = np$.

Lemma 2.4.7 (Expectation of binomial random variables)**.** *Let X be a binomial random variable describing n trials with success probability p. Then* $\mathrm{E}(X) = pn$.

An example of binomial random variable encountered in evolutionary algorithms is during the initialisation. When using random sampling, the number of ones in the bit string of size n is a binomial random variable with $p = 1/2$, therefore, in expectation the bit string contains $\frac{n}{2}$ ones and $\frac{n}{2}$ zeros.

With the same arguments as before we can obtain the expectation of geometric random variables.

Lemma 2.4.8 (Expectation of geometric random variables)**.** *Let X be a geometric random variable with probability p, that is a probability distribution* $\Pr(X = k) = (1-p)^{k-1} p$. *Then* $\mathrm{E}(X) = 1/p$.

This expectation is often referred as the *waiting time argument.* This elementary fact, is helpful when studying the waiting time for a variation operator to find an improvement. For example, in the study of the (1 + 1) EA with mutation probability $1/n$ on $\textsc{Jump}_k$, if we want to know how many evaluations are needed to jump from the local optimum to the global optimum, we first calculate the probability of flipping the correct bits and not flipping the incorrect ones, which is bounded by $n^{-k}(1 - 1/n)^{n-k} \leq n^{-k}$. Given the success probability $p \leq n^{-k}$, using the waiting time argument, we obtain that the expected number of evaluations to jump is at least n^k.

It is often the case that we need to compute the expected value of the sum of a random number of identically distributed random variables. Wald's equation (Lemma 2.4.9) is used for this purpose.

Lemma 2.4.9 (Wald's equation [177])**.** *Let the $X_1, \ldots, X_N$ be a sequence of real-valued, independent and identically distributed random variables with $N \in \mathbb{N}$ being a random variable that is independent of the sequence. If X_N and N have finite expectations, then $\mathrm{E}(X_1 + \cdots + X_N) = \mathrm{E}(N)\mathrm{E}(X_1)$.*

When computing probabilities, some of the most common tools are the *law of total probability* and the *union bound*. We show them in the following lemmas.

Lemma 2.4.10 (Law of total probability)**.** *For an event A and a partition of the probability space $B_1, \ldots, B_k$.*

$$\Pr(A) = \sum_{i=1}^{k} \Pr(A \mid B_i) \Pr(B_i).$$

Lemma 2.4.11 (Union bound)**.** *Let $A_1, \ldots, A_n$ be arbitrary events in some probability space. Then,*

$$\Pr\left(\bigcup_{i=1}^{n} A_i\right) \leq \sum_{i=1}^{n} \Pr(A_i).$$

We sometimes tacitly use the following argument. If in an iterative process there is an event that happens independently in each step with probability at most p, the probability that the event happens during a phase of T steps, T a random variable with $\mathrm{E}(T) < \infty$, is at most

$$\sum_t \Pr(T = t) \cdot tp = p \cdot \mathrm{E}(T). \tag{2.4}$$

Considering that the runtime T is a random variable, it is also useful to understand the deviations from the expected value, and their probabilities. For this purpose there are *tail bounds*, that given an expected value, can help us compute the probability that the random variable exceeds the expectation by a certain amount.

Markov's inequality is a general tail bound, for all non-negative random variables.

Lemma 2.4.12 (Markov's inequality)**.** *Let X be a non-negative random variable. Then for all $a > 0$,*

$$\Pr(X \geq a) \leq \frac{\mathrm{E}(X)}{a}$$

This inequality is quite useful when there is small amount of information. For example, using the previously calculated expected number of ones in a bit string sampled at random, we can also calculate the probability that the number of ones exceeds $2n/3$.

$$\Pr(X \geq 2n/3) \leq \frac{n/2}{2n/3} = \frac{3}{4}$$

Although useful, Markov's inequality gives a "weak" bound on the probability that does not change when n increases. "Stronger" bounds can be achieved using Chernoff Bounds. Chernoff Bounds is a family of bounds concerning the sum of independent random variables. Here we only show two versions of the upper bounds. The reader can find a more thorough review on Chernoff Bounds in the book chapter by Doerr [48].

Lemma 2.4.13 (Chernoff Bounds)**.** *Let $X_i, \ldots, X_n$ be independent random variables taking values in $[0, 1]$. Let $X = \sum_{i=1}^{n} X_i$. Let $\delta \geq 0$. Then,*

$$\Pr(X \geq (1+\delta)\mathrm{E}(X)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mathrm{E}(X)} = \exp(-((1+\delta)\ln(1+\delta) - \delta)\mathrm{E}(X)) \tag{2.5}$$

$$\leq \exp\left(-\frac{\delta^2 \mathrm{E}(X)}{2 + \frac{2}{3}\delta}\right). \tag{2.6}$$

Following the same example as before, with Chernoff bounds we can bound the probability that the number of ones exceeds $2n/3$ by

$$\Pr(X \geq 2n/3) = \Pr(X \geq (1+1/3)n/2) \leq \exp\left(-\frac{n/18}{20/9}\right) = e^{-n/40}$$

As we have seen Chernoff bounds give a bound on the probability that tends to 0 exponentially fast when n grows.

2.4.3 Standard Tools and Methods

In this section we illustrate the most common methods used in the runtime analysis of EAs.

Artificial Fitness Levels

The fitness-level method [178] is a general analysis method for upper bounds on the expected optimisation time for evolutionary algorithms where the fitness of the best individual cannot decrease. We describe it in the context of the (1+1) EA. The method uses so-called f-based partitions, which is a partition of the search space $\{0,1\}^n$ into sets $A_1, \dots, A_{m+1}$ for some $m \in \mathbb{N}$ where all search points A_i are strictly worse than all search points in A_{i+1}, and A_{m+1} exclusively contains all global optima. Each of these sets needs to be left at most once. Suppose that we know that for every search point $x \in A_i$, the probability of creating a search point in a higher fitness-level set is at least s_i, for some expression $s_i > 0$. Then the expected time for leaving set A_i is at most $1/s_i$. Since every fitness level has to be left at most once before reaching A_{m+1} and once A_{m+1} is reached a global optima is found due to its definition, the expected optimisation time of the $(1+1)$ EA is at most $\sum_{i=1}^{m} 1/s_i$.

The fitness-level method is a simple and versatile method in its own right, and it allows researchers to translate bounds on the runtime of the simple $(1+1)$ EA to other elitist algorithms. This has been achieved for parallel evolutionary algorithms [121], ant colony optimisation [92, 146], particle swarm optimisation [175], and artificial immune systems [32]. It further gives rise to tail bounds [183] and lower bounds [53, 171], and the principles extend to non-elitist algorithms as well [33, 54].

Fitness levels may contain search points of different fitness. In the special case where each set A_i contains search points with only one fitness value the partition is called a *canonical* partition.

Typical Run Investigations

The typical run investigations [151] is used under the assumption that we can predict more easily the typical "global" behaviour of a process than a particular "local" behaviour of such process. An example of this is the random initialisation of a bit string of size n. When n is large, we can easily predict the proportion of bits that are one or zero with probability bounds as we did in Section 2.4.2, but it is hard to predict if a particular bit is 1 or 0.

Taking this idea, we can divide the optimisation process in k sufficiently long "phases" with each phase having a specified goal. Then we can more easily compute the expected optimisation time of phase i and bound the probability of an anomalous event happening during phase i by $1-p_i$. We can add the time the algorithm takes to optimise each phase, and this time will hold with a probability $1-\sum_{i=1}^{k} p_i$. Generally, Chernoff bounds are used to obtain the failure probabilities p_i, but here we often use Equation (2.4) for that purpose.

If the goals of each phase are properly set, this phase division is helpful to separate the analysis in smaller sub-analysis that can give stronger results and clearer proofs; owing to this, typical run investigations are widely used in the analysis of evolutionary algorithms, especially on problems that have clear separations of behaviours such as the TRAP, JUMP$_k$ and CLIFF$_d$ functions.

Accounting Method

The accounting method [29, Chapter 17] is a method of amortized analysis, based on accounting. It is used to calculate the total cost of a sequence of operations. It is especially useful when *expensive* operations exist in the sequence, but the cost of such operations decrease when other cheaper operations also occur in the sequence. The method uses operations with certain costs assigned, and all operations need to pay for their cost. Some operations can pay extra cost to a fictional bank account, that can be later used by other (expensive) operations. Provided that no fictional account gets overdrawn, the total amount of money paid bounds the total cost of all operations.

The accounting method has not been widely used in the analysis of evolutionary algorithms, but in combination with the fitness level method it has a lot of potential for the study of parameter control mechanisms. In fact, Lässig and Sudholt [120] used it for the first time in the field to analyse the optimisation time of the $(1+2\lambda, \lambda/2)$ EA which adjusts the offspring population size, giving an upper bound on any function with m fitness values.

Family trees

The use of family trees in the analysis of evolutionary algorithms was first introduced by Witt [180]. A family tree is a directed acyclic graph where the nodes are individuals created by an evolutionary algorithm and the edges represent a direct parent-child connection between individuals. During initialisation, for each initial individual a family tree is created and in each generation the new offspring are added as leaves. Some of these trees or parts of them might become extinct if all nodes of the tree are not in the population.

Family trees can be used to prove lower bounds by showing that after a certain number of generations the tree depth is not too large and the leaves of the tree are not too different from the root. Then, if all global optima are far from the root, the probability that the leaves are located in a global optimum is small.

A different version of this approach was given by Lehre and Yao [129] called non-selective family trees where each generation all the nodes create offspring. This result in a faster growing family tree that grows independently from the fitness function, but the real family tree is a sub-tree of the non-selective family tree. Hence, if an event does not happen in the non-selective family tree it also does not happen in the real family tree.

Drift Analysis

Drift analysis is one of the most useful tools to analyse evolutionary algorithms [131]. It uses the intuitive idea that for a given stochastic process, if we can bound the expected progress towards a target in a single step and the current state of the process is at a given distance according to a distance metric, then we can derive a bound on the expected time to reach the target by analysing the expected decrease in distance in each step. Hence, a general approach for the use of drift analysis is to identify a potential function that adequately captures the progress of the algorithm and the distance from a desired target state (e. g. having found a global optimum). Then we analyse the expected changes in the potential function at every step of the optimisation (drift of the potential) and finally translate this knowledge about the drift into information about the runtime of the algorithm.

Several powerful drift theorems have been developed throughout the years that help with the last step of the above approach, requiring as little information as possible about the potential and its drift. Here we state the drift theorems used in our work.

Theorem 2.4.14 (Additive Drift [98]). *Let $(X_t)_{t\geq 0}$ be a sequence of non-negative random variables over a finite state space $S \subseteq \mathbb{R}^+$. Let T be the random variable that denotes the earliest point in time $t \geq 0$ such that $X_t = 0$. If there exists $\delta > 0$ such that, for all $t < T$,*

$$\mathrm{E}(X_t - X_{t+1} \mid X_t) \geq \delta,$$

then

$$\mathrm{E}(T \mid X_0) \leq \frac{X_0}{\delta}.$$

The following theorem is an extension to Theorem 2.4.14 for an unbounded state space.

Theorem 2.4.15 (Additive Drift, unbounded [117]). *Let $\alpha \leq 0$, let $(X_t)_{t \in \mathbb{N}}$ be random variables over $\mathbb{R}$, and let $T = \inf\{t \mid X_t \leq 0\}$. Furthermore, suppose that,*

(a) for all $t \leq T$, it holds that $X_t \geq \alpha$, and that

(b) there is some value $\delta > 0$ such that, for all $t < T$, it holds that $X_t - \mathrm{E}(X_{t+1} \mid X_0 \dots X_t) \geq \delta$.

Then

$$\mathrm{E}(T \mid X_0) \leq \frac{X_0 - \alpha}{\delta}.$$

The following two theorems both deal with the case that the drift is pointing away from the target, that is, the expected progress is negative in an interval of the state space.

Theorem 2.4.16 (Negative drift theorem [147, 148]). *Let X_t, $t \geq 0$, be real-valued random variables describing a stochastic process over some state space. Suppose there exists an interval $[a, b] \subseteq \mathbb{R}$, two constants $\delta, \varepsilon > 0$ and, possibly depending on $\ell := b - a$, a function $\rho(\ell)$ satisfying $1 \leq \rho(\ell) = o(\ell/\log(\ell))$ such that for all $t \geq 0$ the following two conditions hold:*

1. *$\mathrm{E}(X_{t+1} - X_t \mid X_0, \dots, X_t; a < X_t < b) \geq \varepsilon$.*
2. *$\Pr\left(|X_{t+1} - X_t| \geq j \mid X_0, \dots, X_t; a < X_t\right) \leq \frac{\rho(\ell)}{(1+\delta)^j}$ for $j \in \mathbb{N}_0$.*

Then there exists a constant $c^* > 0$ such that for $T^* := \min\{t \geq 0\colon X_t < a \mid X_0, \dots, X_t; X_0 \geq b\}$ it holds $\Pr\left(T^* \leq 2^{c^* \ell/\rho(\ell)}\right) = 2^{-\Omega(\ell/\rho(\ell))}$.

The following theorem is a variation of Theorem 2.4.16 in which the second condition on large jumps is relaxed.

Theorem 2.4.17 (Negative drift theorem with scaling [149]). *Let X_t, $t > 0$ be real-valued random variables describing a stochastic process over some state space. Suppose there exists an interval $[a, b] \subseteq \mathbb{R}$ and, possibly depending on $\ell := b - a$, a drift bound $\varepsilon := \varepsilon(\ell) > 0$ as well as a scaling factor $\rho := \rho(\ell)$ such that for all $t \geq 0$ the following three conditions hold:*

1. *$\mathrm{E}(X_{t+1} - X_t \mid X_0, \dots, X_t; a < X_t < b) \geq \varepsilon$.*
2. *$\Pr\left(|X_{t+1} - X_t| \geq j\rho \mid X_0, \dots, X_t; a < X_t\right) \leq e^{-j}$ for $j \in \mathbb{N}_0$.*
3. *$1 \leq \rho^2 \leq \varepsilon\ell/(132 \log(\rho/\varepsilon))$.*

Then for the first hitting time $T^* := \min\{t \geq 0\colon X_t < a \mid X_0, \dots, X_t; X_0 \geq b\}$ it holds that $\Pr\left(T^* \leq e^{\varepsilon\ell/(132\rho^2)}\right) = O(e^{-\varepsilon\ell/(132\rho^2)})$.

2.5 Parameter Settings

Evolutionary algorithms come with a range of parameters, such as the size of the parent population, the size of the offspring population or the mutation rate. This means that the behaviour of the algorithms can be modified by adjusting the value of the parameters. It is well known that the optimisation time of an evolutionary algorithm may depend drastically and often unpredictably on their parameters and the problem in hand [51, 140]. Hence, parameter selection is an important and growing field of study.

Early in the history of evolutionary computation, the selection of parameters (parameter tuning) became an important field of study. At first researchers used *manual* parameter tuning, that is, experiment with different parameter values and select the ones that helped the performance of the algorithm. Given the number of possible parameters and their interactions between each other, it is practically impossible to try all the combinations. Because of this, tuning by hand could give suboptimal choices.

Another common practice was (and still is) to choose parameter settings similar to previous parameter settings that have been proved useful for problems similar to the one in hand. One example of this is the mutation probability of $p = 1/n$ for problems of length n which have worked well in a broad range of problems and it was commonly used even before Witt [182] formally proved to be the optimal choice for any linear function.

The problem with this is that even for similar problems the optimal parameter values can be very different. Nowadays there are *automated* parameter tuning tools [79] that help identify reasonable parameter values for the problem in hand.

Even though parameter tuning (manual or automated) can obtain good parameters, it has two main problems. Firstly, there is a lack of rigorous understanding of how and when parameter tuning gives good parameter values for the problem in hand, the only theoretical studies investigating this are made by Hall, Oliveto, and Sudholt [93, 94, 95, 96], secondly, even optimal static parameters may still result in a suboptimal performance in different steps of the optimisation. For example, it can be beneficial to use larger mutation rates at the beginning of the optimisation process to favour exploration, and decrease it later to benefit exploitation.

Throughout the years, theoretical investigations have helped to find optimal static parameters for benchmark problems (see Section 2.3) that have characteristics of real-world problems.

Mutation probability

Early in the study of evolutionary algorithms, empirical studies suggested that the small mutation probability $p = 1/n$ was a good choice and it was often recommended [14]. One of the first theoretical studies to back this choice was made by Garnier, Kallel, and Schoenauer [87], where the authors showed that $p = 1/n$ is asymptotically optimal for OneMax. But, soon after it was shown that this was not always the case: Jansen and Wegener [109], showed that for an artificially constructed function, small values of the mutation probability yield superpolynomial runtime while larger values can find the optimum in polynomal time. Following the result on OneMax, Droste, Jansen, and Wegener [76] proved that all mutation probabilities $p = \Theta(1/n)$ obtain an asymptotically optimal expected optimisation time of $O(n \log n)$ on linear functions.[1]

After a long hiatus, several theoretical results on the importance of the mutation probability where published. The first result to prove an optimal parameter value different from $p = 1/n$ was shown by Böttcher, Doerr, and Neumann [21], where the authors prove that the standard mutation probability of $p = 1/n$ was not optimal for the $(1+1)$ EA on LeadingOnes, and $p \approx 1.59/n$ was the optimal static choice.

Following the results in [52, 76], Witt [182] proved a tight (up to lower order terms) bound of $(1 \pm o(1))\frac{e^r}{r} n \ln n$ for the $(1+1)$ EA using a mutation probability $p = r/n$ (with r constant) on all linear functions. Given that this bound has the factor $\frac{e^r}{r}$ that depends on the mutation parameter r, it can be derived that the mutation probability $p = r/n$ with $r = 1$ is optimal to minimise the specified bound.

This result was later generalised for the $(1+\lambda)$ EA on the OneMax function by Gießen and Witt [89] were they found that the expected parallel optimisation time (i.e. number of generations) for a mutation rate $p = r/n$ with r constant, is equal to

$$(1 \pm o(1)) \left(\frac{e^r}{r} \cdot \frac{n \ln n}{\lambda} + \frac{1}{2} \cdot \frac{n \ln \ln \lambda}{\ln \lambda} \right)$$

[1]Doerr and Goldberg [52] simplified the proof of Droste et al. [76] using Drift Analysis.

From this we can infer that if the value of $\lambda = o(\ln(n)\ln(\ln(n))/\ln(\ln(\ln(n))))$, the first part of the function is predominant and the optimal mutation rate is still $p = 1/n$, and after that threshold any mutation probability $p = r/n$ with a constant value of r does not change the expected runtime, up to lower order terms. More precise results have shown that the optimal mutation probabilities have small variations with changes in λ and n on OneMax [28, 90] and LeadingOnes [73].

It has also been shown that too large mutation probabilities of order $p = \Theta(1/n)$ can be detrimental. Doerr, Jansen, Sudholt, Winzen, and Zarges [56, 59] showed that for some monotone functions large constant mutation rates can drastically affect the performance of the $(1+1)$ EA. In particular the authors present a monotone function where a mutation probability of $p > 16/n$ does not find the optimum within $2^{\Omega(n)}$ iterations. In a similar manner, Lengler and Steger [133] showed that for a monotone function any mutation probability $p > 2.1369/n$ lead to exponential runtimes for the $(1+1)$ EA. Later, Lengler [130] extended this result for the $(1+\lambda)$ EA, the $(\mu+1)$ EA, the $(\mu+1)$ GA and the $(1+(\lambda,\lambda))$ GA.

Although the mutation probability $p = 1/n$ seems to be a good first choice, it is clear that it does not always result in the most efficient optimisation. Doerr, Le, Makhmara, and Nguyen [62], showed that for the $(1+1)$ EA on the Jump$_k$ functions the mutation probability affects the performance greatly. The authors showed that for any mutation probability $p \in [2/n, k/n]$ there is a speedup of at least 2^k evaluations, compared to the standard $p = 1/n$. The authors also showed that the optimal mutation probability is $p = k/n$, and any small deviation ε from the optimal mutation rate, is slower than the optimal choice by a factor at least $\frac{1}{6}\exp(k\varepsilon^2/5)$. Bambury, Bultel, and Doerr [17] showed similar results for a generalised version of the Jump$_k$ functions with the valley of low fitness farther away from the optimum.

Offspring population size

The offspring population size has been less studied than the mutation rate. Most of the theoretical studies that are focused on the behaviour of the offspring population size, have been done on the $(1+\lambda)$ EA. Jansen, De Jong, and Wegener [114] made a broad study of the impact of λ; they found that for simple landscapes setting λ in the range of $1 \leq \lambda \leq \log n$ grant asymptotically equivalent sequential optimisation times (number of evaluations), and specifically for OneMax and LeadingOnes there is no benefit in increasing λ beyond $\lambda = 1$. In contrast with these landscapes the authors also showed that for more complex problems there is a benefit on using large values for λ. In particular, an example function was presented that contain a single narrow steep path to the optimum and several branches guiding to local optima with a more gradual uphill path. This function was called SufSamp and benefitted greatly when $\lambda \gg 1$.

In the study above the focus was made on sequential optimisation time but for parallel optimisation time (number of generations) Jansen et al. [114] showed that the parallel optimisation time is reduced on OneMax and LeadingOnes when increasing λ up to a cut-off value of $\lambda = \Theta(\ln(n)\ln(\ln(n))/\ln(\ln(\ln(n))))$ and $\lambda = \Theta(n)$ respectively. Later Doerr and Künnemann [55] showed that the $(1+\lambda)$ EA finds the optimum in an expected parallel optimisation time of $O(\frac{n \log n}{\lambda} + n)$ on any linear function. A more general result was given by Lehre and Sudholt [127], showing that the λ-parallel (unary) unbiased black-box complexity for any function with a unique optimum is at least $\Omega(\frac{\lambda n}{\max\{1, \log \lambda\}} + n \log n)$.

The parallel optimisation time is important when we can create and evaluate offspring independently in parallel, but parallel evolutionary algorithms can use more complex models such as *island models* that can make the optimisation more efficient. Island models divide the population into subpopulations called *islands* that migrate their best individual(s) at a certain *migration interval* between neighbouring islands. The islands have a *migration topology* that determines what islands are neighbouring. Lässig and Sudholt [121] showed that for certain number of islands λ, different migration topologies can obtain speedups in parallel optimisation time without increasing the sequential optimisation time by more than a constant on unimodal functions and JUMP$_k$ functions. Similar results were shown by Lässig and Sudholt [122] on combinatorial problems. The results from [121, 122] include the $(1 + \lambda)$ EA as a special case of the island models. Sudholt [172] gives a thorough review on parallel evolutionary algorithms.

Another interesting theoretical study was made by Jägersküpper and Storch [106], where the authors study the relation between the parameter λ and the different selection strategies. They show the importance of a correct choice of the selection strategy depending on the offspring population size. And in particular for the $(1, \lambda)$ EA they show that with any $\lambda \leq \ln n/14$, it fails to optimise any function with a unique global optimum when using the mutation probability $p = 1/n$.

Following this study, Rowe and Sudholt [166] extended the results showing that there is a sharp threshold at $\lambda = \log_{\frac{e}{e-1}} n \approx 5 \log_{10} n$ between exponential and polynomial runtimes on ONEMAX. Additionally, they showed that any smaller λ value (by a constant factor) results in exponential time for the $(1, \lambda)$ EA on all functions with one unique optimum. Furthermore, the threshold can shift towards larger values of λ depending on the characteristics of the functions (e. g. in LEADINGONES the threshold is at $\lambda = 2 \log_{\frac{e}{e-1}} n$) or shift towards smaller values if the mutation probability is reduced. The threshold for ONEMAX was refined by Bossek and Sudholt [20] towards $\lambda = \log_{\frac{e}{e-1}} \left(\frac{cn}{\lambda}\right)$ for some constant $c > e^2$.

When considering a parent population μ, Lehre [123] showed that the (μ, λ) EA with $\lambda \leq (1-\varepsilon)e\mu$ for some constant $\varepsilon > 0$ needs an exponential runtime to optimise any function with at most a polynomial number of optima. Follow up studies [33, 47, 124] showed that the (μ, λ) EA with $\lambda = (1 + \varepsilon)e\mu$ for some constant $\varepsilon > 0$ and at least logarithmic in the envisaged runtime can optimise many classic benchmark functions in asymptotically the same time as the $(\mu + \lambda)$ EA. We note that it is often the case that results showing that non-elitist algorithms[2] are efficient need at least a logarithmic offspring population size λ [25, 42, 43, 69].

At last, the optimal parameters for the $(1 + (\lambda, \lambda))$ GA have also been studied. This algorithm has three parameters λ, p and c. Doerr et al. [60] suggested to use $p = \lambda/n$ and $c = 1/\lambda$, reducing to only one parameter. In the same work, they gave an asymptotic bound for the expected optimisation time that is minimised with $\lambda = \Theta(\sqrt{\log n})$. Later Doerr and Doerr [50] improved the aforementioned bound, the new bound suggests that the optimal performance can be achieved with the value $\lambda = \Theta(\sqrt{\log(n) \log\log(n)/\log\log\log(n)})$. The same coupling of parameters to the offspring population size λ has also been shown to be efficient on LEADINGONES [6] and on certain instances of MAX-3SAT [22], but a different parameter selection is preferred for JUMP$_k$ functions [9].

Parent population size

The parent population size μ has mostly been studied separately from the offspring population size in order to understand precisely the effects of this parameter. The first studies were made by Jansen and Wegener [110] and Witt [181], where the authors studied the $(\mu + 1)$ EA with fitness proportional selection for reproduction, and inversely proportional selection for deletion, that is, the better individuals are more likely to stay in the population. They show that for a scaled version of the JUMP$_k$ function and a tailored function this algorithm was asymptotically faster than the $(1 + 1)$ EA. Both of these results where highly dependent on the selection methods, which for the general understanding of the effects of the parent population size is not that informative.

The first to study the parent population size in "isolation" was Witt [180]. The $(\mu + 1)$ EA studied used uniform parent selection and truncation selection for replacement. It was proven that, in expectation, this evolutionary algorithm was slower compared against

[2]with tournament, linear ranking, comma and other selection mechanisms

the $(1+1)$ EA on OneMax, LeadingOnes and Plateau. Nonetheless, the author also showed that large parent population sizes can be beneficial on functions where exploring the search space is important. Gießen and Kötzing [88] showed that large parent population sizes can be useful when encountering noise.

As other parameters, an incorrect selection of the parent population size can sometimes hinder the performance drastically. Lengler and Zou [134] determined that for every choice of mutation probability $p = r/n$ with $r > 0$ being constant, there is a μ_0 such that if $\mu_0 \leq \mu \leq n$ then the $(\mu + 1)$ EA needs a superpolynomial time to optimise some monotone functions. This is caused by a lower selection pressure in some parts of the population allowing them to maintain harmful mutations.

Considering parent and offspring populations in conjunction increases the complexity of the runtime analyses. Chen, He, Sun, Chen, and Yao [27] conducted a study of an elitist $(N + N)$ EA with truncation selection and two tournament selection on unimodal functions. They show that for LeadingOnes and OneMax with $N = O(n/\ln n)$ and $N = O(\ln n/\ln\ln n)$ respectively, both $(N + N)$ EA have asymptotically the same upper bound in expected optimisation time as the $(1+1)$ EA. Antipov and Doerr [4] extended the results on OneMax for all population sizes $\mu \geq 1$ and $\lambda \geq 1$ on the $(\mu + \lambda)$ EA. The authors show that for a parent population size at most $\max\{\log n, \lambda\}$, the $(\mu + \lambda)$ EA needs asymptotically the same number of iterations as the $(1+1)$ EA on OneMax and for $\mu, \lambda = O(\log n)$ it takes the same number of evaluations.

For the (μ, λ) EA we have already discussed that small offspring population sizes $\lambda \leq (1-\varepsilon)e\mu$ can result in exponential runtimes. Antipov, Doerr, and Yang [7] showed that the relationships between the parent and offspring populations are more complex, especially in the threshold $\lambda = (1 \pm \varepsilon)e\mu$. The authors tighten the threshold for λ on OneMax as follows. If $\mu \leq n^{1/2-c}$ for any constant $c > 0$ then any $\lambda \leq e\mu$ result in a super-polynomial runtime. On the other hand if $\mu \geq n^{2/3+c}$ for any constant $c > 0$, then for any $\lambda \geq e\mu$, the runtime is polynomial.

An important thing to notice is that evolutionary algorithms are prone to have *premature convergence*, where all the solutions of the population are the same or very similar to each other. When this happens, the population is no longer useful for global exploration and can affect the performance of the algorithm from maintaining an homogeneous population. Traditionally, to deal with this problem, many mechanisms to diversify the population have been used and studied. Some examples of diversity mechanisms are: avoiding duplicates during the selection process [170], enforcing fitness diversity [83], deterministic crowding [84], etc. Due to the scope of this thesis, we do not expand on the diversity mechanisms, but refer the reader to a recent survey by Squillero and Tonda [169] and the book chapter by Sudholt [174] for more information. We remark that recently Corus, Lissovoi, Oliveto, and Witt [37] proved that reducing the parent selection pressure can address this issue without additional diversity mechanisms.

Crossover probability.

Understanding crossover as a search operator, has been a difficult task. Most of the runtime analyses have been focused on proving the benefits of using crossover. An early study by Jansen and Wegener [111], showed that the $(\mu + 1)$ GA with and without diversity mechanisms on Jump$_k$, has a considerably faster expected optimisation time compared to the $(1+1)$ EA. This proof relied on a unrealistically small crossover probability of $p_c \leq 1/Ckn$ for a large constant C. This parameter value is not normally used in practice. Later, for the same function the analysis was improved by Kötzing, Sudholt, and Theile [118] using a diversity mechanism to allow a crossover probability of $p_c \leq k/n$, which still is unrealistically small. Recently, an analysis by Dang, Friedrich, Kötzing, Krejca, Lehre, Oliveto, Sudholt, and Sutton [40] was able to prove the benefits of crossover on the Jump$_k$ function using constant crossover probabilities. This study proved faster runtimes than mutation only evolutionary algorithms, using several diversity mechanisms. The same authors later showed that without the need of diversity mechanisms the $(\mu+1)$ GA is faster than the $(1+1)$ EA on $\textsc{Jump}_k$ [41], although not as fast as when using the diversity mechanisms from [40]. An important point is that the authors also showed that for mutation probabilities $p = (1+\delta)/n$, for a constant δ, the $(\mu+1)$ GA had an asymptotically better optimisation time.

Other randomised search heuristics have also demonstrate the effectiveness of crossover on the multimodal function class $\textsc{Jump}_k$. Friedrich, Kötzing, Krejca, Nallaperuma, Neumann, and Schirneck [85] showed a $O(n \log n)$ bound on the runtime of an island model with majority vote. Whitley et al. [179] proved that tailored hybrid genetic algorithms using a voting mechanism can optimise $\textsc{Jump}_k$ in expected time $\Theta(n)$ and Buzdalov, Kever, and Doerr [24] designed a tailored black-box algorithm using a voting mechanism that can provably optimise (a slight variation of) $\textsc{Jump}_k$ in expected time $\Theta(n/\log n)$.

The previous examples show the benefits of crossover for functions with local optima, for simple hill-climbing tasks, there have also been several results with [173] and without diversity mechanisms [30, 31] showing improved performance compared to evolutionary algorithms without crossover. Since our focus is on parameter setting the importance of these studies is that they show that when using crossover, the optimal mutation probability and parent population size tend to be larger than without crossover. This shows the complicated interplay between these parameters, making optimal parameter tuning on real-world problems an almost impossible task.

Heavy-tailed parameters and other distributions

In an attempt to create a mutation operator that could hill-climb and escape local optima efficiently, Doerr et al. [62] proposed the mutation operator $\texttt{fmut}_\beta$ that use standard bit mutation with a mutation probability $p = r/n$ where the mutation parameter r is sampled each generation from a power law distribution with exponent $\beta > 1$. Doerr et al. [62] showed that the $(1+1)$ EA using this mutation operator can optimise $\textsc{Jump}_k$ functions in $O\left(k^{\beta-0.5}\left(\frac{en}{k}\right)^k\right)$ which is only a small polynomial factor $O(k^{\beta-0.5})$ slower than the $(1+1)$ EA with best choice of mutation probability using standard bit mutation.

Motivated by this result, Antipov, Buzdalov, and Doerr [8], Antipov, Doerr, and Karavaev [9], Antipov, Buzdalov, and Doerr [10] chose the parameters of the $(1+(\lambda,\lambda))$ GA from power law distributions. For different parameters in the power law distributions it was shown that the $(1+(\lambda,\lambda))$ GA is asymptotically faster than any static parameter choice on $\textsc{OneMax}$ [8] and only a small polynomial factor slower than the $(1+(\lambda,\lambda))$ GA with optimal static parameter choice on $\textsc{Jump}_k$ functions [9, 10].

Other mutation operators that change the distribution of the parameters (mainly the mutation probability) have been proposed. Friedrich, Quinzan, and Wagner [86] proposed a mutation operator $\texttt{cmut}_P$ that with probability P flips one bit and with probability $1-P$ flips r bits with r chosen uniformly at random from $[2\dots n]$. This mutation operator is slower by a linear or constant factor than $\texttt{fmut}_\beta$ on most $\textsc{Jump}_k$ functions but it is faster on $\textsc{Jump}_k$ functions when $n-k$ is constant. Similarly Quinzan, Göbel, Wagner, and Friedrich [159] modified the $\texttt{fmut}_\beta$ operator to flip r bits chosen uniformly at random instead of flipping each bit with probability r/n. This modification was shown to have a similar performance than the original $\texttt{fmut}_\beta$ on $\textsc{Jump}_k$ and $\textsc{OneMax}$ and additionally it was shown to find $(1/3)$ approximations on any non-negative submodular functions and symmetric submodular functions under a single matroid constraint in polynomial time. Corus, Oliveto, and Yazdani [38] proposed a hyper mutation operator for AIS using a symmetric power-law distribution that performs better than classic hyper mutation operator and at least as good or better than the other mutation operators in a wide range of benchmark problems.

The use of different parameter distributions is not limited to increasing the ability of an evolutionary algorithm to escape local optima. Doerr, Doerr, and Kötzing [63] studied how sampling the mutation strength from different distributions affects the performance of RLS on a generalisation of the OneMax problem to a larger alphabet than $\{0,1\}$. Doerr, Doerr, and Kötzing [65] showed that there is a distribution from which, if the mutation probability is sampled from it every generation, the $(1+1)$ EA is able to optimise OneMax and LeadingOnes with unknown solution length at almost no extra cost. Ye, Doerr, and Bäck [184] change the distribution of number of bit-flips of standard bit mutation from a binomial distribution towards a normal distribution allowing an easy way to control the variance, and hence the degree of randomness of mutations.

In a first attempt to understand what are the optimal distributions to sample parameters, Buzdalov and Doerr [23] studied the $(1+\lambda)$ EA and found that for different values of λ the optimal distributions change in unexpected ways.

Parameter tuning vs parameter control

As we have seen, the correct parameter selection of evolutionary algorithms is a complicated task that depends on the interplay of the parameters and the problem in hand. Additionally, even with the best static parameters the algorithms can have suboptimal performance due to changes during the optimisation process. An obvious alternative is to use dynamic parameters, we call these parameter settings *parameter control mechanisms.* Parameter control aims to adjust the parameters during the run in order to identify parameter values that are optimal for the current state of the optimisation process. Dynamic parameter settings allow evolutionary algorithms to adjust to the fitness landscape during different stages of the optimisation process.

In continuous optimisation, parameter control is indispensable to ensure convergence to the optimum, therefore, non-static parameter choices have been standard for several decades. In sharp contrast to this, in the discrete domain parameter control has only become more common in recent years. This is in part owing to theoretical studies demonstrating that parameter control mechanisms can provably outperform the best static parameter settings.

Even though parameter control has shown promising results, there is still a lack of understanding of how and when parameter control mechanisms are better than static parameter settings. The theoretical study of parameter control in evolutionary algorithms is still relatively new compared to the study of static parameter settings.

2.5.1 Taxonomy of Parameter Settings

As mentioned before evolutionary algorithms come with a range of parameters, and it is well known that the optimal parameter settings differ between different optimisation problems and can even change throughout the optimisation process [51].

There are different types of parameter settings. One of the first attempts to classify them was by Eiben, Hinterding, and Michalewicz [78], where the authors proposed a taxonomy scheme that was widely accepted. More recently an updated version was proposed by Doerr and Doerr [51]. Here we use a variation of this classification scheme, shown in Figure 2.2. In contrast to the one proposed in [51], we classify hyper-heuristics as learning-inspired and success-based parameter control mechanisms, because hyper-heuristics that adjust parameters settings have the same behaviour as these parameter control mechanisms.

From Figure 2.2 we can see that the parameter settings are divided in two, parameter tuning, where static parameters are chosen before the run either manually or automatically, and parameter control, where non-static parameters are chosen each generation following certain rules. Parameter control is further divided in several types according to the heuristics used to choose the parameter values. Here we will explain these different types of parameter control. Afterwards, in Chapter 3 we will give a thorough review of theoretical analyses of parameter control mechanisms.

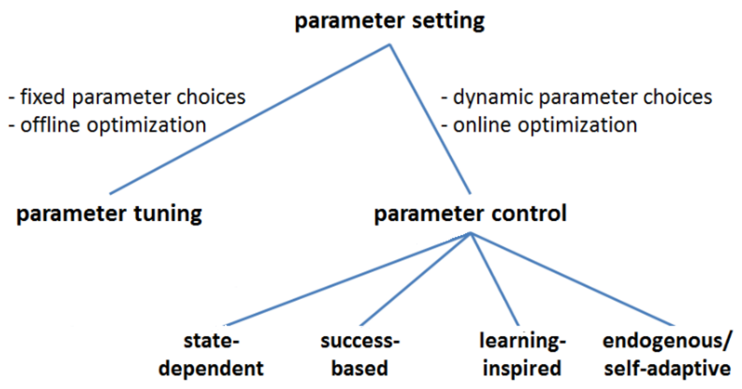

Figure 2.2: Classification of Parameter Control Mechanisms. Adapted from [51].

State-Dependent Parameter Control

State-dependent parameter control mechanisms are the ones that depend directly on the current state of the search process. It can be a value directly related to the search process (eg. fitness value), characteristics of the current population (eg. homogeneity), or other values (eg. iterations). The main characteristic of these mechanisms is that the parameter values are determined by a predefined function that maps the state of the optimisation process to a parameter setting. An important constraint for this type of mechanisms is that, to have an optimal performance they generally need to be tailored to a specific problem which needs a substantial knowledge of the problem in hand [51].

State-dependent parameter control mechanisms have been further subdivided according to the type of information used to adjust the parameters, common subdivisions are [51]:

- Time-Dependent Parameter Control
- Rank-Dependent Parameter Control
- Fitness-Dependent Parameter Control

Success-based parameter control

Success-based parameter control (also called *self-adjusting parameter control*) mechanisms adjust the parameter settings from one iteration to the other. The manner in which the parameters are adjusted is determined by the previous parameter value used and the success of the previous iteration. In contrast to state-dependent mechanisms, success-based mechanisms are not tailored to any specific problem and therefore do not need a deep understanding of the problem in hand for their application.

A common success-based mechanism is the use of multiplicative rules, where the parameters are adjusted using update factors depending on the improvement of fitness in the previous iteration. A well known example of a multiplicative rule is the one-fifth success rule, that suggests to adjust the parameter values so that we obtain an iteration where there is a fitness improvement every five iterations. This rule was first conceived independently by Devroye [46], Rechenberg [163], Schumer and Steiglitz [167]. At first it used several iterations to adjust the parameter but was later simplified by Kern, Müller, Hansen, Büche, Ocenasek, and Koumoutsakos [116] for continuous evolutionary strategies and later applied in discrete optimisation by Doerr et al. [60]. In [60], the rule updates the offspring population size, multiplying its value by a factor $F^{1/4}$ in an unsuccessful iteration and dividing by F otherwise.

Learning-inspired parameter control

Learning-inspired parameter control mechanisms are characterised by learning the parameter values using information from previous iterations. This is a similar approach as the success-based parameter control mechanisms, but without limiting to only the last iteration. Intuitively, during the learning process, a common approach is to give more *weight* to recent iterations than older ones.

Self-adaptive parameter control

Self-adaptive parameter control considers the parameter control as part of the same optimisation problem, i.e. encodes the parameter values inside the individuals and optimises them together with the problem in hand. The motivation behind this type of parameter control is that each individual contains its own parameter value and the selection mechanisms will ensure that good parameter values are maintained in the population. In general, first the encoded mutation rate is mutated and later this new mutation rate is used to mutate the individual. This is necessary for the selection to see the effect of the new mutation rate and hopefully select the most profitable mutation rate.

It has been suggested to not use self-adaptive mechanisms in elitist evolutionary algorithms [51], because they are prone to get stuck with individuals that have a high fitness, but also have a destructive or unprofitable mutation rate. This is because elitist evolutionary algorithms only accept individuals with better or equal fitness, and a destructive mutation rate would not enable the algorithm to accept a new individual nor a better mutation rate. Although elitist algorithms can get stuck in this way, there is no extensive study on the probability of this event happening and in which fitness landscapes this probability increase or decrease.

Chapter 3

State of the Art in Parameter Control

In this section we will give a thorough review of theoretical studies of parameter control mechanisms. Even though our research focus is on the theoretical analyses the review will also contain empirical studies, because some algorithmic concepts that have been first proposed in empirical studies have later been studied by theoretical means, but some others have not, and there can be promising results by studying them.

In the following rather than following the taxonomy defined before (Section 2.5.1), we divide the literature by the underlying algorithms were the parameter control mechanisms are applied or parameter adjusted to be able to compare more easily the results presented.

3.1 Dynamic Mutation Rate for the $(\mu + \lambda)$ EA

In this section we review the history of dynamic mutation rates for the $(\mu + \lambda)$ EA. During this section, we purposely avoid mentioning self-adaptive mechanisms since we will review them in Section 3.3.

One of the first empirical studies showing an improved performance of dynamic mutation rates over static ones was a time-dependent exponentially decreasing mutation rate $p = \frac{0.11375}{240} \cdot 2^{-t}$ by Fogarty [81], but the experimental setup was highly tailored. Later, a fitness-dependent mutation rate for the $(1 + \lambda)$ EA on OneMax was proposed by Bäck [12] which also decrease rapidly when the fitness is increased, from $p = 1$ when the fitness of an individual x is $f(x) \leq n/2$ to $p = 1/n$ when $f(x) = n - 1$. This fitness-dependent mutation rate showed an improved performance compared to static mutation rates on OneMax. In an attempt to generalise the results from [12], Bäck and Schütz [13] proposed a time-dependent mechanism that used the maximum number of generations T to adjust the mutation probability using the formula $p_t = \left(2 + \frac{n-2}{T-1} \cdot t\right)^{-1}$, with $t \in \{0, 1, \dots, T - 1\}$ being the actual generation. This time-dependent mechanism showed improved performance against static parameter values for the knapsack, maximum cut and maximum independent set problems.

A similar concept to these fast decreasing mutation rates is called inversely proportional hypermutations (IPH) and is used in artificial immune systems (AIS). IPH mutates more aggressively search points far away from the optimum and search points near the optimum are only subject to small mutation probabilities. AIS using IPH have been studied on OneMax [185], Cliff and TwoMax [35] showing disappointing results. Corus, Oliveto, and Yazdani [35] proposed an alternative implementation of IPH that is efficient on OneMax, LeadingOnes, and combined with an ageing mechanism it is also efficient on Cliff and TwoMax.

Jansen and Wegener [109] and Droste, Jansen, and Wegener [75] performed the first theoretical studies on runtime analysis of parameter control mechanisms. In these studies a time-dependent mutation rate for the $(1+1)$ EA was studied, the mechanism selected from a set of mutation probabilities $p_t \in \{2^{t-1}/n \mid t = 0, 1, \ldots, \lceil \log n \rceil\}$. The studies showed a faster asymptotic runtime (by a small polynomial factor) than the static $(1+1)$ EA with any value of p on a highly tailored benchmark function but slower asymptotic runtime on the more common problems OneMax and LeadingOnes. The study only showed upper bounds for the runtimes, therefore we cannot confirm that the parameter control mechanism studied is faster. This work was later extended by Jansen and Wegener [112] showing an example function where the dynamic $(1+1)$ EA has a polynomial expected runtime while the static $(1+1)$ EA with any value of p needs superpolynomial expected runtime w. o. p.

Cervantes and Stephens [26] proposed a $(\mu+1)$ EA with rank-based mutation rate and showed promising performance empirically. The intuition behind this mechanism was to exploit and explore the search space simultaneously, mutating the best solutions with small mutation rates and at the same time, mutating the individuals with worse fitness more aggressively. Inspired by this work, Oliveto, Lehre, and Neumann [152] theoretically showed that the rank-based $(\mu+1)$ EA has an improved performance on a constructed function, but they also showed another constructed function where the static $(\mu+1)$ EA performs better.

The first dynamic optimal parameter setting was studied on the LeadingOnes problem by Böttcher et al. [21]. The algorithm proposed was the $(1+1)$ EA with fitness-dependent mutation rate. This algorithm was the first to show a significant advantage over the optimal static parameter settings and no other fitness-dependent mutation rate can achieve a better expected optimisation time on LeadingOnes. Similar to the previous study, Badkobeh et al. [15] and Lehre and Sudholt [127] proposed a $(1+\lambda)$ EA with fitness-dependent mutation rate for the OneMax problem that has an asymptotically optimal runtime amongst all λ-parallel mutation-based unbiased black-box algorithms. In a similar manner Doerr, Doerr, and Yang [67] proposed a $(1+1)$ EA with fitness-dependent mutation rate for the OneMax problem that approaches the optimal runtime for all mutation-based unbiased black-box algorithms by an additive εn term, for an arbitrarily small constant ε.

Following the work from [15] a success-based mutation rate was proposed by Doerr, Gießen, Witt, and Yang [66], called 2-Rate $(1+\lambda)$ EA or self-adjusting $(1+\lambda)$ EA$_{\{2r,r/2\}}$. This algorithm finds the optimum in the same asymptotic expected runtime than the $(1+\lambda)$ EA with optimal fitness-dependent mutation rate on OneMax [15, 127]. The mechanism consists of creating two subpopulations, with half of the offspring each. One of them with a mutation rate half of the current mutation rate and the other with twice the current rate. After an iteration the mutation rate is updated either by randomly choosing one of the two mutation rates used or to the mutation rate used by the subpopulation with the best offspring. They also proposed a version without the random choice and another with three subpopulations, the third population using the actual mutation rate. This 3-Rate $(1+\lambda)$ EA was also proposed earlier by Thierens [176] where the author empirically showed an improved performance over static parameter values on OneMax and the knapsack problem.

Another $(1+1)$ EA with success-based mutation rate mechanism was proposed by Doerr and Wagner [71], where the authors empirically compared the algorithm against the best possible fitness-dependent mutation rates on OneMax and LeadingOnes. The mechanism used a multiplicative update rule for the mutation rate, that is, the mutation rate was multiplied by a factor $A \geq 1$ if the offspring has better or equal fitness than the parent and multiplied by a factor $b \leq 1$ otherwise. Based on this, Doerr, Doerr, and Lengler [68] made a theoretical study of the success-based mechanism. In this study the authors made a relation between the factors A and b in [71] and a generalised version of the one-fifth success rule where the mutation rate is multiplied by F^s in a successful generation and divided by F in an unsuccessful generation, with F being the update strength and s the success rate. The update rule $p = pA$ and $p = pb$ with values $A = 1.11$, $b = 0.66$ is equivalent to $p = pF^s$ and $p = p/F$ with $F = 1.5$ and $s = 4$ (the one-fifth success rule). Using this analogy they found that the success rule with values $F = 1 + o(1)$ and $s = e - 1$ achieves asymptotically optimal running time on LeadingOnes.

Later three new success-based mutation rate mechanisms were proposed by Ye et al. [184], the $(1 + \lambda)$ EA$_{\text{norm}}$, the $(1 + \lambda)$ EA$_{\text{var}}$, and the $(1 + \lambda)$ EA$_{\text{half}}$. The first two algorithms use a normal distribution $N(r, r(1 - r/n))$ to sample the number of bit-flips in a mutation instead of the binomial distribution $\mathcal{B}(n, r/n)$ that the standard bit mutation would sample. The authors argue that the distributions are equivalent because by the central limit theorem the binomial distribution converges to the normal distribution. The algorithms self-adjust the mutation parameter r every iteration by assigning the number of bit-flips that the best offspring did in the last iteration to r. The difference between both algorithms is that the $(1+\lambda)$ EA$_{\text{var}}$ also reduces the variance of the normal distribution by a factor if the previous r is equal to the number of bit flips of the best offspring. The $(1 + \lambda)$ EA$_{\text{half}}$ creates half of the offspring population by flipping exactly r bits from the parent and samples the number of bit-flips from a uniform distribution $U(0, 2r/n)$ for the other half of the offspring population. The value r is adjusted each generation to the number of bit-flips of the best offspring of the last generation. In the study the authors did experiments on OneMax and LeadingOnes comparing their algorithms against the 2-Rate $(1 + \lambda)$ EA and RLS showing an improved performance for the problem sizes tested even against RLS.

When tackling an optimisation problem, it can be useful to use a biased mutation operator, that is, a mutation operator that treats 1-bits different than 0-bits. For example for most of the benchmark problems studied here we can see that flipping 0-bits to 1-bits is more profitable than the other way around. Obviously this is not the case for all optimisation problems and choosing between a biased and an unbiased mutation operator still needs a certain domain knowledge. In an attempt to automate this choice Rajabi and Witt [160] proposed a learning-inspired $(1 + 1)$ EA with asymmetric mutation that uses an independent mutation probability for 1-bits and 0-bits, and adjusts these probabilities after an observation length N. This learning-inspired $(1 + 1)$ EA with asymmetric mutation was shown to have a runtime of $O(n)$ on OneMax which is a logarithmic factor faster than an unbiased mutation operator and a constant factor faster than the previously best known static asymmetric mutation [108]. Despite using an asymmetric mutation the learning-inspired $(1 + 1)$ EA still optimises a generalised OneMax$_a$ function describing the number of matching bits with a fixed target $a \in \{0, 1\}^n$ in $O(n \log n)$ expected number of evaluations.

For multimodal optimisation Rajabi and Witt [161] proposed a learning-inspired mechanism called *stagnation detection* that increases the mutation probability after a set amount of iterations without a fitness improvement. The number of iterations employed ensures w. h. p. that the neighbourhood has already been searched, allowing the algorithm to explore a larger radius. The stagnation detection mechanism applied to the $(1 + 1)$ EA is able to optimise the Jump$_k$ function with the same asymptotic runtime as the $(1 + 1)$ EA with optimal static mutation rate $p = k/n$ without previous knowledge of the jump size k.

Given that there is an increased number of parameter control mechanisms, it is hard to decide which algorithm to use just by looking at each study by itself. In order to help practitioners decide, a number of empirical studies have been made comparing these algorithms. Rodionova, Antonov, Buzdalova, and Doerr [164] did an empirical analysis of the 2-Rate $(1 + \lambda)$ EA, 3-Rate $(1 + \lambda)$ EA from [66] and the $(1 + \lambda)$ EA with success-based mutation rate from [68] with focus on the influence of offspring population size and problem size on performance. Comparisons between the 2-Rate $(1 + \lambda)$ EA and other evolutionary algorithms with self-adjusting offspring population size were made by Doerr, Ye, van Rijn, Wang, and Bäck [73] and Hevia Fajardo [99]. The most complete empirical study, in the sense that compares most of the algorithms showed here was done by Doerr, Ye, Horesh, Wang, Shir, and Bäck [74] using the new benchmarking platform IOHProfiler [72].

3.2 Dynamic Offspring Population Size for the $(1+\lambda)$ EA

For the $(1+\lambda)$ EA there have been a limited amount of parameter control mechanisms proposed that adjust the offspring population size. Jansen et al. [114] proposed an evolutionary algorithm adjusting the offspring population size with a multiplicative success-based rule, that is, the offspring population size is multiplied if there is no improvement in fitness and divided otherwise. The update rule is to multiply by 2 in an unsuccessful generation (i. e. no fitness improvement) and divided by s otherwise, with s being the number of successful offspring. The reasoning given in their work was to set λ to roughly the reciprocal of the success probability, minimising the total expected optimisation time. The mechanism obtained promising results in empirical tests on OneMax, LeadingOnes and SufSamp. This algorithm has been called $(1+\{2\lambda, \lambda/s\})$ EA in [51, 73] and generalised to the $(1+\{F\lambda, \lambda/s\})$ EA in [99] with an update factor $F \geq 1$.

Afterwards Lässig and Sudholt [120] proposed two similar success-based parameter control mechanisms. The update rule used was to double the offspring population size in an unsuccessful generation to help finding improvements. In order to reduce the offspring population size on a successful generation and maintain a "healthy" size they used two approaches. The first approach is to set the offspring population size to 1, this might help if it becomes easier to find an improvement after a success, but if the landscape does not change, it also makes sense to have a similar offspring population size, therefore in the second approach they only halve the offspring population size after a successful generation. Following the same naming conventions from [51, 73] these algorithms are called $(1+\{2\lambda, \lambda/2\})$ EA and $(1+\{2\lambda, 1\})$ EA. In this work Lässig and Sudholt proved that there is an improvement in asymptotic parallel optimisation time (number of generations) without affecting the asymptotic sequential optimisation time (number of evaluations) on OneMax, LeadingOnes and Jump$_k$. These results can be extended to the $(1+\{F\lambda, \lambda/s\})$ EA and other multiplicative rules.

Later Hevia Fajardo [99] proposed another mechanism using the one-fifth rule. The proposed $(1+\{F^{1/4}\lambda, \lambda/F\})$ EA and the other success-based mechanisms proposed in [114, 120] were empirically tested on OneMax, LeadingOnes, SufSamp and Makespan Scheduling. All of the mechanisms showed similar performance. In general the $(1+\{F\lambda, \lambda/s\})$ EA performed slightly better than the other mechanisms for the problems tested.

For parallel $(1+1)$ EAs using λ subpopulations with spatial structures called islands, Mambrini and Sudholt [141] proposed two schemes to self-adjust the interval at which individuals migrate between neighbouring islands, that is, a self-adjusting migration interval τ. In their parallel evolutionary algorithms every island has its own migration interval τ_i, after every migration interval, each island broadcast its best solution to its neighbours. The migration interval τ_i is updated in each island using the following two sets of rules:

- When an improvement is found set $\tau_i = 1$. If τ_i generations have passed migrate and set $\tau_i' = 2\tau_i$
- When an improvement is found set $\tau_i' = \tau_i/2$. If τ_i generations have passed migrate and set $\tau_i' = 2\tau_i$

In their work they analysed the parallel runtime and the total number of migrants sent called *communication effort* of these mechanisms compared against a fixed scheme for τ with Complete, Ring, Torus and Hypercube spatial structures. All the schemes had the same parallel runtime bound, and the adaptive schemes reduced or maintained the communication effort for LeadingOnes, unimodal functions and three of the four structures on Jump$_k$. For three of the four structures on OneMax and one structure on Jump$_k$ the communication effort was increased.

3.3 Self-Adaptive Mutation Rate

Evolutionary algorithms with self-adaptive mutation rate encode the mutation rate in the genomes of the individuals letting the algorithm optimise its own parameters in parallel to the optimisation task. One of the first empirical studies on evolutionary algorithms with self-adaptive mutation rate was made by Bäck [11]. In this study Bäck compared three $(\mu+\lambda)$ GA with two-point crossover to solve continuous problems of l dimensions, encoding the solutions into a pseudo-Boolean bitstring of size $n = 32l$ using Gray code. One of the genetic algorithms had a static mutation rate with $p < 1/n$, the second used a self-adaptive approach that would encode the mutation rate into 20 bits at the end of every individual and the third genetic algorithm used a self-adaptive mechanism that encoded one mutation rate for every dimension l into the end of every individual. The results found that the self-adaptive algorithm with one mutation rate was better than the static choice on all problems and the self-adaptive algorithm with l mutation rates was only better on one multimodal problem.

Later Bäck [12] tested the same self-adaptive mutation rate as before on OneMax, using a (μ, λ) GA. Bäck showed that at every stage of the optimisation process the algorithm contained similar values of mutation rate as a fitness-dependent mutation rate proposed in the same work (which we now know is not the optimal fitness-dependent mutation rate). Judging from the graphs presented, the minimum mutation rates present in the population drop to $p \approx 1/n$ in the end of the optimisation process, which is the optimal static choice.

Following this, Bäck and Schütz [13] empirically investigated the effect of different encodings for the mutation rate of a "Steady State" genetic algorithm with self-adaptive mutation rate. The results were then compared with static mutation rates; the self-adaptive mechanisms show an improvement against them on highly multimodal problem instances. An important point is that the self-adaptive mechanisms use a log-normal distribution to mutate the mutation rate. This mechanism has been recently empirically compared against other self-adjusting mutation rates by Doerr et al. [74] showing poor performance on most of the problems tested. We need to take into account that in this comparison Doerr et al. used a $(1 + 10)$ EA that uses an elitist selection which is different from the selection mechanisms used by Bäck and Schütz [13].

The first rigorous runtime analysis on self-adaptation was made by Dang and Lehre [39]. Using a highly tailored problem Dang and Lehre [39] demonstrate that a self-adaptive mutation rate (that only uses a finite set of mutation rates $\mathcal{M}$) with a non-elitist selection is able to optimise the function in polynomial time. Not only that, but any static mutation rate from $\mathcal{M}$, or randomly changing the mutation rate from $\mathcal{M}$ at each iteration would take exponential expected optimisation time.

In contrast to Dang and Lehre [39], more recently, Doerr et al. [69] did a runtime analysis of a self-adaptive mutation rate on a more common function, OneMax. Their mechanism also used only a discrete selection of mutation probabilities with a mutation parameter $r \in \{1, F, F^2, \ldots, F^{\lfloor \log_F(n/(2F)) \rfloor}\}$ for all $F > 1$. During an iteration the mutation parameter of an individual r (with mutation probability $p = r/n$) is changed uniformly at random either to r/F or rF, and then the individual is mutated with the new mutation rate. In the end of the iteration the best offspring is used as parent for the next iteration; in case of ties the individual with smaller mutation rate is preferred. During the analysis the authors used $F = 32$ to ease the analysis, but they suggest to use smaller values of F in practice. Doerr et al. [69] showed that the self-adaptive $(1, \lambda)$ EA with $\lambda \geq C \log n$ with a sufficiently large constant C has the same asymptotic expected runtime as the optimal fitness-dependent choice from [15, 127].

A self-adaptive (μ, λ) EA was proposed and studied by Case and Lehre [25]. In this analysis the self-adaptive mechanism selects mutation probabilities over a continuous interval. In the self-adaptive (μ, λ) EA the mutation rate of each parent is multiplied by either $A > 1$ with probability p_{inc} or $b \in (0, 1)$ with probability $1 - p_{\text{inc}}$ before creating an offspring via standard bit mutation with the new mutation rate. For a non-trivial selection of parameters $A, b, p_{\text{inc}}, \mu, \lambda$ the self-adaptive (μ, λ) EA showed an expected runtime of $O(k^2)$ on the LeadingOnes$_k$ problem with unknown solution length k and problem size n. This is faster than the best known runtime for this problem shown by Doerr et al. [65].

3.4 Self-Adjusting $(1 + (\lambda, \lambda))$ GA

The $(1 + (\lambda, \lambda))$ GA was proposed by Doerr et al. [60] and studied on OneMax. It was shown that for $\lambda \in [\omega(1), o(\log n)]$ it leads to expected optimization times $o(n \log n)$ breaking the $\Theta(n \log n)$ barrier that applies to all mutation-only algorithms [128]. Additionally, it was proved that a fitness-dependent choice of the offspring population size $\lambda = \left\lceil \sqrt{n/(n - f(x))} \right\rceil$, with $p = \lambda/n$ and $c = 1/\lambda$ could yield an expected optimisation time $\Theta(n)$ on OneMax which is asymptotically faster than any static parameter values.

To aid the complicated task of finding an optimal fitness-dependent choice, Doerr et al. [60] proposed the self-adjusting $(1 + (\lambda, \lambda))$ GA that uses the one-fifth success rule to adjust λ. The self-adjusting $(1 + (\lambda, \lambda))$ GA showed promising empirical results on OneMax, RoyalRoads$_5$ and linear functions. Doerr and Doerr [50] proved that the self-adjusting $(1 + (\lambda, \lambda))$ GA only needs $O(n)$ expected function evaluations on OneMax. This was the first success-based parameter control mechanism proven to reduce the optimisation time of an algorithm by more than a constant factor, compared to optimal static parameter settings.

Following these theoretical results, several empirical studies have been conducted. Goldman and Punch [91] tested the self-adjusting $(1 + (\lambda, \lambda))$ GA on combinatorial problems using a restart mechanism when the offspring population size increased to $\lambda = n$ reporting excellent performance on random instances of the maximum satisfiability problem. Without restarts and a limited budget Foster et al. [82] reported opposite results: the self-adjusting $(1 + (\lambda, \lambda))$ GA with limited budget found worse solutions than all other evolutionary algorithms tested on the same instances of the maximum satisfiability problem from [91]. Hevia Fajardo [99] compared the self-adjusting $(1 + (\lambda, \lambda))$ GA against other success-based evolutionary algorithms showing a poor performance on LeadingOnes, SufSamp and Makespan Scheduling and proposed a new version of the self-adjusting $(1 + (\lambda, \lambda))$ GA using Optimal Stopping Selection that showed a 20% improvement on OneMax. Dang and Doerr [44] investigated empirically the influence of the hyper-parameters on its performance. From the hyper-parameter tuning the authors obtain an improvement of 16% over the original parameters from [50] self-adjusting $(1 + (\lambda, \lambda))$ GA on OneMax.

Inspired by the results shown by Goldman and Punch [91], Buzdalov and Doerr [22] made a runtime analysis on random satisfiability instances showing that the self-adjusting $(1 + (\lambda, \lambda))$ GA is effective on instances with good fitness-distance correlation but for weak fitness-distance correlation the algorithm might increase λ to a value that affects its performance. Bassin and Buzdalov [18] proposed a roll-back mechanism for the parameter λ that slows down the growth of λ and experimentally showed an improved performance on random satisfiability instances. Antipov et al. [6] analysed the $(1 + (\lambda, \lambda))$ GA on LeadingOnes to better understand the behaviour on functions with low fitness-distance correlation. They showed that the self-adjusting $(1 + (\lambda, \lambda))$ GA has the same asymptotic runtime as the $(1 + 1)$ EA on LeadingOnes, but the hidden constants seem to be very large.

Antipov et al. [8], Antipov and Doerr [3] and Antipov et al. [10] exchanged the self-adjusting choice of parameters of the $(1 + (\lambda, \lambda))$ GA for heavy-tailed distributions obtaining the same asymptotic runtime of $O(n)$ on OneMax [8] and a good performance on JUMP$_k$ functions [3, 10]. Since we are interested in understanding parameter control mechanisms we refrain from expanding on these studies in this section.

3.5 Dynamic Parameters on RLS$_k$

The first runtime analysis on RLS$_k$ (with $k \neq 1$) was made by Neumann and Wegener [144] on minimum spanning trees. The authors used a version RLS$_{1,2}$ that can only make either 1-bit flips or 2-bit flips. For the RLS$_{1,2}$ they show that it computes the minimum spanning tree in an expected number of iterations $O(m^2(\log n + \log w_{max}))$ with n vertices and m edges. The same RLS$_{1,2}$ was also studied by Qian, Tang, and Zhou [158] showing that for a constructed multi-objective problem the algorithm needs both 1-bit and 2-bit flips. A related study by Antipov and Doerr [5] showed that the $(1+1)$ EA with any unbiased mutation operator[1], having a constant probability of exactly flipping one bit (this includes any RLS$_k$ that uses 1-bit flips with constant probability), has an expected runtime of $\Theta(n^k)$ on PLATEAU$_k$ with $k \geq 2$ constant (k is the size of the plateau).

The previous results were made with a static probability of selecting any mutation strength k. The first study on varying these probabilities was made by Lehre and Özcan [126]. In this work, the authors first studied the behaviour of the RLS$_{1,2}$ with different static probabilities p_k of performing k-bits flip on ONEMAX and GAPPATH (an improved result on ONEMAX was given by Doerr and Doerr [51]). Later they propose the use of the hyper-heuristic selection mechanism *permutation* to vary the probabilities p_k and showing that the runtime for GAPPATH is finite compared to an infinite runtime using either 1-bit flips or 2-bit flips alone.

Afterwards Alanazi and Lehre [2] studied several hyper-heuristic selection mechanisms on LEADINGONES showing that the asymptotic runtime of all the selection mechanisms is similar. They also proposed an improvement to *random gradient* letting the successful low-level heuristic run for a constant number of iterations regardless of its success. Following this result Lissovoi, Oliveto, and Warwicker [136, 139] show a more precise result proving that the four selection mechanisms have the same expected runtime. The authors also study a similar selection mechanism to the one proposed in [2] called *generalised random gradient* and a new selection mechanism proposed there, called *generalised greedy*. The generalised random gradient lets a successful heuristic run for τ iterations; if an improvement is found during the τ iterations the heuristic is still used for the next τ iterations. The generalised greedy uses all low-level heuristics until an improvement is found and the corresponding heuristic that found the improvement is used the next τ iterations. From these two new selection mechanisms the generalised random gradient has the best possible optimisation time for any RLS$_k$ on LEADINGONES, making it faster than the $(1+1)$ EA and RLS$_1$.

The results in Lissovoi et al. [136, 139] required the correct selection of the learning period τ; this was later solved by Doerr, Lissovoi, Oliveto, and Warwicker [64] using a success-based mechanism in order to self-adjust the value of τ during the run of the algorithm. This mechanism is able to obtain the same optimal asymptotic expected runtime on LEADINGONES without the need to tune the learning period τ. The mechanism counted the number of improvements in a period of time τ, if it exceeded the limit σ, the period was updated to $\tau = \tau F^{-1/\sigma^2}$ and $\tau = \tau F^{1/\sigma}$ otherwise. They show that this mechanism has the best runtime achievable using RLS$_1$ and RLS$_2$, up to lower order terms. Similarly, Lissovoi, Oliveto, and Warwicker [138] showed that the same mechanism is able to find suitable learning periods of time τ on ONEMAX and RIDGE, allowing RLS$_k$ to have an optimal asymptotic expected runtime on both problems.

In parallel to the previous study using hyper-heuristics, Doerr et al. [67] showed an optimal fitness-dependent RLS$_k$ on ONEMAX that maximises the approximated expected

[1]That is, a mutation operator that does not differentiate between 1-bits and 0-bits.

fitness increase. Inspired by this result, Doerr, Doerr, and Yang [61] proposed a learning-inspired mutation strength for RLS$_k$. This mechanism estimates the future progress of a parameter value using learning iterations. This mechanism was theoretically proven to track the optimal values on OneMax and experiments on LeadingOnes and the minimum spanning tree problem show that it outperforms other classical algorithms.

Rajabi and Witt [162] proposed to adapt the mutation strength k of RLS$_k$ using the stagnation detection mechanism from [161] which is able to use all mutation strengths $k \in [1, n]$. Rajabi and Witt [162] showed that RLS$_k$ using this mechanism is able to optimise Jump$_k$ functions faster than the $(1+1)$ EA with the same stagnation detection mechanism by a constant factor.

One of the main reasons to use RLS$_k$ with $k \neq 1$ is to be able to escape local optima, but this is not the only approach one may use to do so. Lissovoi et al. [137] proposed a hyper-heuristic that chooses between elitist and non-elitist selection heuristics for RLS$_1$ that is able to escape local optima with small basins of attraction. This algorithm achieves an expected runtime of $O(n \log n + n^3/d^2)$ on the problem class Cliff$_d$. When $d = \Theta(n)$ this matches the optimal expected runtime for any unary unbiased black-box algorithm [128].

Doerr et al. [63] presented a success-based choice of the mutation strength for an RLS$_k$ variant, proving that it is very efficient for a generalisation of the OneMax problem to a larger alphabet than $\{0, 1\}$.

3.6 Conclusions

In this chapter we have seen that the theoretical study of parameter control mechanisms in evolutionary algorithms is a very fruitful research area, with many results showing that they can outperform the best static parameter values. We gave an overview of parameter control mechanisms adapting mutation rates and offspring populations sizes for the $(\mu + \lambda)$ EA. We then presented state-of-the-art self-adaptive mechanisms for the $(\mu + \lambda)$ EA and (μ, λ) EA. Finally, we gave an overview of dynamic parameter values for the self-adjusting $(1 + (\lambda, \lambda))$ GA and RLS$_k$.

Despite the large body of research in parameter control mechanisms, there is a lack of understanding of how they behave in complex multimodal problems. Most existing runtime analyses focus on simple problems with benign characteristics, for which fixed parameter settings already run efficiently and only moderate performance gains were shown. In this thesis we will use runtime analysis methods to improve our understanding of how success-based parameter control mechanisms can be used to speed up optimisation on multimodal problems.

Chapter 4

Is Success-Based Parameter Control Efficient on Multimodal Problems?

4.1 Introduction

There has been an increasing interest in the theoretical study of success-based parameter control mechanisms in the latest years in order to understand how they work and when they perform better than static parameter settings. Nonetheless, we have a limited understanding of how success-based parameter control mechanisms perform for multimodal optimisation. In this chapter, we aim to understand whether common success-based parameter control mechanisms are efficient when applied on multimodal problems.

We will focus on one of the most successful implementations of success-based parameter control mechanisms: the self-adjusting $(1 + (\lambda, \lambda))$ GA [50, 60]. The self-adjusting $(1 + (\lambda, \lambda))$ GA is the fastest known unbiased genetic algorithm on the popular test function $\text{OneMax}(x) := \sum_{i=1}^{n} x_i$ and has shown excellent performance on NP-hard problems like MaxSat in both empirical [91] and theoretical studies [22]. Despite its success on these problems it has been shown via theoretical [6, 22] and empirical studies [74, 82, 99] that the self-adjusting $(1 + (\lambda, \lambda))$ GA does not behave well on instances with low fitness-distance correlation, that is, instances where the distance of search points to the optimum has a low correlation to their fitness values.

Several of the above works identified that the issue lies in the parameter control mechanism used to control the main parameter of the algorithm: the offspring population size λ. On functions with low fitness-distance correlation, the algorithm can get stuck in situations where λ diverges to its maximum value $\lambda = n$, and then performance deteriorates.

Goldman and Punch [91] suggested to restart the parameter λ to 1 when $\lambda = n$ but also restart the search from a random individual. Buzdalov and Doerr [22] proposed capping the value of λ depending on the fitness-distance correlation of the problem in hand. Lastly, Bassin and Buzdalov [18] proposed a modification where the growth of λ is slowed down for long unsuccessful runs, while still letting the algorithm increase the parameter indefinitely. It achieves this by resetting λ to the parameters of its last successful generation after a certain number of unsuccessful generations and letting the algorithm increase λ a bit more every time it is reset.

At the moment, it is not clear which of these modifications is the best choice. Previous research is fragmented and most of the modifications proposed have only been studied empirically.

In this chapter, we make a step towards understanding the effects that different approaches for parameter control in the self-adjusting $(1 + (\lambda, \lambda))$ GA have, by providing a comprehensive theoretical analysis on the standard Jump_k and Trap benchmark functions (formally defined in Section 2.3). Our main contribution is a comprehensive analysis of three variations of the original parameter control mechanism on the self-adjusting $(1 + (\lambda, \lambda))$ GA showing theoretically and empirically that all the variations studied can improve the performance of the original mechanism on multimodal problems whilst not (or slightly) affecting its performance on simple problems such as OneMax.

4.1.1 Contributions

We consider the standard Jump_k benchmark problem class, a class of multimodal problems on which evolutionary algorithms typically have to make a jump to the optimum at a Hamming distance of k. The parameter k means that Jump_k has an adjustable difficulty and thus represents a whole range from easy to difficult multimodal and even deceptive problems. It was also the first problem for which a drastic advantage from using crossover could be proven with mathematical rigour [111]. More recent analyses have shown that crossover can reduce the runtime to $O(n^{k-1} \log n)$ [41], $O(n \log n + 4^k)$ using additional diversity mechanisms [40] and $O(n \log n)$ using an island model and majority vote [85]. Tailored hybrid genetic algorithms using a voting mechanism can optimise Jump_k in expected time $\Theta(n)$ [179] and tailored black-box algorithms using a voting mechanism can even optimise (a slight variation of) Jump_k in expected time $\Theta(n/\log n)$ [24].

We first present a general method for obtaining upper bounds on the expected optimisation time (the expected number of fitness evaluations to find a global optimum) of the original self-adjusting $(1 + (\lambda, \lambda))$ GA, based on the fitness-level method, in Section 4.2.1. With it we obtain novel bounds including bounds for unimodal functions and the Jump_k function class. Despite its simplicity, we show that it gives tight bounds, up to lower-order terms, on Jump_k functions. Tightness is established through lower bounds in Section 4.2.2. We show that the original self-adjusting $(1 + (\lambda, \lambda))$ GA does not benefit from crossover on Jump_k functions given that it has the same asymptotic runtime as the $(1 + 1)$ EA with standard parameters $p = 1/n$.

Subsequently, in Section 4.3.1 we analyse the performance change when λ is capped to a value less than n. We suggest a generic choice for $\lambda_{\max} := n/2$ that can be advantageous on very hard and deceptive functions without affecting its performance on simple problems. We also show that capping λ can improve the performance of the algorithm on Jump_k, but its behaviour is highly dependent on the choice of $\lambda_{\max}$ defying the point of using a parameter control mechanism. Additionally, with the generic choice of $\lambda_{\max} := n/2$ the self-adjusting $(1 + (\lambda, \lambda))$ GA is faster than the original self-adjusting $(1 + (\lambda, \lambda))$ GA and the $(1 + 1)$ EA with standard parameters for jump sizes $k \leq \log n$ and $k > n/\log n$.

Our analysis in Section 4.3.2 also provides insights into the *parameter landscape*, which describes how parameter values relate to performance and which has recently emerged as a hot topic. The celebrated work by Pushak and Hoos [156] provided evidence that parameter landscapes of prominent algorithms on well-known NP-hard problems like SAT, MIP, and TSP, are surprisingly benign, and parameters often have a unimodal response. Hall et al. [93, 95, 96] rigorously proved that parameters of simple evolutionary algorithms on pseudo-Boolean test problems exhibit a unimodal landscape. These works led to algorithm configurators that exploit unimodal structures [94, 157]. In sharp contrast to the above, we show that the parameter landscape concerning the choice of $\lambda_{\max}$ in the self-adjusting $(1 + (\lambda, \lambda))$ GA on Jump_k is bimodal. More precisely, we regard the time to reach the global optimum from a local optimum with λ at its maximum value as this setting dominates the optimisation time. We give a rigorous but complex formula for this expected time and use semi-rigorous arguments to identify optimal parameters. For appropriate choices of n and k, the landscape features a local optimum located in a wide basin of attraction and a global optimum hidden in a narrow basin of attraction. To the best of our knowledge this is the first proof of a bimodal parameter landscape for the runtime of an evolutionary algorithm on a multimodal benchmark problem. The closest related work and only other such proof we are aware of is for a *unimodal* problem: Lengler, Sudholt, and Witt [135] showed that the parameter landscape for the compact Genetic Algorithm on OneMax has a bimodal structure.

In addition, our analysis in Section 4.3.2 shows that for most jump sizes of k considering the mutation offspring in the selection phase can significantly improve the performance of the self-adjusting $(1 + (\lambda, \lambda))$ GA thanks to the high mutation rates used by the algorithm during the mutation phase.

We also analyse the benefits of resetting λ in Section 4.3.3. With this strategy the algorithm is able to traverse the parameter space, instead of getting stuck with a certain parameter, benefiting the algorithm's behaviour when encountering local optima. In particular, with a clever selection of the update factor F we show that for Jump_k the runtime of the algorithm is only by a factor of $O(n^2/k)$ slower than the runtime of the $(1 + 1)$ EA with optimal parameter choice of $O((en/k)^k)$ [62].

Finally, in Section 4.4 we provide an experimental analysis to test how our theoretical results translate to other fitness landscapes. The experimental results agree with our theoretical results on Jump_k and show that the original self-adjusting $(1 + (\lambda, \lambda))$ GA (from [50]) has a poor performance on most problems tested compared against the other parameter control variations. This demonstrates that our results translate to other common problem settings.

4.1.2 Related Work

We give a brief review of existing theoretical studies on the standard Jump_k benchmark problem class including studies of the $(1 + (\lambda, \lambda))$ GA with non-standard parameter settings previously mentioned in Section 2.5.

The Jump_k function (formally defined in Section 2.3) was designed as a multimodal benchmark function with adjustable difficulty [76] allowing to test the ability of an algorithm to jump over a fitness valley of size k. Crossing this valley can be difficult for evolutionary algorithms. For the $(1 + 1)$ EA using standard bit mutation with the default mutation rate of $p = 1/n$ it takes in expectation $\Theta(n^k)$ evaluations, for $k \leq n/2$. Because of this it has been commonly used in the theory of randomised search heuristics to evaluate their performance on multimodal problems.

For the $(1 + 1)$ EA Doerr et al. [62] showed that for Jump_k with jump size k any mutation rate of $p \in [2/n, k/n]$ is exponentially faster (in k) than the standard choice of $p = 1/n$, with $p = k/n$ being the optimal choice. This showed that the traditional choice of mutation probability is not ideal and as an alternative the authors proposed the mutation operator fmut_β that chooses the mutation rate at random from a heavy-tailed distribution with exponent $\beta > 1$. This new algorithm achieved an asymptotic runtime that is only a factor $k^{\beta-1}$ larger than the optimal mutation rate. A similar mutation operator has been proposed in [86] that can outperform the fmut_β on Jump_k, but only for large jump sizes.

The Jump_k function has also been used as an example where crossover can be beneficial. The first work showing a benefit was [111] showing that a $(\mu + 1)$ *GA* applying uniform crossover only with an unnaturally small crossover probability of $p_c = 1/(kn)$ optimises Jump_k in expected time $O(\mu n^2 k^3 + 4^k/p_c)$. For more natural crossover probabilities, Dang et al. [41] showed a runtime bound of $O(n^{k-1} \log n)$. This can be improved to $O(n \log n + 4^k)$ using additional diversity mechanisms [40] and to $O(n \log n)$ using an island model with majority vote [85].

Recently, for the $(1 + (\lambda, \lambda))$ GA Antipov et al. [9] proved that a non-standard parameter setting of $\lambda = \frac{1}{n}\sqrt{\frac{n}{k}}^k$ and $p = c = \sqrt{\frac{k}{n}}$ can be useful on the multimodal function Jump_k, achieving a runtime of $O(n^{(k+1)/2}e^{O(k)}k^{-k/2})$. However, this result can only be achieved by knowing the jump size k beforehand. To overcome this, Antipov and Doerr [3] and Antipov et al. [10] proposed parameter settings that sample the parameters λ, p and c from power-law distributions. With a non-trivial selection of the distributions these instance-independent algorithms can obtain expected optimisation times that are only a small polynomial factor worse than the aforementioned bound. Note that the $(1+(\lambda,\lambda))$ GA with non-standard parameters from [3, 10] does not use self-adjustment. Since we are interested in understanding self-adjusting mechanisms, in our theoretical work we restrict our attention to the standard parameterisation of the $(1+(\lambda,\lambda))$ GA, that is, we fix the relationship $p = \frac{\lambda}{n}$ and $c = \frac{1}{\lambda}$ and aim to self-adjust λ.

4.2 The Vanilla Self-adjusting $(1+(\lambda,\lambda))$ GA

We recall from Section 2.1 that the $(1+(\lambda,\lambda))$ GA (Algorithm 3) is a crossover-based evolutionary algorithm that uses a mutation phase with a mutation rate higher than usual to assist exploration and a crossover phase as a repair mechanism. The $(1+(\lambda,\lambda))$ GA using the recommended parameters from [50] first creates $\lambda \leq n$ mutants by a process similar to a standard bit mutation with mutation rate λ/n (the only difference to standard GAs being that all mutants flip *the same* number of bits). Then it picks the best mutant and performs λ crossovers with the original parent. A biased uniform crossover is used that independently picks each bit from the mutant with probability $1/\lambda$. After the crossover phase, the algorithm performs an elitist selection using only the offspring from the crossover phase and the parent.

The self-adjusting $(1+(\lambda,\lambda))$ GA (Algorithm 5) adjusts λ every generation with a multiplicative update rule, where λ is multiplied by a factor $F^{1/4}$ if there is no improvement in fitness and divided by F otherwise. We consider $F > 1$ as a constant independent of n unless mentioned otherwise. The parameter λ has an upper limit $\lambda_{\max}$ which is commonly set to n. We note that in lines 5 and 8 of Algorithm 5, following [50], we round λ to its closest integer ($\lfloor\lambda\rceil$); that is, $\lambda = \lfloor\lambda\rfloor$ if the decimal part of λ is less than 1/2 and $\lambda = \lceil\lambda\rceil$ otherwise.

In this work we use a small variation of the algorithm, shown in Line 10 of Algorithm 5, where during the selection step the best offspring from the mutation phase is also considered. This modification has been suggested before in [91, 154] as a way to improve the performance of the $(1+(\lambda,\lambda))$ GA. We believe (and tacitly take for granted) that this change does not invalidate previous theoretical runtime guarantees on problems such as OneMax [50] and LeadingOnes [6]. For OneMax, Doerr and Doerr [50] confirm this fact without proof. Analyses on OneMax and LeadingOnes were based on drift analysis, and the drift can only increase if additional opportunities for improvements are considered. It is less clear how this affects the self-adjusting mechanism; however, previous analyses have shown that the $(1+(\lambda,\lambda))$ GA is very robust in tracking the best parameter setting for these easy unimodal functions and we believe this will still be the case with the modification.

Pinto and Doerr [154] presented additional refinements of the $(1+(\lambda,\lambda))$ GA that they call *implementation-aware*; these can save unnecessary evaluations and decrease some runtime results by constant factors. We only consider the aforementioned change with respect to the selection step for simplicity, and since we are interested in larger performance differences.

Considering the best offspring from the mutation phase is particularly helpful when the algorithm needs to make large jumps, as when encountering local optima. In fact, in Section 4.2.2 and 4.3.2 we show that for large jumps the crossover phase is not very helpful for reaching a higher fitness level, because the crossover phase tends to search near the current parent while the large mutations during the mutation phase can more easily jump out of local optima.

Algorithm 5: The self-adjusting (1 + (λ, λ)) GA with maximum offspring population size $\lambda_{\max} \le n$.

```
1  Initialization: Sample $x \in \{0,1\}^n$ uniformly at random (u.a.r.);
2  Initialize $\lambda \leftarrow 1$, $p \leftarrow \lambda/n$, $c \leftarrow 1/\lambda$;
3  Optimization: for $t = 1, 2, \dots$ do
4      Sample $\ell$ from $\mathcal{B}(n, p)$;
       Mutation phase:
5      for $i = 1, \dots, \lfloor\lambda\rceil$ do
6          Sample $x^{(i)} \leftarrow \text{mut}_\ell(x)$ and query $f(x^{(i)})$;
7      Choose $x' \in \{x^{(1)}, \dots, x^{(\lambda)}\}$ with $f(x') = \max\{f(x^{(1)}), \dots, f(x^{(\lambda)})\}$ u.a.r.;
       Crossover phase:
8      for $i = 1, \dots, \lfloor\lambda\rceil$ do
9          Sample $y^{(i)} \leftarrow \text{cross}_c(x, x')$ and query $f(y^{(i)})$;
10     If $\{x', y^{(1)}, \dots, y^{(\lambda)}\} \backslash \{x\} \ne \emptyset$, choose $y \in \{x', y^{(1)}, \dots, y^{(\lambda)}\} \backslash \{x\}$ with
       $f(y) = \max\{f(x'), f(y^{(1)}), \dots, f(y^{(\lambda)})\}$ u.a.r.;
11     otherwise, set $y := x$;
       Selection and update step:
12     if $f(y) > f(x)$ then $x \leftarrow y$; $\lambda \leftarrow \max\{\lambda/F, 1\}$;
13     if $f(y) = f(x)$ then $x \leftarrow y$; $\lambda \leftarrow \min\{\lambda F^{1/4}, \lambda_{\max}\}$;
14     if $f(y) < f(x)$ then $\lambda \leftarrow \min\{\lambda F^{1/4}, \lambda_{\max}\}$;
```

4.2.1 Fitness-Level Upper Bounds for the Self-Adjusting $(1 + (\lambda, \lambda))$ GA

The self-adjusting $(1 + (\lambda, \lambda))$ GA has only been analysed theoretically on easy unimodal functions like OneMax [50] and LeadingOnes [6] as well as random satisfiability instances [22].[1] Despite these analyses we do not have a clear understanding of its behaviour on other problem settings, especially when it encounters local optima.

In this section we give a new general method to find upper bounds for the runtime of the self-adjusting $(1 + (\lambda, \lambda))$ GA using previously known runtime bounds from the $(1 + 1)$ EA. It is based on the observation that, when λ hits its maximum value of $\lambda = n$, the algorithm temporarily performs n standard bit mutations and thus simulates a generation of a (1+n) EA.

The following theorem gives a fitness-level upper bound tailored to the self-adjusting $(1 + (\lambda, \lambda))$ GA.

Theorem 4.2.1. *Given an arbitrary f-based partition $A_1, \dots, A_{m+1}$. Let d be the number of non-optimal fitness values, $F > 1$ constant, and s_i be a lower bound for the probability that the* $(1 + 1)$ EA *with mutation probability $1/n$ creates a search point in a higher fitness-level set from any search point in fitness level A_i. Then for the self-adjusting* $(1 + (\lambda, \lambda))$ GA *we have*

$$\mathrm{E}\big(T^{\mathrm{eval}}\big) \le O(dn) + 2\sum_{i=1}^{m} \frac{1}{s_i};$$

$$\mathrm{E}(T^{\mathrm{gen}}) \le \lceil 4\log_F(n)\rceil + 6d + \frac{1}{n}\sum_{i=1}^{m} \frac{1}{s_i}.$$

[1]The $(1 + (\lambda, \lambda))$ GA with static parameter choices has also been studied on multimodal functions but only using non-standard parameter choices that do not translate to the self-adjusting mechanism.

To prove Theorem 4.2.1, we analyse the time the algorithm spends in generations with $\lambda < n$ and those in generations with $\lambda = n$.

In the following, we refer to a generation that improves the current best fitness as *successful* and otherwise as *unsuccessful*. We show that a logarithmic number of unsuccessful generations is sufficient to reach the maximum λ value.

Lemma 4.2.2. *Let x and y be the parent and offspring, respectively, as in Algorithm 5. For all values of $F := F(n) > 1$, initial λ value $\lambda_{\text{init}} \geq 1$ and target value $\lambda_{\text{new}} \leq \lambda_{\max}$, the following holds. If in every generation $f(y) \leq f(x)$, the self-adjusting $(1 + (\lambda, \lambda))$ GA needs at most $\left\lceil 4 \log_F \left(\frac{\lambda_{\text{new}}}{\lambda_{\text{init}}}\right)\right\rceil$ generations to grow λ from λ_{init} to at least λ_{new}. During these generations the algorithm makes at most $\frac{4F^{1/4}}{F^{1/4}-1} \cdot \lambda_{\text{new}}$ evaluations.*

For constant $F > 1$ the number of generations is $O(\log \lambda_{\text{new}})$ and the number of evaluations is $O(\lambda_{\text{new}})$.

Proof. Starting with an offspring population size of λ_{init}, after i unsuccessful generations the offspring population size is $\lambda_{\text{init}} \cdot F^{i/4}$. For $i := \left\lceil 4 \log_F \left(\frac{\lambda_{\text{new}}}{\lambda_{\text{init}}}\right)\right\rceil$, we get an offspring population size of

$$\lambda_{\text{init}} \cdot F^{\left\lceil 4 \log_F \left(\frac{\lambda_{\text{new}}}{\lambda_{\text{init}}}\right)\right\rceil/4} \geq \lambda_{\text{init}} \cdot F^{\log_F \left(\frac{\lambda_{\text{new}}}{\lambda_{\text{init}}}\right)} = \lambda_{\text{new}}.$$

The number of unsuccessful generations needed is thus at most

$$\left\lceil 4 \log_F \left(\frac{\lambda_{\text{new}}}{\lambda_{\text{init}}}\right)\right\rceil.$$

Using $\lceil \lambda_{\text{init}} \cdot F^{i/4} \rceil \leq 2\lambda_{\text{init}} \cdot F^{i/4}$, during these generations, the number of evaluations is at most

$$\begin{aligned}
\sum_{i=0}^{\lceil 4\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})\rceil - 1} 2\left\lceil \lambda_{\text{init}} \cdot F^{i/4}\right\rceil &\leq 4\lambda_{\text{init}} \sum_{i=0}^{\lceil 4\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})\rceil - 1} \left(F^{1/4}\right)^i \\
&= 4\lambda_{\text{init}} \cdot \frac{(F^{1/4})^{\lceil 4\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})\rceil} - 1}{F^{1/4} - 1} \\
&\leq 4\lambda_{\text{init}} \cdot \frac{(F^{1/4})^{4\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})+1}}{F^{1/4} - 1} \\
&= 4\lambda_{\text{init}} \cdot \frac{F^{1/4} \cdot \lambda_{\text{new}}/\lambda_{\text{init}}}{F^{1/4} - 1} = \frac{4F^{1/4}}{F^{1/4} - 1} \cdot \lambda_{\text{new}}.
\end{aligned}$$

For constant $F > 1$, the number of generations simplifies to $\lceil 4 \log_F(\lambda_{\text{new}}/\lambda_{\text{init}})\rceil \leq \lceil 4 \log_F \lambda_{\text{new}} \rceil = \lceil 4 \log(\lambda_{\text{new}})/\log(F)\rceil = O(\log \lambda_{\text{new}})$ and the number of evaluations simplifies to $\frac{4F^{1/4}}{F^{1/4}-1} \cdot \lambda_{\text{new}} = O(\lambda_{\text{new}})$. □

Now we bound the number of generations in which the self-adjusting $(1 + (\lambda, \lambda))$ GA operates with $\lambda < n$. To do so, we take into account that the algorithm does not need to increase from λ_{init} every time since we only decrease λ by a factor F each time we find an improvement.

Lemma 4.2.3. *Let $F := F(n) > 1$, $\lambda_{\max} \leq n$, and d be the number of non-optimal fitness values of an arbitrary fitness function. The maximum number of generations in which the self-adjusting $(1 + (\lambda, \lambda))$ GA uses $\lambda < \lambda_{\max}$ is at most $\lceil 4 \log_F(\lambda_{\max})\rceil + 5d$. These generations lead to at most $2d\lambda_{\max} + \frac{4F^{1/4}\lambda_{\max}}{F^{1/4}-1}$ evaluations. For constant $F > 1$ the number of evaluations is $O(d\lambda_{\max})$.*

Proof. In every successful generation, λ is decreased to $\max\{1, \lambda/F\}$ and otherwise it is increased to $\min\{\lambda_{\max}, \lambda \cdot F^{i/4}\}$.

We use the *accounting method* (Section 2.4.3) to account for all generations with $\lambda < \lambda_{\max}$. The basic idea is to create a fictional bank account to which operations are being charged. Some operations are allowed to pay excess amounts, while others can take money from such accounts to pay for their costs. Provided that no fictional account gets overdrawn the total amount of money paid bounds the total cost of all operations.

We start with a fictional bank account and pay costs of $\lceil 4\log_F(\lambda_{\max})\rceil$, since that is the maximum number of consecutive unsuccessful generations before reaching $\lambda = \lambda_{\max}$ (Lemma 4.2.2).

In a successful generation, we pay costs of 1 to cover the cost of the generation, and deposit an additional amount of 4 to the fictional bank account, which will be used to pay for 4 unsuccessful generations needed to increase λ to its original value. Unsuccessful generations that increase λ may withdraw 1 from the fictional account and pay for the cost of this generation. Unsuccessful generations where $\lambda = \lambda_{\max}$ are not charged since they are not counted.

We now need to prove that the fictional bank account is never overdrawn. For any point in time, the number of generations where λ increases is bounded by $T^{\text{inc}} \leq \lceil 4\log_F(\lambda_{\max})\rceil + 4T^{\text{dec}}$ where T^{inc} and T^{dec} are the number of generations increasing and decreasing λ, respectively. This holds by Lemma 4.2.2 and the fact that one successful generation that decreases λ compensates for 4 unsuccessful generations that may increase λ. Considering the initial payment of $\lceil 4\log_F(\lambda_{\max})\rceil$ and transactions for each generation, the current balance is

$$\lceil 4\log_F(\lambda_{\max})\rceil - T^{\text{inc}} + 4T^{\text{dec}} \geq 0,$$

that is, the account is never overdrawn. The number of generations with $\lambda < \lambda_{\max}$ is thus bounded by the sum of all payments. There can only be d successful generations, hence the sum of payments is at most $\lceil 4\log_F(\lambda_{\max})\rceil + 5d$.

It remains to bound the number of evaluations. Since we initialise with $\lambda = 1$, there must be $\lceil 4\log_F(\lambda_{\max})\rceil$ generations $t_1, t_2, \dots$ such that during generation t_i, we have $\lambda \leq \lceil F^{i/4}\rceil$. From Lemma 4.2.2, we know that during these $\lceil 4\log_F(\lambda_{\max})\rceil$ generations, the algorithm uses at most $\frac{4F^{1/4}}{F^{1/4}-1} \cdot \lambda_{\max}$ evaluations. Additionally, in the other at most $5d$ possible generations, the maximum number of evaluations per generation is bounded by $2\lambda_{\max}$. Therefore the algorithm uses at most $2d\lambda_{\max} + \frac{4F^{1/4}\lambda_{\max}}{F^{1/4}-1}$ evaluations with an offspring population size $\lambda < \lambda_{\max}$. For constant $F > 1$, the number of evaluations simplifies to $2d\lambda_{\max} + \frac{4F^{1/4}\lambda_{\max}}{F^{1/4}-1} = O(d\lambda_{\max})$. □

With Lemma 4.2.3 now we have the tools necessary to analyse the runtime of Algorithm 5.

Proof of Theorem 4.2.1. Owing to Lemma 4.2.3, we can focus on bounding the time spent in generations with $\lambda = n$. In these generations, the mutation rate is $p = \lambda/n = 1$ and thus all bits are flipped during mutation. When the current search point is x, mutation thus produces its binary complement, $\overline{x}$. The crossover phase uses a crossover bias of $c = 1/\lambda$, which means that each bit is independently taken from the mutant $\overline{x}$ with probability $1/\lambda$ and otherwise it is taken from x. This is equivalent to a standard bit mutation with mutation probability of $1/\lambda$. Given that $\lambda = n$, during the crossover phase the algorithm creates n independent offspring using standard bit mutation. The crossover phase is then equivalent to the output of a (1+n) EA.

For each fitness level we calculate the number of generations the self-adjusting $(1 + (\lambda, \lambda))$ GA spends on this level while $\lambda = n$ and the self-adjusting $(1 + (\lambda, \lambda))$ GA essentially simulates a (1+n) EA. (We pessimistically ignore the fact that such a situation may not be reached at all; especially on easy problems, λ may not hit the maximum value before the optimum is found.)

We argue as in Lässig and Sudholt [122, Theorem 1] to derive a fitness-level bound for the (1+n) EA. The probability that there is one of n offspring that finds a better fitness level is at least

$$1-(1-s_i)^n \geq 1-\left(\frac{1}{1+s_i n}\right)=\frac{s_i n}{1+s_i n},$$

where the inequality holds by Lemma 2.4.2 (b). The expected number of generations to leave A_i using $\lambda = n$ is at most $\frac{1+s_i n}{s_i n}$. Adding the generations with $\lambda = n$ over all fitness levels and the generations spent with $\lambda < n$, we get

$$\mathrm{E}(T^{\mathrm{gen}}) \leq \lceil 4\log_F(n)\rceil + 5d + \sum_{i=1}^{m}\left(1+\frac{1}{s_i n}\right) \leq \lceil 4\log_F(n)\rceil + 6d + \frac{1}{n}\sum_{i=1}^{m}\frac{1}{s_i},$$

where the last step used $m \leq d$, that is, the number of fitness levels is bounded by the number of fitness values.

By Lemma 4.2.3, the number of evaluations used with $\lambda < n$ is $O(dn)$ when $F > 1$ is a constant. Since with $\lambda = n$ each generation leads to $2n$ evaluations, multiplying the above bound yields the claimed bound on the number of evaluations. □

We show how to apply Theorem 4.2.1 to obtain novel bounds on the expected optimisation time of the self-adjusting $(1+(\lambda,\lambda))$ GA, including the Jump$_k$ function class.

Theorem 4.2.4. *The expected optimisation time* $\mathrm{E}(T^{\mathrm{eval}})$ *of the self-adjusting* $(1+(\lambda,\lambda))$ GA *with* $F > 1$ *constant is at most*

(a) $\mathrm{E}(T^{\mathrm{eval}}) = O(n^n/|\mathrm{OPT}|)$ *and* $\mathrm{E}(T^{\mathrm{gen}}) = O(n^{n-1}/|\mathrm{OPT}|)$ *on any function with a set* OPT *of global optima,*

(b) $\mathrm{E}(T^{\mathrm{eval}}) = O(dn)$ *and* $\mathrm{E}(T^{\mathrm{gen}}) = O(d+\log n)$ *on unimodal functions with* $d+1$ *fitness values,*

(c) $\mathrm{E}(T^{\mathrm{eval}}) \leq (1+o(1))\cdot 2n^k\,(1-1/n)^{-n+k}$ *and* $\mathrm{E}(T^{\mathrm{gen}}) \leq (1+o(1))\cdot 2n^{k-1}\,(1-1/n)^{-n+k}$ *on* Jump$_k$ *with* $k \geq 3$.

Proof. For the general upper bound, we use a fitness level partition with A_1 containing all non-optimal fitness values and A_2 containing the set OPT, then the number of fitness levels is $m+1=2$. We use the corresponding success probability $s_i \geq |\mathrm{OPT}|/n^n$. With this we bound $\sum_{i=1}^{m}\frac{1}{s_i} = O(n^n/|\mathrm{OPT}|)$. All functions have $d < 2^n$ non-optimal fitness values and $n^n/|\mathrm{OPT}| \geq (n/2)^n$, therefore the term $O(dn)$ can be absorbed.

For unimodal functions, we use a canonical f-based partition and success probabilities of $s_i \geq 1/n \cdot (1-1/n)^{n-1} \geq 1/(en)$ where the last inequality holds by Lemma 2.4.1. This yields $\mathrm{E}(T^{\mathrm{eval}}) \leq O(dn) + 2\sum_{i=1}^{d} en = O(dn)$. For the expected number of generations, we get $\mathrm{E}(T^{\mathrm{gen}}) \leq \lceil 4\log_F(n)\rceil + 6d + \frac{1}{n}\sum_{i=1}^{d} en = O(d+\log n)$.

For Jump$_k$ functions with $k \geq 3$ any individual that is not a local or global optimum can find an improvement by increasing or decreasing the number of 1-bits. This yields success probabilities of at least $(n-i)/(en)$ for all search points with $0 \leq i < n-k$ ones and of at least $i/(en)$ for all search points with $n-k < i < n$ ones. For search points with $n-k$ ones, a standard bit mutation can jump to the optimum by flipping the correct k 0-bits and not flipping any other bit. This has a probability of $s_{n-k} = (1/n)^k\,(1-1/n)^{n-k}$. Hence,

$$\begin{aligned}\mathrm{E}(T^{\mathrm{eval}}) &\leq O(n^2) + 2\sum_{i=0}^{n-k-1}\frac{en}{n-i} + 2\sum_{i=n-k+1}^{n-1}\frac{en}{i} + 2n^k\left(1-\frac{1}{n}\right)^{-n+k} \\ &= O(n^2) + 2n^k\left(1-\frac{1}{n}\right)^{-n+k}.\end{aligned}$$

Given that $k \geq 3$ the second term is $\Omega(n^3)$ and the first term can be absorbed, yielding

$$\mathrm{E}\big(T^{\mathrm{eval}}\big) = (1+o(1)) \cdot 2n^k \left(1 - \frac{1}{n}\right)^{-n+k}. \qquad \square$$

In Theorem 4.2.1 as in most of the theoretical bounds proven in other studies for the $(1+(\lambda,\lambda))$ GA, we focus only on the offspring from the crossover phase. But as mentioned before we argue that considering also the offspring from the mutation phase can improve the performance of the $(1+(\lambda,\lambda))$ GA. To exemplify this we consider the Trap function which is the most difficult problem for the standard $(1+1)$ EA having an expected optimisation time of $\Theta(n^n)$ [76]. In contrast, when considering the offspring from the mutation phase, the self-adjusting $(1+(\lambda,\lambda))$ GA is able to optimise the Trap function using only $O(n)$ evaluations.

Theorem 4.2.5. *Let $F > 1$ be a constant. The expected optimisation time of the self-adjusting $(1+(\lambda,\lambda))$ GA on the Trap function is $O(n)$.*

Proof. We divide the proof in two parts, the OneMax phase, and the *trap* phase. While in the OneMax phase, if the optimum is not found the algorithm behaves as on OneMax. From Theorem 9 in [50], we know that the OneMax phase takes $O(n)$ evaluations.

Once it reaches the local optimum (*trap* phase), within at most $\lceil 4 \log_F(n) \rceil$ unsuccessful generations λ has increased to $\lambda = n$. This then implies $p = 1$ and the algorithm will create the global optimum during the mutation phase with probability 1. Therefore, the *trap* phase is solved in $O(\log n)$ generations with probability 1. During these generations, the algorithm uses $O(n)$ evaluations, as shown in Lemma 4.2.2.

Adding the optimisation times of both phases we have shown that in expectation the self-adjusting $(1+(\lambda,\lambda))$ GA takes $O(n)$ evaluations to find the global optimum on Trap. $\square$

We want to point out that in this example the algorithm benefits from the global optimum being exactly at Hamming distance n from the local optimum. Its purpose is to illustrate the possible benefits of considering the offspring from the mutation phase during selection. This inspired us to consider the strategies in the following sections, where we show more extensively the improvements that considering the offspring from the mutation phase during selection can give to the algorithm.

4.2.2 Crossover does not Benefit the Self-Adjusting $(1+(\lambda,\lambda))$ GA on Jump$_k$

We now show that the bound for the self-adjusting $(1+(\lambda,\lambda))$ GA with standard parameter settings on Jump$_k$ from Theorem 4.2.4 (c) is asymptotically tight. This implies that unless k is very small the self-adjusting $(1+(\lambda,\lambda))$ GA is no more efficient on Jump$_k$ than the $(1+1)$ EA and less efficient than other GAs using crossover [40, 41, 111].

Theorem 4.2.6. *Let $F > 1$ be a constant. The expected optimisation time (in terms of fitness evaluations) of the self-adjusting $(1+(\lambda,\lambda))$ GA on the Jump$_k$ function with $4 \leq k \leq (1-\varepsilon)n/2$, for any constant $\varepsilon > 0$, is at least*

$$(1-o(1)) \cdot 2n^k \left(1 - \frac{1}{n}\right)^{-n+k}.$$

We first show the following upper and lower bounds on the probability that the $(1+(\lambda,\lambda))$ GA finds any particular target search point x^* during one mutation phase. Even though we only need the upper bounds in this section, the lower bounds on transition probabilities will be useful later on.

Lemma 4.2.7. *For every current search point x, every target search point x^* and every current parameter λ, let $p_{\text{mut}}^{\lambda}(x, x^*)$ be the probability that the* $(1 + (\lambda, \lambda))$ GA *creates x^* during the mutation phase of one generation.*

If $x^* = \overline{x}$ and $\lambda = n$, $p_{\text{mut}}^{\lambda}(x, x^*) = 1$.

If $x^* \neq \overline{x}$ and $\lambda = n$, $p_{\text{mut}}^{\lambda}(x, x^*) = 0$.

If $x^* \in \{x, \overline{x}\}$ and $\lambda < n$,

$$p_{\text{mut}}^{\lambda}(x, x^*) = (\lambda/n)^{H(x,x^*)}(1 - \lambda/n)^{n-H(x,x^*)}.$$

Otherwise,

$$\frac{\lfloor\lambda\rfloor}{2}(\lambda/n)^{H(x,x^*)}(1 - \lambda/n)^{n-H(x,x^*)} \leq p_{\text{mut}}^{\lambda}(x, x^*) \leq \lceil\lambda\rceil(\lambda/n)^{H(x,x^*)}(1 - \lambda/n)^{n-H(x,x^*)}.$$

From the transition probabilities, the term $(\lambda/n)^{H(x,x^*)}(1 - \lambda/n)^{n-H(x,x^*)}$ equals the probability of a standard bit mutation with mutation probability λ/n creating x^* from x. If $x^* \notin \{x, \overline{x}\}$, the offspring population of λ amplifies this probability by a factor within $[\lfloor\lambda\rfloor/2, \lceil\lambda\rceil]$. If $x^* \in \{x, \overline{x}\}$, the $(1 + (\lambda, \lambda))$ GA does not benefit from its offspring population at all.

Proof of Lemma 4.2.7. The algorithm needs to sample $\ell = H(x, x^*)$, in order to find x^* during the mutation phase. The probability of this event is

$$\Pr(\ell = H(x, x^*)) = \binom{n}{H(x, x^*)}(\lambda/n)^{H(x,x^*)}(1 - \lambda/n)^{n-H(x,x^*)}. \tag{4.1}$$

In the special case where $\lambda = n$ then the mutation probability $p = 1$ and $\ell = n$, hence if $x = \overline{x}$ the target search point is created with probability 1 and if $x^* \neq \overline{x}$ the target search point is created with probability 0. If $\lambda < n$ and $x^* \in \{x, \overline{x}\}$, $\binom{n}{H(x,x^*)} = 1$ and the claim for this case follows as all λ mutants will create x^* for the right choice of ℓ.

Otherwise, the $(1 + (\lambda, \lambda))$ GA also needs to flip the correct bits during the mutation phase. Since there are $\binom{n}{H(x,x^*)}$ possible ways to flip the bits the probability that one offspring flips the correct bits is $\binom{n}{H(x,x^*)}^{-1}$. This gives us the following probability of finding x^* during $\lfloor\lambda\rceil$ mutations (the algorithm creates $\lfloor\lambda\rceil$ offspring), conditional on $\ell = H(x, x^*)$:

$$\Pr(x^* \mid \ell = H(x, x^*)) = 1 - \left(1 - 1/\binom{n}{H(x, x^*)}\right)^{\lfloor\lambda\rceil}.$$

Using the estimate from Lemma 2.4.2 (a) and $\lfloor\lambda\rceil \leq \lceil\lambda\rceil$ this is bounded from above by

$$\lceil\lambda\rceil/\binom{n}{H(x, x^*)}$$

and bounded from below using Lemma 2.4.2 (b) and $\lfloor\lambda\rceil \geq \lfloor\lambda\rfloor$ by

$$\frac{\lfloor\lambda\rfloor/\binom{n}{H(x,x^*)}}{1 + \lfloor\lambda\rfloor/\binom{n}{H(x,x^*)}} \geq \frac{\lfloor\lambda\rfloor/\binom{n}{H(x,x^*)}}{2},$$

where the last inequality follows from $x^* \notin \{x, \overline{x}\}$ yielding $\binom{n}{H(x,x^*)} \geq n$ and thus $1 + \lfloor\lambda\rfloor/n \leq 2$. Since

$$p_{\text{mut}}^{\lambda}(x, x^*) = \Pr(x^* \mid \ell = H(x, x^*)) \Pr(\ell = H(x, x^*)),$$

multiplying (4.1) with the above bounds on $\Pr(x^* \mid \ell = H(x, x^*))$ and observing that the binomial coefficients cancel completes the proof. □

The following lemma gives an upper bound on the probability of hitting any specific target search point x^* during one crossover phase of the $(1 + (\lambda, \lambda))$ GA. Note that, for the original $(1 + (\lambda, \lambda))$ GA that does not consider mutants for selection, Lemma 4.2.8 gives an upper bound for hitting x^* in one generation.

Lemma 4.2.8. *For every current search point x, every target search point x^* and every current parameter λ, the probability that the $(1 + (\lambda, \lambda))$ GA creates x^* during the crossover phase of one generation is at most*

$$\lfloor \lambda \rceil^2 \left(\frac{1}{n}\right)^{H(x,x^*)} \left(1 - \frac{1}{n}\right)^{n-H(x,x^*)}.$$

Before diving into the proof, we give the main idea here. Recall that in every generation, the $(1 + (\lambda, \lambda))$ GA performs $\lfloor \lambda \rceil$ mutations with a radius of ℓ (drawn from a binomial distribution with parameters λ/n and n) and $\lfloor \lambda \rceil$ crossover operations with the best mutant. All mutants are chosen uniformly at random from the Hamming ball of radius ℓ around x. However, the following selection of the best mutant does not preserve uniformity as some offspring on said Hamming ball may have a higher fitness than others. Hence the crossover operations will affect particular regions of the search space more than others. While this is a helpful algorithmic concept (in a sense that this makes the $(1 + (\lambda, \lambda))$ GA solve OneMax in expected time $O(n)$ [60]), it makes it hard to analyse what search points will be generated during crossover as it depends on the fitness function in hand.

As a solution, we borrow an idea similar to *non-selective family trees* by Witt [180] and Lehre and Yao [129] (see Section 2.4.3) that was also used in previous analyses of the $(1 + (\lambda, \lambda))$ GA [50, Proof of Proposition 1]. We consider a variant of the $(1 + (\lambda, \lambda))$ GA that we call *non-selective* $(1 + (\lambda, \lambda))$ GA: instead of performing $\lfloor \lambda \rceil$ crossovers with the best mutant, it performs $\lfloor \lambda \rceil$ crossovers for *all of the* $\lfloor \lambda \rceil$ *mutants*. This results in $\lfloor \lambda \rceil^2$ offspring generated from crossover, in addition to $\lfloor \lambda \rceil$ mutants. Since the offspring created by the original $(1 + (\lambda, \lambda))$ GA form a subset of the offspring generated by the non-selective $(1 + (\lambda, \lambda))$ GA, the probability of the original $(1 + (\lambda, \lambda))$ GA creating x^* in one crossover phase is bounded by the probability of the non-selective $(1 + (\lambda, \lambda))$ GA creating x^* during crossover. Owing to the absence of selection, the output of the $\lfloor \lambda \rceil^2$ crossover operations is independent of the fitness and we obtain a probability bound that only depends on the Hamming distance $H(x, x^*)$.

Proof of Lemma 4.2.8. We argue similarly as the proof of Proposition 1 in [50]. Fix an offspring y created by the non-selective $(1 + (\lambda, \lambda))$ GA. The process for creating y can be described as follows. The algorithm first picks a random value of ℓ according to a binomial distribution with parameters n and λ/n, and then flips ℓ bits chosen at random to create a mutant x'. The creation of x' can alternatively be regarded as a standard bit mutation with a mutation rate of λ/n. To create y, each bit is independently taken from x' with probability $1/\lambda$. Hence, each bit y_i in y attains the value $1 - x_i$ with probability $1/n$, and it attains value x_i with probability $1 - 1/n$, independently from all other bits. Hence the creation of y can be described as a standard bit mutation with mutation rate $1/n$.

The probability of $y = x^*$ is thus $(1/n)^{H(x,x^*)}(1 - 1/n)^{n-H(x,x^*)}$. Note that different offspring are not independent as they use the same random value of ℓ, and every batch of $\lfloor \lambda \rceil$ crossover operations is derived from the same mutant. Taking a union bound over all $\lfloor \lambda \rceil^2$ offspring allows us to conclude, despite these dependencies, that the probability of one offspring generating x^* is at most

$$\lfloor \lambda \rceil^2 \left(\frac{1}{n}\right)^{H(x,x^*)} \left(1 - \frac{1}{n}\right)^{n-H(x,x^*)}. \qquad \square$$

Now we are in a position to prove Theorem 4.2.6.

Proof of Theorem 4.2.6. By standard Chernoff bounds, the probability that the initial search point has at most $(1+\varepsilon)n/2$ ones is $1-2^{-\Omega(n)}$. We assume this to happen and note that then the algorithm will never accept a search point in the fitness valley of $n-k<i<n$ ones.

Let T^{local} be the random number of generations until any search point in the local optimum with $n-k$ ones or the global optimum is reached for the first time. We bound $\mathrm{E}\big(T^{\text{local}}\big)$ from above as follows[2]. This can be done using Theorem 4.2.1 as in Theorem 4.2.4 (c), but with a non-canonical f-based partition where the best fitness level includes the local optimum and the global optimum. This yields $\mathrm{E}\big(T^{\text{local}}\big)=O(n)$ generations.

By Lemmas 4.2.8 and 4.2.7, before we reach the local optimum a generation with parameter λ reaches the optimum with probability at most

$$\lceil\lambda\rceil(\lambda/n)^k(1-\lambda/n)^{n-k}+\lfloor\lambda\rceil^2\, n^{-k}\leq n(k/n)^k(1-k/n)^{n-k}+n^{2-k}=O(1/n^2):=p,$$

since the current search point has Hamming distance at least k from the optimum and $k\geq 4$. By a union bound the probability that the global optimum is found in t steps is at most tp. Then the probability that the global optimum is found during the first T^{local} steps is at most

$$\sum_{t=0}^{\infty}\Pr\big(T^{\text{local}}=t\big)\cdot tp=p\cdot\mathrm{E}\big(T^{\text{local}}\big)=O(1/n).$$

Now assume that the local optimum has been reached and $\lambda<\lambda_{\max}=n$. Since the optimum is the only search point with a strictly larger fitness, λ will be increased in every generation unless the optimum is found. By Lemma 4.2.2, there are at most $\lceil 4\log_F(n)\rceil$ generations before λ has increased to $\lambda_{\max}$. By the same arguments as above, the probability that the optimum is found during this time is $O((\log n)/n^2)$.

Once $\lambda=\lambda_{\max}=n$ has been reached, mutation always creates search points with k ones (i. e. mutation will never find the optimum) and crossover boils down to a standard bit mutation with mutation rate $1/n$. Then the probability of one crossover creating the optimum is $(1/n)^k(1-1/n)^{n-k}$ and the expected number of crossover operations for hitting the optimum is thus

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

Since every batch of λ crossover operations is preceded by λ (useless) fitness evaluations during mutation, this adds a factor of 2 to the above lower bound. The proof is completed by noting that, after adding up all failure probabilities, this case is reached with probability at least $1-O(1/n)$ and $\mathrm{E}(T)\geq\mathrm{E}(T\mid A)\Pr(A)$, where A is the event that we do not reach the global optimum before reaching the local one. $\square$

4.3 Mending the Success-Based Rules

In the previous section, we have analysed the original self-adjusting $(1+(\lambda,\lambda))$ GA and confirmed the previous empirical observations [74, 82, 99] that the parameter λ can easily diverge to its maximum value deteriorating the performance of the algorithm. In this section we will analyse alternative solutions to this problem. We start analysing the performance of capping the value of λ to a value $\lambda_{\max}$ smaller than n in Section 4.3.1. Afterwards, in Section 4.3.2 we study the parameter landscape of $\lambda_{\max}$ on JUMP$_k$ showing that unlike most

[2]The $(1+(\lambda,\lambda))$ GA optimises ONEMAX in expected time $O(n)$, but it is not immediately obvious how to translate the analysis to the ONEMAX-like parts of JUMP$_k$. Note that the $(1+(\lambda,\lambda))$ GA may overshoot the local optimum and then the analysis on ONEMAX breaks down. We suspect this can be fixed with small modifications, but for now we show a more obvious bound as this is sufficient for our purposes.

previously analysed parameter landscapes, it has a bimodal structure, making parameter selection more difficult. Finally, in Section 4.3.3 we explore the benefits of resetting λ to 1 whenever it reaches its maximum. This approach does not have the same issues with parameter selection as capping λ since it is able to cycle through the parameter space instead of settling down into an specific value that might not be appropriate for the rest of the optimisation.

4.3.1 Restricting the Parameters

In previous sections we have shown that the self-adjusting $(1 + (\lambda, \lambda))$ GA can increase λ to its maximum in only a logarithmic amount of steps. This indicates that the algorithm may increase the parameter in a difficult part of the optimisation and afterwards it could end up with a too high population size affecting the performance. In Algorithm 5 the hyper-parameter $\lambda_{\max}$ prevents such a problem by capping the value of λ to some value $\lambda_{\max} \in [1, n]$, allowing the user to prevent the algorithm from increasing λ towards possible sub-optimal parameters.

Buzdalov and Doerr [22] first suggested this strategy, showing that it benefits the algorithm when optimising instances of the maximum satisfiability problem with weak fitness-distance correlation. Similarly Bassin and Buzdalov [18] showed empirically that capping λ at $\log n$ can improve the performance on linear functions with random weights. Antipov et al. [6] capped λ to $n/2$, arguing that since $p = \lambda/n$ larger values of λ would use mutation probabilities larger than $1/2$ which are considered ill-natured because mutation would create offspring that on average are further away from the parent than the average search point. In this section we explore the benefits of capping λ.

Generic Choice for $\lambda_{\max}$

Since the first time the self-adjusting $(1 + (\lambda, \lambda))$ GA was proposed, the algorithm had a limit on the value of λ of n. This was imposed in order to prevent the mutation probability from exceeding 1. As a secondary effect it gives the algorithm a safety net, allowing it to fall back to the behaviour of the (1+n) EA in case λ reaches its maximum value, as we explored on Section 4.2.

We argue that, if the problem is not known, a good generic strategy is to set $\lambda_{\max} = n/2$. This means that, in a situation where no improvements are found quickly and λ increases, the self-adjusting $(1 + (\lambda, \lambda))$ GA is able to simulate random search during the mutation phase whenever $\lambda_{\max}$ is reached. This is a potential advantage when the algorithm is stuck in a very hard local optimum, or on deceptive functions where random search is a viable technique (e. g. $\text{J}\textsc{ump}_k$ with large k such as $k > n/\log n$). Note that the self-adjusting $(1 + (\lambda, \lambda))$ GA still retains its exploitation capability even with $\lambda = n/2$ because the offspring from the crossover phase would still have on average only 1 bit different from the parent. Hence, the algorithm performs wide exploration and exploitation at the same time, in different phases of the same generation. In addition, the algorithm is still able to optimise OneMax efficiently; the cap on λ only kicks in when regular exploitation fails.

To justify our choice of $\lambda_{\max} = n/2$ we present a general bound for the self-adjusting $(1 + (\lambda, \lambda))$ GA with this parameter choice in Theorem 4.3.1. This theorem shows that $\lambda_{\max} = n/2$ only have a worst-case runtime of $O(dn) + \frac{2^{n+4}}{|\text{OPT}|}$ that applies for most functions as opposed to the worst-case expected runtime of $n^{\Theta(n)}$ for the $(1 + 1)$ EA. Note that this theorem does not include Trap functions which we consider later on.

Theorem 4.3.1. *Let f be any function with d non-optimal fitness values and a set* OPT *of global optima such that either* $|\,\text{OPT}\,| \geq 2$ *or* $\text{OPT} = \{x^*\}$ *and its complement* $\overline{x^*}$ *does not have the second-best fitness value. Then for the self-adjusting* $(1 + (\lambda, \lambda))$ GA *with*

$\lambda_{\max} := n/2$ *and* $F > 1$ *constant we have*

$$\mathrm{E}\big(T^{\mathrm{eval}}\big) \le O(dn) + \frac{2^{n+4}}{|\mathrm{OPT}|};$$
$$\mathrm{E}(T^{\mathrm{gen}}) \le O(d + \log n) + \frac{2^{n+3}}{n|\mathrm{OPT}|}.$$

Proof. By Lemma 4.2.3, the algorithm spends $O(dn)$ evaluations and $O(d + \log n)$ generations in settings with $\lambda < \lambda_{\max}$. Hence we can focus on improvement probabilities when $\lambda = \lambda_{\max}$.

If $|\,\mathrm{OPT}\,| \ge 2$, by Lemma 4.2.7, the probability of one generation hitting any search point in OPT that is not the binary complement of the current search point is at least $\lambda_{\max} \cdot 2^{-n-1} = n \cdot 2^{-n-2}$. Since there are at least $|\,\mathrm{OPT}\,| - 1 \ge |\,\mathrm{OPT}\,|/2$ such search points and the probabilities for hitting these are disjoint events, the probability for finding the optimum is at least $n \cdot 2^{-n-3} \cdot |\,\mathrm{OPT}\,|$. Taking the reciprocal gives an upper bound on $\mathrm{E}(T^{\mathrm{gen}})$, and multiplying by $2\lambda_{\max} = n$ yields a bound on $\mathrm{E}\big(T^{\mathrm{eval}}\big)$.

If $\mathrm{OPT} = \{x^*\}$, we use the same argument to show that within $\frac{2^{n+3}}{n|\mathrm{OPT}|}$ generations we either hit an optimum or a second-best search point. From the latter, the probability of hitting the optimum is bounded in the same way, since by assumption the current search point is different from $\overline{x^*}$. Then we proceed as before. □

Trap Functions

The conditions from Theorem 4.3.1 require a fitness function to either contain at least two global optima, or that the complement of the unique global optimum does not have the second-best fitness. Most fitness functions meet this condition. Notable exceptions are Trap functions (defined in Section 2.3) as there the local optimum 1^n with the second-best fitness is precisely the complement of the unique global optimum 0^n. We show that Trap functions were excluded from Theorem 4.3.1 for a good reason since for Trap functions the runtime is higher than the bound from Theorem 4.3.1 by a factor of order $\Omega(n)$.

Theorem 4.3.2. *Let* $F > 1$ *be a constant and* $\lambda_{\max} = n/2$*. The expected optimisation time (in terms of fitness evaluations) of the self-adjusting* $(1 + (\lambda, \lambda))$ *GA on the* Trap *function is* $\Omega(n2^n)$*.*

Proof. We divide the proof in three parts, the *initialisation* phase, the OneMax phase and the *trap* phase.

The expected number of zero bits in the initial individual is $n/2$. By applying Chernoff bounds we can see that, w. o. p. the initial individual has at least $n/3$ one bits. Therefore, with at least the same probability the algorithm does not find the optimum during the *initialisation* phase.

From Theorem 9 in [50], we know that the OneMax phase takes in expectation $\mathrm{E}\big(T^{\textsc{OneMax}}\big) = O(n)$ generations.[3] We now prove that during these generations the algorithm does not find the optimum w. o. p.

By Lemma 4.2.7 the probability to find the optimum during the mutation phase at a Hamming distance of at least $n/3$ is:

$$\Pr(x' = x^*) \le \lceil\lambda\rceil \left(\frac{\lambda}{n}\right)^{n/3} \left(1 - \frac{\lambda}{n}\right)^{2n/3} \le n \cdot 3^{-\frac{n}{3}} \left(\frac{3}{2}\right)^{-\frac{2n}{3}}. \tag{4.2}$$

[3]As long as $\lambda_{\max}$ is bigger than $C_0\sqrt{n/(n - \textsc{OneMax}(x))}$ for a constant C_0 large enough to satisfy Lemma 16 in [50] the bound of $O(n)$ from Theorem 9 in [50] holds.

The second inequality comes from bounding $\lceil\lambda\rceil \leq n$ and using the parameter $\lambda = n/3$ everywhere else, which maximises the term $\left(\frac{\lambda}{n}\right)^{n/3}\left(1-\frac{\lambda}{n}\right)^{2n/3}$ [62, Corollary 2]. For the crossover phase, to find the global optimum we use Lemma 4.2.8, obtaining,

$$\Pr\left(y = x^*\right) \leq \lfloor\lambda\rceil^2 \left(\frac{1}{n}\right)^{n/3}\left(1-\frac{1}{n}\right)^{2n/3} \leq n^{-\frac{n}{3}+2}\cdot e^{-2/3}. \tag{4.3}$$

Adding Equation (4.2) and (4.3) we get a probability of finding the optimum in one iteration of $p = O(n\cdot 3^{-n}\cdot 2^{2n/3})$. Then the probability to find the optimum during T^{OneMax} iterations is at most

$$\sum_{t=0}^{\infty} \Pr\left(T^{\text{OneMax}} = t\right)\cdot tp = p\cdot \mathrm{E}\left(T^{\text{OneMax}}\right) = O(n^2\cdot 3^{-n}\cdot 2^{2n/3}).$$

Finally, during the *trap* phase since $x = \overline{x^*}$ from Lemma 4.2.7 we get

$$\Pr\left(y = x^*\right) = \left(\frac{\lambda}{n}\right)^n \leq 2^{-n}.$$

We know from Lemma 4.2.2 that the algorithm spends at most $\lceil 4\log_F(n/2)\rceil$ generations to get to the parameter $\lambda = n/2$. The probability of finding the optimum during these iterations is $1-(1-2^{-n})^{\lceil 4\log_F(n/2)\rceil} \leq 2^{2-n}\log_F n$. Once the maximum parameter value has been reached, the probability of sampling $\ell = n$ and thus find the optimum is $\Pr\left(y = x^*\right) = 2^{-n}$. Hence, in expectation 2^n further iterations are needed to find the optimum and each one of these iterations will use n evaluations. Since this situation is reached w. o. p., the expected number of evaluations is $\Omega(n\cdot 2^n)$. □

Jump Functions

In this section we explore in detail how capping λ affects the performance of the algorithm on Jump_k functions. In the following theorem, we only consider the mutation phase and assume for simplicity to start in the local optimum.

Theorem 4.3.3. *After reaching the local optimum, the expected number of function evaluations for the self-adjusting* $(1+(\lambda,\lambda))$ GA *with* $F > 1$ *constant and* λ *capped at* $\lambda_{\max} < n$ *is at most*

$$O(\lambda_{\max}) + 4\left(\frac{n}{\lambda_{\max}}\right)^k\left(1-\frac{\lambda_{\max}}{n}\right)^{-n+k}.$$

Proof. By Lemma 4.2.2, λ reaches $\lambda_{\max}$ or the algorithm finds the global optimum within $O(\lambda_{\max})$ evaluations. Then the probability of jumping to the optimum in one generation is at least $\lambda_{\max}/2\cdot(\lambda_{\max}/n)^k(1-\lambda_{\max}/n)^{n-k}$ by Lemma 4.2.7. Taking the reciprocal and multiplying by $2\lambda_{\max}$ yields the claim. □

Note that for $\lambda_{\max} := k$, Theorem 4.3.3 yields an upper bound of

$$O(k) + 4\left(\frac{n}{k}\right)^k\left(1-\frac{k}{n}\right)^{-n+k}.$$

For $k \geq 3$, these bounds match the expected time for the $(1+1)$ EA with the optimal mutation rate of k/n up to constant factors [62]. However, we would need to know k in advance, which defies the goal of parameter control. As mentioned before, an alternative strategy is to set $\lambda_{\max} := n/2$.

Theorem 4.3.4. *Let $k < \frac{n}{4}$. After reaching the local optimum, the expected number of function evaluations for the self-adjusting* $(1 + (\lambda, \lambda))$ GA *with $F > 1$ constant and $\lambda_{\max} := n/2$ on* JUMP$_k$ *is*

$$\mathrm{E}\big(T^{\text{eval}}\big) \leq \begin{cases} O\left(\left(\frac{n}{2}\right)^k\right) & \text{if } k \leq \frac{\log(n/2)}{2}, \\ O\left(\min\{(2n)^{k-1}, 2^n\}\right) & \text{otherwise.} \end{cases}$$

$$\mathrm{E}\big(T^{\text{gen}}\big) \leq \begin{cases} O\left(\left(\frac{n}{2}\right)^{k-1}\right) & \text{if } k \leq \frac{\log(n/2)}{2}, \\ O\left(\min\{(2n)^{k-2}, 2^n/n\}\right) & \text{otherwise.} \end{cases}$$

Note that these upper bounds prove that the self-adjusting $(1 + (\lambda, \lambda))$ GA with $\lambda_{\max} := n/2$ is faster than the $(1 + 1)$ EA with the default mutation rate $1/n$ for all $k \leq \log n$ and $k > n/\log n$ as the latter needs expected time $\Theta(n^k)$ [76].

Before proving Theorem 4.3.4 we need to understand how the crossover phase might help the optimisation process. For the crossover phase to be able to find the optimum, the selected offspring from the mutation phase must flip all 0-bits from the parent. Afterwards the crossover phase is able to repair such offspring to find the global optimum. This was studied by Antipov et al. [9] for any static parameter choice.

Theorem 4.3.5 (Theorem 3.3 in [9])**.** *Let $k \leq \frac{n}{4}$. Assume that $p \geq \frac{2k}{n}$ and $q_\ell \geq 0.1$ is the probability that the number of bits flipped $\ell \in [pn, 2pn]$. After reaching the local optimum, the expected runtime of the* $(1 + (\lambda, \lambda))$ GA *with static parameters on* JUMP$_k$ *is*

$$\mathrm{E}\big(T^{\text{eval}}\big) \leq \frac{8\lambda}{q_\ell \min\{1, \lambda \left(\frac{p}{2}\right)^k\} \min\{1, \lambda c^k (1-c)^{2pn-k}\}};$$

$$\mathrm{E}\big(T^{\text{gen}}\big) \leq \frac{4}{q_\ell \min\{1, \lambda \left(\frac{p}{2}\right)^k\} \min\{1, \lambda c^k (1-c)^{2pn-k}\}}.$$

An extension of this theorem to the self-adjusting $(1 + (\lambda, \lambda))$ GA can be easily shown as follows.

Lemma 4.3.6. *Let $k \leq \frac{n}{4}$. After reaching the local optimum the expected runtime of the self-adjusting* $(1 + (\lambda, \lambda))$ GA *with $p = \lambda/n$, $c = 1/\lambda$, $\lambda_{\max} \geq 2k$ and $F > 1$ constant on* JUMP$_k$ *is*

$$\mathrm{E}\big(T^{\text{eval}}\big) \leq O(n) + \frac{8}{q_\ell \min\left\{1, \lambda_{\max} \left(\frac{\lambda_{\max}}{2n}\right)^k\right\} \left(\frac{1}{\lambda_{\max}}\right)^k \left(1 - \frac{1}{\lambda_{\max}}\right)^{2\lambda_{\max} - k}}.$$

$$\mathrm{E}(T^{\text{gen}}) \leq O(\log n) + \frac{4}{q_\ell \min\left\{1, \lambda_{\max} \left(\frac{\lambda_{\max}}{2n}\right)^k\right\} \left(\frac{1}{\lambda_{\max}}\right)^{k-1} \left(1 - \frac{1}{\lambda_{\max}}\right)^{2\lambda_{\max} - k}}$$

Proof. By Lemma 4.2.2, λ reaches $\lambda_{\max}$ or the algorithm finds the global optimum within $O(\log n)$ generations and $O(n)$ evaluations. Then the algorithm will always use $\lambda = \lambda_{\max}$, therefore the probability of jumping to the optimum during the crossover phase is given by Theorem 4.3.5. □

We now have the necessary tools to prove Theorem 4.3.4.

Proof of Theorem 4.3.4. For the proof we use Lemma 4.3.6 as follows:

$$\mathrm{E}\big(T^{\text{eval}}\big) \leq O(n) + \frac{8}{q_\ell \min\{1, \frac{n}{2} \left(\frac{1}{4}\right)^k\} (\frac{2}{n})^k (1 - \frac{2}{n})^{n-k}}.$$

Note that $k \le \frac{\log(n/2)}{2}$ is equivalent to $\frac{n}{2}\left(\frac{1}{4}\right)^k \ge 1$ and this implies that $\min\{1, \frac{n}{2}\left(\frac{1}{4}\right)^k\} = 1$. Hence,

$$\mathrm{E}(T^{\mathrm{eval}}) \le O(n) + \frac{8}{q_\ell(\frac{2}{n})^k(1-\frac{2}{n})^{n-k}} = O\left(\left(\frac{n}{2}\right)^k\right).$$

If $\frac{\log(n/2)}{2} < k$ then $\min\{1, \frac{n}{2}\left(\frac{1}{4}\right)^k\} = \frac{n}{2}\left(\frac{1}{4}\right)^k$ and

$$\mathrm{E}(T^{\mathrm{eval}}) \le O(n) + \frac{8}{q_\ell \frac{n}{2}\left(\frac{1}{4}\right)^k(\frac{2}{n})^k(1-\frac{2}{n})^{n-k}} \le O(n) + \frac{32}{q_\ell(\frac{1}{2n})^{k-1}(1-\frac{2}{n})^{n-k}} = O\left((2n)^{k-1}\right).$$

Following the same arguments we obtain the bound for the number of generations. In addition, we also consider the upper bounds from Theorem 4.3.1, obtaining $\mathrm{E}(T^{\mathrm{eval}}) = O(2^n)$ and $\mathrm{E}(T^{\mathrm{gen}}) = O(2^n/n)$. □

4.3.2 Parameter Landscape on Jump$_k$

In Section 4.3.1 it was shown that choosing $\lambda_{\max} = k$ gives an expected runtime only a constant factor worse than the $(1+1)$ EA with optimal mutation probability of k/n. However, this runtime guarantee only takes into account the mutation phase of the algorithm. Here we analyse the parameter landscape of the self-adjusting $(1+(\lambda,\lambda))$ GA with $p = \lambda/n$, $c = 1/\lambda$ and $\lambda_{\max} \in [1, n]$. To analyse the effects of the parameter $\lambda_{\max}$ we use the precise expression of the probability $s_k^{\lambda_{\max}}$ of the $(1+(\lambda,\lambda))$ GA with $\lambda = \lambda_{\max}$ to jump from the local optimum to the global optimum in one iteration. Hereby we ignore the initial time to climb up to the local optimum and the time for λ to increase to $\lambda_{\max}$. Our previous analyses have shown that these times are negligible anyway. Antipov and Doerr [3] computed the probability of this event but only considered the offspring from the crossover phase for selection. The probability of jumping from the local optimum to the global optimum is

$$s_k^{\lambda_{\max}} = \sum_{j=0}^{n} \Pr(\ell = j \mid \lambda = \lambda_{\max}) \cdot \Pr(y = x^* \mid \ell = j \wedge \lambda = \lambda_{\max}).$$

Since $\ell \sim \mathcal{B}(n,p)$ the probability of sampling $\ell = j$ is given by

$$\Pr(\ell = j \mid \lambda = \lambda_{\max}) = \binom{n}{j}\left(\frac{\lambda_{\max}}{n}\right)^j\left(1-\frac{\lambda_{\max}}{n}\right)^{n-j}.$$

The probability of finding the optimum depends on the number of bits flipped, ℓ. For the algorithm to find the optimum, ℓ needs to be at least the size of the jump k in order to be able to flip all the 0-bits, meaning that, for any $\ell < k$, $\Pr(y = x^* \mid \ell = j \wedge \lambda = \lambda_{\max}) = 0$. When $\ell = k$, if the mutation phase flips all the 0-bits, the optimum is found and the crossover phase is not needed. This has a probability of

$$\Pr(y = x^* \mid \ell = k \wedge \lambda = \lambda_{\max}) = 1 - \left(1 - \frac{1}{\binom{n}{k}}\right)^{\lambda_{\max}}.$$

Following Antipov and Doerr [3], for $\ell > k$ the global optimum can only be found during the crossover phase because the mutation phase flips more than k bits. In order for the crossover to be able to find the optimum, first the algorithm needs to select a *good offspring* during the mutation phase, that is, an offspring from the set of all possible offspring that flip all the k 0-bits from the parent that we call $X^{(*)}$. Hence, we can calculate the probability of finding the optimum during the crossover phase by taking the probability of selecting an offspring during the mutation phase in $X^{(*)}$ and multiplying it by the conditional probability of finding the optimum given that $x' \in X^{(*)}$. That is, $\Pr(y = x^* \mid \ell = j \wedge \lambda = \lambda_{\max})$ is equivalent to

$$\Pr\left(x' \in X^{(*)} \mid \ell = j \wedge \lambda = \lambda_{\max}\right) \cdot \Pr\left(y = x^* \mid x' \in X^{(*)} \wedge \ell = j \wedge \lambda = \lambda_{\max}\right).$$

The probability of the mutation operator flipping all the 0-bits from the parent is $\binom{n-k}{\ell-k}/\binom{n}{\ell}$. If $\ell \in [k+1, 2k-1]$ any offspring from the mutation phase that flips all 0-bits has more than $n-k$ ones and hence falls into the valley of search points with a fitness less than k. This is worse than any offspring that does not flip all 0-bits. Therefore, in order to select an offspring with all 0-bits flipped the algorithm needs to create all offspring with all k 0-bits flipped, that is,

$$\Pr\left(x' \in X^{(*)} \mid \ell = j \wedge \lambda = \lambda_{\max}\right) = \left(\frac{\binom{n-k}{j-k}}{\binom{n}{j}}\right)^{\lambda_{\max}}.$$

In contrast, for $\ell \geq 2k$ the algorithm always selects an offspring with all k 0-bits flipped if one is created. This is because the algorithm flips at least k 1-bits in all mutants, hence the number of 1-bits in each mutant is at most $n-k$, making an offspring with the k 0-bits flipped have the greatest fitness value. Therefore, the probability of this event is

$$\Pr\left(x' \in X^{(*)} \mid \ell = j \wedge \lambda = \lambda_{\max}\right) = 1 - \left(1 - \frac{\binom{n-k}{j-k}}{\binom{n}{j}}\right)^{\lambda_{\max}}.$$

After selecting an offspring $x' \in X^{(*)}$ during the mutation phase, to generate the global optimum in the crossover phase the algorithm needs to take the k bits which are zero in x from x' and take all the $\ell - k$ bits which are zero in x' from x. This has a probability of $c^k(1-c)^{\ell-k}$ for one offspring, and it needs to happen for at least one of the λ offspring. This yields

$$\Pr\left(y = x^* \mid x' \in X^{(*)} \wedge \ell = j \wedge \lambda = \lambda_{\max}\right) = 1 - \left(1 - \left(\frac{1}{\lambda_{\max}}\right)^k \left(1 - \frac{1}{\lambda_{\max}}\right)^{j-k}\right)^{\lambda_{\max}}.$$

Putting all together and using $\lambda' := \lambda_{\max}$ to improve readability we have shown the following result.

Theorem 4.3.7. *Consider the self-adjusting* $(1 + (\lambda, \lambda))$ GA *capping* λ *at* $\lambda' = \lambda_{\max}$ *on* JUMP_k. *When the algorithm has reached the local optimum and* $\lambda = \lambda'$, *the probability of creating the optimum in one generation is*

$$\begin{aligned} s_k^{\lambda'} = &\binom{n}{k}\left(\frac{\lambda'}{n}\right)^k \left(1 - \frac{\lambda'}{n}\right)^{n-k} \left(1 - \left(1 - \frac{1}{\binom{n}{k}}\right)^{\lambda'}\right) \\ &+ \sum_{j=k+1}^{2k-1} \left(\binom{n}{j}\left(\frac{\lambda'}{n}\right)^j \left(1 - \frac{\lambda'}{n}\right)^{n-j} \left(\frac{\binom{n-k}{j-k}}{\binom{n}{j}}\right)^{\lambda'} \cdot \left(1 - \left(1 - \left(\frac{1}{\lambda'}\right)^k \left(1 - \frac{1}{\lambda'}\right)^{j-k}\right)^{\lambda'}\right)\right) \\ &+ \sum_{j=2k}^{n} \left(\binom{n}{j}\left(\frac{\lambda'}{n}\right)^j \left(1 - \frac{\lambda'}{n}\right)^{n-j} \left(1 - \left(1 - \frac{\binom{n-k}{j-k}}{\binom{n}{j}}\right)^{\lambda'}\right) \cdot \left(1 - \left(1 - \left(\frac{1}{\lambda'}\right)^k \left(1 - \frac{1}{\lambda'}\right)^{j-k}\right)^{\lambda'}\right)\right) \end{aligned}$$

and the expected number of fitness evaluations to find the global optimum is $2\lambda_{\max}/s_k^{\lambda_{\max}}$.

Note that the first line is precisely the probability $p_{\text{mut}}^{\lambda'}(x, x^*)$ that the optimum is found during the mutation phase. We have already bounded this in Lemma 4.2.7 from above and below. For $k \geq 2$, the upper bound is tighter, hence the first line is close to $\lambda'(\lambda'/n)^k(1 -$

$\lambda'/n)^{n-k}$. The term $(\lambda'/n)^k(1-\lambda'/n)^{n-k}$ (without the leading factor λ') is maximised for $\lambda' := k$ [62, Corollary 2]. Hence the first line attains its maximum around $\lambda' = k$.

In the first summation, the term $\left(\frac{\binom{n-k}{j-k}}{\binom{n}{j}}\right)^{\lambda'}$ simplifies to $\left(\frac{j(j-1)\cdot\ldots\cdot(j-k+1)}{n(n-1)\cdot\ldots\cdot(n-k+1)}\right)^{\lambda'}$, which is bounded from above by $(2k/(n-k))^{k\lambda'}$. For $k = o(n)$, these summands are all negligibly small, compared to the other terms from Theorem 4.3.7.

The last summation was bounded from below by Antipov et al. [9] for all $\lambda \geq 2k$ as:

$$\frac{1}{40}\min\left\{1, \lambda\left(\frac{\lambda}{2n}\right)^k\right\}\lambda\left(\frac{1}{\lambda}\right)^k\left(1-\frac{1}{\lambda}\right)^{2\lambda-k}.$$

Since $\left(1-\frac{1}{\lambda}\right)^{2\lambda-k} = \Theta(1)$ for $\lambda \geq 2k$ and it converges to e^{-2} as λ grows, we ignore this factor, and the constant $1/40$, for the time being, and focus on the term

$$\min\left\{1, \lambda\left(\frac{\lambda}{2n}\right)^k\right\}\lambda\left(\frac{1}{\lambda}\right)^k = \min\left\{\lambda\left(\frac{1}{\lambda}\right)^k, \lambda^2\left(\frac{1}{2n}\right)^k\right\}$$

instead. For $\lambda\left(\frac{\lambda}{2n}\right)^k \leq 1$ the minimum simplifies to $\lambda^2\left(\frac{1}{2n}\right)^k$, which is non-decreasing in λ. For larger λ, the minimum simplifies to $\lambda\left(\frac{1}{\lambda}\right)^k$, which is non-increasing in λ. Hence, the above expression is maximised for $\lambda\left(\frac{\lambda}{2n}\right)^k = 1$ or, equivalently, $\lambda = (2n)^{k/(k+1)}$.

This suggests that the probability of finding the global optimum from the local optimum, $s_k^{\lambda_{\max}}$, as stated in Theorem 4.3.7 is maximal around $\lambda_{\max} = k$ and around $\lambda_{\max} = (2n)^{k/(k+1)}$. Since the expected time to find the optimum, $2\lambda_{\max}/s_k^{\lambda_{\max}}$, is proportional to the reciprocal of the improvement probability (with an additional factor of $2\lambda_{\max}$), this would imply that the expected optimisation time is locally minimal around these values. Note that, while Theorem 4.3.7 is rigorous, the above discussion about the possible location of minima is only semi-rigorous and partly based on upper and lower bounds of probabilities. Therefore, to analyse the parameter landscape we compute $2\lambda_{\max}/s_k^{\lambda_{\max}}$ exactly for $n \in \{125, 250, 500\}$, $k \in \{3, 4, 5, 6\}$ and $\lambda \in \{1, \ldots, n\}$ using the exact formula from Theorem 4.3.7. Figure 4.1 shows these computations.

As seen in Figure 4.1, the parameter landscape is a bimodal function for $k \geq 4$ with one minimum at or around $\lambda_{\max} = k$, as predicted by the discussion above. There is another local optimum for much larger values of $\lambda_{\max}$. Our above semi-rigorous arguments predicted a minimum around $\lambda_{\max} = (2n)^{k/(k+1)}$. The real minimum is attained for slightly smaller values of $\lambda_{\max}$. Recall that our prediction was based on a bound of the improvement probability, and we suspect that the bound is not precise enough to predict the exact location of the minimum.

The two local optima are surrounded by a basin of attraction, one narrow and one wide, that changes according to the relation between n and k. For small values of k the wider basin of attraction contains the optimal parameter value and if k is small enough ($k \leq 3$) both basin of attractions merge, transforming the parameter landscape into a unimodal function. For larger k values the narrower basin of attraction contains the optimal parameter and it gets narrower for bigger problem sizes. We note that in this case the optimal parameter maximises the probability of finding the optimum during the mutation phase, hence we obtain much better performance when considering the offspring from both phases versus only the crossover phase. This fluctuating parameter landscape shows that it can be hard to predict the correct parameter values to use for a given problem, especially if the problem is not well understood.

Additionally, since the parameter landscape is bimodal, if gradient descent was used for parameter tuning, it would easily get stuck far away from the optimal parameter value. This is particularly true for values of $k = o(n)$ large enough for $\lambda = k$ to be the best parameter

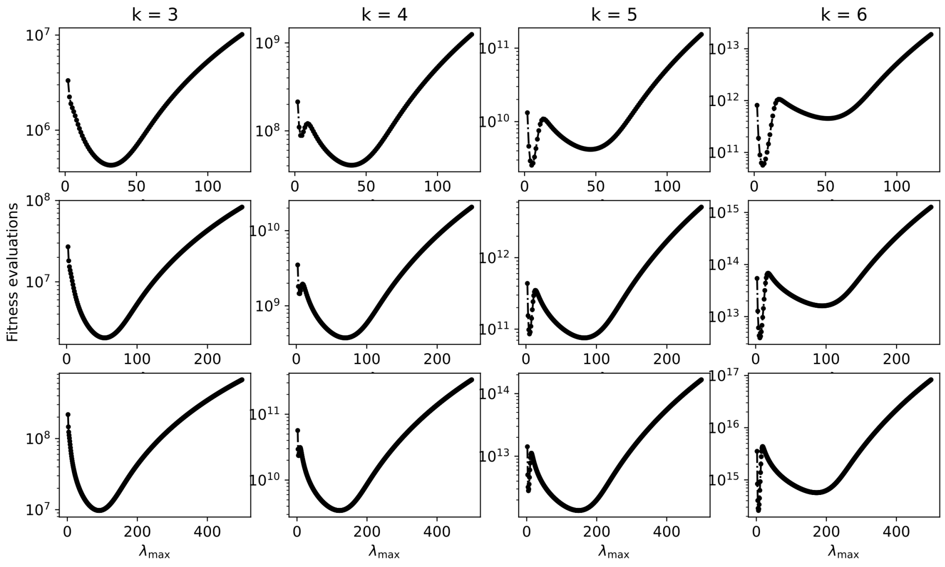

Figure 4.1: $\mathrm{E}\big(T^{\mathrm{eval}}\big)$ of the self-adjusting $(1+(\lambda,\lambda))$ GA on $\textsc{Jump}_k$ with $n \in \{125, 250, 500\}$ and $k \in \{3,4,5,6\}$ varying $\lambda_{\max}$.

(as seen in Figure 4.1 when $k = 6$) because we believe the basin of attraction around $\lambda = k$ has a size of $\Theta(k)$, while the other basin of attraction seems to have a size of $\Theta(n)$. Hence it seems plausible that gradient descent would start in the wrong basin of attraction and only be able to find a locally optimal parameter.

Finally, to complement our runtime predictions, we executed experiments varying $\lambda_{\max}$ with $n = 60$ and $k = 4$ and compared them against the computed $\mathrm{E}\big(T^{\mathrm{eval}}\big)$. The results of the experiments are shown in Figure 4.2. We note that although the average function evaluations follow closely our predictions, the standard deviation was in the same order of magnitude as the mean, meaning that there was a large variation from run to run.

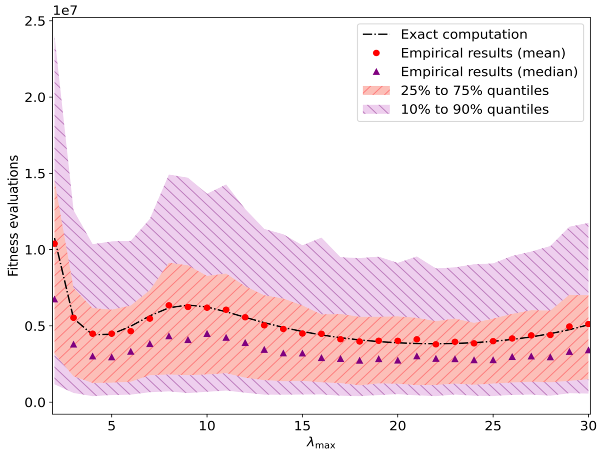

Figure 4.2: $\mathrm{E}\big(T^{\mathrm{eval}}\big)$, mean and median number of evaluations and 10%, 25%, 75%, 90% quantiles of the self-adjusting $(1+(\lambda,\lambda))$ GA on $\textsc{Jump}_k$ with $n = 60$ and $k = 4$ varying $\lambda_{\max}$ over 1000 runs.

4.3.3 Resetting the Parameters

Any generic choice of a maximum $\lambda_{\max}$ bears the risk that the self-adjusting $(1+(\lambda,\lambda))$ GA might get stuck with sub-optimal parameters. A solution to avoid this is to reset λ to 1 if $\lambda = \lambda_{\max}$ and there is another unsuccessful generation. This makes the algorithm cycle through the parameter space in unsuccessful generations. A similar modification was made in [91], where the authors restart the parameter to $\lambda = 1$, but also restart the search from a random individual. We do not restart the search because for functions like JUMP$_k$, restarts would run into the same set of local optima w.o.p. In [18], the authors reset λ, but instead of resetting to 1, they reset to the last successful parameter. We argue that, if the next step of the optimisation needs a lower value of λ than the used in the last successful generation, since λ only increases in unsuccessful generations the algorithm will never use the correct parameter.

4.3.4 Self-Adjusting (1+(λ, λ) GA Resetting λ

In the following, we will analyse the simple strategy of resetting λ to 1 after an unsuccessful generation at $\lambda_{\max} = n$. This strategy takes advantage of two different behaviours. When hill-climbing, the algorithm uses self-adjustment to regulate λ and maintain its value in a *good* parameter range. Because of this, its optimisation time is not affected for problems like ONEMAX. However, when the algorithm encounters a local optimum, its behaviour is similar to the dynamic $(1+1)$ EA [112] that cycles through different parameter regions, like $\lambda \sim n/2$, $\lambda = n$. This helps the algorithm to simulate random search and the (1+n) EA in one cycle. Furthermore, it is not affected by the modality of the parameter landscape. In addition, during every generation the crossover phase is still focusing on exploitation, generating offspring concentrated around the parent.

This strategy is similar to the stagnation detection described in [161, 162] in the sense that when $\lambda = n$ the algorithm is likely to be in a local optimum and the behaviour of the algorithm changes to explore different parameters. Once $\lambda = n$, different approaches could be used, such as sampling λ from a power-law distribution as in [8] or sampling λ from a distribution where $\lambda = i$ with probability $1/(i \ln(n))$, obtaining similar results as the cycling behaviour. If a large jump is expected we could even use the non-standard parameters from [3] and come back to the self-adjusting behaviour once an improvement is found. Although these can be viable solutions we acknowledge that the parameter λ could increase to n without being in a local optima, for example while optimising a function with low fitness-distance correlation or by choosing F as a large constant as explored in [50] where the authors warn that λ can diverge even on simple problems if $F \geq 2.25$. Because of this we decide to study the algorithm without changing its behaviour drastically.

General Method

We show that the fitness-level method can be applied here as well. In contrast to Theorem 4.2.1, here improvement probabilities refer to the transitions of the $(1+(\lambda,\lambda))$ GA, and we consider improvement probabilities across a whole cycle of parameter values.

Theorem 4.3.8. *Given a canonical f-based partition $A_1, \dots, A_{m+1}$, and s_i^{cycle} a lower bound on the probability of finding an improvement on level i during a cycle. Then for the self-adjusting* $(1+(\lambda,\lambda))$ GA *resetting λ to 1, using $F > 1$ which may depend on n, we have*

$$\mathrm{E}(T^{\text{eval}}) \leq O\left(\frac{F^{1/4} n}{F^{1/4}-1}\right) \sum_{i=1}^{m} \frac{1}{s_i^{\text{cycle}}};$$

$$\mathrm{E}(T^{\text{gen}}) \leq O(\lceil \log_F(n) \rceil) \sum_{i=1}^{m} \frac{1}{s_i^{\text{cycle}}}.$$

Algorithm 6: The self-adjusting (1 + (λ, λ)) GA resetting λ.

```
1  Initialization: Sample $x \in \{0,1\}^n$ uniformly at random;
2  Initialize $\lambda \leftarrow 1$, $p \leftarrow \lambda/n$, $c \leftarrow 1/\lambda$;
3  Optimization: for $t = 1, 2, \ldots$ do
4      Sample $\ell$ from $\mathcal{B}(n,p)$;
       Mutation phase:
5      for $i = 1, \ldots, \lfloor\lambda\rceil$ do
6          Sample $x^{(i)} \leftarrow \text{flip}_\ell(x)$ and query $f(x^{(i)})$;
7      Choose $x' \in \{x^{(1)}, \ldots, x^{(\lambda)}\}$ with $f(x') = \max\{f(x^{(1)}), \ldots, f(x^{(\lambda)})\}$ u.a.r.;
       Crossover phase:
8      for $i = 1, \ldots, \lfloor\lambda\rceil$ do
9          Sample $y^{(i)} \leftarrow \text{cross}_c(x, x')$ and query $f(y^{(i)})$;
10     If $\{x', y^{(1)}, \ldots, y^{(\lambda)}\}\backslash\{x\} \neq \emptyset$, choose $y \in \{x', y^{(1)}, \ldots, y^{(\lambda)}\}\backslash\{x\}$ with
       $f(y) = \max\{f(x'), f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ u.a.r.;
11     otherwise, set $y := x$;
       Selection and update step:
12     if $f(y) \geq f(x)$ then $x \leftarrow y$;
13     if $f(y) > f(x)$ then $\lambda \leftarrow \max\{\lambda/F, 1\}$;
14     if $f(y) \leq f(x) \wedge \lambda = n$ then $\lambda \leftarrow 1$;
15     if $f(y) \leq f(x) \wedge \lambda \neq n$ then $\lambda \leftarrow \min\{\lambda F^{1/4}, n\}$;
```

Proof. Since the f-based partition is canonical, the current fitness level is left as soon as we encounter a successful generation. Until this happens, the self-adjusting $(1 + (\lambda, \lambda))$ GA cycles through all parameters for λ. We use Lemma 4.2.2 to show that for every time the algorithm cycles through all the parameters once, it uses $O(\lceil\log_F(n)\rceil)$ generations and $O\left(\frac{F^{1/4}n}{F^{1/4}-1}\right)$ evaluations. The probability of leaving the current fitness level during a cycle is at least s_i^{cycle} by assumption. Hence the expected number of cycles is at most $1/s_i^{\text{cycle}}$. Together, this proves the claimed bounds. □

Upper Bounds on Jump

To showcase how this bound can be used we show an upper bound for JUMP_k. Note that, when applying Theorem 4.3.8, we may bound s_i^{cycle} from below by only considering the best parameter settings the algorithm can use for the current fitness level, making it easy to use. In our case we focus on values of λ close to k since in Section 4.3.2 we showed that for large k it gives the best performance. In addition, we also consider $\lambda = n$ because for small k and steps outside the local optimum this gives a better runtime, decreasing the complexity of the proof.

Theorem 4.3.9. *Let $F = F(n) > 1$ and $k \geq 2$. Then for the self-adjusting* $(1 + (\lambda, \lambda))$ GA *resetting λ to 1 on* JUMP *we have*

$$\mathrm{E}(T^{\text{eval}}) = \min\left\{O\left(\frac{F^{1/4}n^k}{F^{1/4}-1}\right), O\left(\left(\frac{F^{(k+2)/4}}{F^{1/4}-1}\right)\left(\frac{en}{k}\right)^{k+1}\right)\right\};$$
$$\mathrm{E}(T^{\text{gen}}) = \min\left\{O\left(n^{k-1}\log n\right), O\left(\left(\frac{enF^{1/4}}{k}\right)^{k+1}\left(\frac{\log n}{n}\right)\right)\right\}.$$

Proof. For the fitness levels $A_1 \ldots A_{m-1}$ any individual can leave the current fitness level by increasing or decreasing the number of 1-bits. For these fitness levels we bound s_i^{cycle} by only considering generations with $\lambda = n$, therefore similar to Theorem 4.2.1 $s_i^{\text{cycle}} \geq \frac{s_i n}{1+s_i n}$, where s_i is a lower bound for the probability that the $(1+1)$ EA with $p = 1/n$ creates an offspring in $A_{i+1} \dots A_{m+1}$ from a search point in A_i. Using the crude estimate $s_i \geq 1/(en)$, gives us an expected number of generations of $1/s_i^{\text{cycle}} \leq e + 1$ to leave any of these fitness levels.

For the local optimum in fitness level A_m, we use $s_m^{\text{cycle}} \geq \max\{s_m^{k^*}, s_m^n\}$ with $s_m^{k^*}$ and s_m^n being the probability of leaving A_m in one generation with $\lambda \in \left[\frac{k}{F^{1/4}}, k\right]$ and $\lambda = n$, respectively, and bound them separately. To compute s_m^n, as before, we use the probability that the $(1+1)$ EA with $p = 1/n$ finds an improvement, that is, $s_m = (1/n)^k (1-1/n)^{n-k} \geq 1/(en^k)$ to obtain

$$s_m^n \geq \frac{s_m n}{1 + s_m n} \geq \frac{1/(en^{k-1})}{1 + 1/(en^{k-1})} \geq \frac{1}{en^{k-1} + 1}.$$

For $s_m^{k^*}$ we use Lemma 4.2.7 and the range $\lambda \in \left[\frac{k}{F^{1/4}}, k\right]$ as follows,

$$\begin{aligned} s_m^{k^*} &\geq \frac{\lfloor \lambda \rfloor}{2} (\lambda/n)^k (1 - \lambda/n)^{n-k} \\ &\geq \frac{\lambda}{4} (\lambda/n)^k (1 - \lambda/n)^{n-k} \\ &\geq \frac{k}{4F^{1/4}} \left(\frac{k}{F^{1/4} n}\right)^k \left(1 - \frac{k}{n}\right)^{n-k} \\ &\geq \frac{k}{4F^{1/4}} \left(\frac{k}{F^{1/4} en}\right)^k, \end{aligned}$$

where the last inequality holds by Lemma 2.4.1. Applying Theorem 4.3.8 with $s_m^{\text{cycle}} \geq \max\{s_m^{k^*}, s_m^n\}$ and absorbing the expected times for fitness levels $i < m$ in the asymptotic notation proves the claimed bounds. □

Similar to Bassin and Buzdalov [18], we can slow down the growth of λ. We accomplish this by cleverly choosing F in such a way that the algorithm is able to use every $\lambda \leq n$, ensuring that the algorithm uses the best parameter value. Choosing $F = (1 + 1/n)^4$ in Theorem 4.3.9 implies that $F^{(k+2)/4} = (1 + 1/n)^{k+2} \leq (1 + 1/n)^{n+2} = O(1)$, hence this factor can be dropped. Also note that $\frac{1}{F^{1/4}-1} = \frac{1}{(1+1/n)-1} = n$. Together, we obtain the following.

Corollary 4.3.10. *Let $F = (1+1/n)^4$ and $k \geq 2$. The expected optimisation time (in terms of fitness evaluations) of the modified self-adjusting $(1 + (\lambda, \lambda))$ GA on the* Jump$_k$ *function is*

$$\min\left\{O\left(n^{k+1}\right), O\left(n\left(\frac{en}{k}\right)^{k+1}\right)\right\}.$$

Comparing the bound of Corollary 4.3.10 against the expected optimisation time of the $(1+1)$ EA with optimal mutation rate of k/n, which is $T_{\text{opt}} = O\left(\left(\frac{en}{k}\right)^k\right)$ [62], our bound is larger than T_{opt} by a factor of $O\left(n^2/k\right)$ without the need to know the jump size k in advance. Our bound is only by a factor of $O(n^2/k^2)$ larger than the bound for the $(1+1)$ EA with the heavy-tailed ("fast") mutation operators from [62] with the recommended parameter $\beta = 1.5$.

4.4 Experimental Analysis

In the previous sections we performed asymptotic analyses focusing mainly on Jump$_k$ functions (summarised in Table 4.1). In this section, we conduct an experimental analysis that

Mechanism	Bounds	
	General bound	JUMP
Expected number of evaluations		
Vanilla	$O(dn) + 2\sum_{i=1}^{m} \frac{1}{s_i}$	$(1 \pm o(1)) \cdot 2n^k \left(1 - \frac{1}{n}\right)^{-n+k}$
Capping λ at $\lambda_{\max} = k$	–	$O(k) + 4\left(\frac{n}{k}\right)^k \left(1 - \frac{k}{n}\right)^{-n+k}$
Capping λ at $\lambda_{\max} = n/2$	$O(dn) + \frac{2^{n+4}}{\lvert\mathrm{OPT}\rvert}$	$\begin{cases} O\left(\left(\frac{n}{2}\right)^k\right) & \text{if } k \leq \frac{\log(n/2)}{2}, \\ O\left(\min\{(2n)^{k-1}, 2^n\}\right) & \text{otherwise.} \end{cases}$
Resetting λ	$O\left(\frac{F^{1/4}n}{F^{1/4}-1}\right)\sum_{i=1}^{m} \frac{1}{s_i^{\text{cycle}}}$	$\min\left\{O\left(\frac{F^{1/4}n^k}{F^{1/4}-1}\right), O\left(\left(\frac{F^{(k+2)/4}}{F^{1/4}-1}\right)\left(\frac{en}{k}\right)^{k+1}\right)\right\}$
Resetting λ $F = (1+1/n)^4$	–	$\min\left\{O\left(n^{k+1}\right), O\left(n\left(\frac{en}{k}\right)^{k+1}\right)\right\}$
Expected number of generations		
Vanilla	$\lceil 4\log_F(n)\rceil + 6d + \frac{1}{n}\sum_{i=1}^{m} \frac{1}{s_i}$	$(1 + o(1)) \cdot 2n^{k-1} \left(1 - \frac{1}{n}\right)^{-n+k}$
Capping λ at $\lambda_{\max} = n/2$	$O(d + \log n) + \frac{2^{n+3}}{n\lvert\mathrm{OPT}\rvert}$	$\begin{cases} O\left(\left(\frac{n}{2}\right)^{k-1}\right) & \text{if } k \leq \frac{\log(n/2)}{2}, \\ O\left(\min\{(2n)^{k-2}, 2^n/n\}\right) & \text{otherwise.} \end{cases}$
Resetting λ	$O(\lceil\log_F(n)\rceil)\sum_{i=1}^{m} \frac{1}{s_i^{\text{cycle}}}$	$\min\left\{O\left(n^{k-1}\log n\right), O\left(\left(\frac{enF^{1/4}}{k}\right)^{k+1}\left(\frac{\log n}{n}\right)\right)\right\}$

Table 4.1: Example of runtime bounds we obtain for the self-adjusting $(1 + (\lambda, \lambda))$ GA with different self-adjusting mechanisms. Note that some of the results are subject to further conditions, e. g. $4 \leq k \leq (1 - \varepsilon)n/2$.

aims to accomplish the following goals: 1) to complement our theoretical results with precise runtime results for concrete problem instances and jump sizes, 2) to compare the runtime of all the proposed versions of the self-adjusting $(1 + (\lambda, \lambda))$ GA against related algorithms, and 3) to test how the mathematical results obtained translate to other fitness landscapes.

In the experiments of this section, we ran an implementation[4] of all the different modifications of the self-adjusting $(1 + (\lambda, \lambda))$ GA studied in previous sections on different problems and compared them against:

- The $(1 + 1)$ EA with the standard mutation rate $p = 1/n$.
- The $(1 + 1)$ EA with the optimal choice of $p = k/n$ (on JUMP).
- The "fast" (1+1) fEA [62] with a heavy-tailed mutation rate: $p = r/n$ and $r \sim \mathrm{pow}(1.5, n)$.
- The $(1 + (\lambda, \lambda))$ GA with a heavy-tailed choice of $\lambda \sim \mathrm{pow}(2.5, n/2)$ [8] that we call the heavy-tailed $(1 + (\lambda, \lambda))$ GA.
- The $(1 + (\lambda, \lambda))$ GA with non-standard parameters and a heavy-tailed choice of $\lambda \sim \mathrm{pow}(2, \min\{5^{n/10}, 10^6\})$, $p = c = \sqrt{s/n}$ and $s \sim \mathrm{pow}(1, n)$ [3].

The parameter selection for the heavy-tailed algorithms was made following the recommendations of each study. In particular, for the $(1 + (\lambda, \lambda))$ GA with non-standard parameters, the upper bound for λ in the distribution is suggested to be exponential in n but because of memory demand from storing the probabilities in the power-law distribution, during implementation we impose a limit of 10^6 independent of n.

All experiments comprise of 500 runs for each algorithm-problem pair, recording the number of fitness evaluations to reach the optimum and the average is reported unless otherwise stated.

[4]The complete implementation and results can be found on GitHub (https://github.com/mariohevia/Parameter-Control-Mechanisms-Genetic-Algorithm)

In the following we report on results of statistical tests executed as follows. We performed Mann-Whitney U Test for all pairs of algorithms compared in the text. We performed two-sided tests to check whether the two input distributions differ or not, followed by one-sided tests both ways to confirm which algorithm is stochastically faster than the other. We report a comparison as statistically significant if the p-values of the two-sided test and that of the respective one-sided test both satisfied $p \leq 0.01$, that is, a confidence level of 0.01. When comparing one algorithm against several others, we performed a pairwise comparison for each pair as explained before and applied the Bonferroni correction.

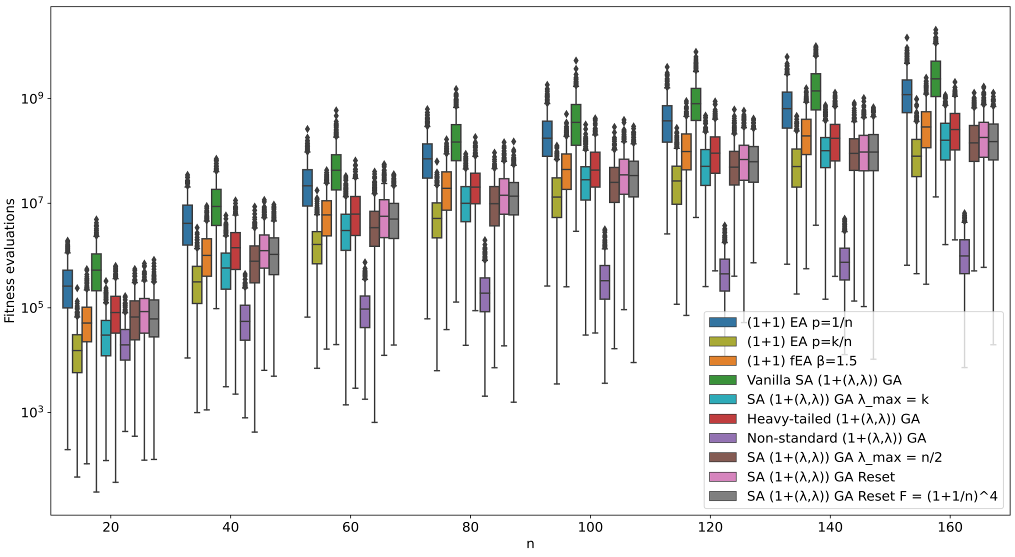

Figure 4.3: Box plot of the number of fitness evaluations on JUMP_k with $k = 4$ and $n = \{20, 40, 60, \ldots, 160\}$ over 500 runs.

4.4.1 Empirical Analyses on Jump Functions

For the class of JUMP_k functions we performed two experiments focused on the effects of the problem size and the jump size separately.

In the first experiment, we used a jump size $k = 4$ and n varying from 20 to 160 shown in Figure 4.3. The first thing to notice is that the $(1 + (\lambda, \lambda))$ GA with non-standard parameters has the best performance for $n \geq 40$; all comparisons are statistically significant. This is expected because the selection of parameters λ, c and p are tailored towards JUMP_k functions. We can also appreciate that the heavy-tailed $(1+(\lambda, \lambda))$ GA is able to use better λ values than the vanilla self-adjusting $(1 + (\lambda, \lambda))$ GA, performing statistically significantly better for all n although it performs statistically significantly worse than both resetting strategies and capping λ to $n/2$ for $n \geq 80$. It is worth pointing out that the self-adjusting $(1 + (\lambda, \lambda))$ GA capping λ to $n/2$ and both versions resetting λ scale better with n than the $(1+1)$ fEA. The self-adjusting $(1 + (\lambda, \lambda))$ GA variants are statistically slower than the $(1 + 1)$ fEA for $n = 20$ but for $n \geq 80$ they are statistically faster. This is most likely because the parameter landscape for instances with large n resemble the ones shown in the second column of Figure 4.1 in Section 4.3.2, where using values of λ that are linear in n help the crossover phase find the optimum even faster than the mutation phase with $\lambda = k$ and by extension faster than any mutation rate for the $(1 + 1)$ EA. Despite this possible advantage, these algorithms perform statistically significantly worse than the $(1 + 1)$ EA with $p = k/n$ on these JUMP_k instances because they waste some evaluations with non-optimal parameters. To inquire further, we performed additional experiments on $\textsc{Jump}_k$ with $n = 160$ and $k = 2$ that are not shown in the figures where the self-adjusting $(1 + (\lambda, \lambda))$ GA resetting λ with $F = (1 + 1/n)^4$ is statistically significantly faster than the $(1 + 1)$ EA with optimal parameter values showing that for some instances of $\textsc{Jump}_k$ the $(1 + (\lambda, \lambda))$ GA with standard parameter settings can be faster than the $(1 + 1)$ EA with the optimal mutation rate.

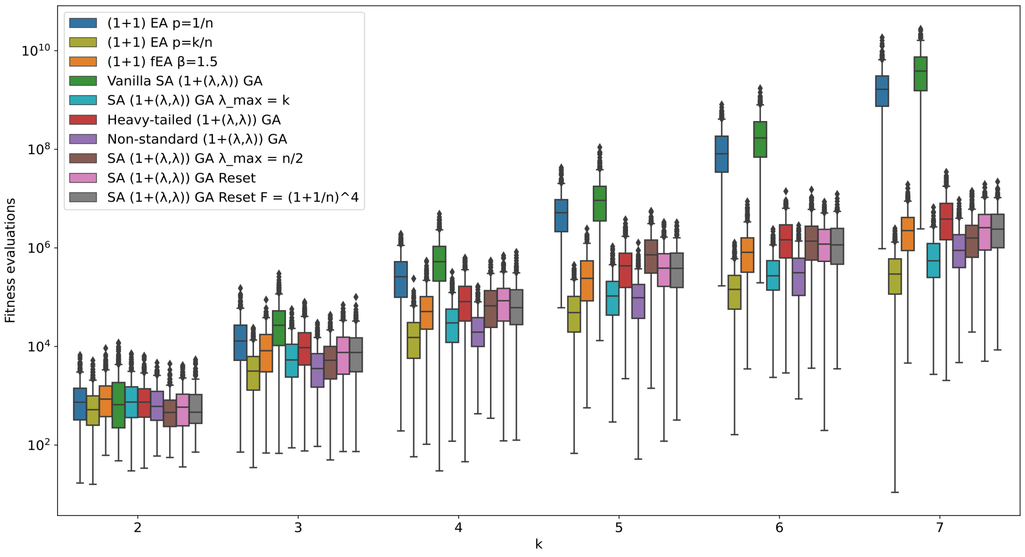

Figure 4.4: Box plot of the number of fitness evaluations on $\textsc{Jump}_k$ with $n = 20$ and $k = \{2, 3, 4, 5, 6, 7\}$ over 500 runs.

For the second experiment, we aimed to explore the effects of the jump size k. We used a small problem size of $n = 20$, varying the jump size from 2 to 7 shown in Figure 4.4. In this case we see that for larger values of k the self-adjusting $(1 + (\lambda, \lambda))$ GA capping λ to $n/2$ and resetting λ do not excel but they are still competitive with the $(1+1)$ fEA. This in conjunction with the good performance of the self-adjusting $(1 + (\lambda, \lambda))$ GA capping λ to k indicates that the mutation phase is much better at finding the optimum for large k. This is in agreement with our analysis of the parameter landscape from Section 4.3.2. Lastly in this experiment the non-standard $(1 + (\lambda, \lambda))$ GA does not excel. We suspect that this is caused by the small problem size $n = 20$.

In both plots shown in Figures 4.3 and 4.4 the average number of fitness evaluations for the self-adjusting $(1 + (\lambda, \lambda))$ GA with standard parameters and with $\lambda_{\max} = k$ is around twice the average for the $(1 + 1)$ EA with $p = 1/n$ and $p = k/n$, respectively, as predicted by our theoretical results.

4.4.2 Empirical Analyses on OneMax

In this section we study the algorithms on $\textsc{OneMax}$. We consider this problem because the self-adjusting $(1 + (\lambda, \lambda))$ GA is the fastest known unbiased genetic algorithm on $\textsc{OneMax}$ (with an expected number of $O(n)$ fitness evaluations, whereas the $(1 + 1)$ EA needs $\Theta(n \log n)$ expected fitness evaluations) and we want to illustrate the effects of the different parameter control strategies on the performance on $\textsc{OneMax}$.

Figure 4.5 shows the average number of fitness evaluations, normalised by the problem size n with the x-axis (problem size) being log-scaled. Figure 4.5 gives us the following three insights. First, the $(1 + (\lambda, \lambda))$ GA with non-standard parameters stands out as the algorithm with the worst performance by far; all comparisons with the non-standard $(1+(\lambda,\lambda))$ GA are statistically significant. This can be attributed to the choice of parameters, mainly the larger offspring population size. Secondly, all the other variants of the $(1+(\lambda,\lambda))$ GA have a good performance and the fact that the normalised time seems to increase slower than the $(1+1)$ EA suggests that they have an asymptotic runtime faster than $n \log n$. Lastly, resetting and capping λ to $n/2$ does not affect the performance of the algorithm on OneMax, compared to the vanilla version.

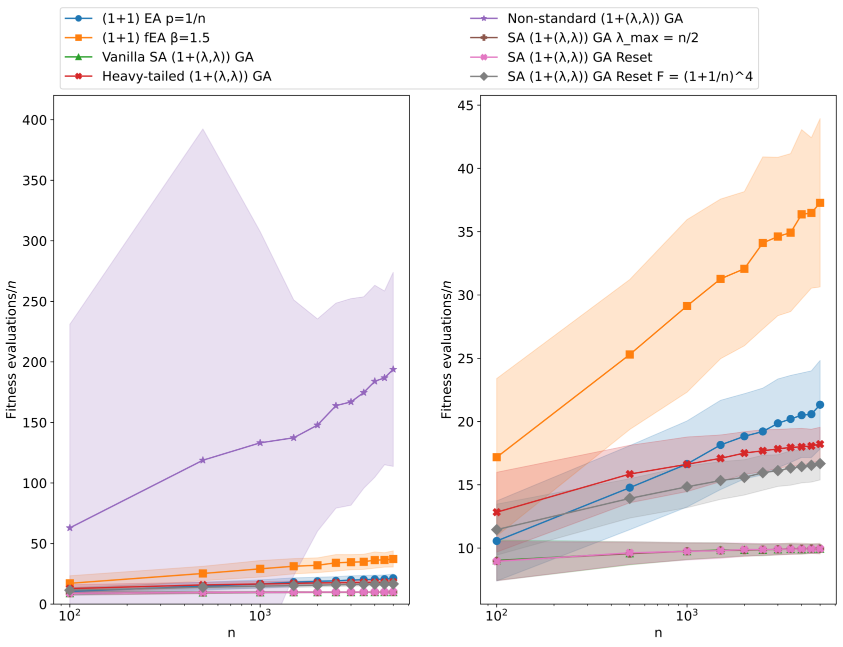

Figure 4.5: Average number of fitness evaluations and standard deviation on OneMax with varying n over 500 runs of all tested algorithms on the left plot and a more detailed view of the best performing algorithms on the right plot.

4.4.3 Empirical Analyses on Other Benchmark Functions

In order to see whether the theoretical results extend to other fitness landscapes we study the algorithms with generic parameter choices on five different functions. During these experiments we put a limit of 2.1 billion evaluations which is close to the upper limit of the variable **int** in C++ ($2^{31}-1$). The mean is computed by assigning the limit of 2.1 billion evaluations to all incomplete runs.

We consider the multimodal functions NearestPeak (NP) and WeightedNearestPeak (WNP) proposed by Jansen and Zarges [113]. These functions are characterised by a set of peaks that define a fitness landscape. The characteristics of their landscape depends on the number, position, slope and height of these peaks. Let p_i be the bit string representing peak i, then the weight W_i of peak i is $W_i = (n - \mathrm{H}(x, p_i))a_i + b_i$ where x is the current solution, a_i is the slope and b_i is the height of the peak. The difference between NP and WNP is that for NP the fitness is the weight of the nearest peak (in Hamming distance), while for WNP the fitness is the biggest weight with respect to the current search point. Both problems have a basin of attraction around the peaks, creating local and global optima.

The other three functions are well known NP-hard problems: Partition, Ising Spin Glass (ISG) and MAX-3SAT. The Partition optimisation problem is to divide a set of positive weights into two disjoint subsets such that the largest subset sum is minimised. The ISG problem is a problem derived from physics. The Ising model consists of discrete variables that represent magnetic dipole moments of atomic "spins" that can be in one of two states ($+1$ or -1). The spins are arranged in a graph describing the interactions strengths (edges) between spins (vertices), here a two-dimensional torus lattice is used. Neighbouring spins with the same state have a lower interaction than those with a different state. The ISG problem is to set the signs of all spins to minimise interactions. The MAX-3SAT problem is related to the more common 3-SAT problem. A MAX-3SAT instance is a Boolean formula in conjunctive normal form, that is, a logical conjunction of one or more clauses, where a clause is a logical disjunction of three binary variables, some of which can be negated. The search space $S = \{0,1\}^n$ encodes the choice of truth values of the binary variables in the Boolean formula and the optimisation problem is to find a solution that satisfies the maximum number of clauses. Since the clauses are disjunctive, they are satisfied if at least one of its literals is satisfied.

For all problems we used the same set of instances (one run per instance) across all algorithms because some instances might be more difficult than others. The instances were generated at random. The NP and WNP instances were generated with a problem size of 50 and different number of peaks with one global optimum at 1^n. The peaks were defined by $[|p_i|_1, a_i, b_i]$ where the values correspond to the number of ones, slope and height of peak i. Every peak was generated by sampling uniformly at random a search point with the number of ones specified in p_i. The following peaks (including the global optimum) were used:

- NearestPeak:
  1. $[50,5,0], [42,2,0], [41,4,0], [42,2,10]$
  2. $[50,5,0], [40,3,0], [42,2,0], [44,1,0], [42,2,10]$
  3. $[50,10,0], [40,9,0], [40,9,0], [40,9,0], [40,9,0], [40,9,0]$
- WeightedNearestPeak:
  1. $[50,10,0], [40,9,0], [41,9,0]$
  2. $[50,10,0], [40,9,0], [40,9,0], [40,9,0], [40,9,0], [40,9,0]$

For Partition the problem size is 500, with weights selected uniformly at random in the range $[0,1]$. Given that Partition is NP-hard we use a deterministic approximation algorithm called Longest Processing Time [145] to set a fitness goal. If an algorithm finds a solution with the same or higher fitness we consider the problem solved. In the case of ISG and MAX-3SAT we use the same 100 instances per problem size used by Goldman and Punch [91] that were also generated at random in their work. Following Goldman and Punch [91], we executed one run per instance, resulting in 100 runs per problem size.

In Figure 4.6 we see the results on NP, WNP and Partition instances. For these problems all algorithms found the optimum on all instances within the time budget, but given that there are a high number of outliers we present the data in box plot form. In Figure 4.6 the number of peaks is mentioned after the problem, where $5p$ means five peaks i.e. four local optima and one global optimum. For these problems we see a common trend: the original self-adjusting $(1 + (\lambda, \lambda))$ GA has the largest outliers in five of the six sets of experiments. This might be caused by the algorithm increasing λ to its maximum when the algorithm gets stuck. The modifications studied in this paper and the heavy-tailed $(1 + (\lambda, \lambda))$ GA seem to reduce the runtime of the outliers while maintaining a similar median, with the capping strategy having a smaller impact in the runtime of the outliers than the other algorithms. We would like to note that for all the $(1 + (\lambda, \lambda))$ GA versions excluding the $(1 + (\lambda, \lambda))$ GA with non-standard parameters the median number of evaluations is similar because the algorithms behave similarly when they do not get stuck in a local optimum as seen in the experiments on OneMax and this happens in most of the runs. In addition, we can see that the $(1 + (\lambda, \lambda))$ GA with non-standard parameters tends to have a slightly larger median runtime in most cases (statistically significantly larger for all comparisons on all problems except for NP 6p). This is most likely because when an easy instance is found

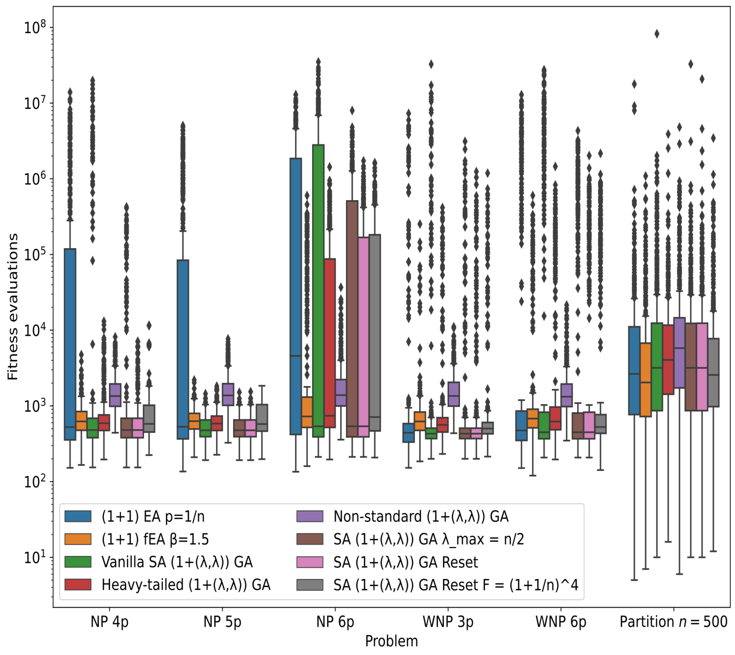

Figure 4.6: Box plot of the number of fitness evaluations on NearestPeak (NP), Weighted-NearestPeak (WNP) with $n = 50$ and Partition with $n = 500$ over 500 runs.

the algorithm spends more time, similar to OneMax. On the other hand, the outliers for the $(1 + (\lambda, \lambda))$ GA with non-standard parameters tend to have a smaller runtime compared with the outliers for other algorithms. We attribute this to the ability of the algorithm to perform large jumps when stuck in a local optima on these *hard* instances.

The next set of experiments were made on the Ising Spin Glass problem. In this case we show in Figure 4.7 a box plot of the number of fitness evaluations, the mean and number of failures. We include the box plot because the mean can be disturbed by failed runs capped at the limit, but the boxplot shows the median that is unaffected by this as long as half of the runs are able to find the optimum. The mean is still useful because it shows the effectiveness of an algorithm to escape local optima. A prime example is the $(1 + (\lambda, \lambda))$ GA with non-standard parameters, which has the smallest mean for most problem sizes, but its median is almost an order of magnitude larger than the smallest median.

Similar to other problems, on ISG the original self-adjusting $(1 + (\lambda, \lambda))$ GA tends to have larger outliers than all other algorithms, which is reflected in the larger mean and higher number of failures. This is attenuated by the variations studied here and the heavy-tailed $(1+(\lambda, \lambda))$ GA. An interesting point is that the $(1 + 1)$ EA is statistically significantly faster than all the $(1 + (\lambda, \lambda))$ GA variations with and without standard parameters (excluding the heavy-tailed $(1 + (\lambda, \lambda))$ GA and the resetting mechanism with $F = (1 + 1/n)^4$) in at least 3 of the problem sizes ($n = 49, 64, 81$), which might indicate that there is a bad fitness-distance correlation for these instances. The results of the heavy-tailed $(1 + (\lambda, \lambda))$ GA and the resetting mechanism with $F = (1 + 1/n)^4$ can be explained by the λ values used by these algorithms. The heavy-tailed $(1 + (\lambda, \lambda))$ GA samples $\lambda = 1$ in most generations and $F = (1 + 1/n)^4$ maintains the value of $\lfloor \lambda \rceil = 1$ for longer, making both algorithms behave exactly as the $(1 + 1)$ EA in these generations therefore they show a similar performance.

The results for the final test problem, MAX-3SAT, are shown in Figure 4.8. In this case there are no statistically significant differences between all the algorithms, but still there are some observations we can obtain. Similar to the ISG problem for MAX-3SAT we

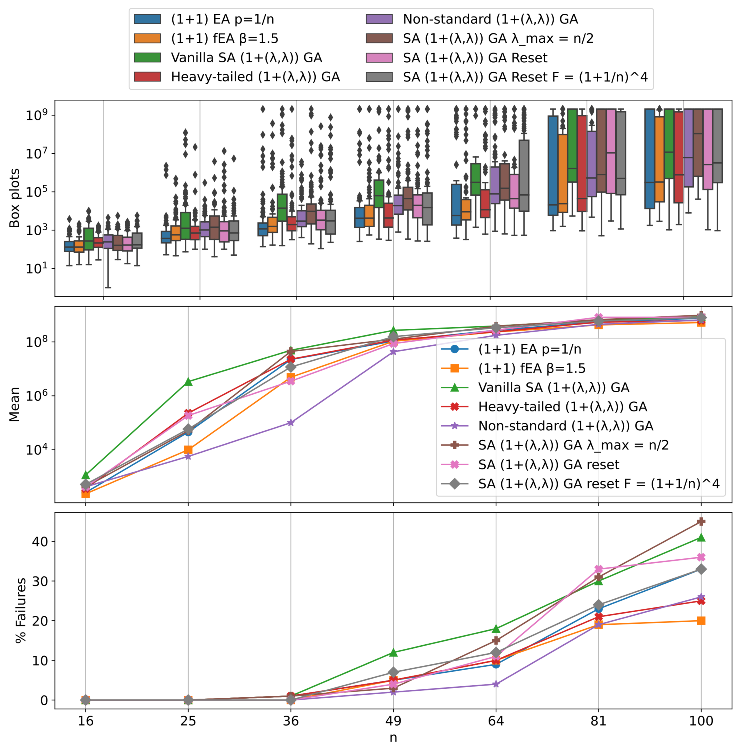

Figure 4.7: Box plot of the number of fitness evaluations (top), average number of fitness evaluations (middle), and percentage of failed runs (bottom) on the Ising Spin Glass problem varying the problem size over 200 runs.

show the median, mean and number of failed runs. For this problem it is clear that both the $(1+1)$ EA and the original self-adjusting $(1+(\lambda,\lambda))$ GA run into local optima and it is hard for them to escape, making the number of outliers, mean and number of failures increase. In contrast, all the self-adjusting $(1+(\lambda,\lambda))$ GA variants have a smaller mean and smaller number of failures. Once again the $(1+(\lambda,\lambda))$ GA with non-standard parameters has the smallest mean and number of failures; we attribute this to its capacity to jump out of local optima faster than all the other algorithms.

4.4.4 Discussion

Overall we have seen that the original self-adjusting $(1+(\lambda,\lambda))$ GA has a poor performance compared to all algorithms studied, on most problems. This shows that, as we expected from our theoretical results in Section 4.2 and 4.2.2, the self-adjusting $(1+(\lambda,\lambda))$ GA easily increases λ to its maximum on difficult parts of the optimisation, degrading the performance of the algorithm. Both resetting and capping λ improved the performance on almost all difficult problems and instances without affecting the performance on easy problems. Slowing down the increase of λ with small values of F can be helpful on difficult problems but as a trade-off it can slightly increase the runtime on easy problems.

On difficult problems both the $(1+1)$ fEA and the $(1+(\lambda,\lambda))$ GA with non-standard parameters tend to have fewer outliers and fewer failures than all the versions of the self-adjusting $(1+(\lambda,\lambda))$ GA. This is largely because both algorithms are tailored to perform large jumps with high probability when stuck in local optima. Because of this, both algorithms have a worse performance on easy problems with the $(1+(\lambda,\lambda))$ GA, with non-

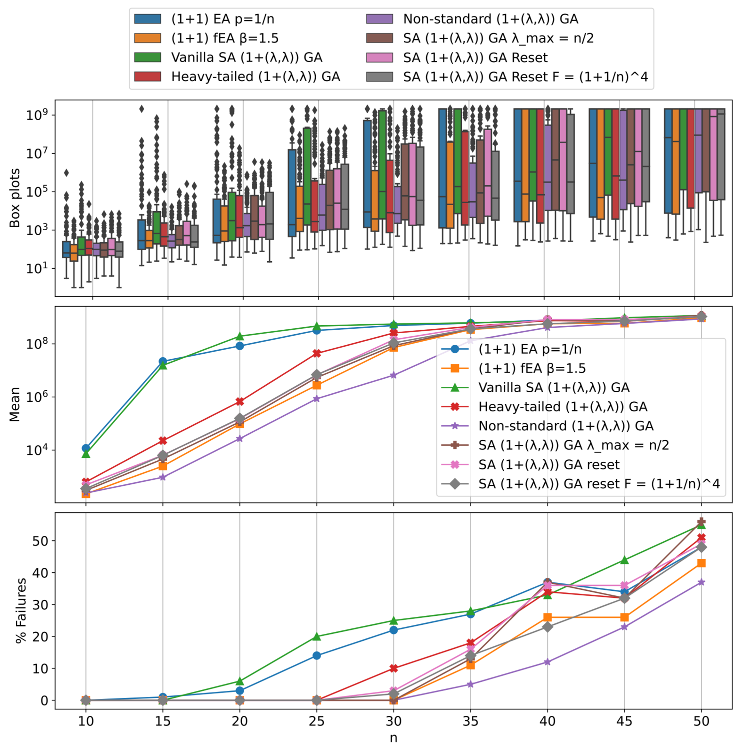

Figure 4.8: Box plot of the number of fitness evaluations (top), average number of fitness evaluations (middle), and percentage of failed runs (bottom) on the MAX-3SAT problem varying the problem size over 100 runs.

standard parameters being the worst algorithm when easy problems (OneMax, WNP and some instances of NP) are encountered.

4.5 Conclusions

We have provided a rigorous runtime analysis of the self-adjusting $(1 + (\lambda, \lambda))$ GA (considering the best offspring from the mutation phase during the selection step) for general function classes by presenting a fitness-level theorem for the self-adjusting $(1 + (\lambda, \lambda))$ GA that is easy to use and enables a transfer of runtime bounds from the $(1 + 1)$ EA to the self-adjusting $(1 + (\lambda, \lambda))$ GA.

The parameter control mechanism in the original self-adjusting $(1 + (\lambda, \lambda))$ GA tends to diverge λ to its maximum on multimodal problems. Then the algorithm effectively simulates a (1+n) EA with the default mutation rate of $1/n$. For the multimodal benchmark problem class Jump_k, we proved upper and lower runtime bounds that are tight up to lower-order terms, showing that, despite using crossover, the self-adjusting $(1 + (\lambda, \lambda))$ GA is not as efficient as other crossover-based algorithms.

Imposing a maximum value $\lambda_{\max}$ can improve performance, however then the problem remains of how to set $\lambda_{\max}$ if no problem-specific knowledge is available. The generic choice $\lambda_{\max} = n/2$ makes the self-adjusting $(1 + (\lambda, \lambda))$ GA perform random search steps during the mutation phase in case the algorithm gets stuck. This guards against deceptive problems and the algorithm still retains its original exploitation capabilities in the crossover phase.

We showed that the parameter landscape with respect to the impact of $\lambda_{\max}$ on the runtime of the self-adjusting $(1 + (\lambda, \lambda))$ GA on Jump_k is bimodal for appropriate n and k, and the optimal parameter changes drastically depending on the problem size n and the jump size k. The parameter landscape features two optima, one located in a wide basin of attraction that maximises the probability of finding the global optimum in the crossover phase and the other hidden in a narrow basin of attraction that maximises the probability of finding the global optimum during the mutation phase. For small k the wider basin of attraction leads to the optimal parameter value, but for larger k the optimal parameter value changes to the narrow basin of attraction. For large k considering the mutation offspring in the selection phase gives a large performance improvement to the $(1 + (\lambda, \lambda))$ GA. This fluctuating parameter landscape combined with it being bimodal makes parameter selection difficult.

Additionally, we investigated resetting λ to 1 after an unsuccessful generation at the maximum value. This makes the self-adjusting $(1 + (\lambda, \lambda))$ GA cycle through the parameter space, approaching optimal or near-optimal parameter values in every cycle in spite of the parameter landscape. We recommend to choose $F = (1 + 1/n)^4$ if a slow growth of λ is desired. For JUMP_k, this strategy gives the same expected runtime as that of the $(1 + 1)$ EA with the optimal mutation rate and fast mutation operators, up to small polynomial factors.

Finally, the empirical results show that the original self-adjusting $(1 + (\lambda, \lambda))$ GA tends to increase λ to its maximum, and this yields the worst performance on most multimodal problems tested, while the modifications of the parameter control mechanism improved its performance. Additionally, these modifications do not significantly affect the algorithm's performance on easy problems. This suggests that for common optimisation problems it is better to use the parameter control variants (capping or resetting λ) studied here than the original self-adjusting $(1 + (\lambda, \lambda))$ GA.

Chapter 5

Do Success-Based Rules Work for Non-elitist Algorithms?

5.1 Introduction

Non-elitist evolutionary algorithms are evolutionary algorithms that use non-elitist selection for survival. The non-elitist selection mechanism is able to pick new search points that in the short-term are less favorable for the optimisation than current best solutions. This short-term loss is taken in the hope that these search points lead to better solutions in the long-term. Despite the fact that non-elitist evolutionary algorithms are often better at escaping from local optima [42, 43, 106] and widely applied in optimisation, non-elitist evolutionary algorithms are rarely theoretically analysed.

This is exacerbated for parameter control mechanisms where most theoretical analyses of parameter control mechanisms focus on so-called *elitist evolutionary algorithms* that always reject worsening moves (with notable exceptions that study self-adaptive mutation rates in the $(1, \lambda)$ EA [69] and the (μ, λ) EA [25], and hyper-heuristics that choose between elitist and non-elitist selection mechanisms [137]). The performance of parameter control mechanisms and especially success-based parameter control in non-elitist algorithms is not well understood. There are many applications of non-elitist evolutionary algorithms for which an improved theoretical understanding of parameter control mechanisms could bring performance improvements matching or exceeding the ones seen for elitist algorithms.

In this chapter we consider the $(1, \lambda)$ EA that in every generation creates λ offspring and selects the best one for survival. The offspring population size λ plays an important role in the performance of the $(1, \lambda)$ EA. The seminal paper by Jägersküpper and Storch [106] showed that the $(1, \lambda)$ EA with small λ-values is inefficient on all functions with one unique optimum, whilst for large λ-values the algorithm behaves as its elitist counterpart the $(1 + \lambda)$ EA with high probability. Rowe and Sudholt [166] showed that there is a sharp threshold at $\lambda = \log_{\frac{e}{e-1}} n$ between exponential and polynomial runtimes on OneMax using standard bit mutation with mutation probability $p = \frac{1}{n}$. A value $\lambda \geq \log_{\frac{e}{e-1}} n$ (relaxed to $\lambda \geq \left\lceil \log_{\frac{e}{e-1}}(cn/\lambda) \right\rceil$ for any constant $c > e^2$ in [20]) ensures that the offspring population size is sufficiently large to ensure that improvements in fitness are made often enough and more importantly fitness losses are avoided resulting in a positive *drift* (expected progress) towards the optimum even on the most challenging fitness levels. For easier fitness levels, smaller values of λ are sufficient.

This is a challenging scenario for self-adjusting the offspring population size λ since too small values of λ can easily make the algorithm decrease its current fitness, moving away from the optimum. For static values of $\lambda \leq (1 - \varepsilon) \log_{\frac{e}{e-1}} n$, for any constant $\varepsilon > 0$, we know that the optimisation time is exponential with high probability for all functions with one unique optimum [166]. Furthermore, this threshold can shift towards larger values of λ depending on the characteristics of the functions (e. g. in LeadingOnes the threshold is at $\lambda = 2\log_{\frac{e}{e-1}} n$) or shift towards smaller values if the mutation probability is reduced [166]. Moreover, too large values for λ can waste function evaluations and blow up the optimisation time.

5.1.1 Contibutions

We consider a self-adjusting version of the $(1, \lambda)$ EA called $(1, \{F^{1/s}\lambda, \lambda/F\})$ EA (self-adjusting $(1, \lambda)$ EA) that uses a success-based rule similar to the one used by the self-adjusting $(1 + (\lambda, \lambda))$ GA from Chapter 4. For an update strength F and a success rate s, in a generation where no improvement in fitness is found, λ is increased by a factor of $F^{1/s}$ and in a successful generation, λ is divided by a factor F. If one out of $s+1$ generations is successful, the value of λ is maintained. The case $s = 4$ is the famous one-fifth success rule [116, 163].

We ask whether the self-adjusting $(1, \lambda)$ EA is able to find and maintain suitable parameter values of λ throughout the run, despite the lack of elitism and without knowledge of the problem in hand.

In Section 5.3.1 we answer this question in the affirmative for the simple OneMax function if the success rate s is chosen correctly. We show that, if s is a constant with $0 < s < 1$, then the self-adjusting $(1, \lambda)$ EA optimises OneMax in $O(n)$ expected generations and $O(n \log n)$ expected fitness evaluations. The bound on evaluations is optimal for all unary unbiased black-box algorithms [67, 128]. However, if s is a sufficiently large constant, $s \geq 18$, the runtime on OneMax becomes exponential with overwhelming probability (see Section 5.3.2). The reason is that then unsuccessful generations increase λ only slowly, whereas successful generations decrease λ significantly. This effect is more pronounced during early stages of a run when the current search point is still far away from the optimum and successful generations are common. We show that then the algorithm gets stuck in a non-stable equilibrium with small λ-values and frequent fallbacks (fitness decreases) at a linear Hamming distance to the optimum. This effect is not limited to OneMax; we show that this negative result easily translates to other functions for which it is easy to find improvements during early stages of a run.

The last remark leaves open whether large success rates are inefficient on all functions. In Section 5.4 we show that the self-adjusting $(1, \lambda)$ EA is robust with respect to the choice of the success rate if the fitness function is sufficiently hard. We define a class of *everywhere hard* fitness functions, where for all search points the probability of finding an improvement is bounded by $O(n^{-\varepsilon})$, for a constant $\varepsilon > 0$. A well-known example is the function LeadingOnes$(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$ that counts the length of the longest prefix of bits set to 1: for every non-optimal search point, an improvement requires the first 0-bit to be flipped.

We show that on all everywhere hard functions λ quickly reaches a sufficiently large value such that fitness decreases become unlikely and the self-adjusting $(1, \lambda)$ EA typically behaves like an elitist algorithm. We then present simple and easy to use general upper bounds for the runtime of the self-adjusting $(1, \lambda)$ EA on everywhere hard functions that apply for all constant success rates s and update strengths $F > 1$. More specifically, we show a general upper bound on the expected number of evaluations using the fitness-level method that asymptotically matches the bound obtained for the $(1 + 1)$ EA. For the expected number of generations, we show an upper bound of $O(d + \log(1/p^+_{\min}))$ where d is the number of non-optimal fitness values and $p^+_{\min}$ is the smallest probability of finding an improvement from any non-optimal search point.

Using our general upper bound, we obtain novel bounds of $O(d)$ expected generations and $O(dn)$ expected evaluations for all everywhere hard unimodal functions with $d + 1 \geq \log n$ fitness levels. This is $O(n)$ expected generations and $O(n^2)$ expected evaluations for

LeadingOnes. We then introduce a problem class called OneMaxBlocks that allows us to tune the difficulty of the fitness levels with a parameter k. Varying this parameter changes the behaviour of the function from a LeadingOnes-like behaviour to a OneMax-like behaviour, allowing us to explore how the difficulty of the function affects the runtime of the self-adjusting $(1, \lambda)$ EA.

In Section 5.5 we complement our runtime analyses with experiments. First, we compare the runtime of the self-adjusting $(1, \lambda)$ EA, the self-adjusting $(1 + \lambda)$ EA and the $(1, \lambda)$ EA with the best known fixed λ on OneMax for different problem sizes. Then, we show a sharp threshold for the success rate at $s \approx 3.4$ where the runtime changes from polynomial to exponential. This indicates that the widely used one-fifth rule ($s = 4$) is inefficient here, but other success rules achieve optimal asymptotic runtime. Next, we show how different values of s affect the fixed target running times, the growth of λ over time and the time spent in each fitness value, shedding light on the non-optimal equilibrium states in the self-adjusting $(1, \lambda)$ EA. Afterwards, we show how reducing the mutation probability allows the self-adjusting $(1, \lambda)$ EA to optimise OneMax with any constant success rate s. Finally, we study the self-adjusting $(1, \lambda)$ EA on LeadingOnes and OneMaxBlocks. The OneMaxBlocks functions allows us to understand better how the difficulty of the fitness levels affect the performance of the self-adjusting $(1, \lambda)$ EA.

5.2 Preliminaries

Using the naming convention from [51] we call the algorithm self-adjusting $(1, \{F^{1/s}\lambda, \lambda/F\})$ EA or self-adjusting $(1, \lambda)$ EA for short (Algorithm 7). Several mutation operators can be plugged into Line 4 to instantiate a particular self-adjusting $(1, \lambda)$ EA. We will consider two operators. Standard bit mutations flip each bit independently with a mutation probability p. The other mutation operator, that has gain popularity recently, is the heavy-tailed mutation operator proposed by Doerr et al. [62]. It performs a standard bit mutation with a mutation probability of $p = r/n$ and r is chosen randomly in each iteration according to a discrete power-law distribution on $[1..n/2]$ with exponent $\beta > 1$.

The self-adjusting $(1, \lambda)$ EA behaves as the conventional $(1, \lambda)$ EA (with the corresponding mutation operator), but in every generation it adjusts the offspring population size depending on the success of the generation. If the fittest offspring y is better than the parent x, the offspring population size is divided by the *update strength* F, and multiplied by $F^{1/s}$ otherwise, with s being the *success rate*.

The idea of the parameter control mechanism is based on the interpretation of the one-fifth success rule from [116]. The parameter λ remains constant if the algorithm has a success every $s + 1$ generations as then its new value is $\lambda \cdot (F^{1/s})^s \cdot 1/F = \lambda$. In pseudo-Boolean optimisation, the one-fifth success rule was first implemented by Doerr et al. [60], and proved to track the optimal offspring population size on the $(1 + (\lambda, \lambda))$ GA in [50]. Our implementation is closer to the one used in [68], where the authors generalise the success rule, implementing the success rate s as a hyper-parameter.

Note that as in Chapter 4 we regard λ to be a real value, so that changes by factors of $1/F$ or $F^{1/s}$ happen on a continuous scale. Following Doerr and Doerr [50] and Chapter 4, we assume that, whenever an integer value of λ is required, λ is rounded to a nearest integer. For the sake of readability, we often write λ as a real value even when an integer is required. Where appropriate, we use the notation $\lfloor \lambda \rceil$ to denote the integer nearest to λ (that is, rounding up if the fractional value is at least 0.5 and rounding down otherwise).

5.2.1 Notation

We now give notation for all $(1, \lambda)$ EA algorithms that will be used in the remaining of this chapter.

Algorithm 7: Self-adjusting $(1, \{F^{1/s}\lambda, \lambda/F\})$ EA.

```
1 Initialization: Choose $x \in \{0,1\}^n$ uniformly at random (u.a.r.) and $\lambda := 1$;
2 Optimization: for $t \in \{1, 2, \dots\}$ do
3     Mutation: for $i \in \{1, \dots, \lambda\}$ do
4         $y'_i \in \{0,1\}^n \leftarrow$ mutate($x$);
5     Selection: Choose $y \in \{y'_1, \dots, y'_\lambda\}$ with $f(y) = \max\{f(y'_1), \dots, f(y'_\lambda)\}$ u.a.r.;
6     Update:
7     if $f(y) > f(x)$ then $x \leftarrow y$; $\lambda \leftarrow \max\{1, \lambda/F\}$;
8     else $x \leftarrow y$; $\lambda \leftarrow F^{1/s}\lambda$;
```

We define $X_0, X_1, \dots$ as the sequence of states of the algorithm, where $X_t = (x_t, \lambda_t)$ describes the current search point x_t and the offspring population size λ_t at generation t. We often omit the subscripts t when the context is obvious.

Since in all $(1, \lambda)$ EA algorithms selection is performed through comparisons of search points and hence ranks of search points, the absolute fitness values are not relevant. Without loss of generality we may therefore assume that the domain of any fitness function is taken as integers $\{0, 1, \dots, d\}$ where $d + 1 > 1$ is the number of different fitness values and all search points with fitness d are global optima. We exclude constant functions with $d = 0$, as these are trivial. We shall refer to all search points with fitness i, for $0 \leq i \leq d$ as fitness level i.

We will use the following notation for all $(1, \lambda)$ EA algorithms.

Definition 5.2.1. In the context of the self-adjusting $(1, \lambda)$ EA with $\lambda_t = \lambda$ for all $0 \leq i < d$ and all search points x with $f(x) < d$ we define:

$$\begin{aligned}
p^-_{x,\lambda} &= \Pr\left(f(x_{t+1}) < f(x_t) \mid x_t = x\right)\\
p^0_{x,\lambda} &= \Pr\left(f(x_{t+1}) = f(x_t) \mid x_t = x\right)\\
p^+_{x,\lambda} &= \Pr\left(f(x_{t+1}) > f(x_t) \mid x_t = x\right)\\
\Delta^-_{x,\lambda} &= \mathrm{E}(f(x_t) - f(x_{t+1}) \mid x_t = x \text{ and } f(x_{t+1}) < f(x_t))\\
\Delta^+_{x,\lambda} &= \mathrm{E}(f(x_{t+1}) - f(x_t) \mid x_t = x \text{ and } f(x_{t+1}) > f(x_t))\\
s_i &= \min_x \{p^+_{x,1} \mid x_t = x \text{ and } f(x) = i\}
\end{aligned}$$

Here s_i is a lower bound on the probability of one offspring finding an improvement from any search point in fitness level i. For functions where all search points x in every fitness level i have the same probabilities $p^+_{x,\lambda}$, $p^0_{x,\lambda}$ and $p^-_{x,\lambda}$ and the same $\Delta^+_{x,\lambda}$ and $\Delta^-_{x,\lambda}$ then instead we use the following notation.

Definition 5.2.2. In the context of the self-adjusting $(1, \lambda)$ EA with $\lambda_t = \lambda$ for all $0 \leq i < d$ where all search points $\{x^{(1)}, x^{(2)}, \dots\}$ in fitness level i satisfy $p^+_{x^{(1)},\lambda} = p^+_{x^{(2)},\lambda} = \dots$, $p^-_{x^{(1)},\lambda} = p^-_{x^{(2)},\lambda} = \dots$, $\Delta^+_{x^{(1)},\lambda} = \Delta^+_{x^{(2)},\lambda} = \dots$ and $\Delta^-_{x^{(1)},\lambda} = \Delta^-_{x^{(2)},\lambda} = \dots$ we define:

$$\begin{aligned}
p^-_{i,\lambda} &= \Pr\left(f(x_{t+1}) < i \mid f(x_t) = i\right)\\
p^0_{i,\lambda} &= \Pr\left(f(x_{t+1}) = i \mid f(x_t) = i\right)\\
p^+_{i,\lambda} &= \Pr\left(f(x_{t+1}) > i \mid f(x_t) = i\right)\\
\Delta^-_{i,\lambda} &= \mathrm{E}(i - f(x_{t+1}) \mid f(x_t) = i \text{ and } f(x_{t+1}) < i)\\
\Delta^+_{i,\lambda} &= \mathrm{E}(f(x_{t+1}) - i \mid f(x_t) = i \text{ and } f(x_{t+1}) > i)
\end{aligned}$$

We note that if the conditions in Definition 5.2.2 are met, then $s_i = p^+_{i,1}$. We often refer to the probability $p^+_{x,1}$ of one offspring improving the current fitness and abbreviate $p^+_x := p^+_{x,1}$ (and likewise $p^+_i := p^+_{i,1}$, $p^-_x := p^-_{x,1}$, $p^-_i := p^-_{i,1}$, etc.).

As in [166], we call $\Delta^+_{x,\lambda}$ and $\Delta^+_{i,\lambda}$ *forward drift*, and $\Delta^-_{x,\lambda}$ and $\Delta^-_{i,\lambda}$ *backward drift*. Note that the forward and backward drift are at least 1 by definition. Now, p^+_x is the probability of one offspring finding a better fitness value and $p^+_{x,\lambda} = 1 - (1 - p^+_x)^\lambda$ since it is sufficient that one offspring improves the fitness. The probability of a fallback is $p^-_{x,\lambda} = (p^-_x)^\lambda$ since all offspring must have worse fitness than their parent. These relationships also hold for $p^+_{i,\lambda}$ and $p^-_{i,\lambda}$ with p^+_i and p^-_i, respectively.

We write $p^+_{\min} := \min_x\{p^+_x \mid f(x) < d\}$ and $p^+_{\max} := \max_x\{p^+_x \mid f(x) < d\}$ to denote the minimum and maximum value for the probability p^+_x among all non-optimal search points x.

5.2.2 Drift Analysis and Potential Functions

Throughout this chapter we will use drift analysis to obtain bounds for the expected optimisation time of the self-adjusting $(1, \lambda)$ EA on different benchmark functions. As we have discussed in Section 2.4.3, in order to apply drift analysis we need to identify a potential function that adequately captures the progress of the algorithm and the distance from a desired target state. A natural candidate for a potential function is the fitness of the current individual $f(x_t)$. However, the self-adjusting $(1, \lambda)$ EA adjusts λ throughout the optimisation, and the expected change in fitness crucially depends on the current value of λ. Therefore, we also need to take into account the current offspring population size λ and capture both fitness and λ in our potential function. Since we study different behaviours of the algorithm depending on the problem in hand and the success rate s we consider an abstract function $h(\lambda_t)$ of the current offspring population sizes. The function $h(\lambda_t)$ will be chosen differently for different contexts, such as proving a positive result for small success rates s and proving a negative result for large success rates.

Definition 5.2.3. Given a function $h\colon \mathbb{R} \to \mathbb{R}$, we define the potential function $g(X_t)$ as

$$g(X_t) = f(x_t) + h(\lambda_t).$$

We do not make any assumptions on $h(\lambda_t)$ at this stage, but we will choose $h(\lambda_t)$ in the following sections as functions of λ_t that reward increases of λ_t, for small values of λ_t. We believe that this approach should be more widespread and that it could be useful for the analysis of a wide range of success-based parameter control mechanisms. Similar approaches have been used before for analysing self-adjusting mutation rates [63, 69] and for continuous domains in [1, 142, 143]. In addition we believe it might be able to simplify previous analysis such as [50, 68].

For every function $h(\lambda_t)$, we can compute the drift in the potential as shown in the following lemma.

Lemma 5.2.4. *Consider the self-adjusting* $(1, \lambda)$ EA*. Then for every function* $h\colon \mathbb{R} \to \mathbb{R}^+_0$ *and every generation* t *with* $f(x_t) < n$ *and* $\lambda_t > F$*,* $\mathrm{E}(g(X_{t+1}) - g(X_t) \mid X_t = X)$ *is*

$$\left(\Delta^+_{x,\lambda} + h(\lambda/F) - h(\lambda F^{1/s})\right) p^+_{x,\lambda} + h(\lambda F^{1/s}) - h(\lambda) - \Delta^-_{x,\lambda} p^-_{x,\lambda}.$$

If $\lambda_t \le F$ *then,* $\mathrm{E}(g(X_{t+1}) - g(X_t) \mid X_t = X)$ *is*

$$\left(\Delta^+_{x,\lambda} + h(1) - h(\lambda F^{1/s})\right) p^+_{x,\lambda} + h(\lambda F^{1/s}) - h(\lambda) - \Delta^-_{x,\lambda} p^-_{x,\lambda}.$$

We note that if all search points in fitness level i have the same probabilities $p^+_{x,\lambda}$, $p^0_{x,\lambda}$ and $p^-_{x,\lambda}$ and the same $\Delta^+_{x,\lambda}$ and $\Delta^-_{x,\lambda}$, then we can replace all subscripts referring to the current search point x by the current fitness level i in Lemma 5.2.4.

Proof of Lemma 5.2.4. When an improvement is found, the fitness increases in expectation by $\Delta^+_{x,\lambda}$ and since $\lambda_{t+1} = \lambda/F$, the λ term changes by $h(\lambda/F) - h(\lambda)$. When the fitness does not change, the λ term changes by $h(\lambda F^{1/s}) - h(\lambda)$. When the fitness decreases the expected decrease is $\Delta^-_{x,\lambda}$ and the λ term changes by $h(\lambda F^{1/s}) - h(\lambda)$. Together with $\lambda > F$, $\mathrm{E}(g(X_{t+1}) - g(X_t) \mid X_t = X)$ is

$$\begin{aligned}
&\left(\Delta^+_{x,\lambda} + h(\lambda/F) - h(\lambda)\right) p^+_{x,\lambda} + \left(h(\lambda F^{1/s}) - h(\lambda)\right) p^0_{x,\lambda} + \left(h(\lambda F^{1/s}) - h(\lambda) - \Delta^-_{x,\lambda}\right) p^-_{x,\lambda} \\
&= \left(\Delta^+_{x,\lambda} + h(\lambda/F) - h(\lambda)\right) p^+_{x,\lambda} + \left(h(\lambda F^{1/s}) - h(\lambda)\right) (p^0_{x,\lambda} + p^-_{x,\lambda}) - \Delta^-_{x,\lambda} p^-_{x,\lambda} \\
&= \left(\Delta^+_{x,\lambda} + h(\lambda/F) - h(\lambda)\right) p^+_{x,\lambda} + \left(h(\lambda F^{1/s}) - h(\lambda)\right) (1 - p^+_{x,\lambda}) - \Delta^-_{x,\lambda} p^-_{x,\lambda} \\
&= \left(\Delta^+_{x,\lambda} + h(\lambda/F) - h(\lambda F^{1/s})\right) p^+_{x,\lambda} + h(\lambda F^{1/s}) - h(\lambda) - \Delta^-_{x,\lambda} p^-_{x,\lambda}
\end{aligned}$$

Line 7 in Algorithm 7 updates λ to $\max\{1, \lambda/F\}$, hence, if $\lambda \leq F$ then $h(\lambda/F)$ needs to be replaced by $h(\lambda/\lambda) = h(1)$. □

5.3 Success Rates Matter

We begin our study of the self-adjusting $(1,\lambda)$ EA using standard bit mutation with a mutation probability $p = 1/n$ on the well-known fitness function $\textsc{OneMax}(x) := \sum_{i=1}^n x_i$. We recall from Section 2.3 that OneMax is one of the simplest benchmark functions and because of this it is traditionally the first function studied on new evolutionary algorithms. We use OneMax to understand better the parameter dynamics of the self-adjusting $(1,\lambda)$ EA and how different values for update strength F and the success rate s affect the performance of the algorithm.

We show that, for all constant s with $0 < s < 1$ the self-adjusting $(1,\lambda)$ EA optimises OneMax in $O(n)$ expected generations and $O(n \log n)$ expected function evaluations. However, this success is only possible when the success rate s is chosen appropriately small, the algorithm requires exponential time with high probability if $s \geq 18$.

To bound the expected number of generations for small success rates on OneMax, we apply drift analysis to a potential function that trades off increases in fitness against a penalty term for small λ-values. In generations where the fitness decreases, λ increases and the penalty is reduced, allowing us to show a positive drift in the potential for all fitness levels and all λ.

To bound the expected number of evaluations, we further use the potential to construct a novel "ratchet argument": we show that, even when the fitness decreases, it does not decrease much below the best fitness seen so far. More precisely, with high probability, if $f(x_t)$ is the current fitness at time t and $f^*_t = \max\{f(x_{t'}) \mid t' \leq t\}$ is the best fitness seen so far, then, with high probability, $f(x_t) \geq f^*_t - \rho \log n$ for an appropriate constant ρ. Then we show that there is a constant probability that the best-so far fitness is increased by $\log n$ in a sequence of generations without fallbacks. We are hopeful that these arguments will prove useful in the analysis of other non-elitist algorithms as well.

We start the analysis with useful bounds for the transition probabilities of the $(1,\lambda)$ EA on OneMax. Using common bounds and standard arguments, we obtain the following lemma.

Lemma 5.3.1. *For any* $(1, \lambda)$ EA *on* OneMax, *the quantities from Definition 5.2.2 are bounded as follows.*

$$1 - \frac{en}{en + \lambda(n-i)} \leq 1 - \left(1 - \frac{n-i}{en}\right)^{\lambda} \leq p^{+}_{i,\lambda} \tag{5.1}$$

$$p^{+}_{i,\lambda} \leq 1 - \left(1 - 1.14\left(\frac{n-i}{n}\right)\left(1 - \frac{1}{n}\right)^{n-1}\right)^{\lambda} \leq 1 - \left(1 - \frac{n-i}{n}\right)^{\lambda} \tag{5.2}$$

If $0.84n \leq i \leq 0.85n$ *and* $n \geq 163$*, then* $p^{+}_{i} \leq 0.069$*.*

$$\left(\frac{i}{n} - \frac{1}{e}\right)^{\lambda} \leq p^{-}_{i,\lambda} \leq \left(1 - \frac{n-i}{en} - \left(1 - \frac{1}{n}\right)^{n}\right)^{\lambda} \leq \left(\frac{e-1}{e}\right)^{\lambda} \tag{5.3}$$

$$1 \leq \Delta^{-}_{i,\lambda} \leq \frac{e}{e-1} \tag{5.4}$$

$$1 \leq \Delta^{+}_{i,\lambda} \leq \sum_{j=1}^{\infty}\left(1 - \left(1 - \frac{1}{j!}\right)^{\lambda}\right) \tag{5.5}$$

If $\lambda \geq 5$*, then* $\Delta^{+}_{i,\lambda} \leq \lceil \log \lambda \rceil + 0.413$*.*

Proof. We start by bounding the probability of one offspring being better than the parent x. For the lower bound a sufficient condition for the offspring to be better than the parent is that only one 0-bit is flipped. Therefore,

$$p^{+}_{i} \geq \frac{n-i}{n}\left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{n-i}{en}. \tag{5.6}$$

Along with $p^{+}_{i,\lambda} = 1 - (1 - p^{+}_{i})^{\lambda}$, this proves one of the lower bounds in Equation (5.1) in Lemma 5.3.1. Additionally, using $(1+x)^{r} \leq \frac{1}{1-rx}$ for all $x \in [-1, 0]$ and $r \in \mathbb{N}$

$$p^{+}_{i,\lambda} \geq 1 - \left(1 - \frac{n-i}{en}\right)^{\lambda} \geq 1 - \frac{1}{1 + \frac{\lambda(n-i)}{en}} = 1 - \frac{en}{en + \lambda(n-i)}.$$

For the upper bound a necessary condition for the offspring to be better than the parent is that at least one 0-bit is flipped, hence

$$p^{+}_{i} \leq \frac{n-i}{n}.$$

Additionally, we use the following upper bound shown in [153]:

$$p^{+}_{i} \leq 1.14\left(\frac{n-i}{n}\right)\left(1 - \frac{1}{n}\right)^{n-1}.$$

Along with $p^{+}_{i,\lambda} = 1 - (1 - p^{+}_{i})^{\lambda}$, this proves the upper bounds in Equation (5.2) in Lemma 5.3.1.

The additional upper bound for p^{+}_{i} when $0.84n \leq i \leq 0.85n$ uses a precise bound for which we will give a complete proof in Section 6.3 of:

$$\begin{aligned} p^{+}_{i} &\leq \left(1 - \frac{1}{n}\right)^{n-2} \sum_{a=0}^{\infty} \sum_{b=a+1}^{\infty} \left(\frac{i}{n}\right)^{a} \left(\frac{n-i}{n}\right)^{b} \frac{1}{a!b!} \\ &\leq \frac{1}{e}\left(1 - \frac{1}{n}\right)^{-2} \sum_{a=0}^{\infty} \sum_{b=a+1}^{\infty} (0.85)^{a} (0.16)^{b} \frac{1}{a!b!} \\ &\leq \left(1 - \frac{1}{n}\right)^{-2} 0.068152. \end{aligned}$$

This implies that, for every $n \geq 163$ and $0.84n \leq i \leq 0.85n$, $p_i^+ \leq 0.069$.

We now calculate p_i^-. For the upper bound we use

$$p_i^- = 1 - p_i^+ - p_i^0.$$

Using Equation (5.6) and bounding p_i^0 from below by the probability of no bit flipping, that is,

$$p_i^0 \geq \left(1 - \frac{1}{n}\right)^n,$$

we get

$$p_i^- \leq 1 - \frac{n-i}{en} - \left(1 - \frac{1}{n}\right)^n. \tag{5.7}$$

Finally, for the lower bound we note that for an offspring to have less fitness than the parent it is sufficient that one of the i 1-bits and none of the 0-bits is flipped. Therefore,

$$\begin{aligned} p_i^- &\geq \left(1 - \left(1 - \frac{1}{n}\right)^i\right)\left(1 - \frac{1}{n}\right)^{n-i} \\ &= \left(1 - \frac{1}{n}\right)^{n-i} - \left(1 - \frac{1}{n}\right)^n \\ &\geq 1 - \frac{n-i}{n} - \frac{1}{e} = \frac{i}{n} - \frac{1}{e}. \end{aligned} \tag{5.8}$$

Using $p_{i,\lambda}^- = (p_i^-)^\lambda$ with Equations (5.7) and (5.8) we obtain

$$\left(\frac{i}{n} - \frac{1}{e}\right)^\lambda \leq p_{i,\lambda}^- \leq \left(1 - \frac{n-i}{en} - \left(1 - \frac{1}{n}\right)^n\right)^\lambda.$$

The upper bound is simplified as follows:

$$\begin{aligned} p_{i,\lambda}^- &\leq \left(1 - \frac{n-i}{en} - \left(1 - \frac{1}{n}\right)^n\right)^\lambda \\ &\leq \left(1 - \frac{1}{en} - \left(1 - \frac{1}{n}\right)^n\right)^\lambda \\ &\leq \left(1 - \frac{1}{en} - \frac{1}{e}\left(1 - \frac{1}{n}\right)\right)^\lambda \\ &= \left(1 - \frac{1}{e}\right)^\lambda = \left(\frac{e-1}{e}\right)^\lambda. \end{aligned}$$

To prove the bounds on the backward drift from Equation (5.4), note that the drift is conditional on a decrease in fitness, hence the lower bound of 1 is trivial.

The backward drift of a generation with λ offspring can be upper bounded by a generation with only one offspring.

We pessimistically bound the backward drift by the expected number of flipping bits in a standard bit mutation. Under this pessimistic assumption, the condition $f(x_{t+1}) < i$ is equivalent to at least one bit flipping. Let B denote the random number of flipping bits in a standard bit mutation with mutation probability $1/n$, then $\mathrm{E}(B) = 1$, $\Pr(B \geq 1) =$

$1-(1-1/n)^n \geq 1-1/e = (e-1)/e$ and

$$\begin{aligned}\Delta^-_{i,\lambda} \leq \mathrm{E}(B \mid B \geq 1) &= \sum_{x=1}^{\infty} \Pr(B = x \mid B \geq 1) \cdot x \\ &= \sum_{x=1}^{\infty} \frac{\Pr(B=x)}{\Pr(B \geq 1)} \cdot x = \frac{\mathrm{E}(B)}{\Pr(B \geq 1)} \leq \frac{e}{e-1}.\end{aligned}$$

The lower bound on the forward drift, Equation (5.5), is again trivial since the forward drift is conditional on an increase in fitness.

To find the upper bound of $\Delta^+_{i,\lambda}$ we pessimistically assume that all bit flips improve the fitness. Then we use the expected number of bit flips to bound $\Delta^+_{i,\lambda}$. Let B again denote the random number of flipping bits in a standard bit mutation with mutation probability $1/n$, then

$$\Pr(B \geq j) = \binom{n}{j}\left(\frac{1}{n}\right)^j \leq \frac{1}{j!}.$$

To compute $\Delta^+_{i,\lambda}$ we use the probability that any of the λ offspring flip at least j bits as follows:

$$\Delta^+_{i,\lambda} \leq \sum_{j=1}^{\infty}\left(1-\left(1-\frac{1}{j!}\right)^{\lambda}\right).$$

For $\lambda \geq 5$ we bound the first $\lceil \log \lambda \rceil$ summands by 1 and apply Bernoulli's inequality:

$$\begin{aligned}\Delta^+_{i,\lambda} &\leq \lceil \log \lambda \rceil + \sum_{j=\lceil \log \lambda \rceil+1}^{\infty}\left(1-\left(1-\frac{1}{j!}\right)^{\lambda}\right) \\ &\leq \lceil \log \lambda \rceil + \lambda \sum_{j=\lceil \log \lambda \rceil+1}^{\infty} \frac{1}{j!} \\ &\leq \lceil \log \lambda \rceil + 2^{\lceil \log \lambda \rceil} \sum_{j=\lceil \log \lambda \rceil+1}^{\infty} \frac{1}{j!}.\end{aligned}$$

The function $f\colon \mathbb{N} \to \mathbb{R}$ with $f(x) := 2^x \sum_{j=x+1}^{\infty} \frac{1}{j!}$ is decreasing with x and thus for all $\lambda \geq 5$ we get $\Delta^+_{i,\lambda} \leq \lceil \log \lambda \rceil + f(3) = \lceil \log \lambda \rceil + \frac{8}{3}(3e-8) < \lceil \log \lambda \rceil + 0.413$. □

We now show the following lemma that establishes a natural limit to the value of λ.

Lemma 5.3.2. *Consider the self-adjusting* $(1, \lambda)$ EA *on any unimodal function with an initial offspring population size of* $\lambda_0 \leq eF^{1/s}n^3$*. The probability that, during a run, the offspring population size exceeds* $eF^{1/s}n^3$ *before the optimum is found is at most* $\exp(-\Omega(n^2))$.

Proof. In order to have $\lambda_{t+1} \geq eF^{1/s}n^3$, a generation with $\lambda_t \geq en^3$ must be unsuccessful. Since there is always a one-bit flip that improves the fitness and the probability that an offspring flips only one bit is $\frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1} \geq \frac{1}{en}$, then the probability of an unsuccessful generation with $\lambda \geq en^3$ is at most

$$\left(1-\frac{1}{en}\right)^{en^3} \leq \exp(-n^2)$$

The probability of finding the optimum in one generation with any λ and any current fitness is at least $n^{-n} = \exp(-n \ln n)$. Hence the probability of exceeding $\lambda = eF^{1/s}n^3$ before finding the optimum is at most

$$\frac{\exp(-n^2)}{\exp(-n \ln n) + \exp(-n^2)} \leq \frac{\exp(-n^2)}{\exp(-n \ln n)} = \exp(-\Omega(n^2)).$$

□

5.3.1 Small Success Rates are Efficient

We show that, for suitable choices of the success rate s and constant update strength F, the self-adjusting $(1, \lambda)$ EA optimises OneMax in $O(n)$ expected generations and $O(n \log n)$ expected evaluations.

Bounding the Number of Generations

We first only focus on the expected number of generations as the number of function evaluations depends on the dynamics of the offspring population size over time and is considerably harder to analyse. The following theorem states the main result of this section.

Theorem 5.3.3. *Let the update strength $F > 1$ and the success rate $0 < s < 1$ be constants. Then for any initial search point and any initial λ the expected number of generations of the self-adjusting $(1, \lambda)$ EA on OneMax is $O(n)$.*

We make use of the potential function from Definition 5.2.3 and define $h(\lambda)$ to obtain the potential function used in this section as follows.

Definition 5.3.4. We define the potential function $g_1(X_t)$ as

$$g_1(X_t) = f(x_t) - \frac{2s}{s+1} \log_F \left(\max \left(\frac{enF^{1/s}}{\lambda_t}, 1 \right) \right).$$

The definition of $h(\lambda)$ in this case is used as a penalty term that grows linearly in $\log_F \lambda$ (since $-\log_F \left(\frac{enF^{1/s}}{\lambda_t} \right) = -\log_F(enF^{1/s}) + \log_F(\lambda_t)$). That is, when λ increases the penalty decreases and vice-versa. The idea behind this definition is that small values of λ may lead to decreases in fitness, but these are compensated by an increase in λ and a reduction of the penalty term.

Since the range of the penalty term is limited, the potential is always close to the current fitness as shown in the following lemma.

Lemma 5.3.5. *For all generations t, the fitness and the potential are related as follows: $f(x_t) - \frac{2s}{s+1} \log_F(enF^{1/s}) \leq g_1(X_t) \leq f(x_t)$. In particular, $g_1(X_t) = n$ implies $f(x_t) = n$.*

Proof. The penalty term $\frac{2s}{s+1} \log_F \left(\max \left(\frac{enF^{1/s}}{\lambda_t}, 1 \right) \right)$ is a non-increasing function in λ_t with its minimum being 0 for $\lambda \geq enF^{1/s}$ and its maximum being $\frac{2s}{s+1} \log_F \left(enF^{1/s} \right)$ when $\lambda = 1$. Hence, $f(x_t) - \frac{2s}{s+1} \log_F(enF^{1/s}) \leq g_1(X_t) \leq f(x_t)$. □

Now we proceed to show that with the correct choice of hyper-parameters the drift in potential is at least a positive constant during all parts of the optimisation.

Lemma 5.3.6. *Consider the self-adjusting $(1, \lambda)$ EA as in Theorem 5.3.3. Then for every generation t with $f(x_t) < n$,*

$$\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t) \geq \frac{1-s}{2e}.$$

for large enough n. This also holds when only considering improvements that increase the fitness by 1.

Proof. Given that $h(\lambda_t) = -\frac{2s}{s+1}\log_F\left(\max\left(\frac{enF^{1/s}}{\lambda_t}, 1\right)\right)$ is a non-decreasing function, if $\lambda \leq F$ then $h(1) \geq h(\lambda/F)$. Hence, by Lemma 5.2.4, for all λ, $\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t)$ is equal to

$$\left(\Delta^+_{i,\lambda} + h(\lambda/F) - h(\lambda F^{1/s})\right) p^+_{i,\lambda} + h(\lambda F^{1/s}) - h(\lambda) - \Delta^-_{i,\lambda} p^-_{i,\lambda}. \tag{5.9}$$

We first consider the case $\lambda_t \leq en$ as then $\lambda_{t+1} \leq enF^{1/s}$ and $h(\lambda_{t+1}) = -\frac{2s}{s+1}(\log_F(enF^{1/s}) - \log_F(\lambda_{t+1})) < 0$. Hence, $\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t \leq en)$ is at least

$$\begin{aligned}
&= \left(\Delta^+_{i,\lambda} + \frac{2s}{s+1}\log_F\left(\frac{\lambda}{F}\right) - \frac{2s}{s+1}\log_F\left(\lambda F^{1/s}\right)\right) p^+_{i,\lambda} + \frac{2s}{s+1}\log_F\left(\lambda F^{1/s}\right) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad - \frac{2s}{s+1}\log_F(\lambda) - \Delta^-_{i,\lambda} p^-_{i,\lambda} \\
&= \left(\Delta^+_{i,\lambda} - \frac{2s}{s+1}\left(\frac{s+1}{s}\right)\right) p^+_{i,\lambda} + \frac{2s}{s+1}\left(\frac{1}{s}\right) - \Delta^-_{i,\lambda} p^-_{i,\lambda} \\
&= \frac{2}{s+1} + \left(\Delta^+_{i,\lambda} - 2\right) p^+_{i,\lambda} - \Delta^-_{i,\lambda} p^-_{i,\lambda}.
\end{aligned}$$

By Lemma 5.3.1 $\Delta^+_{i,\lambda} \geq 1$, hence $\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t \leq en) \geq \frac{2}{s+1} - p^+_{i,\lambda} - \Delta^-_{i,\lambda} p^-_{i,\lambda}$. Using $\frac{2}{s+1} = \frac{s+1+1-s}{s+1} = 1 + \frac{1-s}{s+1}$ yields

$$\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t \leq en) \geq 1 + \frac{1-s}{s+1} - p^+_{i,\lambda} - \Delta^-_{i,\lambda} p^-_{i,\lambda}$$

By Lemma 5.3.1 this is at least

$$\begin{aligned}
&\frac{1-s}{s+1} + \left(1 - 1.14\left(\frac{n-i}{n}\right)\left(1 - \frac{1}{n}\right)^{n-1}\right)^{\lfloor\lambda\rceil} - \left(\frac{e}{e-1}\right)\left(1 - \frac{n-i}{en} - \left(1 - \frac{1}{n}\right)^n\right)^{\lfloor\lambda\rceil} \\
&\geq \frac{1-s}{s+1} + \left(1 - \frac{1.14}{e\left(1 - \frac{1}{n}\right)}\left(\frac{n-i}{n}\right)\right)^{\lfloor\lambda\rceil} - \left(\frac{e}{e-1}\right)\left(1 - \frac{n-i}{en} - \frac{1}{e}\left(1 - \frac{1}{n}\right)\right)^{\lfloor\lambda\rceil} \\
&= \frac{1-s}{s+1} + \left(1 - \frac{1.14}{e}\left(\frac{n-i}{n-1}\right)\right)^{\lfloor\lambda\rceil} - \left(\frac{e}{e-1}\right)\left(\frac{e-1}{e} - \frac{n-i-1}{en}\right)^{\lfloor\lambda\rceil}.
\end{aligned} \tag{5.10}$$

We start taking into account only $\lfloor\lambda\rceil \geq 2$, that is, $\lambda \geq 1.5$ and later on we will deal with $\lfloor\lambda\rceil = 1$. For $\lfloor\lambda\rceil \geq 2$, $\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t, 1.5 \leq \lambda_t \leq en)$ is at least

$$\begin{aligned}
&\frac{1-s}{s+1} + \left(1 - \frac{1.14}{e}\left(\frac{n-i}{n-1}\right)\right)^{\lfloor\lambda\rceil} - \left(\frac{e}{e-1}\right)^{\lfloor\lambda\rceil/2}\left(\frac{e-1}{e} - \frac{n-i-1}{en}\right)^{\lfloor\lambda\rceil} \\
&= \frac{1-s}{s+1} + \underbrace{\left(1 - \frac{1.14}{e}\left(\frac{n-i}{n-1}\right)\right)}_{y_1}{}^{\lfloor\lambda\rceil} - \underbrace{\left(\left(\frac{e-1}{e}\right)^{1/2} - \frac{n-i-1}{(e^2-e)^{1/2}n}\right)}_{y_2}{}^{\lfloor\lambda\rceil}
\end{aligned}$$

Let y_1 and y_2 be the respective bases of the terms raised to $\lfloor\lambda\rceil$ as indicated above. We will now prove that $y_1 \geq y_2$ for all $0 \leq i < n$ which implies that $\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t, 1.5 \leq \lambda_t \leq en) \geq \frac{1-s}{s+1} \geq \frac{1-s}{2e}$.

The terms y_1 and y_2 can be described by linear equations $y_1 = m_1(n-i) + b_1$ and $y_2 = m_2(n-i) + b_2$ with $m_1 = -\frac{1.14}{e(n-1)}$, $b_1 = 1$, $m_2 = -\frac{1}{n\sqrt{e^2-e}}$ and $b_2 = \sqrt{\frac{e-1}{e}} + \frac{1}{n\sqrt{e^2-e}}$. Since $m_2 < m_1$ for all $n \geq 11$, the difference $y_1 - y_2$ is minimised for $n - i = 1$. When

$n-i=1$, then $y_1 = 1 - \frac{1.14}{e(n-1)} > \left(\frac{e-1}{e}\right)^{1/2} = y_2$ for all $n > 3$, therefore $y_1 > y_2$ for all $0 \le i < n$.

When $\lfloor \lambda \rceil = 1$, from Equation (5.10) $\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t \le 1.5) \ge \frac{1-s}{s+1} - \frac{1.14}{e}\left(\frac{n-i}{n-1}\right) + \frac{n-i-1}{(e-1)n}$ which is monotonically decreasing for $0 < i < n-1$ when $n > e/(1.14 - 0.14e) \approx 3.58$, hence $\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t \le 1.5) \ge \frac{1-s}{s+1} - \frac{1.14}{e(n-1)}$ which is bounded by $\frac{1-s}{2e}$ for large enough n.

Finally, for the case $\lambda_t > en$, in an unsuccessful generation the penalty term is capped, hence $h(\lambda F^{1/s}) = h(\lambda)$. Then by Equation (5.9), $\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t > en)$ is at least

$$\begin{aligned}
&\left(\Delta^+_{i,\lambda} + h(\lambda/F) - h(\lambda)\right) p^+_{i,\lambda} - \Delta^-_{i,\lambda} p^-_{i,\lambda} \\
&= \left(\Delta^+_{i,\lambda} + \frac{2s}{s+1}\log_F\left(\frac{\lambda}{F}\right) - \frac{2s}{s+1}\log_F(\lambda)\right) p^+_{i,\lambda} - \Delta^-_{i,\lambda} p^-_{i,\lambda} \\
&= \left(\Delta^+_{i,\lambda} - \frac{2s}{s+1}\right) p^+_{i,\lambda} - \Delta^-_{i,\lambda} p^-_{i,\lambda}
\end{aligned}$$

By Lemma 5.3.1, $\lambda_t > en$ implies $p^+_{i,\lambda} \ge 1 - \left(1 - \frac{1}{en}\right)^{en} \ge 1 - \frac{1}{e}$ and $p^-_{i,\lambda}\Delta^-_{i,\lambda} \le \left(\frac{e-1}{e}\right)^{en} \frac{e}{e-1} = \left(\frac{e-1}{e}\right)^{en-1}$. Together,

$$\begin{aligned}
&\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t > en) \\
&\ge \left(\Delta^+_{i,\lambda} - \frac{2s}{s+1}\right) p^+_{i,\lambda} - \Delta^-_{i,\lambda} p^-_{i,\lambda} \\
&\ge \left(1 - \frac{1}{e}\right)\left(1 - \frac{2s}{s+1}\right) - \left(\frac{e-1}{e}\right)^{en-1}.
\end{aligned}$$

Since $\left(1 - \frac{1}{e}\right) = \frac{1}{e} + \left(1 - \frac{2}{e}\right)$ and $\left(1 - \frac{2}{e}\right)\left(1 - \frac{2s}{s+1}\right)$ is a positive constant, for large enough n this is larger than $\left(\frac{e-1}{e}\right)^{en-1}$ and

$$\mathrm{E}(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t > en) \ge \frac{1}{e}\left(1 - \frac{2s}{s+1}\right) = \frac{1}{e}\left(\frac{1-s}{s+1}\right) \ge \frac{1-s}{2e}.$$

Since $s < 1$, this is a strictly positive constant. □

With this constant lower bound on the drift of the potential, the proof of Theorem 5.3.3 is now quite straightforward.

Proof of Theorem 5.3.3. We bound the time to get to the optimum using the potential function $g_1(X_t)$. Lemma 5.3.6 shows that the potential has a positive constant drift whenever the optimum has not been found, and by Lemma 5.3.5 if $g_1(X_t) = n$ then the optimum has been found. Therefore, we can bound the number of generations by the time it takes for $g_1(X_t)$ to reach n.

To fit the perspective of the additive drift theorem (Theorem 2.4.14) we switch to the function $\overline{g_1}(X_t) := n - g_1(X_t)$ and note that $\overline{g_1}(X_t) = 0$ implies that $g_1(X_t) = f(x_t) = n$. The initial value $\overline{g_1}(X_0)$ is at most $n + \frac{2s}{s+1}\log_F\left(enF^{1/s}\right)$ by Lemma 5.3.5. Using Lemma 5.3.6 and the additive drift theorem, the expected number of generations is at most

$$\frac{n + \frac{2s}{s+1}\log_F\left(enF^{1/s}\right)}{\frac{1-s}{2e}} = O(n).$$

□

Bounding the Number of Evaluations

A bound on the number of generations, by itself, is not sufficient to claim that the self-adjusting $(1, \lambda)$ EA is efficient in terms of the number of evaluations. Obviously, the number of evaluations in generation t equals λ_t and this quantity is being self-adjusting over time. So we have to study the dynamics of λ_t more carefully. Since λ grows exponentially in unsuccessful generations, it could quickly attain very large values. However, we show that this is not the case and only $O(n \log n)$ evaluations are sufficient, in expectation.

Theorem 5.3.7. *Let the update strength $F > 1$ and the success rate $0 < s < 1$ be constants. The expected number of function evaluations of the self-adjusting $(1, \lambda)$ EA on* OneMax *is $O(n \log n)$.*

Bounding the number of evaluations is more challenging than bounding the number of generations as we need to keep track of the offspring population size λ and how it develops over time. Large values of λ lead to a large number of evaluations made in one generation. Small values of λ can lead to a fallback.

If our algorithm was elitist, small values of λ would not be an issue since there would be no fallbacks. We will show later on (Theorem 5.3.12) that then the algorithm will spend $O(n \log n)$ evaluations, refining the amortised analysis from Lässig and Sudholt [120]. This analysis relies on every fitness level being visited at most once. In our non-elitist algorithm, this is not guaranteed. Small values of λ can lead to decreases in fitness, and then the same fitness level can be visited multiple times.

The reader may think that small values of λ only incur few evaluations and that the additional cost for a fallback is easily accounted for. However, it is not that simple. Imagine a fitness level i and a large value of λ such that a fallback is unlikely. But it is possible for λ to decrease in a sequence of improving steps. Then we would have a small value of λ and possibly a sequence of fitness-decreasing steps. Suppose the fitness decreases to a value at most i, then if λ returns to a large value, we may have visited fitness level i multiple times, with large (and costly) values of λ.

It is possible to show that, for sufficiently challenging fitness levels, λ moves towards an equilibrium state, i. e. when λ is too small, it tends to increase. However, this is generally not enough to exclude drops in λ. Since λ is multiplied or divided by a constant factor in each step, a sequence of k improving steps decreases λ by a factor of F^k, which is exponential in k. For instance, a value of $\lambda = \log^{O(1)} n$ can decrease to $\lambda = \Theta(1)$ in only $O(\log \log n)$ generations. We found that standard techniques such as the negative drift theorem, applied to $\log_F(\lambda_t)$, are not strong enough to exclude drops in λ.

We solve this problem as follows. We consider the best-so-far fitness $f_t^* = \max\{f(x_{t'}) \mid 0 \le t' \le t\}$ at time t (as a theoretical concept, as the self-adjusting $(1, \lambda)$ EA is non-elitist and unaware of the best-so-far fitness) and use drift arguments from Section 5.3.1, and the negative drift theorem (Theorem 2.4.16) to show that, with high probability, the current fitness never drops far below f_t^*, that is, $f(x_t) \ge f_t^* - \rho \log n$ for a constant $\rho > 0$. This yields what we call a *ratchet argument*[1]: if the best-so-far fitness increases, the lower bound on the current fitness increases as well. The lower bound thus works like a ratchet mechanism that can only move in one direction.

It remains to show that the best-so-far fitness increases efficiently. We divide the run into fitness intervals of size $\log n$ that we call *blocks*, and bound the time for the best-so-far fitness to reach a better block. This task is easier than bounding the time to go all the way to the optimum since it is sufficient to have a sequence of generations in which λ maintains large enough values (i. e. $\lambda \ge 4 \log n$) such that with high probability the fitness does not decrease and the self-adjusting $(1, \lambda)$ EA temporarily behaves like an elitist algorithm. We

[1]This name is inspired by the term "Muller's ratchet" from biology [80] that considers a ratchet mechanism in asexual evolution, albeit in a different context.

show that, starting from an arbitrary λ-value, the probability of having such a period of generations with temporary elitism is $\Omega(1)$ and that the expected time to reach the next block is only by at most a constant factor larger than that of an elitist algorithm. Adding up expected times for each block then yields the claimed $O(n \log n)$ bound.

Understanding Search Dynamics in the Self-Adjusting $(1, \lambda)$ EA

As a first step, we need to gain a better understanding of the search dynamics and the development of λ over time. We first re-use the potential drift arguments from the proof of Theorem 5.3.3 to show that the number of *generations* to increase the current fitness to a new block is bounded as follows. For $b = a + \log n$, this bound is $O(\log n)$.

Lemma 5.3.8. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 5.3.7. For every* $a, b \in \{0, \dots, n\}$*, the expected number of generations to increase the current fitness from a value at least* a *to at least* b *is at most*

$$\frac{b - a + \frac{2s}{s+1} \log_F \left(enF^{1/s}\right)}{\frac{1-s}{2e}} = O(b - a + \log n).$$

Proof. We use the proof of Theorem 5.3.3 with a revised potential function of $\overline{g_1}'(X_t) := \max(\overline{g_1}(X_t) - (n - b), 0)$ and stopping when $\overline{g_1}'(X_t) = 0$ (which implies that a fitness of at least b is reached) or a fitness of at least b is reached beforehand. Note that the maximum caps the effect of fitness improvements that jump to fitness values larger than b. As remarked in Lemma 5.3.6, the drift bound for $g_1(X_t)$ still holds when only considering fitness improvements by 1. Hence, it also holds for $\overline{g_1}'(X_t)$ and the analysis goes through as before. $\square$

Now we show our ratchet argument and that with high probability the fitness does not decrease when $\lambda \geq 4 \log n$.

Lemma 5.3.9. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 5.3.7. Let* $f_t^* := \max_{t' \leq t} f(x_{t'})$ *be its best-so-far fitness at generation* t *and let* T *be the first generation in which the optimum is found. Then with probability* $1 - O(1/n)$ *the following statements hold for a large enough constant* $\rho > 0$ *(that may depend on* s*).*

1. *For all* $t \leq T$ *in which* $\lambda_t \geq 4 \log n$*, we have* $f(x_{t+1}) \geq f(x_t)$.
2. *For all* $t \leq T$*, the fitness is at least:* $f(x_t) \geq f_t^* - \rho \log n$.

Proof. Let E_1^t denote the event that $\lambda_t < 4 \log n$ or $f(x_{t+1}) \geq f(x_t)$. Hence we only need to consider λ_t-values of $\lambda_t \geq 4 \log n \geq 2 \log_{\frac{e}{e-1}} n$ and by Lemma 5.3.1 we have

$$\Pr\left(\overline{E_1^t}\right) \leq \left(\frac{e-1}{e}\right)^{\lambda_t} \leq \left(\frac{e-1}{e}\right)^{2 \log_{\frac{e}{e-1}} n} = \frac{1}{n^2}.$$

By a union bound, the probability that this happens in the first T generations is at most $\sum_{t=1}^{\infty} \Pr(T = t) \cdot t / n^2 = \mathrm{E}(T)/n^2 = O(1/n)$. For the second statement, let t^* be a generation in which the best-so-far fitness was attained: $f(x_{t^*}) = f_t^*$. By Lemma 5.3.5, abbreviating $\alpha := \frac{2s}{s+1} \log_F(enF^{1/s})$, the condition $f(x_{t^*}) \geq f(x_t) + \rho \log n$ implies $g_1(X_{t^*}) \geq f(x_{t^*}) - \alpha \geq f(x_t) - \alpha + \rho \log n \geq g_1(X_t) - \alpha + \rho \log n$.

Now define events $E_2^t = (\forall t' \in [t+1, n^2] \colon g_1(X_{t'}) \geq g_1(X_t) + \alpha - \rho \log(n))$. We apply the negative drift theorem (Theorem 2.4.16) to bound $\Pr\left(\overline{E_2^t}\right)$ from above. For any $t < n^2$ let $a := g_1(X_t) - \rho \log n + \alpha$ and $b := g_1(X_t) < n$, where $\rho > \alpha$ will be chosen later on. We pessimistically assume that the fitness component of g_1 can only increase by at most 1. Lemma 5.3.6 has already shown that, even under this assumption, the drift is at least a positive constant. This implies the first condition of Theorem 2.4.16. For the second condition, we need to bound transition probabilities for the potential. Owing to our pessimistic assumption, the current fitness can only increase by at most 1. The fitness only decreases by j if *all* offspring are worse than their parent by at least j. Hence, for all λ, the decrease in fitness is bounded by the decrease in fitness of the *first* offspring. The probability of the first offspring decreasing fitness by at least j is bounded by the probability that j bits flip, which is in turn bounded by $1/(j!) \leq 2/2^j$. The possible penalty in the definition of g_1 changes by at most $\max\left(\frac{se}{e-1}, \frac{se}{e-1} \cdot \frac{1}{s}\right) = \frac{e}{e-1} < 1$. Hence, for all t,

$$\Pr\left(|g_1(X_{t-1}) - g_1(X_t)| \geq j+1 \mid g_1(X_t) > a\right) \leq \frac{4}{2^{j+1}},$$

which meets the second condition of Theorem 2.4.16. It then states that there is a constant c^* such that the probability that within $2^{c^*(a-b)/4}$ generations a potential of at most a is reached, starting from a value of at least b, is $2^{-\Omega(a-b)}$. By choosing the constant ρ large enough, we can scale up $a-b$ and thus make $2^{c^*(a-b)/4} \geq n^2$ and $2^{-\Omega(a-b)} = O(1/n^2)$. This yields $\Pr\left(\overline{E_2^t}\right) = O(1/n^2)$.

Arguing as before, the probability that $\overline{E_2^t}$ happens during $O(n)$ expected generations is $O(1/n)$. By Markov's inequality, the probability of not finding the optimum in n^2 generations is $O(1/n)$ as well. Adding up all failure probabilities completes the proof. □

The following lemma bounds the probability of increasing a small λ value to a desired one from below.

Lemma 5.3.10. *Consider the self-adjusting* $(1,\lambda)$ EA *as in Theorem 5.3.7 and assume that the typical events stated in Lemma 5.3.9 occur. Let* $i := f(x_t)$ *and* ρ *be the constant from Lemma 5.3.9. Then for all* $\lambda_{\text{new}} \geq \lambda_{\text{init}} \geq 1$ *the probability that* λ *is increased from* λ_{init} *to* λ_{new} *in a sequence of* $s\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})$ *non-improving generations is at least*

$$1 - \frac{\lambda_{\text{new}} p^+_{i-\rho\log n}}{F^{1/s}-1}.$$

***Proof*.** While no improvement is found, λ is multiplied by $F^{1/s}$ in every iteration. Then λ reaches a value of λ_{new} from λ_{init} in $\gamma := s\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})$ iterations (since $\lambda_{\text{init}} \cdot F^{s\log_F(\lambda_{\text{new}}/\lambda_{\text{init}})/s} = \lambda_{\text{new}}$). The number of evaluations made during this time is at most

$$\begin{aligned}\sum_{j=0}^{\gamma-1} \lambda_{\text{init}} \cdot F^{j/s} = \lambda_{\text{init}} \cdot \sum_{j=0}^{\gamma-1} (F^{1/s})^j &= \lambda_{\text{init}} \cdot \frac{(F^{1/s})^\gamma - 1}{F^{1/s}-1}\\ &= \lambda_{\text{init}} \cdot \frac{\lambda_{\text{new}}/\lambda_{\text{init}} - 1}{F^{1/s}-1} \leq \frac{\lambda_{\text{new}}}{F^{1/s}-1}.\end{aligned}$$

Since, owing to Lemma 5.3.9, the fitness is always at least $i - \rho\log n$ and p_i^+ is non-increasing in i, the probability of an improvement during any offspring creation is at most $p^+_{i-r\log n}$. By a union bound, the probability of having an improvement during $\frac{\lambda_{\text{new}}}{F^{1/s}-1}$ mutations of a search point with fitness i is at most $\frac{\lambda_{\text{new}} p^+_{i-\rho\log n}}{F^{1/s}-1}$. □

Now we bound the probability of returning to a small value of λ.

Lemma 5.3.11. *Consider the self-adjusting* $(1,\lambda)$ EA *as in Theorem 5.3.7 and assume that the typical events stated in Lemma 5.3.9 occur. If the current fitness is* $f(x_t) \geq n - n/\log^3 n$ *and* $\lambda_t \geq 4\log^3 n$*, then, for every constant* $C > 0$*, the probability that within* $C\log n$ *iterations a* λ*-value of at most* $4\log n$ *is reached is at most* $(\log n)^{-\omega(1)}$.

Proof. We assume that $\lambda_t < 4F\log^3 n$ as otherwise we can wait until λ drops below $4F\log^3 n$ (or $C\log n$ generations have passed).

By Lemma 5.3.9, while $\lambda \geq 4\log n$ the current fitness does not decrease. Consequently, the probability of an offspring finding an improvement is always bounded from above by $p_i^+ \leq \frac{1}{\log^3 n}$.

For $k := 2\log_F(\log n)$ we have $4\log^3(n)/F^k = 4\log n$. A necessary condition to decrease λ from a value at least $4\log^3 n$ to a value below $4\log n$ is that for every $j \in \{0,\dots,k-1\}$ there exist generations in which an improvement is found while $\lambda \leq 4\log^3(n)/F^j$. Then, by a union bound, the probability of finding an improvement is at most $4\log^3(n)/F^j \cdot p_i^+$. There are $\binom{C\log n}{k}$ ways to choose these generations. Once these are fixed, the joint probability of having improvements in all these iterations is at most

$$\begin{aligned}\prod_{j=0}^{k-1}\left(\frac{4F\log^3 n}{F^j}\cdot p_i^+\right) &\leq \left(4F\log^3(n)p_i^+\right)^k F^{-\sum_{j=0}^{k-1} j}\\ &\leq (4F)^k F^{-k(k-1)/2}.\end{aligned}$$

Along with a factor of $\binom{C\log n}{k} \leq (eC\log(n)/k)^k$ we bound the sought probability as

$$\begin{aligned}&\left(\frac{\lambda_t p_i^+ eC\log n}{kF^{(k-1)/2}}\right)^k \leq \left(\frac{4F\log^3(n)\cdot 1/(\log^3 n)\cdot eC\log n}{kF^{-1/2}\log n}\right)^k\\ =&\left(\frac{4eCF^{3/2}}{k}\right)^k = (\log\log n)^{-\Omega(\log\log n)} = (\log n)^{-\Omega(\log\log\log n)}.\end{aligned}$$

□

Analysing the Elitist Self-Adjusting (1 + λ) EA

While the fitness does not decrease, the self-adjusting $(1,\lambda)$ EA behaves like an elitist algorithm, i. e. a self-adjusting $(1+\lambda)$ EA. We now consider this elitist algorithm bound its expected time to increase the fitness from a to b.

Theorem 5.3.12. *Consider the elitist self-adjusting* $(1+\lambda)$ EA *on any function with* $d+1$ *fitness values, starting with a fitness of* $f(x_0) \geq a$. *Let* s_i *be a lower bound on the probability of one offspring finding an improvement from any search point in fitness level* i. *For every integer* $b \leq d$, *the expected number of evaluations to reach a fitness of at least* b *is at most*

$$\lambda_0 \cdot \frac{F}{1-F} + \left(\frac{1}{e} + \frac{1-F^{-1/s}}{\ln(F^{1/s})}\right)\cdot\frac{F^{\frac{s+1}{s}}-1}{F-1}\sum_{i=a}^{b-1}\frac{1}{s_i}.$$

For OneMax *this is at most,*

$$\lambda_{\text{init}} \cdot \frac{F}{1-F} + \left(\frac{1}{e} + \frac{1-F^{-1/s}}{\ln(F^{1/s})}\right)\cdot\frac{F^{\frac{s+1}{s}}-1}{F-1}\sum_{i=a}^{b-1}\frac{en}{n-i}.$$

Note that the term $\sum_{i=a}^{b-1}\frac{1}{s_i}$ represents an upper bound for the expected time for the (1+1) EA on any function with $d+1$ fitness values, starting with a fitness of a, to reach a fitness of at least b obtained via the fitness-level method (where the highest fitness level contains all search points with fitness at least b). The bound from Theorem 5.3.12 is thus only by a constant factor and an additive term of $\lambda_{\text{init}}\cdot\frac{F}{1-F}$ larger.

We further remark that Theorem 5.3.12 immediately implies the following.

Corollary 5.3.13. *The expected number of function evaluations of the elitist self-adjusting* $(1+\lambda)$ EA *using any constant parameters* $s > 0, F > 1$ *and* $\lambda_0 = O(n \log n)$ *on* OneMax *is* $O(n \log n)$*.*

To prove Theorem 5.3.12, we argue that the analysis can be boiled down to the number of evaluations made *in unsuccessful generations.* This is made precise in the following lemma.

Lemma 5.3.14. *Consider the elitist self-adjusting* $(1+\lambda)$ EA *on every fitness function with* $f(x_0) \geq a$ *and* $\lambda_0 = \lambda_{\text{init}}$*. Fix an integer* $b \leq d$ *and, for all* $0 \leq i \leq d-1$*, let* U_i *denote the number of function evaluations made during all unsuccessful generations on fitness level* i*. Then the number of evaluations to reach a fitness of at least* b *is at most*

$$\lambda_{\text{init}} \cdot \frac{F}{1-F} + \frac{F^{\frac{s+1}{s}} - 1}{F-1} \sum_{i=a}^{b-1} U_i.$$

A bound on the expected *number of evaluations is obtained by replacing* U_i *with* $\mathrm{E}(U_i)$*.*

Proof. We refine the *accounting method* used in the analysis of the $(1 + \{2\lambda, \lambda/2\})$ EA in [166]. The main idea is: if some fitness level i increases λ to a large value, we charge the costs for increasing λ to that fitness level. In addition, we charge costs that pay for decreasing λ down to 1 in future successful generations. Hence, a successful generation comes for free, at the expense of a constant factor for the cost of unsuccessful generations.

Let $\phi(\lambda_t) := \frac{F}{F-1}\lambda_t$ and imagine a fictional bank account. We make an initial payment of $\phi(\lambda_{\text{init}})$ to that bank account. If a generation t is unsuccessful, we pay λ_t for the current generation and deposit an additional amount of $\phi(\lambda_t F^{1/s}) - \phi(\lambda_t)$. If an improvement is found, we withdraw an amount of λ_t to pay for generation t.

We show by induction: for every generation t, the account's balance is $\phi(\lambda_t)$. This is true for the initial generation owing to the initial payment of $\phi(\lambda_{\text{init}})$. Assume the statement holds for time t. If generation t is unsuccessful, the new balance is

$$\phi(\lambda_t) + (\phi(\lambda_t F^{1/s}) - \phi(\lambda_t)) = \phi(\lambda_t F^{1/s}) = \phi(\lambda_{t+1}).$$

If generation t is successful, $\lambda_{t+1} = \lambda_t / F$ and the new balance is

$$\phi(\lambda_t) - \lambda_t = \left(\frac{F}{F-1} - 1\right)\lambda_t = \frac{1}{F-1} \cdot \lambda_t = \frac{F}{F-1} \cdot \lambda_{t+1} = \phi(\lambda_{t+1}).$$

Now the costs of an unsuccessful generation t are

$$\lambda_t + \phi(\lambda_t F^{1/s}) - \phi(\lambda_t) = \lambda_t \left(1 + \frac{F^{\frac{s+1}{s}}}{F-1} - \frac{F}{F-1}\right) = \lambda_t \cdot \frac{F^{\frac{s+1}{s}} - 1}{F-1},$$

that is, by a factor of $\frac{F^{\frac{s+1}{s}}-1}{F-1}$ larger than the number of evaluations in that generation. Recall that successful generations incur no costs as the additional factor in unsuccessful generation has already paid for these evaluations. This implies that, if U_i denotes the number of evaluations during all unsuccessful generations on fitness level i, the costs incurred on fitness level i are $U_i \cdot \frac{F^{\frac{s+1}{s}}-1}{F-1}$. Summing up over all non-optimal i and adding costs $\phi(\lambda_{\text{init}}) = \lambda_{\text{init}} \cdot \frac{F}{F-1}$ to account for the initial value of λ yields the claimed bound.

The final statement on the *expected* number of evaluations follows from taking expectations and exploiting linearity of expectations. □

The next step is to bound $\mathrm{E}(U_i)$, that is, the expected number of evaluations *in unsuccessful generations* on an arbitrary but fixed fitness level i.

Lemma 5.3.15. *Consider the self-adjusting* $(1 + \lambda)$ EA *on every fitness function starting on fitness level i with an offspring population size of* λ*. For every initial* λ*, the expected number of evaluations in unsuccessful generations for the self-adjusting* $(1 + \lambda)$ EA *on fitness level i is at most*

$$\mathrm{E}(U_i) \le \frac{1}{s_i} \cdot \left(\frac{1}{e} + \frac{1 - F^{-1/s}}{\ln(F^{1/s})} \right).$$

Note that the bound from Lemma 5.3.15 does not depend on the initial value of λ. Roughly speaking, for small values of λ, the algorithm will typically increase λ to a value where improvements become likely, and then the number of evaluations essentially depends on the difficulty of the fitness level, expressed through the factor of $\frac{1}{s_i}$ from the lemma's statement. If λ is larger than required, with high probability the first generation is successful and then there are no evaluations in an unsuccessful generation on fitness level i.

In order to prove Lemma 5.3.15, we first need two technical lemmas.

Lemma 5.3.16. *Let* $f\colon \mathbb{R}_0^+ \to \mathbb{R}_0^+$ *be an integrable function with a unique maximum at* α*. Then*

$$\sum_{i=0}^{\infty} f(i) \le f(\alpha) + \int_0^{\infty} f(i) \,\mathrm{d}i.$$

Proof. We assume that $\int_0^\infty f(i)\,\mathrm{d}i < \infty$ as otherwise the claim is trivial. Since f is non-decreasing in $[0, \lfloor\alpha\rfloor]$, for all $i = 0, \dots, \lfloor\alpha\rfloor - 1$ we have $f(i) \le \int_i^{i+1} f(i)\,\mathrm{d}i$. This yields

$$\sum_{i=0}^{\lfloor\alpha\rfloor} f(i) \le \int_0^{\lfloor\alpha\rfloor} f(i)\,\mathrm{d}i + f(\lfloor\alpha\rfloor).$$

Likewise, since f is non-increasing in $[\lfloor\alpha\rfloor + 1, \infty)$, for all $i = \lfloor\alpha\rfloor + 2, \dots$ we have $f(i) \le \int_{i-1}^{i} f(i)\,\mathrm{d}i$. This yields

$$\sum_{i=\lfloor\alpha\rfloor+1}^{\infty} f(i) \le \int_{\lfloor\alpha\rfloor+1}^{\infty} f(i)\,\mathrm{d}i + f(\lfloor\alpha\rfloor).$$

Assume $f(\lfloor\alpha\rfloor) \le f(\lfloor\alpha\rfloor + 1)$, then $f(i) \ge f(\lfloor\alpha\rfloor)$ for all $i \in [\lfloor\alpha\rfloor, \lfloor\alpha\rfloor + 1]$ and thus $f(\lfloor\alpha\rfloor) \le \int_{\lfloor\alpha\rfloor}^{\lfloor\alpha\rfloor+1} f(i)\,\mathrm{d}i$. This implies

$$f(\lfloor\alpha\rfloor) + f(\lfloor\alpha\rfloor + 1) \le f(\alpha) + \int_{\lfloor\alpha\rfloor}^{\lfloor\alpha\rfloor+1} f(i)\,\mathrm{d}i.$$

The case $f(\lfloor\alpha\rfloor) > f(\lfloor\alpha\rfloor + 1)$ is symmetric and leads to the same statement. Together, this implies the claim. $\square$

We will also need a closed form for the following integral.

Lemma 5.3.17.

$$\int_{j=0}^{\infty} \alpha^j \cdot \exp\left(-\beta \cdot \alpha^j\right)\,\mathrm{d}j = \frac{e^{-\beta}}{\beta \ln(\alpha)}.$$

Proof. The integral can be written as

$$\frac{1}{\ln(\alpha)} \int_{j=0}^{\infty} \alpha^j \ln(\alpha) \cdot \exp\left(-\beta \cdot \alpha^j\right)\,\mathrm{d}i.$$

For $f(x) := \exp(-\beta x)$ and $\varphi(x) = \alpha^x$, along with $\varphi'(x) = \alpha^x \ln(\alpha)$, this is

$$\frac{1}{\ln(\alpha)} \int_{j=0}^{\infty} \varphi'(x) \cdot f(\varphi(x))\,\mathrm{d}x.$$

The rule of integration by substitution states that $\int_a^b \varphi'(x) \cdot f(\varphi(x))\ \mathrm{d}x = \int_{\varphi(a)}^{\varphi(b)} f(u)\ \mathrm{d}u$. Using $\int \exp(-\beta u)\ \mathrm{d}u = -\frac{e^{-\beta u}}{\beta}$, the above is

$$\frac{1}{\ln(\alpha)} \cdot \lim_{b\to\infty} \int_{\varphi(0)}^{\varphi(b)} f(u)\ \mathrm{d}u = \frac{1}{\ln(\alpha)} \cdot \lim_{u\to\infty} \frac{e^{-\beta} - e^{-\beta u}}{\beta} = \frac{1}{\ln(\alpha)} \cdot \frac{e^{-\beta}}{\beta}.$$

□

Now we use these technical lemmas to prove Lemma 5.3.15.

Proof of Lemma 5.3.15. Since we are considering an elitist algorithm, fitness level i is left for good once we have a success from a current search point on level i. Starting with an offspring population size of λ, if generations $0, \dots, j$ are unsuccessful, the j-th unsuccessful generation has an offspring population size of $\lambda F^{j/s}$. However, it is only counted in the expectation we are aiming to bound if there is no success in generations $0, \dots, j$. There is no success in generations $0, \dots, j$ if and only if the algorithm makes $\lambda \sum_{\ell=0}^{j} F^{\ell/s} = \lambda \frac{F^{\frac{j+1}{s}}-1}{F^{1/s}-1}$ evaluations without generating an improvement. Hence the expected number of evaluations in unsuccessful generations is equal to

$$\begin{aligned} &\sum_{j=0}^{\infty} \lambda F^{j/s} \cdot \Pr\left(\text{no success in } \lambda \cdot \frac{F^{\frac{j+1}{s}}-1}{F^{1/s}-1} \text{ evaluations}\right) \\ &= \sum_{j=0}^{\infty} \lambda F^{j/s} \cdot (1-s_i)^{\lambda \cdot \frac{F^{\frac{j+1}{s}}-1}{F^{1/s}-1}} \\ &\leq \sum_{j=0}^{\infty} \lambda F^{j/s} \cdot \exp\left(-s_i \lambda \cdot \frac{F^{\frac{j+1}{s}}-1}{F^{1/s}-1}\right). \end{aligned}$$

We will use Lemma 5.3.16 to bound the above sum by an integral and the maximum of the function $\lambda F^{j/s} \cdot \exp\left(-s_i\lambda \cdot \frac{F^{\frac{j+1}{s}}-1}{F^{1/s}-1}\right)$.

We define the (simpler) function $\xi(x) := x \cdot \exp(-s_i x)$ and note that its maximum value is $1/(es_i)$. This value is an upper bound for the sought maximum since $\sum_{\ell=0}^{j} F^{\ell/s} = \frac{F^{\frac{j+1}{s}}-1}{F^{1/s}-1}$ implies $\frac{F^{\frac{j+1}{s}}-1}{F^{1/s}-1} \geq F^{j/s}$ and thus

$$\lambda F^{j/s} \cdot \exp\left(-s_i\lambda \cdot \frac{F^{\frac{j+1}{s}}-1}{F^{1/s}-1}\right) \leq \lambda F^{j/s} \cdot \exp\left(-s_i \lambda F^{j/s}\right) = \xi(\lambda F^{j/s}) \leq \frac{1}{es_i}.$$

Plugging this into the bound obtained by invoking Lemma 5.3.16, the sum is thus at most

$$\begin{aligned} &\frac{1}{es_i} + \int_{j=0}^{\infty} \lambda F^{j/s} \cdot \exp\left(-s_i\lambda \cdot \frac{F^{\frac{j+1}{s}}-1}{F^{1/s}-1}\right)\ \mathrm{d}i \\ &= \frac{1}{es_i} + \exp\left(\frac{s_i\lambda}{F^{1/s}-1}\right) \int_{j=0}^{\infty} \lambda F^{j/s} \cdot \exp\left(-s_i\lambda \cdot \frac{F^{\frac{j+1}{s}}}{F^{1/s}-1}\right)\ \mathrm{d}i \qquad (5.11) \end{aligned}$$

Let $\alpha := F^{1/s}$ and $\beta := s_i\lambda \cdot \frac{F^{1/s}}{F^{1/s}-1}$, then we can write

$$\int_{j=0}^{\infty} \lambda F^{j/s} \cdot \exp\left(-s_i\lambda \cdot \frac{F^{\frac{j+1}{s}}}{F^{1/s}-1}\right)\ \mathrm{d}i = \lambda \int_{j=0}^{\infty} \alpha^j \cdot \exp(-\beta \cdot \alpha^j).$$

By Lemma 5.3.17, this equals

$$\lambda \cdot \frac{e^{-\beta}}{\beta \ln(\alpha)} = \frac{\exp\left(-s_i\lambda \cdot \frac{F^{1/s}}{F^{1/s}-1}\right)}{s_i \cdot \frac{F^{1/s}}{F^{1/s}-1} \ln\left(F^{1/s}\right)} = \frac{\exp\left(-s_i\lambda \cdot \frac{F^{1/s}}{F^{1/s}-1}\right)(F^{1/s}-1)}{s_i \cdot F^{1/s} \ln\left(F^{1/s}\right)}.$$

Plugging this back into (5.11) yields

$$\begin{aligned}
&\frac{1}{es_i} + \exp\left(\frac{s_i\lambda}{F^{1/s}-1}\right) \cdot \frac{\exp\left(-s_i\lambda \cdot \frac{F^{1/s}}{F^{1/s}-1}\right)(F^{1/s}-1)}{s_i \cdot F^{1/s} \ln\left(F^{1/s}\right)} \\
={}& \frac{1}{es_i} + \exp\left(\frac{s_i\lambda}{F^{1/s}-1} \cdot (1-F^{1/s})\right) \cdot \frac{F^{1/s}-1}{s_i \cdot F^{1/s} \ln\left(F^{1/s}\right)} \\
={}& \frac{1}{es_i} + \exp\left(-s_i\lambda\right) \cdot \frac{F^{1/s}-1}{s_i \cdot F^{1/s} \ln\left(F^{1/s}\right)}.
\end{aligned}$$

In this upper bound, we can see that the worst value for λ is $\lambda = 1$. Using $s_i\lambda \geq 0$ for all λ and thus bounding $\exp(-s_i\lambda) \leq 1$, we get an upper bound of

$$\frac{1}{es_i} + \frac{F^{1/s}-1}{s_i F^{1/s} \ln(F^{1/s})} = \frac{1}{es_i} + \frac{1-F^{-1/s}}{s_i \ln(F^{1/s})} = \frac{1}{s_i} \cdot \left(\frac{1}{e} + \frac{1-F^{-1/s}}{\ln(F^{1/s})}\right).$$

□

Now proving Theorem 5.3.12 is quite straightforward.

Proof of Theorem 5.3.12. Combining Lemma 5.3.14 with Lemma 5.3.15 yields an upper bound of

$$\lambda_{\text{init}} \cdot \frac{F}{1-F} + \left(\frac{1}{e} + \frac{1-F^{-1/s}}{\ln(F^{1/s})}\right) \cdot \frac{F^{\frac{s+1}{s}}-1}{F-1} \sum_{i=a}^{b-1} \frac{1}{s_i}.$$

Plugging in $s_i = p_i^+ \geq (n-i)/(en)$ yields the claimed bound for OneMax. □

Analysing Periods of Temporary Elitism in the Self-Adjusting $(1,\lambda)$ EA

Now we return to the non-elitist self-adjusting $(1,\lambda)$ EA and show how the bound for the elitist algorithm can be incorporated. We first bound the expected number of evaluations to raise the best-so-far fitness value by at least $\log n$.

Lemma 5.3.18. *Consider the self-adjusting* $(1,\lambda)$ EA *as in Theorem 5.3.7 and assume that the typical events stated in Lemma 5.3.9 occur. Let f_t^* be the best-so-far fitness at time t. If $f_t^* \geq n - n/\log^3 n$ and $f_t^* \leq n - \log n$, the expected number of evaluations made until the best-so-far fitness increases to at least $f_t^* + \log n$ is at most*

$$\beta \cdot \left(\lambda_t + \log^3(n)\log\log(n) + \sum_{i=f_t^*-\rho\log n}^{f_t^*+\log n} \frac{n}{n-i}\right)$$

for some constant $\beta > 0$ (that depends on s and F).

Proof. We first show that the expected number of evaluations until either a fitness of at least $f_t^* + \log n$ is reached or $\lambda \geq 4\log^3 n$ is reached is bounded by $O(\log^3(n)\log\log n)$.

By Lemma 5.3.10, the probability that in a sequence of $O(\log\log n)$ generations a λ-value of at least $4\log^3 n$ is reached is at least

$$1 - \frac{4\log^3(n)p^+_{i-\rho\log n}}{F^{1/s}-1} \geq 1 - \frac{4\log^3(n)}{F^{1/s}-1} \cdot \frac{n/\log^3 n + \rho\log n}{en} = \Omega(1).$$

Hence, the expected number of phases of $O(\log\log n)$ generations for this to happen is $O(1)$. As every generation makes at most $4F^{1/s}\log^3 n$ evaluations, the expected number of evaluations is $O(\log^3(n)\log\log n)$.

Now assume $\lambda \geq 4\log^3 n$. We say that a failure occurs if λ drops below $4\log n$ before the fitness is increased to at least $f_t^* + \log n$. By Lemma 5.3.8, the expected number of generations to increase the fitness from f_t^* to at least $f_t^* + \log n$ is at most $c_{\text{gen}}\log n$, for a constant c_{gen}. By Markov's inequality, the probability that more than $2c_{\text{gen}}\log n$ generations are needed is at most $1/2$. By Lemma 5.3.11, the probability of decreasing λ to less than $4\log n$ in the next $2c_{\text{gen}}\log n$ generations is $(\log n)^{-\omega(1)}$. By a union bound, the probability of a failure is at most $1/2 + (\log n)^{-\omega(1)}$.

While no failure occurs, the algorithm displays temporary elitism and behaves like a $(1 + \{F^{1/s}\lambda, \lambda/F\})$ EA. Applying Theorem 5.3.12 with parameters $a := f_t^* - \rho\log n$ and $b := f_t^* + \log n$, the expected number of evaluations until a fitness of at least $f_t^* + \log n$ is found or a fallback occurs is at most

$$\lambda_{\text{init}} \cdot \frac{F}{1-F} + \left(\frac{1}{e} + \frac{1-F^{-1/s}}{\ln(F^{1/s})}\right) \cdot \frac{F^{\frac{s+1}{s}} - 1}{F-1} \sum_{i=f_t^*-\rho\log n}^{f_t^*+\log n - 1} \frac{en}{n-i}.$$

In case of a failure we iterate the above arguments; this increases the expected number of evaluations only by a factor $2 + o(1)$. Choosing an appropriately large constant $\beta > 0$ completes the proof. $\square$

Since the bound from Lemma 5.3.18 contains an additive term of $O(\lambda_t)$, we also provide a lemma that bounds the expectation of λ_t.

Lemma 5.3.19. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 5.3.7. If the best-so-far fitness at time t is at most i then*

$$\mathrm{E}(\lambda_t \mid \lambda_0) \leq \lfloor \lambda_0/F^t \rfloor + \frac{en}{n-i} \cdot \left(F^{1/s} + \frac{F^{1/s}}{\ln F}\right).$$

Proof. Since the fitness in all generations $t' \leq t$ is at most i, the probability of one offspring finding an improvement at time t' is at least $\frac{n-i}{en}$. We have

$$\mathrm{E}(\lambda_t \mid \lambda_0) = \sum_{x=1}^{\infty} \Pr(\lambda_t \geq x \mid \lambda_0)$$

and bound the latter probabilities from above. If $\lambda_0 \geq x \cdot F^t$ then $\lambda_t \geq x$ with probability 1. If $\lambda_0 < x \cdot F^t$, $\lambda_t \geq x$ only holds if there has been at least one generation where λ was increased. Let $t - k > 0$ be the last such generation, that is, $\lambda_{t-k} = \lambda_{t-k-1}F^{1/s}$. Since all generations $t-k, \dots, t-1$ are successful, $\lambda_t \geq x$ implies $\lambda_{t-k} \geq xF^k$ and hence $\lambda_{t-k-1} \geq xF^{k-1/s}$.

Hence, for all $x > \lambda_0/F^t$,

$$\begin{aligned}\Pr(\lambda_t \geq x \mid \lambda_0) &\leq \sum_{k=1}^{\infty} \Pr\left(\text{no success with } xF^k/F^{1/s} \text{ offspring}\right)\\ &\leq \sum_{k=1}^{\infty} \left(1 - \frac{n-i}{en}\right)^{xF^k/F^{1/s}}.\end{aligned}$$

Using $F^k = e^{k\ln F} \geq 1 + k\ln F$, this is at most

$$\begin{aligned}&\left(1 - \frac{n-i}{en}\right)^{x/F^{1/s}} \sum_{k=1}^{\infty} \left(\left(1 - \frac{n-i}{en}\right)^{x\ln(F)/F^{1/s}}\right)^k\\ =\ &\left(1 - \frac{n-i}{en}\right)^{x/F^{1/s}} \frac{\left(1 - \frac{n-i}{en}\right)^{x\ln(F)/F^{1/s}}}{1 - \left(1 - \frac{n-i}{en}\right)^{x\ln(F)/F^{1/s}}}\end{aligned}$$

For $x > \lambda_0/F^t$ and $x \geq \frac{en}{n-i} \cdot \frac{F^{1/s}}{\ln F}$, the last fraction is at most $\frac{1/e}{1-1/e} < 1$, thus for all such x

$$\Pr\left(\lambda_t \geq x \mid \lambda_0\right) \leq \left(1 - \frac{n-i}{en}\right)^{x/F^{1/s}}.$$

Bounding the first $\alpha := \max\left(\lfloor \lambda_0/F^t \rfloor, \left\lfloor \frac{en}{n-i} \cdot \frac{F^{1/s}}{\ln F} \right\rfloor\right)$ terms trivially by 1, we obtain

$$\begin{aligned}
\sum_{x=1}^{\infty} \Pr\left(\lambda_t \geq x \mid \lambda_0\right) &\leq \alpha + \sum_{x=\alpha+1}^{\infty} \Pr\left(\lambda_t \geq x \mid \lambda_0\right) \\
&\leq \alpha + \sum_{x=\alpha+1}^{\infty} \left(1 - \frac{n-i}{en}\right)^{x/F^{1/s}} \\
&\leq \alpha + \sum_{x=0}^{\infty} \left(\left(1 - \frac{n-i}{en}\right)^{1/F^{1/s}}\right)^{x} \\
&= \alpha + \frac{1}{1 - \left(1 - \frac{n-i}{en}\right)^{1/F^{1/s}}}.
\end{aligned}$$

By Lemma 2.4.2 (a), $(1+x)^r \leq 1 + rx$ for every $0 \leq r \leq 1$ and $x \geq -1$, then this is at most

$$\alpha + \frac{1}{1 - \left(1 - F^{-1/s} \cdot \frac{n-i}{en}\right)} = \alpha + F^{1/s} \cdot \frac{en}{n-i}.$$

Plugging in α implies the claim. □

Now we are in a position to prove Theorem 5.3.7.

Proof of Theorem 5.3.7. We divide the optimisation in several phases. Phase 1 ends when the distance to the optimum is at most $n/\log n$. By Lemma 5.3.8 the expected number of *generations* spent in Phase 1, called T_1, is $\mathrm{E}(T_1) = O(n)$.

The expected number of function evaluations during this time is $\mathrm{E}(\lambda_0 + \lambda_1 + \cdots + \lambda_{T_1-1}) = \sum_{t=0}^{T_1-1} \mathrm{E}(\lambda_t \mid \lambda_0)$. We bound all summands by Lemma 5.3.19, applied with a worst case fitness of $i := n - n/\log n$. This yields a random variable λ^* with

$$\mathrm{E}(\lambda^*) \leq \frac{en}{n/\log n} \cdot \left(F^{1/s} + \frac{F^{1/s}}{\ln F}\right) = e \log n \cdot \left(F^{1/s} + \frac{F^{1/s}}{\ln F}\right)$$

and $\mathrm{E}(\lambda^*) \geq \mathrm{E}(\lambda_t \mid \lambda_0)$ for all $t < T_1$. Thus, the expected time in Phase 1 can be bounded by T_1 iid variables λ^*. Since T_1 is itself a random variable, we apply Wald's equation (Lemma 2.4.9) to conclude that

$$\sum_{t=0}^{T_1-1} \mathrm{E}(\lambda^*) = \mathrm{E}(T_1) \cdot \mathrm{E}(\lambda^*) = O(n \log n).$$

Phase 2 ends when the distance to the optimum is at most $n/\log^2 n$. Again, by Lemma 5.3.8 the expected number of *generations* is T_2 with $\mathrm{E}(T_2) = O(n/\log n)$ and the expected number of evaluations in one generation is bounded by $e \log^2 n \cdot \left(F^{1/s} + \frac{F^{1/s}}{\ln F}\right)$ using Lemma 5.3.19 with a worst-case fitness of $i := n - n/\log^2 n$. Wald's equation (Lemma 2.4.9) then yields a bound of $O(n/\log n) \cdot O(\log^2 n) = O(n \log n)$.

Phase 3 ends with the distance to the optimum is at most $n/\log^3 n$ and we obtain another bound of $O(n \log n)$ in the same way.

Phase 4 ends when the optimum is found. We divide the distance to the optimum in *blocks* of length $\log n$. Let T_i be the random number of evaluations to increase the best fitness from at least $n - n/\log^3(n) + (i-1)\log n$ to a fitness of at least $n - n/\log^3(n) + i \log n$. Let λ_i^* be the λ-value in the first generation where this block i is reached. Then the expected number of evaluations to find the optimum from a best fitness of at least $n - n/\log^3 n$ is at most

$$\mathrm{E}\left(\sum_{i=1}^{n/\log^4 n} T_i\right) = \sum_{i=1}^{n/\log^4 n} \mathrm{E}(T_i) = \sum_{i=1}^{n/\log^4 n} \mathrm{E}(\mathrm{E}(T_i \mid \lambda_i^*))$$

By Lemma 5.3.18, this is at most

$$\sum_{i=1}^{n/\log^4 n} \mathrm{E}\left(\beta\left(\lambda_i^* + \log^3(n)\log\log(n) + \sum_{j=n-n/\log^3(n)+(i-1-\rho)\log n}^{n-n/\log^3(n)+i\log(n)-1} \frac{1}{n-j}\right)\right).$$

The terms $\beta \log^3(n) \log\log n$ sum up to $O(n \log\log n)$. Note that

$$\beta \sum_{i=1}^{n/\log^4 n} \sum_{j=n-n/\log^3(n)+(i-1-\rho)\log n}^{n-n/\log^3(n)+i\log(n)-1} \frac{1}{n-j} \leq \beta \sum_{i=0}^{n-1} \frac{\rho+1}{n-j} = O(n \log n)$$

since every summand $\frac{1}{n-j}$ appears in at most $\rho + 1$ blocks. Finally, by Lemma 5.3.19, $\sum_{i=1}^{n/\log^4 n} \mathrm{E}(\lambda_i^*) = O(n \log n)$ with room to spare.

In case a failure occurs, we repeat the above arguments. Note that then we start the analysis with λ_0 being the λ-value at the time of the failure. A large λ-value may incur additional costs. However, applying Lemma 5.3.19 with a fitness of $i = n$ shows that then the expected λ-value is at most $O(n)$. And, since the $\lfloor \lambda_0/F^t \rfloor$ term in Lemma 5.3.19 decays exponentially in t, a value $\lambda_0 = O(n)$ only incurs additional costs of at most $\sum_{t=0}^{\infty} \lambda_0/F^t = \lambda_0 \cdot \frac{F}{F-1}$ in subsequent generations. Since failures have a probability of $O(1/n)$, the expected number of repetitions is $1 + O(1/n)$ and all additional costs can be absorbed in the $O(n \log n)$ bound. $\square$

5.3.2 Large Success Rates Fail

In this section, we show that the choice of the success rate is crucial as when s is a large constant, the runtime becomes exponential.

Theorem 5.3.20. *Let the update strength $F \leq 1.5$ and the success rate $s \geq 18$ be constants. With probability $1 - e^{-\Omega(n/\log^4 n)}$ the self-adjusting $(1,\lambda)$ EA needs at least $e^{\Omega(n/\log^4 n)}$ evaluations to optimise* OneMax.

The reason why the algorithm takes exponential time is that now $F^{1/s}$ is small ($F^{1/s}$ converges to 1 as s grows) and λ only increases slowly in unsuccessful generations, whereas successful generations decrease λ by a much larger factor of F. This is detrimental during early parts of the run where it is *easy* to find improvements and there are frequent improvements that decrease λ. When λ is small, there are frequent fallbacks, hence the algorithm stays in a region with small values of λ, where it finds improvements with constant probability, but also has fallbacks with constant probability. We show, using another potential function based on Definition 5.2.3, that it takes exponential time to escape from this equilibrium.

Definition 5.3.21. We define the potential function $g_2(X_t)$ as

$$g_2(X_t) := f(x_t) + 2.2 \log_F^2 \lambda_t.$$

While $g_1(X_t)$ used a (capped) linear contribution of $\log_F(\lambda_t)$ for $h(\lambda_t)$, here we use the function $\log_F^2(\lambda_t)$ that is convex in $\log_F(\lambda_t)$, so that changes in λ_t have a larger impact on the potential. We show that, in a given fitness interval, the potential $g_2(X_t)$ has a negative drift.

Lemma 5.3.22. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 5.3.20. Then there is a constant* $\delta > 0$ *such that for every* $0.84n + 2.2\log^2(4.5) < g_2(X_t) < 0.85n$,

$$\mathrm{E}(g_2(X_{t+1}) - g_2(X_t) \mid X_t) \le -\delta.$$

Proof of Lemma 5.3.22. We abbreviate $\Delta_{g_2} := \mathrm{E}(g_2(X_{t+1}) - g_2(X_t) \mid X_t)$. Given that for all $\lambda \ge 1$

$$h(\lambda/F) = 2.2\log_F^2(\lambda/F) = 2.2\left(\log_F(\lambda) - 1\right)^2 \ge 0 = h(1)$$

then by Lemma 5.2.4, for all λ, Δ_{g_2} is at most

$$\begin{aligned}
&\left(\Delta^+_{i,\lambda} + h(\lambda/F) - h(\lambda F^{1/s})\right) p^+_{i,\lambda} + h(\lambda F^{1/s}) - h(\lambda) - \Delta^-_{i,\lambda} p^-_{i,\lambda} \\
&= \left(\Delta^+_{i,\lambda} + 2.2\log_F^2(\lambda/F) - 2.2\log_F^2(\lambda F^{1/s})\right) p^+_{i,\lambda} + \\
&\qquad\qquad 2.2\log_F^2(\lambda F^{1/s}) - 2.2\log_F^2(\lambda) - \Delta^-_{i,\lambda} p^-_{i,\lambda} \\
&= \left(\Delta^+_{i,\lambda} + 2.2(\log_F(\lambda) - 1)^2 - 2.2(\log_F(\lambda) + 1/s)^2\right) p^+_{i,\lambda} + \\
&\qquad\qquad 2.2(\log_F(\lambda) + 1/s)^2 - 2.2\log_F^2(\lambda) - \Delta^-_{i,\lambda} p^-_{i,\lambda} \\
&= \left(\Delta^+_{i,\lambda} - \frac{s+1}{s} \cdot 4.4\log_F(\lambda) + 2.2 - \frac{2.2}{s^2}\right) p^+_{i,\lambda} + \frac{4.4\log_F\lambda}{s} + \frac{2.2}{s^2} - \Delta^-_{i,\lambda} p^-_{i,\lambda} \\
&\le \left(\Delta^+_{i,\lambda} - \frac{19}{18} \cdot 4.4\log_F\lambda + 2.2 - \frac{2.2}{324}\right) p^+_{i,\lambda} + \frac{4.4\log_F\lambda}{18} + \frac{2.2}{324} - \Delta^-_{i,\lambda} p^-_{i,\lambda}. \qquad (5.12)
\end{aligned}$$

We note that in Equation (5.12), $\lambda \in \mathbb{R}_{\ge 1}$ but since the algorithm creates $\lfloor\lambda\rceil$ offspring, the forward drift and the probabilities are calculated using $\lfloor\lambda\rceil$. In the following in all the computations the last digit is rounded up if the value was positive and down otherwise to ensure the inequalities hold. We start taking into account only $\lfloor\lambda\rceil \ge 5$, that is, $\lambda \ge 4.5$ and later on we will deal with smaller values of λ. With this constraint on λ we use the simple bound $p^-_{i,\lambda} \ge 0$. Bounding $\Delta^+_{i,\lambda} \le \lceil\log\lambda\rceil + 0.413$ using Lemma 5.3.1,

$$\Delta_{g_2} \le \left(2.613 + \lceil\log\lambda\rceil - \frac{19}{18} \cdot 4.4\log_F\lambda - \frac{2.2}{324}\right) p^+_{i,\lambda} + \frac{4.4\log_F\lambda}{18} + \frac{2.2}{324}.$$

For all $\lambda \ge 1$, $f(x_t) \ge 0.85n$ implies $g_2(X_t) \ge 0.85n$. By contraposition, our precondition $g_2(X_t) < 0.85n$ implies $f(x_t) < 0.85n$. Therefore, using Lemma 5.3.1 with the worst case $f(x_t) = 0.85n$ and $\lfloor\lambda\rceil = 5$ we get $p^+_{i,\lambda} \ge 1 - \frac{e}{e+0.15\lfloor\lambda\rceil} \ge 1 - \frac{e}{e+5\cdot 0.15} > 0.216$. Substituting these bounds we obtain

$$\begin{aligned}
\Delta_{g_2} &\le \left(2.613 + \lceil\log\lambda\rceil - \frac{19}{18} \cdot 4.4\log_F\lambda - \frac{2.2}{324}\right) 0.216 + \frac{4.4\log_F\lambda}{18} + \frac{2.2}{324} \\
&\le 0.5562 + 0.216\lceil\log\lambda\rceil - 0.7587\log_F\lambda
\end{aligned}$$

Given that $\log_F\lambda \ge \log\lambda$ for all $F \le 2$ and $\lceil\log\lambda\rceil \le \log(\lambda) + 1$,

$$\begin{aligned}
\Delta_{g_2} &\le 0.5562 + 0.216(\log(\lambda) + 1) - 0.7587\log\lambda \\
&= 0.7722 - 0.5427\log\lambda \\
&\le 0.7722 - 0.5427\log 4.5 \le -0.4054.
\end{aligned}$$

Up until now we have proved that $\Delta_{g_2} \leq -0.4058$ for all $f(x_t) < 0.85n$ and $\lfloor\lambda\rceil \geq 5$. Now we need to consider $\lfloor\lambda\rceil < 5$. For $\lfloor\lambda\rceil < 5$, that is, $\lambda < 4.5$, the precondition $g_2(X_t) > 0.84n + 2.2\log^2(4.5)$ implies that $f(x_t) > 0.84n$. Therefore, the last part of this proof focuses only on $0.84n < f(x_t) < 0.85n$ and $\lfloor\lambda\rceil < 5$. For this region we use Equation (5.12). By Lemma 5.3.1, $p^-_{i,\lambda} \geq \left(\frac{f(x_t)}{n} - \frac{1}{e}\right)^{\lfloor\lambda\rceil} \geq \left(0.84 - \frac{1}{e}\right)^{\lfloor\lambda\rceil}$ and bounding $\Delta^+_{i,\lambda}$ and $\Delta^-_{i,\lambda}$ using Lemma 5.3.1 yields:

$$\Delta_{g_2} \leq \underbrace{\left(\sum_{j=1}^{\infty}\left(1-\left(1-\frac{1}{j!}\right)^{\lfloor\lambda\rceil}\right) - \frac{19}{18}\cdot 4.4\log_F\lambda + 2.2 - \frac{2.2}{324}\right)}_{\alpha} p^+_{i,\lambda}$$
$$+\frac{4.4\log_F\lambda}{18} + \frac{2.2}{324} - \left(0.84-\frac{1}{e}\right)^{\lfloor\lambda\rceil}. \quad (5.13)$$

We did not bound $p^+_{i,\lambda}$ in the first term yet because the factor α in brackets preceding it can be positive or negative. We now calculate precise values for $\sum_{j=1}^{\infty}\left(1-\left(1-\frac{1}{j!}\right)^{\lfloor\lambda\rceil}\right)$ giving $e-1$, 2.157, 2.4458 and 2.6511 for $\lfloor\lambda\rceil = 1,2,3,4$, respectively. Given that $F \leq 1.5$ the factor α is negative for all $1.5 \leq \lambda < 4.5$, because

$$\begin{aligned}&\left(\sum_{j=1}^{\infty}\left(1-\left(1-\frac{1}{j!}\right)^{\lfloor\lambda\rceil}\right) - \frac{19}{18}\cdot 4.4\log_F\lambda + 2.2 - \frac{2.2}{324}\right)\\ &\leq \left(\sum_{j=1}^{\infty}\left(1-\left(1-\frac{1}{j!}\right)^{\lfloor\lambda\rceil}\right) - 4.4\log_F(\lambda) + 2.2\right)\\ &\leq \begin{cases} 4.8511 - 4.4\log_{1.5}(3.5) = -8.74 & 3.5 \leq \lambda < 4.5\\ 4.6458 - 4.4\log_{1.5}(2.5) = -5.29 & 2.5 \leq \lambda < 3.5\\ 4.357 - 4.4\log_{1.5}(1.5) = -0.043 & 1.5 \leq \lambda < 2.5\end{cases}\end{aligned}$$

On the other hand, for $\lambda < 1.5$ and $\lfloor\lambda\rceil = 1$, α is positive when $\lambda < F^\gamma$ for $\gamma = \frac{1933}{7524} + \frac{45e}{209} \approx 0.8422$ and negative otherwise. With this we evaluate different ranges of λ separately using Equation (5.13). For $1 \leq \lambda < F^\gamma$, we get $\lfloor\lambda\rceil = 1$ and by Lemma 5.3.1 if $0.84n \leq i \leq 0.85n$ and $n \geq 163$ then $p^+_{i,\lambda} \leq 0.069$, thus

$$\begin{aligned}\Delta_{g_2} &\leq \left(e + 1.2 - \frac{19}{18}\cdot 4.4\log_F(\lambda) - \frac{2.2}{324}\right)0.069 + \frac{4.4}{18}\log_F(\lambda) + \frac{2.2}{324} - \left(0.84 - \frac{1}{e}\right)\\ &\leq -0.076\log_F(\lambda) - 0.195 \leq -0.195.\end{aligned}$$

For $F^\gamma \leq \lambda < 1.5$, by Lemma 5.3.1 we bound $p^+_{i,\lambda} \geq \frac{n-f(x_t)}{en} \geq 0.0551$:

$$\begin{aligned}\Delta_{g_2} &\leq \left(e + 1.2 - \frac{19}{18}\cdot 4.4\log_F(\lambda) - \frac{2.2}{324}\right)0.0551 + \frac{4.4}{18}\log_F(\lambda) + \frac{2.2}{324} - \left(0.84 - \frac{1}{e}\right)\\ &\leq -0.0114\log_F(\lambda) - 0.2498 \leq -0.2498.\end{aligned}$$

For $1.5 \leq \lambda < 2.5$, by Lemma 5.3.1 we bound $p^+_{i,\lambda} \geq 1 - \frac{e}{e+0.3} \geq 0.0993$

$$\begin{aligned}\Delta_{g_2} &\leq \left(4.357 - \frac{19}{18}\cdot 4.4\log_F(\lambda) - \frac{2.2}{324}\right)0.0993 + \frac{4.4}{18}\log_F(\lambda) + \frac{2.2}{324} - \left(0.84 - \frac{1}{e}\right)^2\\ &\leq 0.2159 - 0.2167\log_F(\lambda)\\ &\leq 0.2159 - 0.2167\log_{1.5}(1.5) \leq -0.0008.\end{aligned}$$

For $2.5 \le \lambda < 3.5$ we use $p^+_{i,\lambda} \ge 1 - \frac{e}{e+0.45} = 0.142$,

$$\begin{aligned}\Delta_{g_2} &\le \left(4.6458 - \frac{19}{18}\cdot 4.4\log_F(\lambda) - \frac{2.2}{324}\right)0.142 + \frac{4.4}{18}\log_F(\lambda) + \frac{2.2}{324} - \left(0.84 - \frac{1}{e}\right)^3\\ &\le 0.5612 - 0.415\log_F(\lambda)\\ &\le 0.5612 - 0.415\log_{1.5}(2.5) \le -0.376.\end{aligned}$$

Finally for $3.5 \le \lambda < 4.5$ we use $p^+_{i,\lambda} \ge 1 - \frac{e}{e+0.6} = 0.1808$,

$$\begin{aligned}\Delta_{g_2} &\le \left(4.8511 - \frac{19}{18}\cdot 4.4\log_F(\lambda) - \frac{2.2}{324}\right)0.1808 + \frac{4.4}{18}\log_F(\lambda) + \frac{2.2}{324} - \left(0.84 - \frac{1}{e}\right)^4\\ &\le 0.832 - 0.5952\log_F(\lambda)\\ &\le 0.832 - 0.5952\log_{1.5}(3.5) \le -1.006.\end{aligned}$$

With these results we can see that the potential is negative with $\lambda \in [1, 4.5)$ and $0.84n < f(x_t) < 0.85n$. Hence, for every $0.84n + \log^2(4.5) < g_2(X_t) < 0.85n$, and $\delta = 0.0008$, $\Delta_{g_2} \le -\delta$. $\square$

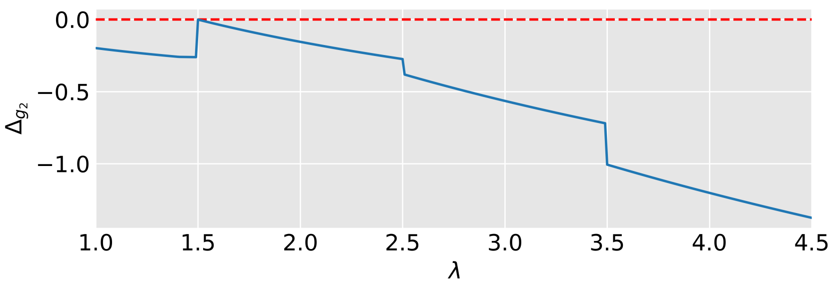

Figure 5.1: Bounds on Δ_{g_2} with a maximum of -0.0008 for $\lambda = 1.5$.

Finally, with Lemmas 5.3.22 and 5.3.2, we now prove Theorem 5.3.20.

Proof of Theorem 5.3.20. We apply the negative drift theorem with scaling (Theorem 2.4.17). We switch to the potential function $\overline{h}(X_t) := \max\{0, n - g_2(X_t)\}$ in order to fit the perspective of the negative drift theorem. In this case we can pessimistically assume that if $\overline{h}(X_t) = 0$ the optimum has been found.

The first condition of the negative drift theorem with scaling (Theorem 2.4.17) can be established with Lemma 5.3.22 for $a = 0.15n$ and $b = 0.16n - 2.2\log^2(4.5)$. Furthermore, with Chernoff bounds we can prove that at initialization $\overline{h}(X_t) \ge b$ with probability $1 - 2^{-\Omega(n)}$.

To prove the second condition we need to show that the probability of large jumps is small. Starting with the contribution that λ makes to the change in $\overline{h}(X_t)$, we use Lemma 5.3.2 to show that this contribution is at most $2.2\log^2(eF^{1/s}n^3) \le 4.79 + 19.8\log^2 n \le 20\log^2 n$ with probability $\exp(-\Omega(n^2))$, where the last inequality holds for large enough n.

The only other contributor is the change in fitness. The probability of a jump in fitness away from the optimum maximised when there is only one offspring. On the other hand the bigger the offspring population the higher the probability of a large jump towards the optimum. Taking this into account and considering that every bit flip either decreases the fitness in the first case or increases the fitness in the latter we get the following probabilities.

$$\Pr\left(f(x_t) - f(x_{t+1}) \ge \kappa\right) \le \frac{1}{\kappa!}$$

$$\Pr\left(f(x_{t+1}) - f(x_t) \geq \kappa\right) \leq 1 - \left(1 - \frac{1}{\kappa!}\right)^{\lambda}$$

Given that $\frac{1}{\kappa!} \leq 1 - \left(1 - \frac{1}{\kappa!}\right)^{\lambda}$, and that $\lambda \leq eF^{1/s}n^3$.

$$\begin{aligned}\Pr\left(|f(x_{t+1}) - f(x_t)| \geq \kappa\right) &\leq 1 - \left(1 - \frac{1}{\kappa!}\right)^{eF^{1/s}n^3} \\ &\leq \frac{eF^{1/s}n^3}{\kappa!} \\ &\leq \frac{e^{\kappa+1}F^{1/s}n^3}{\kappa^\kappa}\end{aligned}$$

Joining both contributions, we get

$$\Pr\left(|g_2(X_{t+1}) - g_2(X_t)| \geq \kappa + 20\log^2 n\right) \leq \frac{e^{\kappa+1}F^{1/s}n^3}{\kappa^\kappa}. \tag{5.14}$$

To satisfy the second condition of the negative drift theorem with scaling (Theorem 2.4.17) we use $\rho = 21\log^2 n$ and $\kappa = j\log^2 n$ in order to have $\kappa + 20\log^2 n \leq j\rho$ for $j \in \mathbb{N}$. For $j = 0$ the condition $\Pr\left(|g_2(X_{t+1}) - g_2(X_t)| \geq j\rho\right) \leq e^0$ is trivial. From Equation (5.14), we obtain

$$\Pr\left(|g_2(X_{t+1}) - g_2(X_t)| \geq j\rho\right) \leq \frac{eF^{1/s}e^{(j\log^2 n)}n^3}{(j\log^2 n)^{j\log^2 n}}$$

We simplify the numerator using

$$e^{(j\log^2 n)}n^3 = e^{(j\log(n)\ln(n)/\ln(2))}n^3 = n^{(3+j\log(n)/\ln(2))}$$

and bound the denominator as

$$(j\log^2 n)^{j\log^2 n} \geq (\log n)^{2j\log^2 n} = n^{2j\log(n)\log\log(n)},$$

yielding

$$\begin{aligned}\Pr\left(|g_2(X_{t+1}) - g_2(X_t)| \geq jr\right) &\leq eF^{1/s}n^{(3+j\log(n)/\ln(2) - 2j(\log n)\log\log n)} \\ &= eF^{1/s}n^{(3+j(\log(n)/\ln(2) - 2(\log n)\log\log n))},\end{aligned}$$

which for large enough n is bounded by e^{-j} as desired.

The third condition is met with $\rho = 21\log^2 n$ given that $\delta\ell/(132\log((21\log^2 n)/\delta)) = \Theta(n/\log\log n)$, which is larger than $\rho^2 = \Theta(\log^4 n)$ for large enough n.

With this we have proved that the algorithm needs at least $e^{\Omega(n/\log^4 n)}$ generations with probability $1 - e^{-\Omega(n/\log^4 n)}$. Since each generation uses at least one fitness evaluation, the claim is proved. □

We note that although Theorem 5.3.20 is applied for OneMax specifically, the conditions used in the proof of Theorem 5.3.20 and Lemma 5.3.22 apply for several other benchmark functions. This is because our result only depends on some fitness levels of OneMax and other functions have fitness levels that are symmetrical or resemble these fitness levels. We show this in the following theorem. To improve readability we use $|x|_1 := \sum_{i=1}^n x_i$ and $|x|_0 := \sum_{i=1}^n (1 - x_i)$.

Theorem 5.3.23. *Let the update strength $F \leq 1.5$ and the success rate $s \geq 18$ be constants. With probability $1 - e^{-\Omega(n/\log^4 n)}$ the self-adjusting $(1,\lambda)$ EA needs at least $e^{\Omega(n/\log^4 n)}$ evaluations to optimise:*

- $\textsc{Jump}_k(x) := \begin{cases} n - |x|_1 & \text{if } n-k < |x|_1 < n, \\ k + |x|_1 & \text{otherwise,} \end{cases}$
  with $k = o(n)$,
- $\textsc{Cliff}_d(x) := \begin{cases} |x|_1 & \text{if } |x|_1 \leq d, \\ |x|_1 - d + 1/2 & \text{otherwise,} \end{cases}$
  with $d = o(n)$,
- $\textsc{ZeroMax}(x) := |x|_0$,
- $\textsc{TwoMax}(x) := \max\{|x|_1, |x|_0\}$,
- $\textsc{Ridge}(x) := \begin{cases} n + |x|_1 & \text{if } x = 1^i 0^{n-i}, i \in \{0, 1, \dots, n\}, \\ |x|_0 & \text{otherwise.} \end{cases}$

Proof. For $\textsc{Jump}_k$ and $\textsc{Cliff}_d$, given that k and d are $o(n)$ the algorithm needs to optimise a OneMax-like slope with the same transition probabilities as in Lemma 5.3.22 before the algorithm reaches the local optima. Hence, we can apply the negative drift theorem with scaling (Theorem 2.4.17) as in Theorem 5.3.20 to prove the statement.

For ZeroMax the algorithm will behave exactly as in OneMax, because it is unbiased towards bit-values. Similarly, for TwoMax, independently of the slope the algorithm is optimising, it needs to traverse through a OneMax-like slope needing at least the same number of function evaluations as in OneMax.

Finally, for Ridge, unless the algorithm finds a search point on the ridge ($x \in 1^i 0^{n-i}$ with $i \in \{0, 1, \dots, n\}$) beforehand, the first part of the optimisation behaves as ZeroMax and similar to Theorem 5.3.20 by Lemma 5.3.22 and the negative drift theorem with scaling (Theorem 2.4.17) it will need at least $e^{Cn/\log^4 n}$ generations with probability $e^{-Cn/\log^4 n}$ to reach a point with $|x|_1 \leq 0.15n$ for some constant $C > 0$.

It remains to show that the ridge is not reached during this time, with high probability. We first imagine the algorithm optimising ZeroMax and note that the behaviour on Ridge and ZeroMax is identical as long as no point on the ridge is discovered. Let $x_0, x_1, \dots$ be the search points created by the algorithm on ZeroMax in order of creation. Since ZeroMax is symmetric with respect to bit positions, for any arbitrary but fixed t we may assume that the search point x_t with $d = |x_t|_1$ is chosen uniformly at random from the $\binom{n}{d}$ search points that have exactly d 1-bits. There is only one search point $1^d 0^{n-d}$ that on the function Ridge would be part of the ridge. Thus, for $d \geq 0.15n$ the probability that x_t lies on the ridge is at most

$$\binom{n}{d}^{-1} \leq \binom{n}{0.15n}^{-1} \leq \left(\frac{n}{0.15n}\right)^{-0.15n} = \left(\frac{20}{3}\right)^{-0.15n}.$$

(Note that these events for t and t' are not independent; we will resort to a union bound to deal with such dependencies.) By Lemma 5.3.2, during the optimisation of any unimodal function every generation uses $\lambda \leq eF^{1/s}n^3$ with probability $1 - \exp(-\Omega(n^2))$. By a union bound over $e^{Cn/\log^4 n}$ generations, for an arbitrary constant $C > 0$, each generation creating at most $eF^{1/s}n^3$ offspring, the probability that a point on the ridge is reached during this time is at most

$$e^{Cn/\log^4 n} \cdot eF^{1/s}n^3 \cdot \left(\frac{20}{3}\right)^{-0.15n} = e^{-\Omega(n)}.$$

Adding up all failure probabilities, the algorithm will not create a point on the ridge before $e^{Cn/\log^4 n}$ generations have passed with probability $1 - e^{-\Omega(n/\log^4 n)}$, and the algorithm needs at least $e^{\Omega(n/\log^4 n)}$ evaluations to solve Ridge with probability $1 - e^{-\Omega(n/\log^4 n)}$. □

5.4 Hard Problems are Easier for Success-Based Parameter Control

In Section 5.3 we studied the self-adjusting $(1,\lambda)$ EA using standard bit mutation with mutation probability $1/n$ on OneMax. We showed that if the success rate s is chosen appropriately small constant $s < 1$ the self-adjusting $(1,\lambda)$ EA optimises OneMax efficiently, but for a large constant $s > 18$ it needs exponential time w. o. p. This behaviour is not limited to OneMax; it holds for other common benchmark functions that have easy slopes.

In this section, we extend our analysis to the self-adjusting $(1,\lambda)$ EA using either standard bit mutation with mutation probability $p = r/n$ with $p = O(1/n)$ and $p = n^{-O(1)}$, or the heavy-tailed mutation operator with constant $\beta > 1$. We study a class of functions that are characterised by not having easy fitness levels throughout the optimisation. We now define this class of functions.

Definition 5.4.1. We say that a function f is *everywhere hard* with respect to a black-box algorithm $\mathcal{A}$ if and only if $p^+_{\max} = O\left(n^{-\varepsilon}\right)$ for some constant $0 < \varepsilon < 1$.

Owing to non-elitism, the $(1,\lambda)$ EA may decrease its current fitness if all offspring are worse. This is often a desired characteristic that allows the algorithm to escape local optima, but if this happens too frequently as seen in Section 5.3.2, then the algorithm may not be able to converge to good solutions. Hence, it is important that the probability of an offspring having a lower fitness than its parent is sufficiently small. The probability of this event depends on the mutation operator used. The following lemma shows general bounds on transition probabilities for standard bit mutation and heavy-tailed mutations.

Lemma 5.4.2. *For all* $(1,\lambda)$ EA *algorithms using standard bit mutation with a mutation probability in* $O(1/n)$ *and* $n^{-O(1)}$ *or heavy-tailed mutation operators with a constant* $\beta > 1$*, there is a constant* $\gamma > 1$ *such that* $p^-_x \leq \gamma^{-1}$ *for all non-optimal search points* x.

In addition, for all $0 \leq i \leq d-1$, $p^+_x \geq n^{-O(n)}$.

Proof. Let C denote the event that an offspring is the exact copy of a parent. When using standard bit mutation, if r denotes the implicit constant in the bound $O(1/n)$ on the mutation probability,

$$\Pr(C) \geq \left(1 - \frac{r}{n}\right)^n = \left(1 - \frac{r}{n}\right)^{n-r} \left(1 - \frac{r}{n}\right)^r \geq e^{-r} \cdot \left(1 - \frac{r^2}{n}\right)$$

where the inequality follows from Lemma 2.4.1 and Lemma 2.4.2 (a). Since C implies that in this generation the current search point cannot worsen, we have $p^-_x \leq 1 - \Pr(C)$, therefore there is a constant $\gamma > 1$ for which $p^-_x \leq \gamma^{-1}$.

The probability of the heavy-tailed mutation operator choosing a mutation rate of $1/n$ is

$$\left(\sum_{i=1}^{n/2} i^{-\beta}\right)^{-1} \geq \left(\sum_{i=1}^{\infty} i^{-\beta}\right)^{-1} = \Theta(1).$$

Therefore the probability of creating an exact copy of the parent using a heavy-tailed mutation operator is at least $\Theta(1) \cdot \left(1 - \frac{1}{n}\right)^n = \Theta(1)$ and there is a constant $\gamma > 1$ for which $p^-_x \leq \gamma^{-1}$.

For the second statement, note that the probability of generating a global optimum in one standard bit mutation is at least $\left(n^{-O(1)}\right)^n = n^{-O(n)}$. For heavy-tailed mutations, the probability of choosing a mutation rate of $1/n$ is $\Theta(1)$ as shown above, and then the probability of generating an optimum is at least n^{-n}. □

We now give a helpful definition for specific λ-values as follows.

Definition 5.4.3. Consider a function f with d fitness values that is everywhere hard for a self-adjusting $(1, \lambda)$ EA with success rate s that meets the conditions from Lemma 5.4.2. Let γ and ε be the parameters from Lemma 5.4.2 and Definition 5.4.1, respectively. Then we define $\lambda_{\text{safe}} := 4 \max\left(\log_\gamma(2d(s+1)), \log_\gamma(n \log n)\right)$ and $\lambda_{\text{inc}} := n^{\varepsilon/2}$.

We consider λ_{safe} as a threshold for λ such that λ-values larger than λ_{safe} are considered "safe" because the probability of a fitness loss is small. We aim to show that the algorithm will typically use values larger than λ_{safe} throughout the optimisation. The value λ_{inc} is a threshold for λ such that any λ-value with $\lambda \leq \lambda_{\text{inc}}$ has a relatively small success probability. We will show that λ has a tendency to increase whenever $\lambda \leq \lambda_{\text{inc}}$.

5.4.1 Bounding the Number of Generations

We first focus on bounding the expected number of generations as this bound will be used to bound the expected number of function evaluations later on. The main result of this section is as follows.

Theorem 5.4.4. *Consider a self-adjusting* $(1, \lambda)$ EA *using either standard bit mutation with mutation probability* $p \in O(1/n) \cap n^{-O(1)}$ *or a heavy-tailed mutation operator with a constant* $\beta > 1$*, a constant update strength* $F > 1$ *and a constant success rate* $s > 0$*. For all everywhere hard functions* f *with* $d + 1 = n^{o(\log n)}$ *function values the following holds. For every initial search point and every initial offspring population size* λ_0 *the self-adjusting* $(1, \lambda)$ EA *optimises* f *in an expected number of generations bounded by*

$$O\left(d + \log\left(1/p^+_{\min}\right)\right).$$

This result is related to Theorem 3 in [120] which shows the same asymptotic upper bound for the elitist (1+$\{2\lambda, \lambda/2\}$) EA (i.e. fixing $F = 2$ and $s = 1$) on functions on which fitness levels can only become harder as fitness increases. Our Theorem 5.4.4 applies to everywhere hard functions on which easy and hard fitness levels are mixed in arbitrary ways. And, quite surprisingly, the upper bound only depends on the hardest fitness level.

To bound the number of generations we first need to study how the offspring population size behaves throughout the run. We start by showing that in the beginning of the run λ grows fast.

Lemma 5.4.5. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 5.4.4. Let* τ *be first generation where* $\lambda_\tau \geq \lambda_{\text{inc}}$ *(cf. Definition 5.4.3). Then* $\mathrm{E}(\tau) = O(\log \lambda_{\text{inc}})$*. During these* τ *generations the algorithm only makes* $\lambda_0 + O(\lambda_{\text{inc}} \log \lambda_{\text{inc}})$ *function evaluations in expectation.*

Proof. If the initial offspring population size λ_0 is at least λ_{inc} then $\tau = 1$ and λ_0 evaluations are made. Hence we assume $\lambda_0 < \lambda_{\text{inc}}$.

The parameter λ is multiplied in each unsuccessful generation by $F^{1/s}$ and divided by F otherwise. The probability of an unsuccessful generation is at most $(1 - p_x^+)^\lambda$ and the probability of a successful generation is at least $1 - (1 - p_x^+)^\lambda$.

Hence the expected drift of $\log_F(\lambda)$ is at least

$$\begin{aligned}
&\mathrm{E}(\log_F(\lambda_{t+1}) - \log_F(\lambda_t) \mid \lambda_t = \lambda, \lambda_t \le \lambda_{\mathrm{inc}}, x_t = x) \\
&= \log_F\left(\lambda F^{1/s}\right)\left(1-p_x^+\right)^\lambda + \log_F\left(\frac{\lambda}{F}\right)\left(1-\left(1-p_x^+\right)^\lambda\right) - \log_F(\lambda) \\
&= \left(\log_F(\lambda) + \frac{1}{s}\right)\left(1-p_x^+\right)^\lambda + \left(\log_F(\lambda) - 1\right)\left(1-\left(1-p_x^+\right)^\lambda\right) - \log_F(\lambda) \\
&= \frac{s+1}{s}\left(1-p_x^+\right)^\lambda - 1 \ge \frac{s+1}{s}\left(1-\lambda p_x^+\right) - 1 \\
&= \frac{1-(s+1)\lambda p_x^+}{s} \ge \frac{1-(s+1)\lambda_{\mathrm{inc}} p_x^+}{s} = \frac{1}{s} - O\left(n^{-\varepsilon/2}\right) \ge \frac{1}{2s} \qquad (5.15)
\end{aligned}$$

where the last inequality holds for sufficiently large n, since s is constant.

We apply additive drift as stated in Theorem 2.4.15 as it allows for an unbounded state space. We use the potential function

$$q(\lambda_t) = \log_F(\lambda_{\mathrm{inc}}) - \log(\lambda_t),$$

which implies that when $q(\lambda_t) \le 0$, λ_t is at least λ_{inc}. By Equation (5.15), the drift of $q(\lambda_t)$ is

$$\begin{aligned}
&\mathrm{E}(q(\lambda_t) - q(\lambda_{t+1}) \mid \lambda_t = \lambda, \lambda_t \le \lambda_{\mathrm{inc}}) \\
&= \mathrm{E}(\log(\lambda_{t+1}) - \log(\lambda_t) \mid \lambda_t = \lambda, \lambda_t \le \lambda_{\mathrm{inc}}) \ge \frac{1}{2s}.
\end{aligned}$$

The initial value $q(\lambda_0)$ is at most $\log_F(\lambda_{\mathrm{inc}})$ since we assumed $\lambda_0 \le \lambda_{\mathrm{inc}}$. Now τ denotes the expected number of generations to reach $q(\lambda_t) \le 0$ for the first time, and $q(\lambda_t) \ge -\frac{1}{s}$ for all $t \le \tau$ since $q(\lambda_{t-1}) > 0$ and $q(\lambda_{t-1}) - q(\lambda_t) \le -\log_F(\lambda_{t-1}) + \log_F(F^{1/s}\lambda_{t-1}) = \frac{1}{s}$. Applying Theorem 2.4.15 with $\alpha := -\frac{1}{s}$, we obtain

$$\mathrm{E}(\tau) \le \frac{\log_F(\lambda_{\mathrm{inc}}) + \frac{1}{s}}{1/(2s)} = 2s\log_F(\lambda_{\mathrm{inc}}) + 2 = O(\log \lambda_{\mathrm{inc}}).$$

Given that all generations use $\lambda \le \lambda_{\mathrm{inc}}$, the expected number of evaluations during the $O(\log \lambda_{\mathrm{inc}})$ expected generations is $O(\lambda_{\mathrm{inc}} \log \lambda_{\mathrm{inc}})$. □

Now we show that, once λ reaches a value of at least λ_{inc}, the algorithm maintains a large λ with high probability.

Lemma 5.4.6. *Consider the self-adjusting* $(1,\lambda)$ EA *as in Theorem 5.4.4 at some point of time t^*. For every offspring population size $\lambda_{t^*} \ge \lambda_{\mathrm{inc}}$ the probability that within the next $n^{o(\log n)}$ generations the offspring population size drops below λ_{safe} is at most $n^{-\Omega(\log n)}$.*

Proof. We first note that $\lambda_t \ge F\lambda_{\mathrm{inc}}$ implies $\lambda_{t+1} \ge \lambda_{\mathrm{inc}}$ with probability 1. Thus, the interval $[\lambda_{\mathrm{safe}}, \lambda_{\mathrm{inc}})$ can only be reached if $\lambda_t < F\lambda_{\mathrm{inc}}$. We may assume that $\lambda_{t^*} < F\lambda_{\mathrm{inc}}$ as otherwise we can simply wait for the first point in time t^{**} where $\lambda_{t^{**}} < F\lambda_{\mathrm{inc}}$ and redefine $t^* := t^{**}$. If no such point in time t^{**} exists, or if $t^{**} - t^* \ge n^{o(\log n)}$, there is nothing to show.

Assuming $\lambda_{t^*} < F\lambda_{\mathrm{inc}}$, we show that an improbably large number of successes are needed for the population size to drop below λ_{safe} before returning to a population size of at least $F\lambda_{\mathrm{inc}}$. We define a *trial* as the random time period starting at time t^* and ending when either $\lambda_t < \lambda_{\mathrm{safe}}$ or $\lambda_t \ge \lambda_{t^*}$ for some $t > t^*$. The length of a trial is given by

$$\alpha := \inf\{t - t^* \mid \lambda_t < \lambda_{\mathrm{safe}} \vee \lambda_t \ge \lambda_{t^*}, t > t^*\}$$

and at the end of the trial, either $\lambda_{t^*+\alpha} < \lambda_{\text{safe}}$ or $\lambda_{t^*+\alpha} \geq \lambda_{t^*}$ holds.

An important characteristic of the self-adjusting mechanism is that if there are 1 or 0 successful generations every $\lceil s+1 \rceil$ generations, λ will either grow or maintain its previous value, because $\lambda \cdot (F^{1/s})^{\lceil s \rceil} \cdot 1/F \geq \lambda$. Hence, if from the start of a trial there are at most κ successful generations during $\lceil s+1 \rceil \kappa$ generations for every $\kappa \in \mathbb{N}$ then $\lambda_{t+\lceil s+1 \rceil \kappa} \geq \lambda_t$, implying that the trail has ended with an offspring population size of at least λ_{t^*} and $\alpha \leq \lceil s+1 \rceil \kappa$.

We now consider $\kappa^* := \left\lceil \log_F \left(\frac{\lambda_{\text{inc}}}{\lambda_{\text{safe}}} \right) \right\rceil - 1$ and show that, to end a trial with $\lambda_{t+\alpha} \leq \lambda_{\text{safe}}$, more than κ^* successful generations are needed. For every $\lambda_{t^*} \geq \lambda_{\text{inc}}$, after κ^* consecutive successful generations the offspring population size is

$$\lambda_{t^*+\kappa^*} = \frac{\lambda_{t^*}}{F^{\kappa^*}} = \frac{\lambda_{t^*}}{F^{\left\lceil \log_F \left(\frac{\lambda_{\text{inc}}}{\lambda_{\text{safe}}} \right) \right\rceil - 1}} > \frac{\lambda_{\text{inc}}}{F^{\log_F \left(\frac{\lambda_{\text{inc}}}{\lambda_{\text{safe}}} \right)}} = \lambda_{\text{safe}}.$$

If the successful generations are not consecutive, then the number of successful generations needed to reduce the λ value can only increase. Therefore, to reach $\lambda \leq \lambda_{\text{safe}}$ there must be more than κ^* successful generations.

Now, we know that if there are less than κ^* successful generations within the first $\lceil s+1 \rceil \kappa^*$ generations of the trial then $\lambda_{t^*+\lceil s+1 \rceil \kappa^*} \geq \lambda_{t^*}$ and we end the trial without dropping below λ_{safe}. In every generation of a trial, at most $F\lambda_{\text{inc}}$ offspring are created, thus by a union bound, the probability of a successful generation is at most $F\lambda_{\text{inc}} p_x^+$.

Let X be the number of successful generations within the first $\lceil s+1 \rceil \kappa^*$ generations of a trial, then $0 < \mathrm{E}(X) \leq \lceil s+1 \rceil F \lambda_{\text{inc}} p_x^+ \kappa^*$. Using $\delta := \kappa^* \mathrm{E}(X)^{-1} - 1$ and Chernoff bounds (Lemma 2.4.13),

$$\begin{aligned}
&\Pr(X \geq \kappa^*) = \Pr(X \geq \mathrm{E}(X)(1+\delta)) \\
&\leq \exp\left(- \left(\kappa^* \mathrm{E}(X)^{-1} \ln \left(\kappa^* \mathrm{E}(X)^{-1} \right) - \kappa^* \mathrm{E}(X)^{-1} + 1 \right) \mathrm{E}(X) \right) \\
&= \exp\left(- \left(\kappa^* \ln \left(\kappa^* \mathrm{E}(X)^{-1} \right) - \kappa^* + \mathrm{E}(X) \right) \right) \\
&= e^{-\mathrm{E}(X)} \left(\frac{e\mathrm{E}(X)}{\kappa^*} \right)^{\kappa^*} \leq e^0 \left(\frac{e\mathrm{E}(X)}{\kappa^*} \right)^{\kappa^*} \\
&\leq \left(e \lceil s+1 \rceil F \lambda_{\text{inc}} p_x^+ \right)^{\left\lceil \log_F \left(\frac{\lambda_{\text{inc}}}{\lambda_{\text{safe}}} \right) \right\rceil - 1} = n^{-\Omega(\log n)}
\end{aligned}$$

where the last equation uses that the base is $\Theta(\lambda_{\text{inc}} p_x^+) = O(n^{\varepsilon/2} \cdot n^{-\varepsilon}) = O(n^{-\Omega(1)})$ and simplifying the exponent using

$$\log_F(\lambda_{\text{inc}}/\lambda_{\text{safe}}) = \log_F(n^{\varepsilon/2}) - \log_F(\lambda_{\text{safe}}) = \varepsilon/2 \cdot \log_F(n) - o(\log n)) = \Omega(\log n).$$

Hence, with probability $n^{-\Omega(\log n)}$ a trial ends with an offspring population size of $\lambda_{t^*+\alpha} \geq \lambda_{t^*}$ and without dropping below λ_{safe}. Each trial uses at least one generation. By a union bound over $n^{o(\log n)}$ possible number of trials, the probability of reaching $\lambda \leq \lambda_{\text{safe}}$ within $n^{o(\log n)}$ generations is still $n^{-\Omega(\log n)}$. □

We now define a potential function $g_3(X_t)$ using Definition 5.2.3. We recall that the potential function is a sum of the current search point's fitness and another function $h(\lambda_t)$ that takes into account the current offspring population size: $g_3(X_t) = f(x_t) + h(\lambda_t)$.

Definition 5.4.7. We define the potential function $g_3(X_t)$ as

$$g_3(X_t) = f(x_t) - \frac{s}{s+1} \log_F \left(\max \left(\frac{F^{1/s}}{p_{\min}^+ \lambda_t}, 1 \right) \right).$$

The function $h(\lambda_t) = \frac{s}{s+1}\log_F\left(\max\left(\frac{F^{1/s}}{p_{\min}^{+}\lambda_t}, 1\right)\right)$ is a straightforward generalisation of our approach in Section 5.3.1 in which the specific value $p_{\min}^{+} = 1/(en)$ was used in the context of OneMax.

Similar to the potential function $g_1(X_t)$ used in Section 5.3.1 the potential $g_3(X_t)$ is always close to the current fitness.

Lemma 5.4.8. *For all generations t, the fitness and the potential are related as follows: $f(x_t) - \frac{s}{s+1}\log_F\left(\frac{F^{1/s}}{p_{\min}^{+}}\right) \leq g_3(X_t) \leq f(x_t)$. In particular, $g_3(X_t) = d$ implies $f(x_t) = d$.*

Proof. The term $\frac{s}{s+1}\log_F\left(\max\left(\frac{F^{1/s}}{p_{\min}^{+}\lambda_t}, 1\right)\right)$ is a non-increasing function in λ_t with its minimum being 0 for $\lambda_t \geq F^{1/s}/p_{\min}^{+}$ and its maximum being $\frac{s}{s+1}\log_F\left(\frac{F^{1/s}}{p_{\min}^{+}}\right)$ when $\lambda_t = 1$. Hence, $f(x_t) - \frac{s}{s+1}\log_F\left(\frac{F^{1/s}}{p_{\min}^{+}}\right) \leq g_3(X_t) \leq f(x_t)$. □

Given that we have shown that λ grows fast and stays at a large value, we now show that the expected drift of the potential $g_3(X_t)$ is a positive constant whenever λ is at least λ_{safe}.

Lemma 5.4.9. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 5.4.4. Then for every generation t with $f(x_t) < d$ and $\lambda_t \geq \lambda_{\text{safe}}$,*

$$\mathrm{E}(g_3(X_{t+1}) - g_3(X_t) \mid X_t) \geq \frac{1}{2(s+1)}$$

for large enough n. This also holds when only considering improvements that increase the fitness by 1.

Proof. We consider only $\lambda \geq \lambda_{\text{safe}} > F$, hence, by Lemma 5.2.4, for all $\lambda \geq \lambda_{\text{safe}}$, $\mathrm{E}(g_3(X_{t+1}) - g_3(X_t) \mid X_t)$ is at least

$$\left(\Delta_{x,\lambda}^{+} + h(\lambda/F) - h(\lambda F^{1/s})\right) p_{x,\lambda}^{+} + h(\lambda F^{1/s}) - h(\lambda) - \Delta_{x,\lambda}^{-} p_{x,\lambda}^{-}. \tag{5.16}$$

We first consider the case $\lambda_t \leq 1/p_{\min}^{+}$ as then $\lambda_{t+1} \leq F^{1/s}/p_{\min}^{+}$ and the first term in the maximum of $h(\lambda_{t+1})$ is at least 1, yielding

$$h(\lambda_{t+1}) = -\frac{s}{s+1}\left(\log_F\left(\frac{F^{1/s}}{p_{\min}^{+}}\right) - \log_F(\lambda_{t+1})\right) < 0.$$

Hence, $\mathrm{E}\big(g_3(X_{t+1}) - g_3(X_t) \mid X_t, \lambda_t \leq 1/p_{\min}^{+}\big)$ is at least

$$\begin{aligned} &\left(\Delta_{x,\lambda}^{+} - \frac{s}{s+1}\left(\frac{s+1}{s}\right)\right) p_{x,\lambda}^{+} + \frac{s}{s+1}\left(\frac{1}{s}\right) - \Delta_{x,\lambda}^{-} p_{x,\lambda}^{-} \\ &= \frac{1}{s+1} + \left(\Delta_{x,\lambda}^{+} - 1\right) p_{x,\lambda}^{+} - \Delta_{x,\lambda}^{-} p_{x,\lambda}^{-}. \end{aligned}$$

By definition $\Delta_{x,\lambda}^{+} \geq 1$, hence

$$\mathrm{E}\big(g_3(X_{t+1}) - g_3(X_t) \mid X_t, \lambda_t \leq 1/p_{\min}^{+}\big) \geq \frac{1}{s+1} - \Delta_{x,\lambda}^{-} p_{x,\lambda}^{-}.$$

By Lemma 5.4.2, $p_{x,\lambda_t}^{-} = (p_x^{-})^{\lambda_t} \leq \gamma^{-\lambda_t}$. Along with the trivial bound $\Delta_{x,\lambda}^{-} \leq d$, the right-hand side of the previous inequality is at least

$$\frac{1}{s+1} - d\gamma^{-\lambda}.$$

Since $\lambda_t \geq \lambda_{\text{safe}} \geq \log_\gamma(2d(s+1))$ the second term is at most $\frac{1}{2(s+1)}$, thus $\mathrm{E}\big(g_3(X_{t+1}) - g_3(X_t) \mid X_t, \lambda_t \leq 1/p^+_{\min}\big) \geq \frac{1}{2(s+1)}$.

Finally, for the case $\lambda_t > 1/p^+_{\min}$, in an unsuccessful generation the penalty term is capped, hence we only know that $h(\lambda F^{1/s}) \geq h(\lambda)$ (which holds with equality if $\lambda_t \geq F^{1/s}/p^+_{\min}$). By Equation (5.16),

$$\begin{aligned}\mathrm{E}\big(g_3(X_{t+1}) - g_3(X_t) \mid X_t, \lambda_t > 1/p^+_{\min}\big) &\geq \Big(\Delta^+_{x,\lambda} + h(\lambda/F) - h(\lambda)\Big)\, p^+_{x,\lambda} - \Delta^-_{x,\lambda} p^-_{x,\lambda} \\ &= \left(\Delta^+_{x,\lambda} - \frac{s}{s+1}\right) p^+_{x,\lambda} - \Delta^-_{x,\lambda} p^-_{x,\lambda}.\end{aligned}$$

By definition $\Delta^+_{x,\lambda} \geq 1$, hence

$$\mathrm{E}\big(g_3(X_{t+1}) - g_3(X_t) \mid X_t, \lambda_t \leq 1/p^+_{\min}\big) \geq \left(\frac{1}{s+1}\right) p^+_{x,\lambda} - \Delta^-_{x,\lambda} p^-_{x,\lambda}.$$

$\lambda_t > 1/p^+_{\min}$ implies $p^+_{x,\lambda} \geq 1 - (1-p^+_x)^{1/p^+_{\min}} \geq 1 - \frac{1}{e}$ and by Definition 5.4.1, $p^-_{x,\lambda}\Delta^-_{x,\lambda} \leq d\gamma^{-1/p^+_{\min}}$. Together,

$$\begin{aligned}\mathrm{E}\big(g_3(X_{t+1}) - g_3(X_t) \mid X_t, \lambda_t \leq 1/p^+_{\min}\big) &\geq \left(\frac{1}{s+1}\right)\left(1 - \frac{1}{e}\right) - d\gamma^{-1/p^+_{\min}} \\ &= \left(\frac{1}{s+1}\right)\left(1 - \frac{1}{e}\right) - o(1) \geq \frac{1}{2(s+1)}\end{aligned}$$

where the penultimate step follows from $d \leq n^{o(\log n)}$ and $p^+_{\min} \leq p^+_{\max} \leq n^{-\varepsilon/2}$, which implies $\gamma^{-1/p^+_{\min}} \leq \gamma^{-n^{\varepsilon/2}} = n^{-\Omega(n^{\varepsilon/2}/\log n)}$. The last inequality holds if n is large enough. □

With the previous lemmas we are now able to prove Theorem 5.4.4.

Proof of Theorem 5.4.4. If $\lambda_0 < \lambda_{\text{inc}}$ then by Lemma 5.4.5 in expected $O(\log \lambda_{\text{inc}})$ generations λ will grow to $\lambda \geq \lambda_{\text{inc}}$. Afterwards, by Lemma 5.4.6, with probability $1 - n^{-\Omega(\log n)}$, the offspring population size will be $\lambda \geq \lambda_{\text{safe}}$ in the next $n^{o(\log n)}$ generations. Assuming in the following that this happens, we note that then the drift bound from Lemma 5.4.9 is in force.

Now, similar to Theorem 5.3.3 in Section 5.3.1 we bound the number of generations to reach the global optimum using the potential $g_3(X_t)$. Lemma 5.4.9 shows that the potential has a positive constant drift whenever the optimum has not been found, and by Lemma 5.4.8 if $g_3(X_t) = d$ then the optimum has been found. Therefore, we can bound the number of generations to find a global optimum by the time it takes for $g_3(X_t)$ to reach d.

To fit the perspective of the additive drift theorem (Theorem 2.4.14) we switch to the function $\overline{g_3}(X_t) := d - g_3(X_t)$ and note that $\overline{g_3}(X_t) = 0$ implies that $g_3(X_t) = f(x_t) = d$. The initial value $\overline{g_3}(X_0)$ is at most $d + \frac{s}{s+1}\log_F\left(\frac{F^{1/s}}{p^+_{\min}}\right)$ by Lemma 5.4.8. Using Lemma 5.4.9 and the additive drift theorem, the expected number of generations, assuming no failure occurs, is at most

$$\mathrm{E}(T) \leq \frac{d + \frac{s}{s+1}\log_F\left(\frac{F^{1/s}}{p^+_{\min}}\right)}{\frac{1}{2(s+1)}} = 2(s+1)\cdot d + O\left(\log\left(1/p^+_{\min}\right)\right).$$

Finally, by Lemma 5.4.2 we have $p^+_{\min} \geq n^{-O(n)}$ and thus $\mathrm{E}(T) = O(d + n\log n)$ in case of no failures. Since failures have a probability of $n^{-\Omega(\log n)}$ over $n^{o(\log n)}$ generations and $O(d + n\log n) = n^{o(\log n)}$ using the assumption $d+1 = n^{o(\log n)}$, if we restart the proof every time a failure happens, the expected number of repetitions is $1 + n^{-\Omega(\log n)}$ and all additional costs can be absorbed in the previous bounds. □

5.4.2 Bounding the Number of Evaluations

Now we consider the expected number of fitness evaluations and give the following general result.

Theorem 5.4.10. *Consider the self-adjusting* $(1, \lambda)$ EA *using any mutation operator that ensures $p_x^+ > 0$ for all non-optimal search points x. Let the update strength $F > 1$ and the success rate $s > 0$ be constants. Consider an arbitrary everywhere hard function f with $d + 1 = n^{o(\log n)}$ function values. Then for every initial search point and every initial offspring population size $\lambda_0 = O\left(\sum_{i=0}^{d-1} \frac{1}{s_i}\right)$ the expected number of evaluations to optimise f is at most*

$$O\left(\sum_{i=0}^{d-1} \frac{1}{s_i}\right).$$

The condition $p_x^+ > 0$ is met by standard bit mutation and heavy-tailed mutations. The term $\sum_{i=0}^{d-1} \frac{1}{s_i}$ equals the fitness-level upper bound for the (1+1) EA using the same mutation operator as the considered self-adjusting $(1, \lambda)$ EA. A similar result to Theorem 5.4.10 was shown for the (elitist) self-adjusting $(1+\{2\lambda, \lambda/2\})$ EA from [120]. Our result shows that the same bound also applies in the context of non-elitism, if the fitness function is everywhere hard.

The main proof idea is that given that λ maintains a large value with high probability throughout the optimisation, the algorithm with high probability behaves as an elitist algorithm. This is shown in the next lemma, adapted from Lemma 5.3.9 in Section 5.3.1.

Lemma 5.4.11. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 5.4.10. Let T be the first generation in which the optimum is found. Then for all $t \leq T$ in which $\lambda_t \geq \lambda_{\text{safe}}$, we have $f(x_{t+1}) \geq f(x_t)$ with probability $1 - O(1/(n \log n))$.*

Proof. Let E^t denote the event that $\lambda_t < \lambda_{\text{safe}}$ or $f(x_{t+1}) \geq f(x_t)$. Hence we only need to consider λ_t-values of $\lambda_t \geq \lambda_{\text{safe}}$. We note that

$$\begin{aligned}\lambda_{\text{safe}} &= 4 \max\left(\log_\gamma(2d(s+1)), \log_\gamma(n \log n)\right) \\ &\geq 2\left(\log_\gamma(2d(s+1)) + \log_\gamma(n \log n)\right).\end{aligned}$$

Then by Lemma 5.4.2 we have

$$\Pr\left(\overline{E^t}\right) \leq \gamma^{-\lambda_t} \leq \gamma^{-2\left(\log_\gamma(2d(s+1)) + \log_\gamma(n \log n)\right)} = \frac{1}{(2d(s+1)n \log n)^2}.$$

By a union bound, the probability that this happens in the first T generations is at most

$$\frac{\sum_{t=1}^{\infty} \Pr(T = t) \cdot t}{(2d(s+1)n \log n)^2} = \frac{\mathrm{E}(T)}{(2d(s+1)n \log n)^2}.$$

By Theorem 5.4.4, this is

$$O\left(\frac{d + n \log n}{(dn \log n)^2}\right) = O\left(\frac{1}{d(n \log n)^2} + \frac{1}{d^2 n \log n}\right) = O\left(\frac{1}{n \log n}\right). \qquad \square$$

If the algorithm behaves as an elitist algorithm with high probability, we can bound its expected optimisation time by the expected optimisation time of its elitist version, i. e. a self-adjusting $(1 + \lambda)$ EA. We have already used this argument in Section 5.3.1 for the function OneMax, and the expected optimisation time of the elitist self-adjusting $(1 + \lambda)$ EA was bounded from above in Theorem 5.3.12. Note that the bound in Theorem 5.3.12 is $O\left(\lambda_0 + \sum_{i=a}^{b-1} \frac{1}{s_i}\right)$.

The following lemma bounds the expectation of λ at each step t in order to deal with the case where the self-adjusting $(1, \lambda)$ EA does not behave as an elitist algorithm. It generalises Lemma 5.3.19 for all functions. It follows from the proof of Lemma 5.3.19 for OneMax when replacing the specific lower bound of $\frac{n-i}{en}$ on the success probability on OneMax by the general lower bound $p^+_{\min}$.

Lemma 5.4.12. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 5.4.10. The expected value of* λ *at time* t *is*

$$\mathrm{E}(\lambda_t \mid \lambda_0) \leq \lfloor \lambda_0 / F^t \rfloor + \frac{1}{p^+_{\min}} \cdot \left(F^{1/s} + \frac{F^{1/s}}{\ln F} \right).$$

Now we are in a position to prove Theorem 5.4.10.

Proof of Theorem 5.4.10. By Lemma 5.4.5, λ will grow to λ_{inc} using $O(\lambda_{\text{inc}} \log \lambda_{\text{inc}}) = o(\sqrt{n} \log(n))$ expected evaluations. Afterwards, by Lemma 5.4.6 the offspring population size will maintain a value of at least λ_{safe} with probability $1 - n^{-\Omega(\log n)}$ throughout the optimisation and by Lemma 5.4.11 with probability $1 - O(1/(n \log n))$ the algorithm will behave as an elitist algorithm until the optimum is found. Considering the above rare, undesired events as *failures*, we define $\mathrm{E}(T^*)$ be the expected time of a run with $\lambda_0 = O\left(\sum_{i=0}^{d-1} \frac{1}{s_i}\right)$ until either a global optimum is found or a failure occurs. As long as no failure occurs, Theorem 5.3.12 can be applied with $a := 0$ and thus we obtain $\mathrm{E}(T^*) = O\left(\sum_{i=0}^{d-1} \frac{1}{s_i}\right)$.

Since failures have a probability of $O(1/(n \log n))$, we can restart our arguments whenever a failure happens and in expectation there would be at most $1 + O(1/(n \log n))$ attempts. Arguing as in Theorem 5.3.7, in each restart of the analysis λ_0 would take the λ-value at the time of a failure, denoted as λ_{fail}. By Lemma 5.4.12,

$$\mathrm{E}(\lambda_{\text{fail}}) \leq \lambda_0 + O\left(\frac{1}{p^+_{\min}}\right) \leq O\left(\sum_{i=0}^{d-1} \frac{1}{s_i}\right)$$

and this term, multiplied by $O(1/(n \log n))$, can easily be absorbed in our claimed upper bound. □

5.4.3 Bounds on Unimodal Functions

We now show how to apply Theorems 5.4.4 and 5.4.10 to obtain novel bounds on the expected optimisation time of the self-adjusting $(1, \lambda)$ EA on *everywhere hard* unimodal functions and give specific bounds for the benchmark function LeadingOnes and a new function class that we call OneMaxBlocks that allows us to vary the difficulty of the easiest fitness levels (see Section 2.3 for their definitions).

Recall that LeadingOnes returns the number of 1-bits in the longest prefix that only contains ones. The proposed function class, OneMaxBlocks, has a similar structure. It is comprised of *blocks* of k bits. A block is *complete* if it only contains 1-bits and *incomplete* otherwise. The function returns the number of 1-bits in the longest prefix of completed blocks plus the number of 1-bits in the first incomplete block. Evolutionary algorithms typically optimise this function by optimising each OneMax-like block of size k from left to right until the global optimum 1^n is reached. We can tune the maximum success probability $p^+_{\max}$ by assigning different values to k. If $k = 1$ the function equals LeadingOnes where $p^+_{\max} = \Theta(1/n)$. Increasing k increases the maximum success probability. If $k = n$ the function equals OneMax and $p^+_{\max} = \Theta(1)$.

Theorem 5.4.13. *Let $s > 0$ and $F > 1$ be constants. The expected number of generations and evaluations of the self-adjusting $(1, \lambda)$ EA using standard bit mutation with mutation probability r/n, $r = \Theta(1)$, or heavy-tailed mutations with constant $\beta > 1$ is at most*

1. *$O(d)$ expected generations and $O(dn)$ expected evaluations on all unimodal functions with $\log n \leq d+1 = n^{o(\log n)}$ fitness values that are everywhere hard for the considered algorithm,*
2. *$O(n)$ expected generations and $O(n^2)$ expected evaluations on* LeadingOnes *, and*
3. *$O(n)$ expected generations and $O\left(\frac{n^2 \log k}{k}\right)$ expected evaluations on* OneMaxBlocks *with $1 < k \leq n^{1-\varepsilon}$ for some constant $0 < \varepsilon < 1$ and, additionally, $k \leq n^{\beta-1-\varepsilon}$ if heavy-tailed mutations are used.*

Proof. The set of everywhere hard unimodal functions by definition meet the conditions in Definition 5.4.1, therefore we can apply Theorems 5.4.4 and 5.4.10 directly. We only need to bound p^+_{min}. Given that every search point has a strictly better Hamming neighbour, the success probability of all fitness levels is at least as large as the probability of flipping only one specific bit. For standard bit mutations with mutation probability r/n, this is at least

$$p^+_{\text{min}} \geq \frac{r}{n}\left(1 - \frac{r}{n}\right)^{n-1} \geq \frac{r}{e^r n} \cdot \left(1 - \frac{r}{n}\right)^{1-r} = \Omega(1/n).$$

For heavy-tailed mutations we also have $p^+_{\text{min}} = \Omega(1/n)$ as there is a constant probability of choosing a mutation rate of $1/n$. Applying Theorems 5.4.4 and 5.4.10 yields $O(d + \log n) = O(d)$ generations and $O(dn)$ evaluations in expectation.

For the statement on LeadingOnes we need to show that the function is everywhere hard for the considered algorithms. A necessary condition for an offspring to be better than the parent is to flip the leftmost 0-bit, hence the success probability of standard bit mutation on all fitness levels is bounded by $p^+_x \leq \frac{r}{n}$ and LeadingOnes meets the conditions of an everywhere hard unimodal function. Thus, it is covered by the previous statement with $d := n$. For heavy-tailed mutations, a much larger mutation rate might be used, hence we need to be more careful. Heavy-tailed mutation chooses a mutation probability r^*/n according to a power-law distribution with parameter β, truncated at $n/2$. The probability of choosing a certain mutation rate r/n is

$$\Pr(r^* = r) = \frac{r^{-\beta}}{\sum_{j=1}^{n/2} j^{-\beta}} \leq \frac{r^{-\beta}}{\zeta(\beta) - \rho},$$

where the inequality is taken from [62] and $\zeta(\beta)$ is the Riemann zeta function ζ evaluated at β and $\rho = \frac{\beta}{\beta-1}\left(\frac{n}{2}\right)^{-\beta+1} = o(1)$. Given a mutation probability of r/n (where r is no longer restricted to a constant), the probability of flipping the first 0-bit is r/n. Hence, we have

$$p^+ \leq \sum_{r=1}^{n/2} \frac{r^{-\beta}}{\zeta(\beta) - \rho} \cdot \frac{r}{n} = \frac{\sum_{r=1}^{n/2} r^{1-\beta}}{n(\zeta(\beta) - \rho)} = \Theta\left(\frac{1}{n}\right) \cdot \sum_{r=1}^{n/2} r^{1-\beta}.$$

Since $\beta > 1$, $r^{1-\beta}$ is strictly decreasing with r and, using $r^{1-\beta} \leq \int_{r-1}^{r} x^{1-\beta}\,\mathrm{d}x$, we can bound the sum by an integral:

$$\sum_{r=1}^{n/2} r^{1-\beta} \leq \int_0^{n/2} r^{1-\beta}\,\mathrm{d}r = \frac{(n/2)^{2-\beta}}{2-\beta}.$$

Together, we get

$$p^+ \leq \Theta\left(\frac{1}{n}\right) \cdot \frac{(n/2)^{2-\beta}}{2-\beta} = O(n^{1-\beta})$$

which, recalling $\beta > 1$, meets the definition of everywhere hardness for $\varepsilon := \beta - 1 > 0$.

For OneMaxBlocks, a search point of fitness $i < n$ has $i \bmod k$ 1-bits in its first incomplete block. All non-optimal search points can only be improved by flipping at least one 0-bit in the first incomplete block. Hence, a necessary condition for an offspring to be better than the parent is to flip one of the $k - (i \bmod k) \leq k$ 0-bits in the first incomplete block. Hence, all success probabilities can be bounded by k times the bound on the success probability for LeadingOnes. Thus, for all non-optimal x, $p_x^+ \leq r \cdot k/n \leq rn^{-\varepsilon}$ for standard bit mutations using the assumption $k \leq n^{1-\varepsilon}$ and $p_x^+ \leq O(k \cdot n^{1-\beta}) = O(n^{-\varepsilon})$ for heavy-tailed mutation using the assumption $k \leq n^{\beta-1-\varepsilon}$. For both operators, OneMaxBlocks meets the conditions of an everywhere hard unimodal function.

A sufficient condition for an offspring to be better than its parent of fitness i is to flip only one of the $k - (i \bmod k)$ 0-bits in the first incomplete block, yielding a success probability for standard bit mutations of

$$s_i \geq \frac{r(k - (i \bmod k))}{n} \left(1 - \frac{r}{n}\right)^{n-1}.$$

By Theorem 5.4.4, the self-adjusting $(1, \lambda)$ EA optimises OneMaxBlocks with $k \leq n^{1-\varepsilon}$ in $O(n)$ generations. By Theorem 5.4.10 the expected number of evaluations is at most

$$O\left(\sum_{i=1}^{d-1} \frac{1}{s_i}\right) = O\left(\sum_{i=1}^{n} \frac{\left(1 - \frac{r}{n}\right)^{-n+1} n}{r(k - (i \bmod k))}\right)$$
$$= O\left(\frac{n}{k} \cdot \left(1 - \frac{r}{n}\right)^{-n+1} \sum_{j=1}^{k} \frac{n}{rj}\right) = O\left(\frac{n^2 \log k}{k}\right).$$

If $k = 1$ OneMaxBlocks is equivalent to LeadingOnes and the expected number of evaluations is $O(n^2)$. For heavy-tailed mutations we again use that a mutation rate of $1/n$ is used with constant probability, hence the above asymptotic probability bounds apply as well. □

5.4.4 Very Small Mutation Rates Make All Functions Everywhere Hard

In this short section we remark that, if the self-adjusting $(1, \lambda)$ EA uses standard bit mutation with mutation rate $1/n^{1+\varepsilon}$ for some constant $0 < \varepsilon$ then all functions are everywhere hard since any mutation will create a copy of the parent with probability at least $1 - n^{-\varepsilon}$ and thus the probability of an offspring improving the fitness is at most $p_{\max}^+ \leq n^{-\varepsilon}$. This means that functions like OneMax, where large constant values of s result in exponential runtimes (Theorem 5.3.20 and 5.3.23 in Section 5.3.2), can be solved in polynomial expected time for arbitrary constant values of s. For example, the self-adjusting $(1, \lambda)$ EA with every constant $s > 1$ can solve OneMax in $O(n)$ expected generations and $O(n^{1+\varepsilon} \log n)$ expected evaluations. In the following corollary we make this explicit.

Corollary 5.4.14. *Let $0 < \varepsilon < 1$, the update strength $F > 1$ and the success rate $s > 0$ be constants. For every function f with $d + 1 = n^{o(\log n)}$ fitness levels, the self-adjusting $(1, \lambda)$ EA using standard bit mutation with mutation rate $1/n^{1+\varepsilon}$ optimises f in $O\left(d + \log\left(1/p_{\min}^+\right)\right)$ expected generations and $O\left(\sum_{i=1}^{d-1} \frac{1}{s_i}\right)$ expected evaluations.*

The conclusion is that making an algorithm less efficient (in the sense of introducing large self-loops) can improve performance as the algorithm shows a more stable search behaviour. Similar observations were made before for the $(1, \lambda)$ EA with fixed λ [166] and, in a wider sense, in evolution with partial information [40].

5.5 Experimental Analysis

Due to the complex nature of our analyses there are still open questions about the behaviour of the self-adjusting $(1, \lambda)$ EA. In this section we show some elementary experiments to enhance our understanding of the parameter control mechanism and address these unknowns. All the experiments were performed using the IOHProfiler [74].

5.5.1 Empirical Analyses on OneMax

In Section 5.3.1 we have shown that both the self-adjusting $(1, \lambda)$ EA and the self-adjusting $(1 + \lambda)$ EA have an asymptotic runtime of $O(n \log n)$ evaluations on OneMax. This is the same asymptotic runtime as the static parameters $\lambda = \left\lceil \log_{\frac{e}{e-1}}(n) \right\rceil$ for the $(1, \lambda)$ EA [166]. Unfortunately the asymptotic notation may hide large constants, therefore, our first experiments focus on the comparison of these three algorithms on OneMax. We remark that very recently the conditions for efficient offspring population sizes have been relaxed to $\lambda \geq \left\lceil \log_{\frac{e}{e-1}}(cn/\lambda) \right\rceil$ for any constant $c > e^2$ in [20]. However, this only reduces the best known value of λ by 1 or 2 for the considered problem sizes, and so we stick to the simpler formula of $\lambda = \left\lceil \log_{\frac{e}{e-1}}(n) \right\rceil$, i. e. the best static parameter value reported in [166]. Following Rowe and Sudholt [166] we consider an algorithm inefficient on OneMax if it does not find the optimum within $500n$ generations, hence, all runs in this section were stopped once the optimum was found or after $500n$ generations were reached, unless stated otherwise.

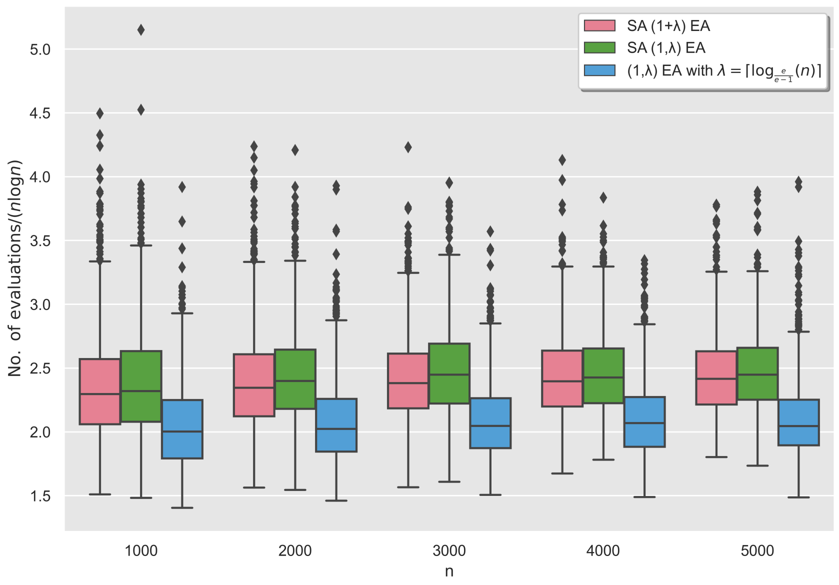

Figure 5.2: Box plots of the number of evaluations used by the self-adjusting $(1, \lambda)$ EA, the self-adjusting $(1+\lambda)$ EA with $s = 1$, $F = 1.5$ and the $(1, \lambda)$ EA over 1000 runs for different n on OneMax. The number of evaluations is normalised by $n \log n$.

Figure 5.2 displays box plots of the number of evaluations over 1000 runs for different problem sizes on OneMax. From Figure 5.2 we observe that the difference between both self-adjusting algorithms is relatively small. This indicates that there are only a small number of fallbacks in fitness and such fallbacks are also small. We also observe that the best static parameter choice $\lambda = \left\lceil \log_{\frac{e}{e-1}}(n) \right\rceil$ from [166] is only a small constant factor faster than the self-adjusting algorithms.

In the results of Sections 5.3.1 and 5.3.2 there is a gap between $s < 1$ and $s \geq 18$ where we do not know how the algorithm behaves on OneMax. In our second experiment, we explore how the algorithm behaves in this region by running the self-adjusting $(1, \lambda)$ EA on OneMax using different values for s shown in Figure 5.3. We found a sharp threshold at $s \approx 3.4$, indicating that the widely used one-fifth rule ($s = 4$) is inefficient here, but other success rules achieve optimal asymptotic runtime.

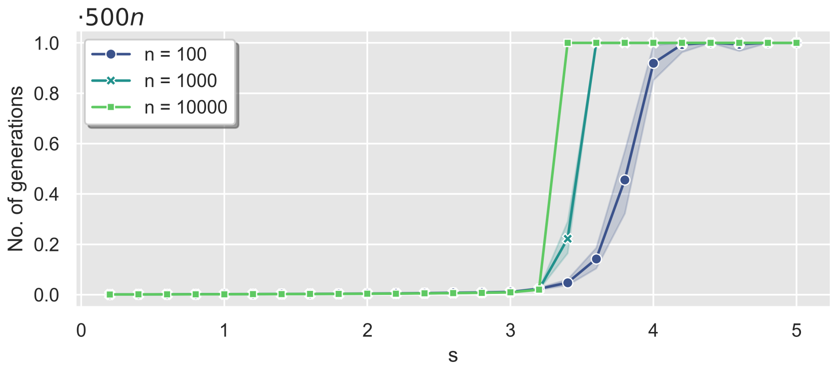

Figure 5.3: Average number of generations with 99% bootstrapped confidence interval of the self-adjusting $(1, \lambda)$ EA with $F = 1.5$ in 100 runs for different n, normalised and capped at $500n$ generations.

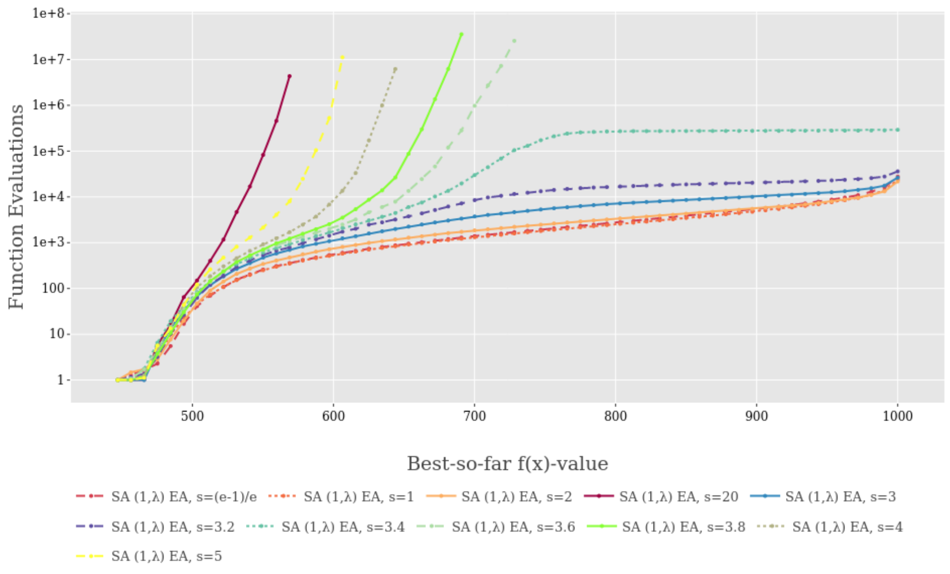

Figure 5.4: Fixed target results for the self-adjusting $(1, \lambda)$ EA on OneMax with $n = 1000$ (100 runs).

Additionally, in Figure 5.4 we plot fixed target results, that is, the average time to reach a certain fitness, for $n = 1000$ for different s. No points are graphed for fitness values that were not reached during the allocated time ($500n$ generations). We note that the plots do not start exactly at $n/2 = 500$; this is due to the random effects of initialisation. From here we found that the range of fitness values with negative drift is wider than what we where able to prove in Section 5.3.2. Already for $s = 3.4$, there is an interval on the scale of number of ones around $0.7n$ where the algorithm spends a large amount of evaluations to traverse this interval. Interestingly, as s increases the algorithm takes longer to reach points farther away from the optimum.

We also explored how the parameter λ behaves throughout the optimisation depending on the value of s. In Figure 5.5 we can see the average λ at every fitness value for $n = 1000$. As expected, on average λ is larger when s is smaller. For $s \geq 3$ we can appreciate that on average $\lambda < 2$ until fitness values around $0.7n$ are reached. This behaviour is what creates the non-stable equilibrium slowing down the algorithm.

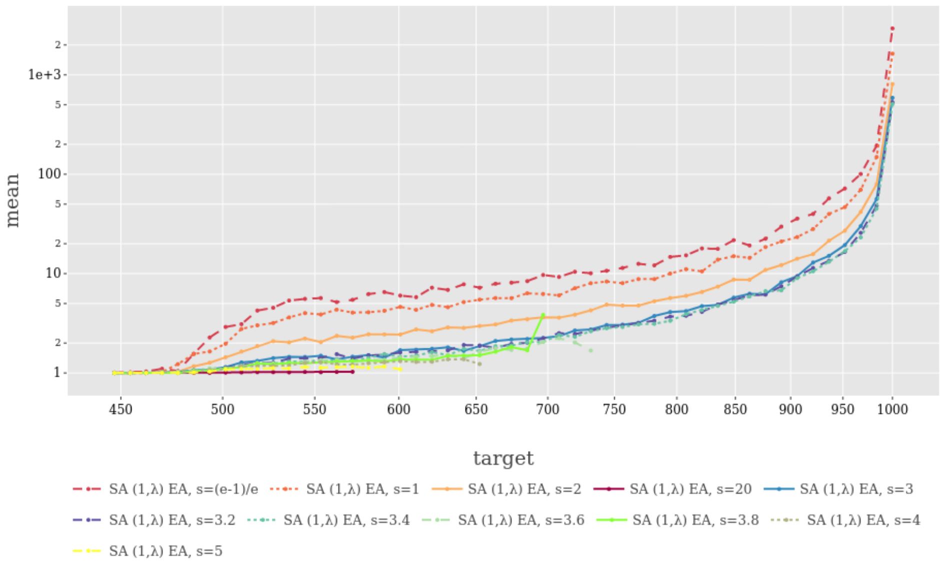

Figure 5.5: Average λ values for each fitness level of the self-adjusting $(1, \lambda)$ EA on OneMax with $n = 1000$ (100 runs).

To identify the area of attraction of the non-stable equilibrium we implemented a set of experiments (100 runs) with $n = 100$ and plotted (Figure 5.6) the percentage of time spent in each fitness level for different s values near the transition between polynomial and exponential. Runs were stopped when the optimum was found or when 1,500,000 function evaluations were made. The first thing to notice is that for $s = 20$ the algorithm is attracted and spends most of the time near $n/2$ ones, which suggests that it behaves similar to a random walk. When s decreases, the area of attraction moves towards the optimum but stays at a linear distance from it. For $s \leq 3.4$ most of the evaluations are spent near the optimum on the harder fitness levels where λ tends to have linear values.

We now explore how small mutation rates help the self-adjusting $(1, \lambda)$ EA optimise OneMax with any constant success rate $s > 0$ as suggested by Corollary 5.4.14. We use a mutation probability $p = 1/n^{1.3}$ and success rates $s \in \{2, 4, \ldots, 20\}$. In Figure 5.7 we see that independent of the value of s chosen the algorithm spends a similar number of evaluations. In addition we also plot the number of evaluations without evaluating *clones* (offspring that is an exact copy of the parent). This is a simple way to save unnecessary evaluations caused by the small mutation rates used. With this simple change the runtime of the self-adjusting $(1, \lambda)$ EA with mutation probability $p = 1/n^{1.3}$ is competitive with its

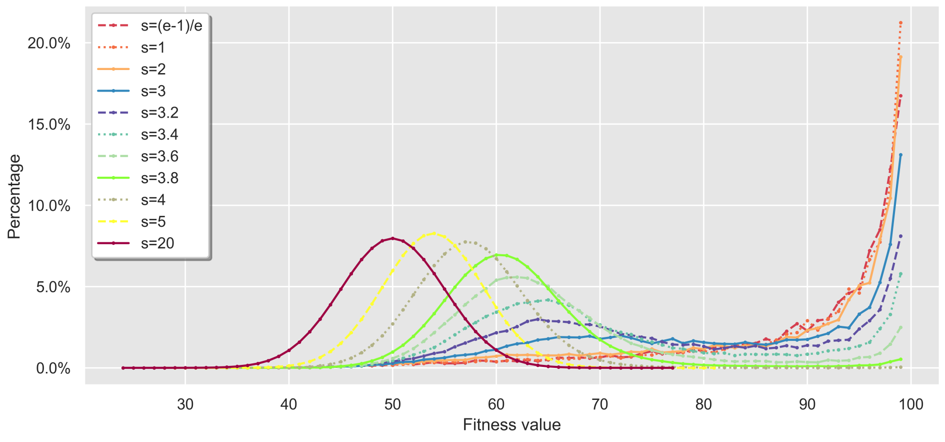

Figure 5.6: Percentage of fitness function evaluations used per fitness value for the self-adjusting $(1, \lambda)$ EA on OneMax with $n = 100$ over 100 runs (runs were stopped when the optimum was found or when 1,500,000 function evaluations were made).

elitist counterpart using the standard mutation rate $p = 1/n$ independently of the success rate chosen.

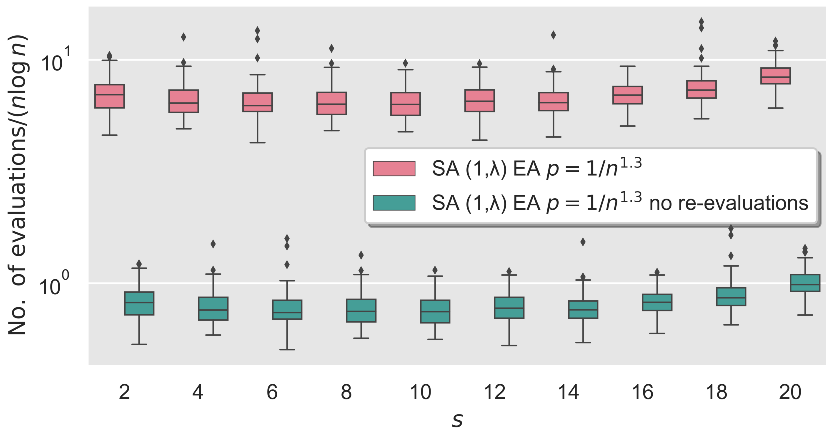

Figure 5.7: Box plots of the number of evaluations used by the self-adjusting $(1, \lambda)$ EA with $F = 1.5$ and $p = 1/n^{1.3}$ over 100 runs for different success rates s on OneMax with $n = 1000$. The number of evaluations is normalised by $n \log n$.

5.5.2 Empirical Analyses on Other Benchmark Functions

Figure 5.8 displays box plots of the number of evaluations over 1000 runs for different problem sizes on LeadingOnes. From Figure 5.8 we observe that the difference between both self-adjusting algorithms is relatively small. This last observation was expected from our analysis, because λ grows to at least a sublinear value before finding an improvement and stays large during the optimisation behaving as an elitist algorithm with high probability. We also observe that the best known static parameter of $\lambda = \left\lceil 2 \log_{\frac{e}{e-1}}(n) \right\rceil$ [166] is only a small constant factor faster than the self-adjusting algorithms.

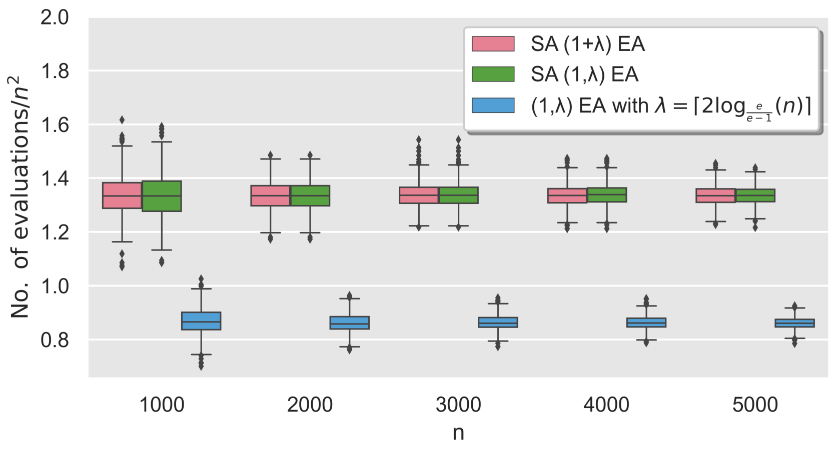

Figure 5.8: Box plots of the number of evaluations used by the self-adjusting $(1, \lambda)$ EA, the self-adjusting $(1+\lambda)$ EA with $s = 1$, $F = 1.5$ and the $(1, \lambda)$ EA over 1000 runs for different n on LeadingOnes. The number of evaluations is normalised by n^2.

From Theorem 5.4.13 we can see that for OneMaxBlocks if k is sufficiently small then the self-adjusting $(1, \lambda)$ EA with any constant $s > 0$ solves the function in polynomial time. But if $k = n$ then OneMaxBlocks is equivalent to OneMax and then the self-adjusting $(1, \lambda)$ EA using a large constant s needs exponential time to solve the function (Theorem 5.3.20). Therefore, there must be phase transition for k between polynomial and exponential runtime when a large constant s is used. In Figure 5.9 we show the number of evaluations over 100 runs of the self-adjusting $(1, \lambda)$ EA and the self-adjusting $(1 + \lambda)$ EA with $s = 10$ and $F = 1.5$ on OneMaxBlocks for $n = 2^{12}$ and different values of k, where large k represent "easy" functions. All runs that did not find the optimum before $10 \cdot 2^{24}$ evaluations are not shown.

We can see that for all $k \leq \sqrt{n}$ ("hard" functions) both algorithms have a similar runtime, but for larger k-values the runtime of the self-adjusting $(1, \lambda)$ EA increases rapidly. This agrees with our theoretical results and follows the behaviour that we expected.

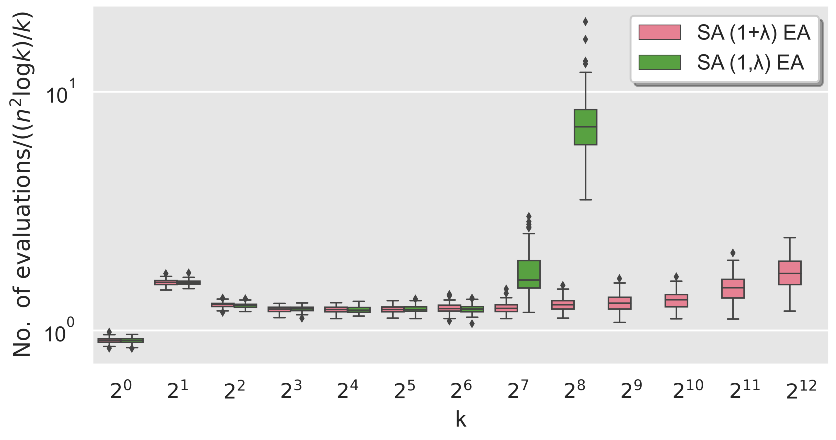

Figure 5.9: Box plots of the number of evaluations used by the self-adjusting $(1, \lambda)$ EA, the self-adjusting $(1 + \lambda)$ EA with $s = 10$, $F = 1.5$ over 100 runs for different k on OneMaxBlocks with $n = 2^{12}$. The number of evaluations is normalised by $(n^2 \log k)/k$.

5.6 Discussion and Conclusions

We have shown that simple success-based rules, embedded in a $(1, \lambda)$ EA, are able to optimise OneMax in $O(n)$ generations and $O(n \log n)$ evaluations. The latter is best possible for any unary unbiased black-box algorithm [67, 128].

However, this result depends crucially on the correct selection of the success rate s. The above holds for constant $0 < s < 1$ and, in sharp contrast, the runtime on OneMax (and other common benchmark problems) becomes exponential with overwhelming probability if $s \geq 18$. Then the algorithm stagnates in an equilibrium state at a linear distance to the optimum where successes are common. Simulations showed that, once λ grows large enough to escape from the equilibrium, the algorithm is able to maintain large values of λ until the optimum is found. Hence, we observe the counterintuitive effect that for too large values of s, optimisation is harder when the algorithm is far away from the optimum and becomes easier when approaching the optimum. (To our knowledge, such an effect was only reported before on HotTopic functions [134] and Dynamic BinVal functions [132].)

There is a gap between the conditions $s < 1$ and $s \geq 18$. Further work is needed to close this gap. In our experiments we found a sharp threshold at $s \approx 3.4$, indicating that the widely used one-fifth rule ($s = 4$) is inefficient here, but other success rules achieve optimal asymptotic runtime.

Additionally, we showed that when s is large the self-adjusting $(1, \lambda)$ EA has an exponential runtime with overwhelming probability on Jump_k, Cliff_d, ZeroMax, TwoMax and Ridge. We believe that these results can be extended for many other functions: we conjecture that for any function that has a large number of contiguous fitness levels that are easy, that is, that the probability of a successful generation with $\lambda = 1$ is constant, then there is a (large) constant success rate s for which the self-adjusting $(1, \lambda)$ EA would have an exponential runtime. We suspect that many combinatorial problem instances are easy somewhere, for example problems like minimum spanning trees, graph colouring, knapsack and MaxSat tend to be easy in the beginning of the optimisation. Furthermore, given that for large values of s the algorithm gets stuck on easy parts of the optimisation and that OneMax is the easiest function with a unique optimum for the $(1 + 1)$ EA [58], we conjecture that any s that is efficient on OneMax would also be a good choice for any other problem.

In stark contrast, we have also shown that the non-elitist self-adjusting $(1, \lambda)$ EA is not affected by the choice of the success rate (from positive constants) if the problem in hand is everywhere hard, that is, improvements are always found with a probability of at most $n^{-\varepsilon}$.

This last results apply to both standard bit mutation as well as heavy-tailed mutations. The expected number of evaluations is bounded by the same fitness-level upper bound as known for the $(1 + 1)$ EA using the same mutation operator. Self-adjusting the offspring population size drastically reduces the number of generations to just $O(d + \log(1/p^{+}_{\min}))$, that is, roughly to the number of fitness values, improving and generalising previous results [120].

As a byproduct of our analysis, we have also shown an upper bound for the expected number of evaluations of the elitist self-adjusting $(1 + \lambda)$ EA on arbitrary fitness functions.

Chapter 6

Benefits of Using Success-Based Rules for Non-elitist Algorithms on Multimodal Problems

6.1 Introduction

In Chapter 4 we have analysed how the one-fifth rule, a common success-based parameter control mechanism, embedded in the elitist $(1 + (\lambda, \lambda))$ GA behaves on multimodal problems. We have shown that the parameter control mechanism tends to diverge its parameters whenever a local optimum is reached, deteriorating the algorithm's performance. We proved that a simple reset mechanism is able to improve the performance of the self-adjusting $(1 + (\lambda, \lambda))$ GA, by allowing the algorithm to cycle through the parameter space whenever a local optima is reached. In each cycle the algorithm is able to use optimal or near-optimal parameters, enhancing its ability to escape local optima.

We have argued in Chapter 5 that analysing the runtime of parameter control mechanisms for non-elitist evolutionary algorithms is an important direction for research. One reason is that non-elitist algorithms are frequently used in practice and understanding the dynamics of non-elitist evolutionary algorithms is vital to narrow the gap between theory and practice. In Chapter 5 we contributed to this effort by analysing a generalisation of the one-fifth rule (the one-$(s + 1)$-th rule) adjusting λ for the non-elitist $(1, \lambda)$ EA. We have shown that for a sufficiently small constant success rate s the self-adjusting $(1, \lambda)$ EA is able to optimise OneMax efficiently. In addition we have shown that if the problem in hand is sufficiently hard, any constant success rate s allows the self-adjusting $(1, \lambda)$ EA to optimise the problem in at most the same asymptotic time as its elitist counterpart, the self-adjusting $(1 + \lambda)$ EA.

In this chapter we will bring everything together by using the lessons learned in Chapter 4 on how to efficiently apply success-based parameter control mechanisms for multimodal optimisation and draw on the insights from our analysis in Chapter 5 to analyse a non-elitist evolutionary algorithm on a multimodal problem where the potential for performance improvements is much greater.

6.1.1 Contributions

We provide an example of significant performance improvements through parameter control for a multimodal problem. We study the $(1, \lambda)$ EA on the multimodal problem Cliff (see Section 2.3) with a mechanism to self-adjust the choice of the offspring population size λ. The function Cliff typically requires evolutionary algorithms to jump down a "cliff" by accepting a huge loss in fitness, and then to climb up a slope towards the global optimum. Elitist evolutionary algorithms are unable to accept this fitness loss and typically need to jump directly to the global optimum (Theorem 8 in [153] gives a tight bound for the (1+1) EA). The $(1,\lambda)$ EA is able to jump down the cliff if and only if *all* offspring are generated at the bottom of the cliff. Hence, the smaller the population size, the higher the probability of jumping down the cliff. However, the $(1,\lambda)$ EA also needs to be able to climb up a OneMax-like slope towards the cliff and towards the global optimum. The offspring population size λ must be large enough to enable hill climbing. Rowe and Sudholt [166] showed that there is a phase transition on OneMax at $\log_{\frac{e}{e-1}} n$. More specifically, $\lambda \geq \log_{\frac{e}{e-1}} n$ is sufficient to optimise OneMax efficiently (relaxed to $\lambda \geq \left\lceil \log_{\frac{e}{e-1}}(cn/\lambda) \right\rceil$ for any constant $c > e^2$ in [20]), in expected $O(n\log(n) + \lambda n)$ evaluations, but all $\lambda \leq (1-\varepsilon)\log_{\frac{e}{e-1}} n$ lead to exponential optimisation times. Every fixed value of λ must strike a delicate balance to enable jumps down the cliff and at the same time being able to hill climb. Jägersküpper and Storch [106] showed a bound of $O(e^{5\lambda})$ for $\lambda \geq 5\ln n$, which gives an upper bound of $O(n^{25})$ for $\lambda = 5\ln n$. To our knowledge, this is the best known upper bound for the runtime of the $(1,\lambda)$ EA to date. A lower bound of $\min\{n^{n/4}, e^{\lambda/4}\}/3$ for all λ was shown in [106]. Comparing the term $e^{\lambda/4} \approx 1.284^\lambda$ to the upper bound of order $e^{5\lambda} \approx 148.413^\lambda$, the exponents (to the base of e) differ by a factor of 20 and the bases to the power of λ differ by a factor larger than 115. This leaves a large polynomial gap between upper and lower bounds for $\lambda = \Theta(\log n)$.

We refine results from [106] and show that the runtime is $\Omega(\lambda\xi^\lambda)$ and $O(\lambda\xi^\lambda \log n)$ for a base of $\xi \approx 6.196878$, for reasonable values of λ. For the recommended fixed λ, we show that the expected runtime is $O(n^\eta \log^2 n)$ for a constant $\eta \approx 3.976770136$, and that it grows faster than any polynomial of degree less than η.

More importantly, we then show that parameter control is highly beneficial in this scenario. We present a self-adjusting $(1,\lambda)$ EA that self-adjusts λ and prove that it is able to optimise Cliff in $O(n)$ expected generations and $O(n\log n)$ expected evaluations. This is faster than any static parameter choice by a factor of $\Omega(n^{2.9767})$ and it is asymptotically the best possible runtime for any unary unbiased black-box algorithm [128].

We remark that this is the first bound of $O(n\log n)$ for a standard evolutionary algorithm on Cliff; previously, $O(n\log n)$ bounds were only achieved by using additional mechanisms such as ageing [36] and hyper-heuristics [137].

Our analysis builds on our previous work in Chapter 5 that analysed a similar algorithm on OneMax. The considered algorithm works using a variant of the famous one-fifth success rule: in a generation in which the current fitness is increased (a success), λ is decreased by a factor of F, where F is a parameter. In an unsuccessful generation, λ is increased by a factor of $F^{1/s}$. The parameter s is called the *success rate* and it implies that, if on average one in $s+1$ generations is successful, the current value of λ is maintained (as we have one success and s unsuccessful generations and so $\lambda_{t+s+1} = \lambda_t \cdot (F^{1/s})^s \cdot 1/F = \lambda_t$).

To make the self-adjusting $(1,\lambda)$ EA work in multimodal optimisation, we need to tackle an important challenge that requires a redesign of the self-adjusting $(1,\lambda)$ EA from Chapter 5. Success-based parameter control mechanisms can be problematic on multimodal problems because once a local optimum is reached the success of previous generations does not give a good indication of what parameters are needed to escape the local optimum. In Chapter 4 we have explored two strategies to solve this problem for the elitist self-adjusting $(1+(\lambda,\lambda))$ GA: capping the value of the parameter and resetting the parameter once it has reached a certain value, allowing the algorithm to cycle through the possible parameter values.

Here, we enhance the self-adjusting $(1,\lambda)$ EA studied in Chapter 5 with a resetting mechanism: whenever λ exceeds a predefined maximum of $\lambda_{\max}$, it is reset to $\lambda = 1$. When such a reset happens at the top of the cliff, there is a good probability of jumping down the cliff.

Note that the resetting mechanism may have unwanted side effects: resets may happen in difficult fitness levels, for instance on Cliff resets may happen when climbing up the slope to the global optimum and successes become rare. Hence we need to choose $\lambda_{\max}$ sufficiently large to mitigate this risk and enhance the analysis of the self-adjusting $(1, \lambda)$ EA on OneMax with additional arguments.

6.2 Preliminaries

We present a runtime analysis of the self-adjusting $(1, \lambda)$ EA using standard bit mutation with mutation probability $p = 1/n$ on the n-dimensional pseudo-Boolean benchmark function Cliff.

The Cliff benchmark function (defined in Section 2.3) was first proposed by Jägersküpper and Storch [106] as an example where non-elitism helps the optimisation process. The function is designed to guide hill-climbing algorithms towards a local optimum (*cliff*) where the global optimum is the only other search point with a higher fitness value but it is at a linear distance from the local optimum. An elitist algorithm then needs to perform a large jump to find the global optimum; a non-elitist algorithm instead can escape the local optimum by performing a fitness-decreasing step that leads to another slope guiding to the global optimum.

Throughout the analysis we assume n is divisible by 3. Also following [137], we say that all search points x with $|x|_1 \leq 2n/3$ form the *first slope* and all other search points form the *second slope.*

The Cliff function was used as a benchmark in several other works, including studies of the Strong Selection Weak Mutation (SSWM) model of evolution [153], artificial immune systems [36] and hyper-heuristics [137].

The self-adjusting $(1, \lambda)$ EA uses the generalised success based rule (one-$(s+1)$-th success rule) to adjust the offspring population size λ. If the fittest offspring y is better than the parent x, the offspring population size is divided by the *update strength* F, and multiplied by $F^{1/s}$ otherwise, with s being the *success rate.*

In this chapter we consider a variation of the self-adjusting $(1, \lambda)$ EA with a resetting mechanism for λ (see Algorithm 8) where λ is reset to 1 if $\lambda = \lambda_{\max}$ and there is an unsuccessful generation. This strategy has also been successfully applied to the self-adjusting mechanism of the $(1 + (\lambda, \lambda))$ GA in Chapter 4. In addition, the strategy is similar to the stagnation detection from [161, 162] in that if $\lambda_{\max}$ is set appropriately when $\lambda = \lambda_{\max}$ the algorithm is likely to be in a local optimum and the behaviour of the algorithm changes. In this case when λ is large enough the algorithm maintains its fitness with high probability, but when λ is reset to 1 the behaviour changes to a pure random walk allowing the algorithm to escape local optima.

As in previous chapters we regard λ to be a real value, so that changes by factors of $1/F$ or $F^{1/s}$ happen on a continuous scale. Following Doerr and Doerr [50], we assume that, whenever an integer value of λ is required, λ is rounded to a nearest integer. For the sake of readability, we often write λ as a real value even when an integer is required.

6.2.1 Transition Probabilities

We now define and estimate transition probabilities that apply to all $(1, \lambda)$ EA variants (with or without self-adjustment) using standard bit mutation in the context of OneMax and Cliff.

Algorithm 8: Self-adjusting $(1, \{F^{1/s}\lambda, \lambda/F\})$ EA resetting λ.

```
1 Initialization: Choose $x \in \{0,1\}^n$ uniformly at random (u.a.r.) and set $\lambda := 1$;
2 Optimization: for $t \in \{1, 2, \dots\}$ do
3     Mutation: for $i \in \{1, \dots, \lfloor\lambda\rceil\}$ do
4         $y'_i \in \{0,1\}^n \leftarrow$ standard bit mutation($x$);
5     Selection: Choose $y \in \{y'_1, \dots, y'_{\lfloor\lambda\rceil}\}$ with $f(y) = \max\{f(y'_1), \dots, f(y'_{\lfloor\lambda\rceil})\}$ u.a.r.;
6     Update: $x \leftarrow y$;
7     if $f(y) > f(x)$ then $\lambda \leftarrow \max\{\lambda/F, 1\}$;
8     if $f(y) \le f(x) \wedge \lambda = \lambda_{\max}$ then $\lambda \leftarrow 1$;
9     if $f(y) \le f(x) \wedge \lambda \ne \lambda_{\max}$ then $\lambda \leftarrow \min\{\lambda F^{1/s}, \lambda_{\max}\}$;
```

Definition 6.2.1. For all $\lambda \in \mathbb{N}$ we state the following definitions from Chapter 5 in the context of OneMax:

$$\begin{aligned}
p^{+}_{i,\lambda} &= \Pr\left(|x_{t+1}|_1 > i \mid |x_t|_1 = i\right)\\
p^{-}_{i,\lambda} &= \Pr\left(|x_{t+1}|_1 < i \mid |x_t|_1 = i\right)\\
\Delta^{-}_{i,\lambda} &= \mathrm{E}(i - |x_{t+1}|_1 \mid |x_t|_1 = i \text{ and } |x_{t+1}|_1 < i)
\end{aligned}$$

The following probabilities and expectations are tailored to the Cliff function. This includes probabilities for jumping down the cliff ($p^{\downarrow}_{i,\lambda}$), that is, jumping from the first slope to the second slope, jumping back up the cliff ($p^{\uparrow}_{i,\lambda}$), that is, jumping from the second slope to the first slope, increasing the fitness without jumping back up the cliff ($p^{\rightarrow}_{i,\lambda}$), and decreasing the fitness without jumping down the cliff ($p^{\leftarrow}_{i,\lambda}$).

Definition 6.2.2. For all $\lambda \in \mathbb{N}$ we define:

$$p^{\downarrow}_{i,\lambda} = \begin{cases} \Pr\left(|x_{t+1}|_1 > 2n/3 \mid 2n/3 \ge |x_t|_1 = i\right) & \text{if } i \le 2n/3\\ 0 & \text{otherwise.}\end{cases}$$

$$p^{\uparrow}_{i,\lambda} = \begin{cases} \Pr\left(|x_{t+1}|_1 \le 2n/3 \mid 2n/3 < |x_t|_1 = i\right) & \text{if } i > 2n/3\\ 0 & \text{otherwise.}\end{cases}$$

$$p^{\rightarrow}_{i,\lambda} = \begin{cases} \Pr\left(i < |x_{t+1}|_1 \le 2n/3 \mid |x_t|_1 = i\right) & \text{if } i \le 2n/3,\\ \Pr\left(i < |x_{t+1}|_1 \mid |x_t|_1 = i\right) & \text{otherwise.}\end{cases}$$

$$p^{\leftarrow}_{i,\lambda} = \begin{cases} \Pr\left(|x_{t+1}|_1 < i \mid |x_t|_1 = i\right) & \text{if } i \le 2n/3,\\ \Pr\left(2n/3 < |x_{t+1}|_1 < i \mid |x_t|_1 = i\right) & \text{otherwise.}\end{cases}$$

$$\Delta^{\leftarrow}_{i,\lambda} = \begin{cases} \mathrm{E}(i - |x_{t+1}|_1 \mid |x_t|_1 = i, |x_{t+1}|_1 < i) & \text{if } i \le 2n/3,\\ \mathrm{E}(i - |x_{t+1}|_1 \mid |x_t|_1 = i, 2n/3 < |x_{t+1}|_1 < i) & \text{otherwise.}\end{cases}$$

$$\Delta^{\rightarrow}_{i,\lambda} = \begin{cases} \mathrm{E}(|x_{t+1}|_1 - i \mid |x_t|_1 = i, i < |x_{t+1}|_1 \le 2n/3) & \text{if } i \le 2n/3,\\ \mathrm{E}(|x_{t+1}|_1 - i \mid |x_t|_1 = i, i < |x_{t+1}|_1) & \text{otherwise.}\end{cases}$$

Finally, for $i > 2n/3$,

$$\Delta^{\uparrow}_{i,\lambda} = \mathrm{E}(i - |x_{t+1}|_1 \mid 2n/3 < |x_t|_1 = i \text{ and } |x_{t+1}|_1 \le 2n/3).$$

The events underlying the probabilities from Definition 6.2.2 are mutually disjoint. They relate to the probabilities defined for OneMax in Chapter 5 as follows:

$$p^{+}_{i,\lambda} = p^{\rightarrow}_{i,\lambda} + p^{\downarrow}_{i,\lambda} \tag{6.1}$$

$$p^{-}_{i,\lambda} = p^{\leftarrow}_{i,\lambda} + p^{\uparrow}_{i,\lambda} \tag{6.2}$$

The following lemma collects bounds on the above transition probabilities that will be used throughout the remainder of this chapter.

Lemma 6.2.3. *For any* $(1,\lambda)$ EA *on* Cliff*, the quantities from Definition 6.2.2 are bounded as follows:*

For all $i \in \{1, \dots, n\}$,

$$p^{\leftarrow}_{i,\lambda} \le p^{-}_{i,\lambda} \le \left(\frac{e-1}{e}\right)^{\lambda} \tag{6.3}$$

$$\Delta^{\leftarrow}_{i,\lambda} \le \Delta^{-}_{i,\lambda} \le \frac{e}{e-1}. \tag{6.4}$$

For all $i \in \{2n/3, \dots, n\}$*, letting* $d := i - 2n/3$,

$$p^{\uparrow}_{i,\lambda} \le \frac{\lambda(i/n)^d}{d!} \tag{6.5}$$

$$\Delta^{\uparrow}_{i,\lambda} \le d+1. \tag{6.6}$$

Finally,

$$p^{\rightarrow}_{i,\lambda} \ge \begin{cases} 1 - \left(1 - \frac{1}{3e}\right)^{\lambda} & \text{if } i < 2n/3, \\ 1 - \left(1 - \frac{n-i}{en}\right)^{\lambda} - p^{\uparrow}_{i,\lambda} & \text{if } i > 2n/3. \end{cases} \tag{6.7}$$

***Proof*.** The first inequality in (6.3), $p^{\leftarrow}_{i,\lambda} \le p^{-}_{i,\lambda}$, follows from (6.2). The stated upper bound on $p^{-}_{i,\lambda}$ was shown in Chapter 5 (Lemma 5.3.1).

We have $\Delta^{\leftarrow}_{i,\lambda} \le \Delta^{-}_{i,\lambda}$ since for $i \le 2n/3$, $\Delta^{\leftarrow}_{i,\lambda} = \Delta^{-}_{i,\lambda}$ by definition of $\Delta^{\leftarrow}_{i,\lambda}$ and otherwise $\Delta^{\leftarrow}_{i,\lambda}$ is capped as only target search points with more than $2n/3$ ones are considered. The second inequality in (6.4) was shown in Chapter 5 (Lemma 5.3.1).

To bound $p^{\uparrow}_{i,\lambda}$, we argue that a necessary requirement for creating an offspring with at most $2n/3$ ones is that d one-bits flip. There are $\binom{i}{d}$ ways for choosing d one-bits and the probability that d specific bits are flipped is $(1/n)^d$. Thus,

$$p^{\uparrow}_{i,1} \le \binom{i}{d}\left(\frac{1}{n}\right)^d \le \frac{i^d}{d!} \cdot \left(\frac{1}{n}\right)^d \le \frac{(i/n)^d}{d!}.$$

Using the union bound over all offspring, we get

$$p^{\uparrow}_{i,\lambda} \le \frac{\lambda(i/n)^d}{d!}.$$

To bound $\Delta^{\uparrow}_{i,\lambda}$ we bound the number of one-bits flipped by the number of bit-flips during a generation conditional on flipping d bits. Let B denote the random number of flipping bits in a standard bit mutation with mutation probability $1/n$, then using Lemma 1.7.3 from [48] we get $\mathrm{E}(B \mid B \ge d) \le d + \mathrm{E}(B) = d+1$. Increasing the offspring population size does not increase $\Delta^{\uparrow}_{i,\lambda}$ because if multiple offspring jump up the cliff, the algorithm will transition to an offspring with a maximum number of ones on the first slope.

To bound $p^{\rightarrow}_{i,\lambda}$ we argue that, for all $i < 2n/3$, if there is an offspring with $i+1$ ones then the number of ones increases. For $|x_t|_1 < 2n/3$ the probability that an offspring flips only one 0-bit is

$$p^{+}_{i,1} \ge \frac{n-i}{n}\left(1 - \frac{1}{n}\right)^{n-1} \ge \frac{n-i}{en}. \tag{6.8}$$

The probability that any of the λ offspring flips only one 0-bit is

$$1 - \left(1 - p^{+}_{i,1}\right)^{\lambda} \geq 1 - \left(1 - \frac{n-i}{en}\right)^{\lambda} \geq 1 - \left(1 - \frac{1}{3e}\right)^{\lambda}.$$

This proves the claimed lower bound on $p^{\rightarrow}_{i,\lambda}$ if $|x_t|_1 < 2n/3$. If $|x_t|_1 > 2n/3$ then there is an offspring with $i+1$ ones with probability

$$1 - \left(1 - \frac{n-i}{en}\right)^{\lambda}.$$

In this case either the number of ones increases or the algorithm goes back up the cliff. Hence,

$$p^{\rightarrow}_{i,\lambda} \geq 1 - \left(1 - \frac{n-i}{en}\right)^{\lambda} - p^{\uparrow}_{i,\lambda}. \qquad \square$$

We also give a lower bound for a mutation creating an offspring from the top of the cliff (that is, a parent with $2n/3$ ones) that increases the number of ones by at least $\frac{c \log\log n}{\log\log\log n}$, for an arbitrary constant $c > 0$.

Lemma 6.2.4. *For every constant $c > 0$ and all $n \geq 2^{2^{2^{3c}}}$ the probability that a standard bit mutation of a search point with $2n/3$ ones yields an offspring with at least $2n/3 + \frac{c \log\log n}{\log\log\log n}$ ones is at least $(\log n)^{-c}$.*

Proof. Let $\kappa := \frac{c \log\log n}{\log\log\log n}$, then a sufficient condition for the sought event is that κ 0-bits flip. Since there are $\binom{n/3}{\kappa}$ ways to choose these flipping bits, the probability is at least

$$\binom{n/3}{\kappa}\left(\frac{1}{n}\right)^{\kappa} \geq \left(\frac{n/3}{\kappa}\right)^{\kappa}\left(\frac{1}{n}\right)^{\kappa} = \left(\frac{1}{3\kappa}\right)^{\kappa} = (3\kappa)^{-\kappa}.$$

Plugging in κ, we get

$$(3\kappa)^{-\kappa} = \left(\frac{3c \log\log n}{\log\log\log n}\right)^{-\frac{c \log\log n}{\log\log\log n}} = 2^{-\frac{c \log\log n}{\log\log\log n} \cdot \log\left(\frac{3c \log\log n}{\log\log\log n}\right)}.$$

By assumption on n, we have $\log\log\log n \geq 3c$, thus $\frac{3c \log\log n}{\log\log\log n} \leq \log\log n$ and we bound the sought probability by

$$2^{-\frac{c \log\log n}{\log\log\log n} \cdot \log\log\log n} = 2^{-c \log\log n} = (\log n)^{-c}. \quad \square$$

6.3 Non-elitist Algorithms with Static Parameters

We first consider the performance of the $(1, \lambda)$ EA with a static choice of λ. The main result in this section is the following theorem that gives upper and lower bounds for the expected optimisation time of the $(1, \lambda)$ EA on Cliff.

Theorem 6.3.1. *The expected optimisation time $\mathrm{E}(T)$ of the $(1, \lambda)$ EA with static λ on Cliff is*

$$\begin{aligned} &\mathrm{E}(T) = \Omega\left(\lambda \xi^{\lambda}\right) && \text{if } \lambda = O(n) \text{ and} \\ &\mathrm{E}(T) = O\left(\lambda \xi^{\lambda} \cdot \log n\right) && \text{if } \log_{\frac{e}{e-1}} n \leq \lambda = O(\log n), \end{aligned}$$

where $\xi^{-1} := \frac{1}{e}\sum_{a=0}^{\infty}\sum_{b=a+1}^{\infty}\left(\frac{2}{3}\right)^{a}\left(\frac{1}{3}\right)^{b}\frac{1}{a!b!} \approx 0.1613715804$ and thus $\xi \approx 6.196878$.

The lower bound is exponential in λ for all $\lambda = O(n)$. The constant ξ^{-1} roughly represents the probability of increasing the number of ones in a mutation of a parent at the top of the cliff, i. e. a parent with $2n/3$ ones. For $\lambda \leq (1-\varepsilon)\log_{\frac{e}{e-1}} n$, that is, if the offspring population size is too small to allow for hill climbing on OneMax, a much stronger lower bound of $2^{cn^{\varepsilon/2}}$, for some constant $c > 0$, was shown in [166] for all functions with a unique optimum. In the (arguably more interesting) parameter regime $\lambda > (1-\varepsilon)\log_{\frac{e}{e-1}} n$ our result improves upon the only other lower bound we are aware of: [106] showed a lower bound for all λ of $\min\{n^{n/4}, e^{\lambda/4}\}/3$. The term $e^{\lambda/4}$ is roughly 1.284^{λ} and hence considerably lower than our lower bound of $\lambda \cdot 6.196^{\lambda}$. In this parameter regime our lower bound is nearly tight: for the recommended values of λ for OneMax [166], the upper bounds only differ from the lower bound by a logarithmic factor.

To prove Theorem 6.3.1, we first show that, for all search points with at most $3n/4$ ones, improvements are found easily if $\lambda \geq \log_{\frac{e}{e-1}} n$. Note that the considered set of search points includes search points on the first slope as well as search points on the second slope at a linear distance past the top of the cliff. We will see that, once a search point with at least $3n/4$ ones has been reached, the algorithm will not jump back up the cliff, with high probability. The choice of the constant $3/4$ is somewhat arbitrary; we could have chosen any other constant in $(2/3, 1)$.

Lemma 6.3.2. *Consider the* $(1,\lambda)$ EA *with* $\log_{\frac{e}{e-1}} n \leq \lambda = O(\log n)$ *and a current search point* x_t *with* $|x_t|_1 \leq 3n/4$. *Then the following statements hold.*

1. *The probability of creating an offspring with* $|x_t|_1 + 1$ *ones is at least* $1 - n^{-0.2}$.
2. *For all* x_t *with* $|x_t|_1 \in [0, 3n/4] \setminus [2n/3, 2n/3 + \log n]$, *the drift in the number of ones is* $\mathrm{E}(|x_{t+1}|_1 - |x_t|_1 \mid x_t) \geq 1 - o(1)$.
3. *For every* $\kappa \in [0, n/12]$, *the expected number of generations until a search point with exactly* $2n/3$ *ones or at least* $2n/3 + \kappa$ *ones is reached is* $O(n)$.

Proof. The probability that one fixed offspring has more than $|x|_1$ ones is at least $(n/4)/(en) = 1/(4e)$. The probability that there is an offspring that increases the number of ones is at least

$$\begin{aligned}1 - \left(1 - \frac{1}{4e}\right)^{\log_{\frac{e}{e-1}} n} &\geq 1 - \left(\frac{4e}{4e-1}\right)^{-\log_{\frac{4e}{4e-1}}(n)/\log_{\frac{4e}{4e-1}}(\frac{e}{e-1})} \\ &= 1 - n^{-1/\log_{\frac{4e}{4e-1}}(\frac{e}{e-1})} \geq 1 - n^{-0.2}.\end{aligned}$$

For the second statement, let A^{+1} denote the event from the first statement, that is, creating an offspring with $|x_t|_1+1$ ones and let $A^{\uparrow}$ be the event that $|x_t|_1 > 2n/3$ and $|x_{t+1}|_1 \leq 2n/3$. By the law of total probability, abbreviating $\Delta := (|x_{t+1}|_1 - |x_t|_1 \mid x_t)$,

$$\begin{aligned}\mathrm{E}(\Delta) = \mathrm{E}\Big(\Delta \mid A^{+1}, \overline{A^{\uparrow}}\Big) \cdot \Pr\left(A^{+1}, \overline{A^{\uparrow}}\right) + \mathrm{E}\Big(\Delta \mid \overline{A^{+1}}, \overline{A^{\uparrow}}\Big) \cdot \Pr\left(\overline{A^{+1}}, \overline{A^{\uparrow}}\right) \\ + \mathrm{E}\big(\Delta \mid A^{\uparrow}\big) \cdot \Pr\left(A^{\uparrow}\right).\end{aligned}$$

The first term is at least $1 \cdot \left(1 - \Pr\left(\overline{A^{+1}}\right) - \Pr\left(A^{\uparrow}\right)\right) \geq 1 - n^{-0.2} - p^{\uparrow}_{i,\lambda}$ by the first statement and a union bound. The second term is at least $-(\log(n) + n^{-\omega(1)} \cdot n) n^{-0.2} = -o(1)$, since the probability of flipping at least $\log n$ bits is $n^{-\omega(1)}$, also under conditions $\overline{A^{+1}}, \overline{A^{\uparrow}}$, and using the trivial bound n if this happens nevertheless. The third term is $-\Delta^{\uparrow}_{i,\lambda} p^{\uparrow}_{i,\lambda}$ by definition. Plugging this together, we get

$$\begin{aligned}\mathrm{E}(\Delta) &\geq 1 - n^{-0.2} - p^{\uparrow}_{i,\lambda} - o(1) - \Delta^{\uparrow}_{i,\lambda} p^{\uparrow}_{i,\lambda} \\ &= 1 - n^{-0.2} - (\Delta^{\uparrow}_{i,\lambda} + 1) p^{\uparrow}_{i,\lambda} - o(1).\end{aligned}$$

For $|x_t|_1 < 2n/3$, $p^{\uparrow}_{i,\lambda} = 0$ and the claim follows. For $|x_t|_1 > 2n/3 + \log n$, by Lemma 6.2.3 with $d \geq \log n$,

$$(\Delta^{\uparrow}_{i,\lambda} + 1)p^{\uparrow}_{i,\lambda} \leq \frac{\lambda(d+2)}{d!} \leq \frac{\lambda(\log(n)+2)}{(\log n)!} = n^{-\omega(1)}.$$

This implies the second statement.

For the third statement, we first note that the second statement also holds for the drift on the function OneMax, for all x_t with $|x_t|_1 \leq 3n/4$, as the negative terms involving $p^{\uparrow}_{i,\lambda}$ disappear. Let us first consider the case that the current search point has at most $2n/3$ ones. Then, by the additive drift theorem (Theorem 2.4.14), the expected time until a search point with at least $2n/3$ ones is reached is $O(n)$. Note that is it possible (though unlikely) that the top of the cliff is skipped and the algorithm jumps down the cliff from a search point with at most $2n/3 - 1$ ones. By the first statement of this lemma, the conditional probability of this happening, conditional on increasing the number of ones, is $O(n^{-0.2})$. If it happens regardless, we consider the following case of having more than $2n/3$ ones.

If the current search point has more than $2n/3$ ones, we argue that on OneMax, by the same drift arguments as above, the expected time to reach a search point with at least $2n/3 + \kappa$ ones is $O(\kappa) = O(n)$. We only see a difference to OneMax if the algorithm jumps back up the cliff. Then we are left with a search point of at most $2n/3$ ones and we apply the above arguments.

If $T(n)$ denotes the worst-case time with respect to the initial number of ones $i < 2n/3+\kappa$, we have shown the recurrence: $T(n) \leq O(n) + O(n^{-0.2}) \cdot T(n)$. It is easy to see that $T(n) = O(n)$. □

Another important step for proving Theorem 6.3.1 is estimating the probability of a standard bit mutation of a parent at the top of the cliff increasing the number of ones, $p^{+}_{2n/3,1}$. This is because in order to jump down the cliff, all offspring must increase the number of ones, which has a probability of $(p^{+}_{2n/3,1})^{\lambda}$. To prove the claimed upper and lower bounds in Theorem 6.3.1 we need precise estimations of $p^{+}_{2n/3,1}$ as it appears in the base of an expression exponential in λ; the commonly used inequalities $\frac{n-2n/3}{en} \leq p^{+}_{2n/3,1} \leq \frac{n-2n/3}{n}$ (that is, $\frac{1}{3e} \leq p^{+}_{2n/3,1} \leq \frac{1}{3}$) are too loose. The following lemma gives precise upper and lower bounds on $p^{+}_{i,1}$ for almost all values of i as this generality is achieved quite easily and the lemma may be of independent interest.

Lemma 6.3.3. *For all $i \in \{0, \dots, n-1\}$,*

$$p^{+}_{i,1} \leq \left(1 - \frac{1}{n}\right)^{n-2} \sum_{a=0}^{\infty} \sum_{b=a+1}^{\infty} \left(\frac{i}{n}\right)^{a} \left(\frac{n-i}{n}\right)^{b} \frac{1}{a!b!}. \tag{6.9}$$

For all $i \in \{\lceil \log n \rceil, \dots, n - \lceil \log n \rceil\}$,

$$p^{+}_{i,1} \geq \left(1 - \frac{1}{n}\right)^{n-2} \sum_{a=0}^{\infty} \sum_{b=a+1}^{\infty} \left(\frac{i}{n}\right)^{a} \left(\frac{n-i}{n}\right)^{b} \frac{1}{a!b!} \left(1 - \frac{2\log^2 n}{\min\{i, n-i\}}\right). \tag{6.10}$$

Proof. Let f^{ℓ}_{k} denote the probability of flipping k bits out of a set of ℓ bits. We have an increase in the number of ones if more zeros flip than ones. Since mutation treats ones and zeros independently,

$$p^{+}_{i,1} = \sum_{a=0}^{\infty} \sum_{b=a+1}^{\infty} f^{i}_{a} \cdot f^{n-i}_{b}. \tag{6.11}$$

(We let these sums run to ∞ to ease notation, but only finitely many terms with $a \leq i$ and $b \leq n - i$ are non-zero.)

We bound f_k^ℓ from above and below to show the claim. For all $1 \le k \le \ell \le n$,

$$\begin{aligned}
f_k^\ell &= \binom{\ell}{k}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-k}\\
&= \frac{\ell(\ell-1)\cdot\ldots\cdot(\ell-k+1)}{k!}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-k}\\
&\le \frac{\ell(\ell-\ell/n)^{k-1}}{k!}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-k}\\
&= \frac{1}{k!}\left(\frac{\ell}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-1}.
\end{aligned}$$

For $k=0$ we have $f_k^\ell = \left(1-\frac{1}{n}\right)^\ell \le \frac{1}{k!}\left(\frac{\ell}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-1}$ as well. Plugging this into (6.11) establishes the claimed upper bound.

To bound $p_{i,1}^+$ from below, we note that the probability of flipping at least $\log n$ bits is at most $1/((\log n)!) \le (e/\log n)^{\log n} = n^{-\Omega(\log\log n)}$. Thus,

$$p_{i,1}^+ - \sum_{a=0}^{\log n}\sum_{b=a+1}^{\log n} f_a^i \cdot f_b^{n-i} = n^{-\Omega(\log\log n)}.$$

Now, for $1 \le k \le \log n$ and $k \le \ell \le n$,

$$\begin{aligned}
f_k^\ell &= \binom{\ell}{k}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-k}\\
&= \frac{\ell(\ell-1)\cdot\ldots\cdot(\ell-k+1)}{k!}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-k}\\
&\ge \frac{(\ell-\log n)^k}{k!}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-1}\\
&= \frac{1}{k!}\left(\frac{\ell}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-1}\left(1-\frac{\log n}{\ell}\right)^k\\
&\ge \frac{1}{k!}\left(\frac{\ell}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-1}\left(1-\frac{k\log n}{\ell}\right)\\
&\ge \frac{1}{k!}\left(\frac{\ell}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-1}\left(1-\frac{\log^2 n}{\ell}\right).
\end{aligned}$$

For $k=0$ we have $f_k^\ell = \left(1-\frac{1}{n}\right)^\ell = \frac{1}{k!}\left(\frac{\ell}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-1}\left(1-\frac{1}{n}\right) \ge \frac{1}{k!}\left(\frac{\ell}{n}\right)^k\left(1-\frac{1}{n}\right)^{\ell-1}\left(1-\frac{\log^2 n}{\ell}\right)$ as well. Plugging this into (6.11), the product of error terms for $\ell = i$ and $\ell = n-i$ becomes

$$\left(1-\frac{\log^2 n}{i}\right)\left(1-\frac{\log^2 n}{n-i}\right) \ge 1 - \frac{2\log^2 n}{\min\{i, n-i\}} + \frac{\log^4 n}{i(n-i)}.$$

Since $p_{i,1}^+ \ge \Omega(1/n)$, we drop the $\frac{\log^4 n}{i(n-i)}$ term to compensate for the error term $n^{-\Omega(\log\log n)}$ that accounts for flipping at least $\log n$ bits. This establishes the claimed lower bound. $\square$

For the specific value $i = 2n/3$, we obtain the following special case. Along with $\frac{1}{e} \le \left(1-\frac{1}{n}\right)^{n-2} \le \frac{1}{e}\cdot(1+O(1/n))$, Equations (6.9) and (6.10) in Lemma 6.2.3 imply the following.

Corollary 6.3.4.

$$p^+_{2n/3,1} = \frac{1}{e}\sum_{a=0}^{\infty}\sum_{b=a+1}^{\infty}\left(\frac{2}{3}\right)^a\left(\frac{1}{3}\right)^b\frac{1}{a!b!} \pm O\left(\frac{\log^2 n}{n}\right)$$

which is approximately $0.1613715804 \pm O(\log^2(n)/n)$.

Now we are ready to prove Theorem 6.3.1.

Proof of Theorem 6.3.1. By Chernoff bounds, with probability $1 - e^{-\Omega(n)}$, the initial search point has at most $2n/3$ ones. In the following, we assume that in the first $\Theta(\xi^\lambda)$ expected generations, no mutation ever flips at least $n/3$ bits. The probability of flipping at least $n/3$ bits in one mutation is at most $1/((n/3)!) = n^{-\Omega(n)}$ and a union bound over λ offspring and $\Theta(\xi^\lambda)$ expected generations (cf. (2.4)) still yields a probability of $n^{-\Omega(n)}$.

Under this assumption, a necessary condition for finding the optimum is that a transition from a search point with at most $2n/3$ ones to a search point with more than $2n/3$ ones is made (i.e. a jump down the cliff). By Lemma 1 in [182], the probability of this event is maximised if the parent has exactly $2n/3$ ones; then it is $(p^+_{2n/3,1})^\lambda$. By Equation (6.9) in Lemma 6.2.3, along with $(1-1/n)^{n-2} \le 1/e \cdot (1-1/n)^{-2} \le 1/e \cdot (1+2/n)$,

$$(p^+_{2n/3,1})^\lambda \le \left(\xi^{-1}\left(1+\frac{2}{n}\right)\right)^\lambda \le \xi^{-\lambda}\cdot e^{2\lambda/n} = O(\xi^{-\lambda}).$$

The expected waiting time for this transition to happen is at least $\Omega(\xi^\lambda)$. This establishes a lower bound for the number of generations of $(1 - e^{-\Omega(n)} - n^{-\Omega(n)}) \cdot \Omega(\xi^\lambda) = \Omega(\xi^\lambda)$. Multiplying the expected number of generations by λ yields the claimed lower bound for the number of evaluations.

Now we show the upper bound. We consider the number of generations T_κ until a search point with at least $2n/3 + \kappa$ ones is found, for $\kappa := \frac{2\log\log n}{\log\log\log n}$, assuming that the current search point is at the top of the cliff, that is, the current search point has $2n/3$ ones.

The expected number of generations to return to the top of the cliff, or to find a search point with at least $2n/3 + \kappa$ ones, is bounded by $O(n)$ by Lemma 6.3.2.

Let p^- denote the probability of accepting a search point with less than $2n/3$ ones from the top of the cliff and let $p^+ = (p^+_{2n/3,1})^\lambda$ denote the probability of accepting a search point with more than $2n/3$ ones from the top of the cliff.

By Lemma 6.2.4 with $c := 2$, the probability of creating an offspring at distance at least $\kappa := \frac{2\log\log n}{\log\log\log n}$ from the cliff is at least $1/\log^2 n$. This clearly also holds under the condition of the event underlying p^+, that is, that all offspring have more than $2n/3$ ones. The probability that there is one such offspring is at least

$$1 - \left(1 - \frac{1}{\log^2 n}\right)^\lambda \ge \frac{\lambda/\log^2 n}{1+\lambda/\log^2 n} \ge \frac{\lambda}{2\log^2 n} \ge \frac{1}{4\log n},$$

where the first inequality follows from Lemma 2.4.1 (d) and the last inequality follows from $\lambda \ge \log_{\frac{e}{e-1}} n \ge \frac{1}{2}\log_2 n$.

Together, we have established a recurrence for $\mathrm{E}(T_\kappa)$:

$$\begin{aligned}\mathrm{E}(T_\kappa) &\le 1 + p^-(O(n) + \mathrm{E}(T_\kappa)) \\ &\quad + p^+\left(O(n) + \left(1 - \frac{1}{4\log n}\right)\mathrm{E}(T_\kappa)\right) + (1 - p^+ - p^-)\mathrm{E}(T_\kappa).\end{aligned}$$

This is equivalent to

$$\frac{p^+}{4\log n}\cdot \mathrm{E}(T_\kappa) \le 1 + p^- O(n) + p^+ O(n). \tag{6.12}$$

We argue that, at the top of the cliff, the probability of moving to a search point with a different number of ones is $p^- + p^+ = O(1/n)$. This is because if there is a mutation that does not flip any bits, the next search point will be at the top of the cliff again. Hence, in order to move to a search point with a different number of ones, all offspring must flip at least one bit. The probability of this event is at most, using $(1-1/n)^n = (1-1/n)\cdot(1-1/n)^{n-1} \geq 1/e - 1/(en)$,

$$\begin{aligned} p^- + p^+ &\leq \left(1 - \left(1 - \frac{1}{n}\right)^n\right)^\lambda \\ &\leq \left(1 - \frac{1}{e} + \frac{1}{en}\right)^\lambda = \left(1 - \frac{1}{e}\right)^\lambda \left(1 + \frac{1}{(e-1)n}\right)^\lambda \\ &= \left(\frac{e-1}{e}\right)^{\log_{\frac{e}{e-1}} n} \left(1 + \frac{1}{(e-1)n}\right)^\lambda \\ &\leq \frac{1}{n} \cdot e^{\lambda/((e-1)n)} = O(1/n) \end{aligned}$$

using $\lambda = O(n)$ in the last step. Plugging this in to (6.12) and multiplying both sides by $4\log(n)/p^+$, we get

$$\mathrm{E}(T_\kappa) \leq O\left(\frac{\log n}{p^+}\right).$$

Now assume that a search point with $2n/3 + d$ ones has been reached, where $\kappa \leq d \leq \log n$. By Lemma 6.3.2 and Lemma 6.2.3, the probability that in one generation the number of ones is increased (i.e. the algorithm does not jump up the cliff again) is at least

$$1 - n^{-0.2} - \frac{\lambda(3/4)^d}{d!}.$$

By a union bound, the probability that this happens for all $d = \kappa \dots n^{0.1}$ is at least

$$1 - \frac{n^{0.1}}{n^{0.2}} - \lambda \sum_{d=\kappa}^{n^{0.1}} \frac{(3/4)^d}{d!}.$$

The sum is bounded from above, using $d! \geq (d/e)^e$, as

$$\begin{aligned} \sum_{d=\kappa}^{n^{0.1}} \frac{(3/4)^d}{d!} \leq \sum_{d=\kappa}^{n^{0.1}} \left(\frac{3e}{4d}\right)^d &\leq \sum_{d=\kappa}^{\infty} \left(\frac{3e}{4\kappa}\right)^d \\ &= \left(\frac{3e}{4\kappa}\right)^\kappa \cdot \sum_{d=0}^{\infty} \left(\frac{3e}{4\kappa}\right)^d \\ &= \left(\frac{3e}{4\kappa}\right)^\kappa \cdot \frac{1}{1 - \frac{3e}{4\kappa}} \leq 2\left(\frac{3e}{4\kappa}\right)^\kappa. \end{aligned}$$

Plugging in κ, we get

$$\begin{aligned} 2\left(\frac{3e \log\log\log n}{4\log\log n}\right)^{\frac{2\log\log n}{\log\log\log n}} &\leq 2 \cdot 2^{\frac{2\log\log n}{\log\log\log n}\cdot \log\left(\frac{3e\log\log\log n}{4\log\log n}\right)} \\ &= 2 \cdot 2^{-\frac{2\log\log n}{\log\log\log n}\cdot \log\left(\frac{4\log\log n}{3e\log\log\log n}\right)}. \end{aligned}$$

For large enough n, we have

$$\frac{4\log\log n}{3e\log\log\log n} \geq 2(\log\log n)^{1/2}$$

and then

$$\log\left(\frac{4\log\log n}{3e\log\log\log n}\right) \le 1 + \frac{1}{2}\log\log\log n.$$

Plugging this in, we bound the sum by

$$2\cdot 2^{-\frac{2\log\log n}{\log\log\log n}\cdot\left(1+\frac{1}{2}\log\log\log n\right)} \le 2\cdot 2^{-\log\log n - \frac{2\log\log n}{\log\log\log n}} = o\left(\frac{1}{\log n}\right).$$

Thus, the probability of reaching a search point with at least $2n/3 + n^{0.1}$ ones before going back up the cliff is at least

$$1 - \frac{n^{0.1}}{n^{0.2}} - \lambda\sum_{d=\kappa}^{\log n}\frac{(3/4)^d}{d!} \ge 1 - o(1).$$

From that point on, for any search point with at least $2n/3 + n^{0.1}/2$ ones, we can use arguments from the analysis of the $(1,\lambda)$ EA on OneMax in [166] since the $(1,\lambda)$ EA must flip at least $n^{0.1}/2$ bits to return to the cliff, and this has exponentially small probability. It is not difficult to show using the negative drift theorem (Theorem 2.4.16) and the second statement of Lemma 6.3.2, that, once we have reached a distance of at least $n^{0.1}$ from the cliff, reducing that distance to at most $n^{0.1}/2$ has an exponentially small probability. Assuming we never go back up the cliff, the remaining expected optimisation time is $O(n\log n)$ [166].

In total, the expected optimisation time (number of evaluations) is

$$O(1)\cdot\lambda\mathrm{E}(T_\kappa) = O\left(\frac{\lambda\log n}{p^+}\right) = O\left(\frac{\lambda\log n}{(p^+_{2n/3,1})^\lambda}\right).$$

This implies the claim since $p^+_{2n/3,1} = \xi^{-1} - O((\log^2 n)/n)$ by Corollary 6.3.4 and

$$\begin{aligned}\left(p^+_{2n/3,1}\right)^\lambda &= \left(\xi^{-1} - \frac{O(\log^2 n)}{n}\right)^\lambda = \xi^{-\lambda}\left(1 - \frac{O(\log^2 n)}{n}\right)^\lambda\\ &\ge \xi^{-\lambda}\left(1 - \frac{O(\lambda\log^2 n)}{n}\right) = \Omega(\xi^{-\lambda}).\end{aligned}$$

□

Along with the exponential lower bound from [166] for small λ-values, Theorem 6.3.1 shows that the expected optimisation time for any λ grows faster than any polynomial with degree less than $\eta \approx 3.976770136$.

Theorem 6.3.5. *Let $\eta := 1/\log_\xi\left(\frac{e}{e-1}\right) \approx 3.976770136$. The expected optimisation time of the $(1,\lambda)$ EA with static λ is $\omega(n^{\eta-\varepsilon}\log n)$ for every constant $\varepsilon > 0$ and every λ and $O(n^\eta\log^2 n)$ for $\lambda = \lceil\log_{\frac{e}{e-1}} n\rceil$.*

Proof. By Theorem 6.3.1, the expected optimisation time for $\lambda = \lceil\log_{\frac{e}{e-1}} n\rceil$ is $O(\xi^\lambda\log^2 n)$. Using

$$\xi^\lambda \le \xi^{\log_{1+\frac{e}{e-1}} n} = \xi^{1+\log_\xi(n)/\log_\xi(\frac{e}{e-1})} = \xi n^{1/\log_\xi(\frac{e}{e-1})} = \xi n^\eta$$

establishes the claimed upper bound.

For the lower bound, we exploit that for every constant $\varepsilon' > 0$, for all $\lambda \le (1-\varepsilon')\log_{\frac{e}{e-1}} n$ the expected optimisation time of the $(1,\lambda)$ EA on every function with a unique optimum is at least $2^{cn^{\varepsilon'/2}}$, for some constant $c > 0$, by Theorem 10 in [166]. This is clearly in $\omega(n^\eta\log n)$. For $\lambda = \omega(n)$ the lower bound $\min\{n^{n/4}, e^{\lambda/4}\}/3$ from Theorem 8 in [106] is exponential. It thus suffices to consider $\lambda > (1-\varepsilon')\log_{\frac{e}{e-1}} n$ and $\lambda = O(n)$. The lower bound from Theorem 6.3.1 then becomes $\Omega(\xi^{(1-\varepsilon')\log_{\frac{e}{e-1}} n}\log n) = \Omega(n^{(1-\varepsilon')\eta}\log n)$. Choosing $\varepsilon' := \varepsilon/(2\eta)$, this is $\Omega(n^{\eta-\varepsilon/2}\log n) = \omega(n^{\eta-\varepsilon}\log n)$ as claimed. □

6.4 Provable Performance Gains applying Success-Based Rules

In this section we show that the self-adjusting $(1, \lambda)$ EA is faster than the $(1, \lambda)$ EA with static parameter choice by a polynomial factor of $\Theta(n^{2.9767})$, achieving the best possible asymptotic runtime for any unary unbiased black-box algorithm of $O(n \log n)$ [128]. The main result of this section is shown in Theorem 6.4.1.

Theorem 6.4.1. *Let the update strength $F > 1$ and the success rate $0 < s < 1$ be constants and $enF^{1/s} \leq \lambda_{\max} = \text{poly}(n)$. Then for any initial search point and any initial $\lambda_0 \leq \lambda_{\max}$ the self-adjusting* $(1, \lambda)$ EA *resetting λ optimises* Cliff *in $O(n)$ expected generations and $O(\lambda_{\max} \log n)$ expected function evaluations.*

For $\lambda_{\max} = \lceil enF^{1/s} \rceil$ we get $O(n \log n)$ evaluations in expectation.

The proof of our result is divided in four phases: reaching the cliff, jumping down the cliff, climbing away from the cliff and finding the global optimum.

6.4.1 Reaching the Cliff

We note that the algorithm studied here is the same as the algorithm studied in Chapter 5 as long as all generations that use $\lambda = \lambda_{\max}$ are successful. Here we show that before reaching the cliff the probability of an unsuccessful generation with $\lambda = \lambda_{\max}$ is sufficiently small to not affect the optimisation. Additionally, the results from Chapter 5 on OneMax can be applied when only considering improvements that increase the fitness by 1. Hence, they can be translated to the Cliff function to calculate the time the algorithm takes to reach the cliff, giving the following bounds.

Lemma 6.4.2 (Adapted from Theorem 5.3.7). *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 6.4.1. For every initial offspring population size $\lambda_0 \leq \lambda_{\max}$ and every initial search point x_0 with $|x_0|_1 = 2n/3 - a$ for $a \geq 1$ the algorithm evaluates a solution x_t with $|x_t|_1 \geq 2n/3$ using in expectation $O(a + \log n)$ generations and $O(a + \log n + \lambda_0)$ evaluations.*

We translate the results from Chapter 5 on OneMax to the first slope of Cliff, therefore, before giving a proof for Lemma 6.4.2 we first show that w. o. p. the self-adjusting $(1, \lambda)$ EA does not reset λ in this region and consequently it behaves as the algorithm studied in Chapter 5. By (2.4) this holds for any random time period of polynomial expected length. The following lemma concerns a larger region of search points with up to $3n/4$ ones as this will be useful later on.

Lemma 6.4.3. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 6.4.1. The probability that in a generation t with $|x_t|_1 \leq 3n/4$ and $|x_t|_1 \neq 2n/3$ the self-adjusting* $(1, \lambda)$ EA *resets λ is at most $e^{-\Omega(n)}$.*

Proof. In order to reset λ a generation using $\lambda = \lambda_{\max}$ must not increase the fitness. By Lemma 6.2.3 with $\lambda = \lambda_{\max} \geq enF^{1/s}$ the probability of this event is at most

$$1 - p^{\rightarrow}_{\lambda_{\max}} \leq \left(1 - \frac{1}{3e}\right)^{enF^{1/s}} \leq \exp\left(-\frac{nF^{1/s}}{3}\right) = e^{-\Omega(n)}. \quad \square$$

We now show the relevant definitions and lemmas adapted from Chapter 5 including the necessary modifications to the proofs to translate them to Cliff. We start by defining a potential function $g_4(X_t)$ using Definition 5.2.3.

Definition 6.4.4. We define the potential function $g_4(X_t)$ as:

$$g_4(X_t) = |x_t|_1 - \frac{2s}{s+1} \log_F \left(\max \left(\frac{enF^{1/s}}{\lambda_t}, 1 \right) \right).$$

Using Definition 6.4.4 we can see that given that $\lambda_{\max} \geq enF^{1/s}$ the drift of the potential does not change as long as λ does not reset to 1. Hence the following lemma still holds.

Lemma 6.4.5 (Adapted from Lemma 5.3.6)**.** *Consider the self-adjusting* $(1,\lambda)$ EA *resetting* λ *as in Theorem 6.4.1 and assume that the event stated in Lemma 6.4.3 does not occur. Then for every generation* t *with* $|x_t|_1 < 2n/3$*,*

$$\mathrm{E}(g_4(X_{t+1}) - g_4(X_t) \mid X_t) \geq \frac{1-s}{2e} > 0$$

for large enough n*.*

Again, as long as λ does not reset, the following lemma from Chapter 5 that describes the expected value of λ holds.

Lemma 6.4.6 (Lemma 5.3.19)**.** *Consider the self-adjusting* $(1,\lambda)$ EA *as in Theorem 6.4.1 and assume that the event stated in Lemma 6.4.3 does not occur. If the best-so-far fitness at time* t *is at most* i *then*

$$\mathrm{E}(\lambda_t \mid \lambda_0) \leq \lfloor \lambda_0/F^t \rfloor + \frac{en}{n-i} \cdot \left(F^{1/s} + \frac{F^{1/s}}{\ln F}\right).$$

For completeness we state the following lemma taken from Chapter 5.

Lemma 6.4.7 (Lemma 5.3.5)**.** *For all generations* t*,* $|x|_1$ *and the potential are related as:* $|x|_1 - \frac{2s}{s+1}\log_F(enF^{1/s}) \leq g_4(X_t) \leq |x|_1$*.*

With the previous lemmas we can now prove Lemma 6.4.2.

Proof of Lemma 6.4.2. Following the arguments of the proof of Theorem 5.3.7 to bound the number of generations to reach $|x_t|_1 \geq 2n/3$ we use the potential function $g_4(X_t)$. To fit the perspective of the additive drift theorem (Theorem 2.4.14) we switch to the potential function $\overline{g_4}(X_t) := \max(2n/3 - g_4(X_t), 0)$ and stop when $\overline{g_4}(X_t) = 0$ (which implies that $|x_t|_1$ is least $2n/3$) or $|x_t|_1$ of at least $2n/3$ is reached beforehand. Note that the maximum caps the effect of generations that jump down the cliff. Lemma 6.4.5 shows that the potential $g_4(X_t)$ has a positive constant drift whenever $|x_t|_1 < 2n/3$, and given that the drift bound for $g_4(X_t)$ still holds when only considering fitness improvements by 1 it also holds for $\overline{g_4}(X_t)$.

The initial value $\overline{g_4}(X_0)$ is at most $a + \frac{2s}{s+1}\log_F\left(enF^{1/s}\right)$ by Lemma 6.4.7. Using Lemma 6.4.5 and the additive drift theorem (Theorem 2.4.14), the expected number of generations is

$$\mathrm{E}(T_1) \leq \frac{a + \frac{2s}{s+1}\log_F\left(enF^{1/s}\right)}{\frac{1-s}{2e}} = O(a + \log n).$$

The expected number of function evaluations during this time is $\mathrm{E}(\lambda_0 + \lambda_1 + \cdots + \lambda_{T_1-1}) = \mathrm{E}\left(\sum_{t=0}^{T_1-1} \lambda_t \mid \lambda_0\right)$. We bound all summands by Lemma 6.4.6, applied with a worst case fitness of $i := 2n/3$. This yields a random variable λ^* with

$$\mathrm{E}(\lambda^*) \leq \frac{en}{n/3} \cdot \left(F^{1/s} + \frac{F^{1/s}}{\ln F}\right) = e/3 \cdot \left(F^{1/s} + \frac{F^{1/s}}{\ln F}\right)$$

and $\mathrm{E}(\lambda^*) \geq \mathrm{E}(\lambda_t \mid \lambda_0) - \lfloor \lambda_0/F^t \rfloor$ for all $t < T_1$. Thus, the expected time can be bounded by T_1 i.i.d. variables λ^* and $\sum_{t=0}^{\infty} \lfloor \lambda_0/F^t \rfloor \leq \frac{F\lambda_0}{F-1} = O(\lambda_0)$. Since T_1 is itself a random variable, we apply Wald's equation (Lemma 2.4.9) to conclude that

$$O(\lambda_0) + \mathrm{E}\left(\sum_{t=0}^{T_1-1} \lambda^*\right) = O(\lambda_0) + \mathrm{E}(T_1) \cdot \mathrm{E}(\lambda^*) = O(a + \log n + \lambda_0).$$

Finally, if the failure from Lemma 6.4.3 occurs we restart the analysis with a worst-case value of n for a. Since the failure has an exponentially small probability, this does not affect the claimed expectations. □

6.4.2 Jumping Down the Cliff

After reaching the cliff, the algorithm needs to jump down the cliff. This requires a generation in which *all* offspring lie on the second slope. We have seen in Section 6.3 that this probability is exponentially small in λ_t. The resetting mechanism implies that when λ reaches its maximum value $\lambda_{\max}$ and the following generation is unsuccessful, we reach small values of λ_t and jumps down the cliff become likely.

We also know from Section 6.3 that we need a sufficiently large jump to prevent the algorithm from jumping straight back up the cliff. This probability decreases with the distance to the cliff (that is, $|x_t|_1 - 2n/3$) and it increases with λ_t as many offspring can amplify the probability of a jump back up the cliff. The following definition captures states from which the probability of jumping up the cliff is sufficiently small.

Definition 6.4.8. Given some even value $\kappa \in \mathbb{N}$, a state (x_t, λ_t) is called *κ-safe* if $|x_t|_1 \geq 2n/3 + \kappa$ and $\lambda_t \leq 2^{-\kappa/2} \cdot (\kappa/2)!$.

Note that $2^{-\kappa/2} \cdot (\kappa/2)!$ is non-decreasing in κ.

In this subsection we give upper bounds on the expected number of generations and the expected number of function evaluations to reach a κ-safe state for the specific value $\kappa := \frac{\log\log n}{\log\log\log n}$. We also consider search points with at least $3n/4$ ones as safe, regardless of the value of λ_t; reaching such a search point will be the goal of the following phase.

Lemma 6.4.9. *Consider the self-adjusting* $(1,\lambda)$ EA *with resetting* λ *as in Theorem 6.4.1. Let* $\kappa := \frac{\log\log n}{\log\log\log n}$. *For every initial* $\lambda_0 \leq \lambda_{\max}$ *and every initial search point* x_0 *with* $|x_0|_1 \geq 2n/3$ *the algorithm reaches a* κ*-safe state, or a search point with at least* $3n/4$ *ones, in* $O(\log(\lambda_{\max})\log n)$ *expected generations and* $O(\lambda_{\max}\log n)$ *expected evaluations.*

The main idea of the proof of Lemma 6.4.9 is that, once the algorithm reaches a local optimum with $2n/3$ ones, λ will increase until it resets to 1 and this is repeated until the algorithm leaves its local optimum. Every time $\lambda = 1$ the algorithm will accept any offspring, then we can wait until a lucky mutation step can directly jump to a search point with at least $2n/3 + \frac{\log\log n}{\log\log\log n}$ ones. Finally, we account for the time that the algorithm takes to reset λ to 1, including the time spent outside of local optima in states that are not safe.

One such set of non-safe states is that of states with at least $2n/3+\kappa$ ones but violating the upper bound for λ_t from Definition 6.4.8. For these states we cannot exclude that the algorithm will return to the first slope before it reaches a safe state, but we can bound the time the algorithm spends before either event happens and later on account for this time. This is done in the following lemma.

Lemma 6.4.10. *Consider the self-adjusting* $(1,\lambda)$ EA *with resetting* λ *as in Theorem 6.4.1. Let* $\kappa := \frac{\log\log n}{\log\log\log n}$. *From any state* (x_0, λ_0) *with* $|x_0|_1 \geq 2n/3 + \kappa$ *in expectation the algorithm needs at most* $O(\log\lambda_{\max})$ *generations and* $O(\lambda_{\max})$ *evaluations to reach a* κ*-safe state, to return to the first slope or to find a search point* x_t *with* $|x_t|_1 \geq 3n/4$.

The main proof idea is to show that with a large value of λ, with high probability all generations are successful and λ is quickly reduced to a safe value, unless any of the other two events happens.

Proof of Lemma 6.4.10. In every successful generation the algorithm will decrease λ and either increase the number of ones or go back to the first slope. In the latter case, we are done. In the first case, the algorithm moves towards a κ-safe state as λ_t is decreased.

If the algorithm only has successful generations it will reach a κ-safe state once λ has decreased below the threshold of $2^{-\kappa/2}\cdot(\kappa/2)!$. Assuming that every generation is successful, after i generations the algorithm will reduce the offspring population size to $\lambda = \frac{\lambda_0}{F^i}$. For $i = \alpha := \left\lceil \log_F\left(\frac{\lambda_0}{2^{-\kappa/2}\cdot(\kappa/2)!}\right)\right\rceil$, we get an offspring population size of

$$\frac{\lambda_0}{F^{\left\lceil \log_F\left(\frac{\lambda_0}{2^{-\kappa/2}\cdot(d_t/2)!}\right)\right\rceil}} \leq \frac{\lambda_0}{F^{\log_F\left(\frac{\lambda_0}{2^{-\kappa/2}\cdot(\kappa/2)!}\right)}} = 2^{-\kappa/2}\cdot(\kappa/2)!.$$

Hence the algorithm needs at most α consecutive successful generations to obtain $\lambda \leq 2^{-\kappa/2}\cdot(\kappa/2)!$. During these generations, the number of evaluations is at most

$$\begin{aligned}\sum_{i=0}^{\alpha-1}\left\lceil\frac{\lambda_0}{F^i}\right\rceil &\leq \sum_{i=0}^{\alpha-1}\left(\frac{\lambda_0}{F^i}+1\right) \leq \alpha + \lambda_0\sum_{i=0}^{\infty}\left(\frac{1}{F}\right)^i \\ &= \alpha + \lambda_0\left(\frac{1}{1-1/F}\right) = \alpha + \lambda_0\left(\frac{F}{F-1}\right) \\ &= O(\lambda_0) = O(\lambda_{\max}).\end{aligned}$$

We now show that the algorithm finds an improvement in every generation with probability $1-o(1)$. The algorithm will increase λ in all generations that are unsuccessful. Since we only consider $|x_t|_1 \leq 3n/4$ the probability of an offspring being better than the parent is at least $1/(4\mathrm{e})$ and the probability of an unsuccessful generation is at most $\left(1-\frac{1}{4\mathrm{e}}\right)^\lambda$. By the union bound the probability that there is at least one unsuccessful generation during α generations is at most

$$\sum_{j=0}^{\alpha-1}\left(1-\frac{1}{4e}\right)^{\frac{\lambda_0}{F^j}} \leq \sum_{j=0}^{\alpha-1} e^{-\frac{\lambda_0}{4eF^j}}.$$

We now show by induction that for every $\alpha \geq 1$ the inequality $\sum_{j=0}^{\alpha-1} e^{-\frac{\lambda_0}{4eF^j}} \leq 2e^{-\frac{\lambda_0}{4eF^{\alpha-1}}}$ holds. Starting with the base case $\alpha = 1$ the statement is clearly true:

$$\sum_{j=0}^{0} e^{-\frac{\lambda_0}{4eF^j}} = e^{-\frac{\lambda_0}{4e}} \leq 2e^{-\frac{\lambda_0}{4e}}.$$

We now show that if the statement holds for $\alpha-1$ then it also holds for α. First we note that, since $2^{-\kappa/2}\cdot(\kappa/2)! = \omega(1)$, the following holds for large enough n:

$$\begin{aligned}2 &\leq \exp\left(\frac{F-1}{4e}\cdot\frac{2^{-\kappa/2}\cdot(\kappa/2)!}{F}\right) \\ &= \exp\left(\frac{F-1}{4e}\cdot\frac{\lambda_0}{F^{\log_F\left(\frac{\lambda_0}{2^{-\kappa/2}\cdot(\kappa/2)!}\right)+1}}\right) \\ &\leq \exp\left(\frac{F-1}{4e}\cdot\frac{\lambda_0}{F^{\left\lceil\log_F\left(\frac{\lambda_0}{2^{-\kappa/2}\cdot(\kappa/2)!}\right)\right\rceil}}\right) = \exp\left(\frac{F-1}{4e}\cdot\frac{\lambda_0}{F^\alpha}\right)\end{aligned}$$

Hence, using the inductive hypothesis the following statement holds:

$$\begin{aligned}\sum_{j=0}^{\alpha-1} e^{-\frac{\lambda_0}{4eF^j}} &\leq 2e^{-\frac{\lambda_0}{4eF^{\alpha-1}}} \leq e^{\frac{(F-1)\lambda_0}{4eF^\alpha}}\cdot e^{-\frac{\lambda_0}{4eF^{\alpha-1}}} \\ &= e^{\left(\frac{(F-1)\lambda_0}{4eF^\alpha}-\frac{\lambda_0}{4eF^{\alpha-1}}\right)} = e^{\left(\frac{(F-1)\lambda_0-F\lambda_0}{4eF^\alpha}\right)} = e^{-\frac{\lambda_0}{4eF^\alpha}}.\end{aligned}$$

Now adding $e^{-\frac{\lambda_0}{4eF^\alpha}}$ to the first and last term we obtain

$$\sum_{j=0}^{\alpha} e^{-\frac{\lambda_0}{4eF^j}} \leq 2e^{-\frac{\lambda_0}{4eF^\alpha}},$$

proving the hypothesis.

Therefore, the probability that there are α consecutive successful generations yielding $\lambda \leq 2^{-\kappa/2} \cdot (\kappa/2)!$ is at least

$$\begin{aligned} 1 - 2e^{-\frac{\lambda_0}{4eF^{\alpha-1}}} &\geq 1 - 2e^{-\frac{\lambda_0}{4e\left(\frac{\lambda_0}{2^{-\kappa/2}\cdot(\kappa/2)!}\right)}} \\ &= 1 - 2e^{-\frac{2^{-\kappa/2}\cdot(\kappa/2)!}{4e}} = 1 - o(1). \end{aligned}$$

With the same probability the algorithm will reduce λ to a value $\lambda_t \leq 2^{-\kappa/2} \cdot (\kappa/2)!$ in

$$\left\lceil \log_F \left(\frac{\lambda_0}{2^{-\kappa/2} \cdot (\kappa/2)!} \right) \right\rceil = O(\log(\lambda_0)) = O(\log \lambda_{\max})$$

generations and $O(\lambda_{\max})$ evaluations. If there is an unsuccessful generation we can restart the arguments. Since there are no unsuccessful generations with probability $1-o(1)$, we need to restart the arguments at most $1 + o(1)$ times and obtain the same asymptotic expected number of generations and evaluations. □

With Lemma 6.4.10 we are able to prove Lemma 6.4.9.

Proof of Lemma 6.4.9. We define a cycle as a sequence of generations that begins after λ is reset to 1 and $|x_t|_1 = 2n/3$. Since $|x_0|_1 \geq 2n/3$ and there is no assumption on λ_0 we need to take into a account the time taken to start the first cycle; this will be accounted later on.

By Lemma 6.2.4 with $c := 1$ in the first generation of every cycle (with $\lambda = 1$) there is a probability of at least $1/\log n$ to create and accept an offspring with at least $2n/3 + \frac{\log\log n}{\log\log\log n}$ ones. The next generation will have $\lambda = F^{1/s}$, which for a sufficiently large n satisfies $\lambda \leq 2^{-\kappa/2} \cdot (\kappa/2)!$. Hence, in expected $\log n$ cycles the sought event will happen. Now it remains to bound the expected number of generations and evaluations in each cycle.

If during a cycle all generations maintain the current fitness value, after j generations the offspring population size is $F^{j/s}$. For $j := \lceil s \log_F \lambda_{\max} \rceil$, we get an offspring population size of

$$F^{\lceil s \log_F \lambda_{\max} \rceil / s} \geq F^{\log_F \lambda_{\max}} = \lambda_{\max}.$$

Therefore, the number of generations needed to reset λ is at most $\lceil s \log_F \lambda_{\max} \rceil + 1$. Using $\lceil F^{j/s} \rceil \leq 2F^{j/s}$, during these generations, the number of evaluations is at most

$$\begin{aligned} \sum_{j=0}^{\lceil s \log_F \lambda_{\max} \rceil} \left\lceil F^{j/s} \right\rceil &\leq 2 \sum_{j=0}^{\lceil s \log_F \lambda_{\max} \rceil} \left(F^{1/s} \right)^j \\ &= 2 \cdot \frac{(F^{1/s})^{\lceil s \log_F(\lambda_{\max}) \rceil + 1} - 1}{F^{1/s} - 1} \\ &\leq 2 \cdot \frac{(F^{1/s})^{s \log_F(\lambda_{\max}) + 2}}{F^{1/s} - 1} \\ &= \frac{2F^{2/s}}{F^{1/s} - 1} \cdot \lambda_{\max} = O(\lambda_{\max}). \end{aligned}$$

We now show that when the algorithm is in a local optimum, with constant probability all following generations will maintain the current fitness value until λ is reset to 1. When $|x_t|_1 = 2n/3$, in order for a generation to maintain the number of one-bits, it is sufficient to create at least one copy of the parent. Hence, the probability of the event is at least

$$\begin{aligned}1 - \left(1 - \left(1 - \frac{1}{n}\right)^n\right)^\lambda &\geq \frac{\lambda\left(1-\frac{1}{n}\right)^n}{1+\lambda\left(1-\frac{1}{n}\right)^n} = \frac{1}{1+\frac{1}{\lambda\left(1-\frac{1}{n}\right)^n}} \\ &\geq \frac{1}{\exp\left(\frac{1}{\lambda\left(1-\frac{1}{n}\right)^n}\right)} = \exp\left(-\frac{1}{\lambda\left(1-\frac{1}{n}\right)^n}\right).\end{aligned}$$

The probability that a cycle is comprised only of generations that maintain the fitness value is at least

$$\begin{aligned}&\prod_{j=0}^{\lceil s \log_F \lambda_{\max}\rceil} \exp\left(-\frac{1}{F^j\left(1-\frac{1}{n}\right)^n}\right) \geq \prod_{j=0}^{\infty} \exp\left(-\frac{1}{F^j\left(1-\frac{1}{n}\right)^n}\right) \\ &= \exp\left(-\frac{1}{\left(1-\frac{1}{n}\right)^n}\sum_{j=0}^{\infty} F^{-j}\right) = \exp\left(-\frac{1}{\left(1-\frac{1}{n}\right)^n}\cdot\frac{F}{F-1}\right) = \Omega(1). \qquad (6.13)\end{aligned}$$

Therefore, in each cycle the algorithm will directly increase λ to $\lambda_{\max}$ and then reset to 1 with constant probability using $O(\log \lambda_{\max})$ generations and $O(\lambda_{\max})$ evaluations.

Since the self-adjusting $(1, \lambda)$ EA is non-elitist, there is still a possibility for the algorithm to either jump down the cliff (but not to the desired κ-safe state) or to reduce the number of ones. We consider the total time T_{c} until a cycle finishes, that is, λ resets to 1 with the current search point has $2n/3$ ones. We use as superscript gen and eval to denote that we are considering number of generations or evaluations respectively. Similar to the proof of Theorem 6.3.1, let p^- denote the probability of accepting a search point with less than $2n/3$ ones from the top of the cliff and let p^+ denote the probability of accepting a search point with more than $2n/3$ ones from the top of the cliff. According to (6.13) $p^+ + p^- = 1 - \Omega(1)$.

Now, let T_1^{gen} denote the number of generations and T_1^{eval} denote the number of evaluations to return to the top of the cliff after accepting a search point with less than $2n/3$ ones. If the number of one-bits is reduced, by Lemma 5.3.9 with probability $1 - O(1/n)$ the number of one-bits will never drop below $2n/3 - O(\log n)$ ones before reaching a point with $2n/3$ ones. By Lemma 6.4.2, it will take $O(\log n)$ generations and $O(\log n + \lambda_0) = O(\lambda_{\max})$ evaluations in expectation to return to a local optimum. With probability $O(1/n)$ the number of one-bits will drop below $2n/3 - O(\log n)$ and again by Lemma 6.4.2 in expectation it will take at most $O(n)$ generations and $O(n + \lambda_0) = O(\lambda_{\max})$ to return to a local optimum. Hence,

$$\begin{aligned}\mathrm{E}\big(T_1^{\mathrm{gen}}\big) &= (1 - O(1/n))O(\log n) + O(1/n)O(n) = O(\log n), \\ \mathrm{E}\big(T_1^{\mathrm{eval}}\big) &= (1 - O(1/n))O(\lambda_{\max}) + O(1/n)O(\lambda_{\max}) = O(\lambda_{\max}).\end{aligned}$$

Let T_2^{gen} and T_2^{eval} denote the number of generations and evaluations to return to the first slope or finding κ-safe state from a search point in the second slope that is not in a κ-safe state. Using the same arguments as in Lemma 6.4.2 we can see that the algorithm will use $O(\log n)$ generations and $O(\lambda_{\max})$ evaluations to either find a solution with at least $2n/3 + \frac{\log\log n}{\log\log\log n}$ ones or jump back to the first slope. If the algorithm finds a solution with at least $2n/3 + \frac{\log\log n}{\log\log\log n}$ ones but with $\lambda > 2^{-\kappa/2} \cdot (\kappa/2)!$ then by Lemma 6.4.10 it will take $O(\log \lambda_{\max})$ generations and $O(\lambda_{\max})$ evaluations in expectation to either reduce λ to

$\lambda \leq 2^{-\kappa/2} \cdot (\kappa/2)!$ or to return to the first slope. Hence, we obtain

$$\begin{aligned}\mathrm{E}\big(T_2^{\mathrm{gen}}\big) &= O(\log n) + O(\log \lambda_{\max}) = O(\log \lambda_{\max}),\\ \mathrm{E}\big(T_2^{\mathrm{eval}}\big) &= O(\lambda_{\max}) + O(\lambda_{\max}) = O(\lambda_{\max}).\end{aligned}$$

Finally, when jumping back to the first slope, by Lemma 6.2.3, in expectation the number of ones is reduced by $\Delta^{\uparrow}_{i,\lambda} \leq d+1$, that is, in expectation the algorithm jumps to a point that has $a \leq 1$ less ones than the local optimum. Let T_3^{eval} be the time to go back to $|x|_1 = 2n/3$ from $|x|_1 = 2n/3 - a$. By the law of total expectation and Lemma 6.4.2, $\mathrm{E}\big(T_3^{\mathrm{eval}}\big) = \mathrm{E}(\mathrm{E}(T \mid a)) \leq \mathrm{E}(O(a + \log n + \lambda_0)) = O(\mathrm{E}(a)) + O(\log n + \lambda_0)$. Given that $\mathrm{E}(a) \leq 1$ we obtain $\mathrm{E}\big(T_3^{\mathrm{eval}}\big) = O(\log n + \lambda_0) = O(\lambda_{\max})$ evaluations. For the number of generations T_3^{gen} we use the same arguments and obtain $O(\log n)$ generations.

Therefore, if the algorithm moves out of the local optimum it will return to it in

$$\begin{aligned}\mathrm{E}\big(T_{\mathrm{c}}^{\mathrm{gen}}\big) &= O(\log \lambda_{\max})\Omega(1) + p^{-}\mathrm{E}\big(T_1^{\mathrm{gen}}\big) + p^{+}\big(\mathrm{E}\big(T_2^{\mathrm{gen}}\big) + \mathrm{E}\big(T_3^{\mathrm{gen}}\big)\big) = O(\log \lambda_{\max}),\\ \mathrm{E}\big(T_{\mathrm{c}}^{\mathrm{eval}}\big) &= O(\lambda_{\max})\Omega(1) + p^{-}\mathrm{E}\big(T_1^{\mathrm{eval}}\big) + p^{+}\big(\mathrm{E}\big(T_2^{\mathrm{eval}}\big) + \mathrm{E}\big(T_3^{\mathrm{eval}}\big)\big) = O(\lambda_{\max}).\end{aligned}$$

It remains to account for the time before the first cycle. Using the same arguments as before for any $\lambda_0 \leq \lambda_{\max}$ and $|x_0|_1 \geq 2n/3$ the algorithm will spend $O(\log \lambda_{\max})$ generations and $O(\lambda_{\max})$ evaluations to start the first cycle or reach the desired state. Noting that the expected number of cycles is $\log n$ proves the claim. □

6.4.3 After Jumping Down the Cliff

Now we show that, with probability $\Omega(1)$, we reach a search point with at least $3n/4$ ones when starting from a κ-safe state with $\kappa := \frac{\log\log n}{\log\log\log n}$. As in Section 6.3, the target of reaching $3n/4$ ones is chosen such that the probability of an improving mutation is always $\Omega(1)$.

Proving this claim is not straightforward for several reasons. It is always possible to have a mutation jumping back up the cliff. This probability decreases with the distance to the cliff (that is, $|x_t|_1 - 2n/3$) and it increases with λ_t as many offspring can amplify the probability of a jump back up the cliff (cf. Lemma 6.2.3). Fortunately, the notion of κ-safe states implies that we start with a distance of at least κ to the cliff and a small λ_t, so that initially this probability amplification does not pose a huge risk.

But small values of λ are risky for another reason. Since the number of ones is larger than $2n/3$ and hence significantly larger than $n/2$, the expected number of ones in any offspring is smaller than that of its parent. With $\lambda_t \approx 1$ there is a constant negative drift towards decreasing the number of zeros and "slipping down" the second slope. Fortunately, λ_t will increase during unsuccessful generations and we will see that this effect prevents the algorithm from slipping down the second slope.

In Chapter 5 we showed that, on OneMax, for the potential function from Definition 6.4.4 there is a positive drift in the potential throughout the run: on OneMax, Lemma 6.4.5 holds for all non-optimal search points. We further constructed a so-called "ratchet argument", arguing that significant decreases in the potential are unlikely. In our approach, the potential and the fitness only differ by a term of $\Theta(\log n)$, as stated in Lemma 6.4.7. Hence we concluded that, with high probability, the best fitness never decreases by a term of $r \log n$, for some constant $r > 0$.

Unfortunately, this ratchet argument is not directly applicable here, since we can only guarantee a distance of $\kappa := \frac{\log\log n}{\log\log\log n} \ll r \log n$ to the cliff. Hence the ratchet argument from Chapter 5 has far too much slack.

The proof of the ratchet argument in Chapter 5 applies the negative drift theorem (Theorem 2.4.16) to an interval on the potential scale of size $\Theta(\log n)$, in order to obtain failure probabilities that are polynomially small (that is, exponentially small in the interval length).

We refine the ratchet argument here by defining a revised potential function tailored to a fitness range up to $3n/4$ ones, where the fitness and the potential only differ by an additive term of $\Theta(1)$. Then we apply the negative drift theorem (Theorem 2.4.16) to an interval of size $\kappa/2 - O(1) = \frac{\log\log n}{2\log\log\log n} - O(1)$ on the potential scale to show that the number of ones does not drop below $2n/3 + \kappa/2$ in a time that is exponential in the interval length. More specifically, the time period will be determined as γ^κ, for some constant $\gamma > 1$. During this time, the potential has a positive drift and with good probability the algorithm moves sufficiently far away from the cliff, that is, to a distance of $\Theta(\gamma^\kappa)$.

Since $\gamma^{\frac{\log\log n}{\log\log\log n}} = o(\log n)$, we can only guarantee a sub-logarithmic increase in the distance and the failure probability from the negative drift theorem is $\omega(1/\log n)$. Thus, we iterate this argument three times, with exponentially increasing values for κ, until we reach a search point with at least $3n/4$ ones (or we return to the first slope).

Throughout these arguments, we also show that λ_t is bounded from above as $\lambda_t \le 2^{-\kappa/2} \cdot (\kappa/2)!$ as in the definition of κ-safe states. This definition requires a distance of at least κ from the top of the cliff, however we can only guarantee a distance of at least $\kappa/2$. We call such states *weakly κ-safe.*

Definition 6.4.11. A state (x_t, λ_t) is called *weakly κ-safe* if $|x_t|_1 \ge 2n/3 + \kappa/2$ and $\lambda_t \le 2^{-\kappa/2} \cdot (\kappa/2)!$.

We start by revising the potential function as follows.

Definition 6.4.12. Let $\varepsilon := \left(\frac{e-1}{e}\right)\left(\frac{1-s}{s+1}\right)$. We define the potential function $g_5(X_t)$ as

$$g_5(X_t) = |x_t|_1 - \frac{2s}{s+1}\log_F\left(\max\left(\frac{8e + \log_{\frac{e}{e-1}}(2/\varepsilon)F^{1/s}}{\lambda_t}, 1\right)\right).$$

Lemma 6.4.13. *Consider the self-adjusting* $(1,\lambda)$ EA *as in Theorem 6.4.1. For all states* (x_t, λ_t) *with* $d := |x_t|_1 - 2n/3 = \omega(1)$, $|x_t|_1 \le 3/4$ *and* $\lambda_t \le 2^{-d/2} \cdot (d/2)!$,

$$\mathrm{E}(g_5(X_{t+1}) - g_5(X_t) \mid X_t) \ge \frac{1-s}{4e} > 0$$

for large enough n. This also holds when only considering improvements that increase the fitness by 1.

Proof. The proof follows the proof of Lemma 5.3.6, using additional arguments to consider jumps up the cliff and the possibility that λ is reset when $\lambda_t = \lambda_{\max}$. In this proof, we use $p^0_{i,\lambda} := 1 - p^{\rightarrow}_{i,\lambda} - p^{\leftarrow}_{i,\lambda} - p^{\uparrow}_{i,\lambda}$ to denote the probability of not changing the current number of ones. We first assume that $\lambda_{\max} > 2^{-d/2} \cdot (d/2)!$, which implies that resets are impossible under the assumption $\lambda_t \le 2^{-d/2} \cdot (d/2)!$.

We first consider the case $\lambda_t \le 8e + \log_{\frac{e}{e-1}}(2/\varepsilon)$ as then $\lambda_{t+1} \le 8e + \log_{\frac{e}{e-1}}(2/\varepsilon)F^{1/s}$ and $g_5(X_{t+1}) = |x_{t+1}|_1 - \frac{2s}{s+1}(\log_F(8e + \log_{\frac{e}{e-1}}(2/\varepsilon)F^{1/s}) - \log_F(\lambda_{t+1}))$. When the number of ones increases, in expectation they do so by $\Delta^{\rightarrow}_{i,\lambda}$ and since $\lambda_{t+1} = \lambda_t/F$, the penalty term $\frac{2s}{s+1}(\log_F(8e + \log_{\frac{e}{e-1}}(2/\varepsilon)F^{1/s}) - \log_F(\lambda_t))$ increases by $\frac{2s}{s+1}$ (unless $\lambda_{t+1} = 1$ is reached, in which case the increase might be lower). When the number of ones does not change, the penalty decreases by $\frac{2}{s+1}$. When the number of ones decreases, conditional on $|x_{t+1}|_1 > 2n/3$, the expected decrease is at most $\Delta^{\leftarrow}_{i,\lambda}$ and the penalty decreases by $\frac{2}{s+1}$. Finally, then when the algorithm creates an offspring up the cliff the expected decrease in the number of ones is $\Delta^{\uparrow}_{i,\lambda}$ and the penalty increases by $\frac{2s}{s+1}$. Together,

$$\begin{aligned}
&\mathrm{E}\Big(g_5(X_{t+1}) - g_5(X_t) \mid X_t, \lambda_t \le 8e + \log_{\frac{e}{e-1}}(2/\varepsilon)\Big)\\
&\ge p^{\rightarrow}_{i,\lambda}\left(\Delta^{\rightarrow}_{i,\lambda} - \frac{2s}{s+1}\right) + p^{0}_{i,\lambda}\cdot\frac{2}{s+1} + p^{\leftarrow}_{i,\lambda}\left(-\Delta^{\leftarrow}_{i,\lambda} + \frac{2}{s+1}\right) + p^{\uparrow}_{i,\lambda}\left(-\Delta^{\uparrow}_{i,\lambda} - \frac{2s}{s+1}\right)\\
&= p^{\rightarrow}_{i,\lambda}\left(\Delta^{\rightarrow}_{i,\lambda} - \frac{2s}{s+1}\right) + \left(p^{0}_{i,\lambda} + p^{\leftarrow}_{i,\lambda}\right)\frac{2}{s+1} - \Delta^{\leftarrow}_{i,\lambda}p^{\leftarrow}_{i,\lambda} - p^{\uparrow}_{i,\lambda}\left(\Delta^{\uparrow}_{i,\lambda} + \frac{2s}{s+1}\right)\\
&= p^{\rightarrow}_{i,\lambda}\left(\Delta^{\rightarrow}_{i,\lambda} - \frac{2s}{s+1}\right) + \left(1 - p^{\rightarrow}_{i,\lambda} - p^{\uparrow}_{i,\lambda}\right)\frac{2}{s+1} - \Delta^{\leftarrow}_{i,\lambda}p^{\leftarrow}_{i,\lambda} - p^{\uparrow}_{i,\lambda}\left(\Delta^{\uparrow}_{i,\lambda} + \frac{2s}{s+1}\right)\\
&= p^{\rightarrow}_{i,\lambda}\left(\Delta^{\rightarrow}_{i,\lambda} - 2\right) + \frac{2}{s+1} - \Delta^{\leftarrow}_{i,\lambda}p^{\leftarrow}_{i,\lambda} - p^{\uparrow}_{i,\lambda}\left(\Delta^{\uparrow}_{i,\lambda} + 2\right).
\end{aligned}$$

Using $\Delta^{\rightarrow}_{i,\lambda} \ge 1$ (which also holds when only considering fitness increases by 1), $\Delta^{\leftarrow}_{i,\lambda} \le \frac{e}{e-1}$ and $\Delta^{\uparrow}_{i,\lambda} \le d+1$ by Lemma 6.2.3, this is at least

$$1 + \frac{1-s}{s+1} - p^{\rightarrow}_{i,\lambda} + p^{\leftarrow}_{i,\lambda}\Delta^{\leftarrow}_{i,\lambda} - p^{\uparrow}_{i,\lambda}(d+3).$$

Following the calculations from Lemma 5.3.6 $1 + \frac{1-s}{s+1} - p^{\rightarrow}_{i,\lambda} + p^{\leftarrow}_{i,\lambda}\Delta^{\leftarrow}_{i,\lambda}$ is at least $\frac{1-s}{2e}$. Lemma 6.2.3 shows that $p^{\uparrow}_{i,\lambda} \le \frac{\lambda(3/4)^d}{d!}$, if $\lambda = O(1)$ and $d = \omega(1)$ then $p^{\uparrow}_{i,\lambda}(d+3) = o(1)$, hence for a sufficiently large n,

$$\begin{aligned}
&\mathrm{E}\Big(g_5(X_{t+1}) - g_5(X_t) \mid X_t, \lambda_t \le 8e + \log_{\frac{e}{e-1}}(2/\varepsilon)\Big)\\
&\ge \frac{1-s}{2e} - o(1) \ge \frac{1-s}{4e}.
\end{aligned}$$

For the case $8e + \log_{\frac{e}{e-1}}(2/\varepsilon) < \lambda_t \le 2^{-d/2}\cdot(d/2)!$, in an unsuccessful generation the penalty term is capped at its maximum and we pessimistically bound the positive effect on the potential from below by 0. However, the probability of increasing the number of ones is large enough to show a positive drift.

By assumption $\lambda_t \le 2^{-d/2}\cdot(d/2)!$. Along with (6.5) from Lemma 6.2.3,

$$p^{\uparrow}_{i,\lambda} \le \frac{\lambda(3/4)^d}{d!} \le \left(\frac{3}{8}\right)^d.$$

We also have $\lambda_t > 8e + \log_{\frac{e}{e-1}}(2/\varepsilon) > 8e$ since $1/\varepsilon \ge \frac{e}{e-1}$. Then, by Lemma 6.2.3, $8e + \log_{\frac{e}{e-1}}(2/\varepsilon) < \lambda_t \le 2^{-d/2}\cdot(d/2)!$ implies the following two statements.

$$\begin{aligned}
p^{\rightarrow}_{i,\lambda} &\ge 1 - \left(1 - \frac{1}{4e}\right)^{8e} - \left(\frac{3}{8}\right)^d \ge 1 - \frac{1}{2e} - \left(\frac{3}{8}\right)^d\\
p^{\leftarrow}_{i,\lambda}\Delta^{\leftarrow}_{i,\lambda} &\le \left(\frac{e-1}{e}\right)^{8e+\log_{\frac{e}{e-1}}(2/\varepsilon)}\cdot\frac{e}{e-1}\\
&= \left(\frac{e-1}{e}\right)^{8e-1+\log_{\frac{e}{e-1}}(2/\varepsilon)} \ge \left(\frac{e-1}{e}\right)^{\log_{\frac{e}{e-1}}(2/\varepsilon)} = \frac{\varepsilon}{2}.
\end{aligned}$$

Together,

$$\begin{aligned}
&\mathrm{E}\Big(g_5(X_{t+1}) - g_5(X_t) \mid X_t, 8e + \log_{\frac{e}{e-1}}(2/\varepsilon) < \lambda_t < 2^{-d/2}\cdot(d/2)!\Big)\\
&\ge\ p^{\rightarrow}_{i,\lambda}\left(1 - \frac{2s}{s+1}\right) + p^{\leftarrow}_{i,\lambda}\left(-\Delta^{\leftarrow}_{i,\lambda}\right) - p^{\uparrow}_{i,\lambda}\left(d + 1 + \frac{2s}{s+1}\right)\\
&\ge\ \left(1 - \frac{1}{2e} - \left(\frac{3}{8}\right)^d\right)\left(1 - \frac{2s}{s+1}\right) - \frac{\varepsilon}{2} - \left(\frac{3}{8}\right)^d(d+2)
\end{aligned}$$

given that $d = \omega(1)$,

$$\geq \left(1 - \frac{1}{e}\right)\left(1 - \frac{2s}{s+1}\right) - \frac{\varepsilon}{2} - o(1) = \left(\frac{e-1}{e}\right)\left(\frac{1-s}{s+1}\right) - \frac{\varepsilon}{2} - o(1)$$

using the definition $\varepsilon := \left(\frac{e-1}{e}\right)\left(\frac{1-s}{s+1}\right)$,

$$= \left(\frac{e-1}{2e}\right)\left(\frac{1-s}{s+1}\right) - o(1).$$

Since $\left(\frac{e-1}{2e}\right) = \frac{1}{2e} + \left(\frac{e-2}{2e}\right)$ and $\left(\frac{e-2}{2e}\right)\left(1 - \frac{2s}{s+1}\right)$ is a positive constant, for large enough n this is larger than $o(1)$ and

$$\mathrm{E}\Big(g_5(X_{t+1}) - g_5(X_t) \mid X_t, \lambda_t \leq 8e + \log_{\frac{e}{e-1}}(2/\varepsilon)\Big) \geq \frac{1}{2e}\left(\frac{1-s}{s+1}\right) \geq \frac{1-s}{4e}.$$

Since $s < 1$, this is a strictly positive constant.

Now, if $\lambda_{\max} \leq 2^{-d/2} \cdot (d/2)!$ then resets may happen if $\lambda_t = \lambda_{\max}$. A reset decreases the potential by at most $n + O(1)$ as this is the range of the potential scale. The probability of a reset is at most $e^{-\Omega(n)}$ by Lemma 6.4.3. Hence, this only affects the drift by an additive term $-O(n) \cdot e^{-\Omega(n)} = -o(1)$, which can easily be absorbed in the $-o(1)$ terms from the above calculations. □

The following lemma shows that λ typically does not grow beyond the threshold $2^{-\kappa/2} \cdot (\kappa/2)!$ from the definition of weakly κ-safe states.

Lemma 6.4.14. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 6.4.1. Then for all* $\kappa \geq 324F^{1/s}$ *the following holds. If the current state* (x_t, λ_t) *has* $\lambda_t \leq 2^{-\kappa/2} \cdot (\kappa/2)!$ *and* $2n/3 < |x_t|_1 < 3n/4$*, then with probability at least* $1 - 2^{-2\kappa}$ *we have* $\lambda_{t+1} \leq 2^{-\kappa/2} \cdot (\kappa/2)!$.

***Proof*.** Since $\lambda_t \leq 2^{-\kappa/2} \cdot (\kappa/2)!$, a necessary condition for $\lambda_{t+1} > 2^{-\kappa/2} \cdot (\kappa/2)!$ is that generation t is unsuccessful. Since $|x_t|_1 < 3n/4$, the probability of finding an improvement in any mutation is at least $1/(4e)$ and the probability of an unsuccessful generation is at most

$$\begin{aligned}\left(1 - \frac{1}{4e}\right)^{F^{-1/s}2^{-\kappa/2}(\kappa/2)!} &\leq \left(\frac{4e}{4e-1}\right)^{-F^{-1/s}(2e)^{-\kappa/2}(\kappa/2)^{\kappa/2}} \\ &= 2^{-F^{-1/s}(\kappa/(4e))^{\kappa/2}\log\left(\frac{4e}{4e-1}\right)}. \end{aligned} \tag{6.14}$$

The condition $\kappa \geq 324F^{1/s}$ implies

$$\frac{\kappa}{4e} \geq \left(1 + \frac{4}{\log\left(\frac{4e}{4e-1}\right)}\right)F^{1/s} \geq \left(1 + \frac{4F^{1/s}}{\log\left(\frac{4e}{4e-1}\right)}\right).$$

We bound the absolute value of the exponent in (6.14) using $(1+y)^x \geq xy$ for $x \in \mathbb{N}_0$, $y \geq 0$, as follows.

$$\begin{aligned} &F^{-1/s}(\kappa/(4e))^{\kappa/2}\log\left(\frac{4e}{4e-1}\right) \\ &\geq F^{-1/s}\left(1 + \frac{4F^{1/s}}{\log\left(\frac{4e}{4e-1}\right)}\right)^{\kappa/2}\log\left(\frac{4e}{4e-1}\right) \\ &\geq F^{-1/s}\frac{4F^{1/s}}{\log\left(\frac{4e}{4e-1}\right)} \cdot \kappa/2 \cdot \log\left(\frac{4e}{4e-1}\right) = 2\kappa. \end{aligned}$$

Hence the probability of an unsuccessful generation is at most $2^{-2\kappa}$. □

The following lemma now generalises and refines the "ratchet argument" from Chapter 5.

Lemma 6.4.15. *Consider the self-adjusting* $(1,\lambda)$ EA *as in Theorem 6.4.1. Let* $T_{3n/4} = \inf\{t \mid |x_t|_1 \geq 3n/4\}$ *be the number of generations until a search point with at least* $3n/4$ *ones is reached.*

There are constants $\gamma := \gamma(s,F) \in (1,2]$ *and* $\kappa_0 := \kappa_0(s,F,\gamma) \geq 2$ *such that for all* $\kappa \geq \kappa_0$ *the following holds. If the initial state* (x_0,λ_0) *is* κ*-safe with* $|x_0|_1 < 3n/4$ *then with probability at least* $1-\gamma^{-\Omega(\kappa)}$ *all states during the next* $\min\{\gamma^\kappa, T_{3n/4}\}$ *generations are weakly* κ*-safe.*

Proof. Let (x_0,λ_0) denote the initial state of the self-adjusting $(1,\lambda)$ EA. If $|x_0|_1 \geq 3n/4$ the statement is trivial, hence we assume $|x_0|_1 < 3n/4$. As in the proof of Lemma 5.3.9, we are setting up to apply the negative drift theorem (Theorem 2.4.16).

The value of γ will be determined later on, ensuring $1 < \gamma \leq 2$. Choosing $\kappa_0 \geq 324F^{1/s}$ and recalling that the initial state is κ-safe, Lemma 6.4.14 states that, with probability at least $1-2^{-2\kappa}$, $\lambda_1 \leq 2^{-\kappa/2}\cdot(\kappa/2)!$ as long as the number of ones is smaller than $3n/4$. By induction and a union bound, this holds for the first $\min\{\gamma^\kappa, T_{3n/4}\}$ generations with probability at least $1-2^{-2\kappa}\cdot\min\{\gamma^\kappa, T_{3n/4}\} \geq 1-2^{-2\kappa}\cdot\gamma^\kappa \geq 1-\gamma^{-k}$ as $\gamma \leq 2$, unless the algorithm jumps back to the first slope. We assume in the following that the bound on λ_t always applies, while the algorithm remains on the second slope (and has not found a search point with at least $3n/4$ ones yet).

Let $\alpha := \frac{se}{e-1}\log_F\left(8e + \log_{\frac{e}{e-1}}(2/\varepsilon)F^{1/s}\right) = O(1)$ abbreviate the maximum difference between the potential g_5 and the number of ones, then we start with a potential of at least $2n/3+\kappa-\alpha$. We apply the negative drift theorem (Theorem 2.4.16) to an interval $[a,b] := [2n/3+\kappa/2, 2n/3+\kappa-\alpha]$ with respect to the current potential. By choosing $\kappa_0 \geq 6\alpha$, we can ensure that $b-a = \kappa-\alpha-\kappa/2 = \kappa/3+\kappa/6-\alpha \geq \kappa/3$.

We pessimistically assume that the number-of-ones component of g_5 can only increase by at most 1. Lemma 6.4.13 has already shown that, even under this assumption, the drift is at least a positive constant. This implies the first condition of Theorem 2.4.16. For the second condition, we need to bound transition probabilities for the potential. Owing to our pessimistic assumption, the number of ones can only increase by at most 1.

For jumps decreasing the number of ones, we need to argue more carefully. Let $i = |x_t|_1 \geq a$ be the current number of ones and let $p_{i,j}$ be the probability that $|x_{t+1}|_1 = i-j$. Note that a jump back to the first slope it is sufficient that one offspring has at most $2n/3$ ones. A necessary requirement is that j bits flip, which has probability at most $1/(j!)$. By a union bound over λ offspring, $p_{i,j} \leq \lambda/(j!) \leq 2^{-\kappa/2}\cdot(\kappa/2)!/(j!)$ using our bound on λ. For $j \geq \kappa/2$ (as $\kappa \geq 2$), we have $j! \geq (\kappa/2)!\cdot 2^{j-\kappa/2}$ and $p_{i,j} \leq 2^{-j}$. This implies for all $i \geq a$ and all j with $i-j \leq 2n/3$:

$$\Pr\left(|x_t|_1 - |x_{t+1}|_1 \geq j\right) \leq \sum_{j' \geq j} 2^{-j'} \leq 2\cdot 2^{-j}.$$

In particular, $p^{\uparrow}_{i,\lambda} \leq \sum_{j=\kappa/2}^{n-i} p_{i,j} \leq \sum_{j=\kappa/2}^{\infty} 2\cdot 2^{-j} = 4\cdot 2^{-\kappa/2}$.

For $i-j > 2n/3$ the number of ones only decreases by j if *all* offspring decrease their number of ones by at least j, or if there is one offspring on the first slope. The probability of all offspring decreasing their number of ones by j is bounded by the probability that the first offspring decreases its number of ones by j. This is bounded by the probability of j bits flipping, which is at most $1/(j!) \leq 2/2^j$. Hence,

$$\forall i-j > 2n/3\colon \Pr\left(|x_t|_1 - |x_{t+1}|_1 \geq j\right) \leq 2\cdot 2^{-j} + p^{\uparrow}_{i,\lambda} \leq 6\cdot 2^{-j}.$$

The possible penalty in the definition of g_5 changes by at most $\max\left(\frac{2s}{s+1}, \frac{2s}{s+1} \cdot \frac{1}{s}\right) = \frac{2}{s+1} < 2$. Hence, for all t,

$$\Pr\left(|g_5(X_{t-1}) - g_5(X_t)| \geq j + 2 \mid g_5(X_t) > a\right) \leq \frac{24}{2^{j+2}},$$

which meets the second condition of Theorem 2.4.16 when choosing $\delta := 1$ and $\rho(\ell) := 24$.

The negative drift theorem (Theorem 2.4.16) now implies that there exists a constant c^* such that the probability of the number of ones dropping below a in $2^{c^*(b-a)/24} \geq 2^{c^*\kappa/72}$ generations (or reaching a search point with at least $3n/4$ ones) is $2^{-\Omega((b-a)/24)} = 2^{-\Omega(\kappa)}$. Choosing $\gamma := \min\{2^{c^*/72}, 2\}$, this is at least γ^κ generations and a probability of $\gamma^{-\Omega(\kappa)}$ as claimed. Taking a union bound over this failure probability and that from Lemma 6.4.14 proves the claim. □

Now we show that with probability $\Omega(1)$ a search point with at least $3n/4$ ones is reached, without returning to the first slope and without resetting λ. Thus, with the claimed probability the algorithm behaves as the self-adjusting $(1, \lambda)$ EA from [101] on OneMax throughout this part of the run.

Lemma 6.4.16. *Consider the self-adjusting* $(1, \lambda)$ EA *as in Theorem 6.4.1. Assume the conditions from Lemma 6.4.15 hold for constants γ and $\kappa := \frac{\log\log n}{\log\log\log n}$. Then with probability $\Omega(1)$ a search point with at least $3n/4$ ones is reached within $O(n)$ generations.*

Moreover, with the claimed probability the algorithm does not go back to the first slope and does not reset λ before a search point with at least $3n/4$ ones is reached.

***Proof*.** The statement of Lemma 6.4.15 satisfies the preconditions of Lemma 6.4.13 for $d = \kappa$. Then Lemma 6.4.13 implies a positive drift

$$\mathrm{E}(g_5(X_{t+1}) - g_5(X_t) \mid X_t) \geq \frac{1-s}{4e} =: \delta$$

for the next γ^κ generations, unless a search point with at least $3n/4$ ones has been reached. In the latter case we are done, hence we assume that the drift is bounded from below as stated through the next $t_\kappa := \min\{\gamma^\kappa, n/\delta\}$ generations. By the additive drift theorem (Theorem 2.4.14), the expected time to increase the potential by $\delta/12 \cdot \min\{\gamma^\kappa, n/\delta\}$, while the drift bound holds, is at most $\frac{\delta/12 \cdot \min\{\gamma^\kappa, n/\delta\}}{\delta} = \min\{\gamma^\kappa/12, n/(12\delta)\}$. By Markov's inequality, the probability that after t_κ steps the potential has not increased by $\delta/12 \cdot \min\{\gamma^\kappa, n/\delta\} = \min\{\delta/12 \cdot \gamma^\kappa, n/12\}$ is at most $1/12$.

Assuming that the potential has increased by $\min\{\delta/12 \cdot \gamma^\kappa, n/12\}$, by Definition 6.4.13 the number of ones has increased by $\min\{\delta/12 \cdot \gamma^\kappa, n/12\} - O(1)$. Since we start with at least $2n/3 + \kappa = 2n/3 + \omega(1)$ ones, there is a generation amongst the next γ^κ generations in which the number of ones is at least $\min\{2n/3 + \delta/12 \cdot \gamma^\kappa, 3n/4\}$.

Let $\kappa_0 := \kappa = \frac{\log\log n}{\log\log\log n}$ and define $\kappa_i := \delta/12 \cdot \gamma^{\kappa_{i-1}}$ for $i > 0$. Note that we have just showed that we have found a search point with at least $\min\{\kappa_1, 3n/4\}$ ones. If the number of ones is less than $3n/4$, the current state (x_t, λ_t) is κ_1-safe as it is weakly κ_0-safe and so $\lambda_t \leq 2^{-\kappa_0/2} \cdot (\kappa_0/2)! \leq 2^{-\kappa_1/2} \cdot (\kappa_1/2)!$.

Iterating the above argument with κ_1 instead of κ_0, we find a search point with at least $\min\{2n/3 + \kappa_2, 3n/4\}$ ones within the next $\min\{\gamma^{\kappa_1}, n/\delta\}$ generations, with probability at least $1 - 1/12 - \gamma^{-\Omega(\kappa_1)}$. We again iterate the argument with κ_2 and once again with κ_3. We claim that $t_{\kappa_2} := \min\{\gamma^{\kappa_2}, n/\delta\} = n/\delta$ and show this by bounding κ_1, κ_2 and κ_3 from below.

$$\begin{aligned}\kappa_1 = \frac{\delta}{12}\gamma^{\frac{\log\log n}{\log\log\log n}} &= 2^{\log(\gamma)\cdot\frac{\log\log n}{\log\log\log n} + \log(\delta/12)} \\ &\geq 2^{\log\left(\frac{2}{\log(\gamma)}\log\log n\right)} = \frac{2}{\log(\gamma)}\log\log n.\end{aligned}$$

Now, κ_2 is at least

$$\kappa_2 = \frac{\delta}{12}\gamma^{\kappa_1} \geq \frac{\delta}{12}\gamma^{\frac{2}{\log(\gamma)}\log\log n} = \frac{\delta}{12}\log^2 n \geq \frac{2}{\log(\gamma)}\log n$$

for n large enough. Likewise,

$$\kappa_3 = \frac{\delta}{12}\gamma^{\kappa_2} \geq \frac{\delta}{12}\cdot\gamma^{\frac{2}{\log(\gamma)}\log n} = \frac{\delta}{12}\cdot n^2.$$

Together, we have that within $\min\{\gamma^{\kappa_0}, n/\delta\} + \min\{\gamma^{\kappa_1}, n/\delta\} + \min\{\gamma^{\kappa_2}, n/\delta\} = O(n)$ generations, with probability at least $1 - \frac{3}{12} - \gamma^{-\Omega(\kappa_0)} - \gamma^{-\Omega(\kappa_1)} - \gamma^{-\Omega(\kappa_2)} \geq \frac{3}{4} - 3\gamma^{-\Omega(\kappa_0)} = \Omega(1)$ we have reached a search point with at least $3n/4$ ones, without going back to the first slope. The probability of a reset during $O(n)$ expected generations is exponentially small by Lemma 6.4.3 and (2.4), hence this failure probability can be absorbed in the $\Omega(1)$ probability bound. This completes the proof. □

6.4.4 Finding the Global Optimum

Once the self-adjusting $(1,\lambda)$ EA moves far away from the cliff, the probability of jumping back up the cliff is reduced, and the next part of the optimisation resembles OneMax. The algorithm still can reset λ to 1. Such a steep decrease of λ would typically make the algorithm slip down the second slope until λ recovers to large enough values that support hill climbing. Hence, resets would break the runtime analysis made in Chapter 5. We show in Lemma 6.4.17 that, with probability $\Omega(1)$, the algorithm neither jumps back up the cliff nor resets λ during the last part of the optimisation. This allows us to apply the previous analysis from Chapter 5 on OneMax.

Lemma 6.4.17. *Consider the self-adjusting* $(1,\lambda)$ EA *as in Theorem 6.4.1. For any initial* $\lambda_0 \leq \lambda_{\max}$ *with* $\lambda_0 = O(n\log n)$ *and any initial search point* x_0 *with* $|x_0|_1 \geq 3n/4$ *the probability that the self-adjusting* $(1,\lambda)$ EA *creates the optimum without jumping back up the cliff or resetting* λ *to 1 is at least* $1 - \frac{1}{e-1} - O\left(\frac{\log^3(n)}{n}\right)$.

Proof. As long as the self-adjusting $(1,\lambda)$ EA does not jump back up the cliff, the self-adjusting $(1,\lambda)$ EA behaves as the self-adjusting $(1,\lambda)$ EA on OneMax. Additionally, if it does not have an unsuccessful generation with $\lambda = \lambda_{\max}$ it will never reset to 1, behaving as the self-adjusting $(1,\lambda)$ EA studied in Chapter 5.

From Theorems 5.3.3 and 5.3.7 we know that the self-adjusting $(1,\lambda)$ EA solves OneMax in expected $O(n)$ generations and $O(n\log n)$ evaluations. Therefore, within these expected times our algorithm either finds the global optimum, jumps back up the cliff or resets 1. We show that the with probability $\Omega(1)$, a global optimum is reached.

In order for λ to reset at the same time as there is a jump back up the cliff, at least one offspring must flip $n/3$ one-bits and all other offspring must not increase their fitness. The probability of flipping $n/3$ bits is $n^{-\Omega(n)}$, hence the probability of both events happening at the same time is at most $n^{-\Omega(n)}$.

By Lemma 5.3.9 if the initial search point x_0 has $|x_0|_1 \geq 3n/4$, with probability $1-O(1/n)$ the number of one-bits will never drop below $3n/4 - O(\log n)$ before finding the optimum. This means that, for the algorithm to jump back up the cliff at least one offspring must flip a linear amount of bits. The probability that one offspring flips a linear amount of bits is $n^{-\Omega(n)}$. By (2.4), the probability that this happens during $O(n\log n)$ expected evaluations is still $n^{-\Omega(n)}$. In the following we assume that we never return to the first slope.

To show that there is never an unsuccessful generation with $\lambda = \lambda_{\max}$ (i.e. λ never resets to 1) we divide the optimisation in two phases. The first phase ends the first time a state (x_t, λ_t) is found with $\lambda_t \geq 4\log n$ and $|x_t|_1 \geq n - 3\ln n$ or the optimum is found, and the second phase ends when the optimum is found.

During the first phase, since $\lambda_{\max} > 4\log n$, we can only reach $\lambda = \lambda_{\max}$ if $|x|_1 < n - 3\ln n$ otherwise we would start phase two. In order to reach $\lambda = \lambda_{\max}$ at least one generation with $\lambda \geq en$ must be unsuccessful. The probability of an unsuccessful generation with $\lambda \geq en$ is at most

$$1 - p_{i,\lambda}^{\rightarrow} + p_{i,\lambda}^{\uparrow} \leq \left(1 - \frac{n-i}{en}\right)^{en} \leq \left(1 - \frac{3\ln n}{en}\right)^{en} \leq e^{-3\ln n} = n^{-3}.$$

Given that the optimum is found after $O(n)$ expected generations, by (2.4) the probability of reaching $\lambda = \lambda_{\max}$ during the first phase is $O(1/n^2)$.

For the second phase we first argue that the current fitness does not decrease, with high probability. The second phase starts with $\lambda \geq 4\log n$ and by Lemma 5.3.9 while $\lambda_t \geq 4\log n$ the fitness is not reduced before reaching the optimum with probability $1 - O(1/n)$. We now show that $\lambda \geq 4\log n$ throughout the remainder of the run with high probability.

By Lemma 5.3.8 from $|x|_1 \geq n - 3\ln n$ in expectation the optimum will be reached in $O(\log n)$ generations. To reduce λ to a value smaller than $4\log n$, a generation with $\lambda < 4F\log n$ must be successful. This event has a probability of at most

$$1 - \left(\frac{i}{n}\right)^{\lambda} \leq 1 - \left(1 - \frac{3\ln n}{n}\right)^{4F\log n} \leq \frac{12F\log^2 n}{n\log e} = O\left(\frac{\log^2(n)}{n}\right).$$

By (2.4) the probability that λ is reduced to a value less than $4\log n$ during the next $O(\log n)$ generations is $O\left(\frac{\log^3(n)}{n}\right)$. Accounting for both failures with probability $1-O\left(\frac{\log^3(n)}{n}\right)$ each fitness value is left at most once.

Now we can calculate the probability of resetting λ by considering at most one generation with $\lambda = \lambda_{\max}$ per fitness value. We only have a reset of λ if one such generation is unsuccessful. Thus, the probability of resetting λ during the second phase is at most

$$\begin{aligned}
\sum_{i=n-3\ln n}^{n-1} \left(1 - p_{i,\lambda_{\max}}^{+}\right) &\leq \sum_{i=n-3\ln n}^{n-1} \left(1 - \frac{n-i}{en}\right)^{enF^{1/s}} \\
&\leq \sum_{i=n-3\ln n}^{n-1} e^{-F^{1/s}(n-i)} = \sum_{j=1}^{3\ln n} e^{-F^{1/s}j} \leq \sum_{j=1}^{\infty} e^{-F^{1/s}j} \\
&= \frac{1}{1-e^{-F^{1/s}}} - 1 = \frac{1}{\exp(F^{1/s}) - 1} \leq \frac{1}{e-1}.
\end{aligned}$$

Adding up all failure probabilities completes the proof. □

6.4.5 Putting Things Together

Now we are able to prove the claimed bounds of $O(n)$ expected generations and $O(n\log n)$ expected evaluations from Theorem 6.4.1.

Proof of Theorem 6.4.1. From any initial state, by Lemma 6.4.2 we reach a solution x_t with $|x_t|_1 \geq 2n/3$ in expected $O(n)$ generations and $O(n + \lambda_{\max})$ evaluations.

Then, by Lemma 6.4.9, the algorithm reaches a κ-safe state (for $\kappa := \frac{\log\log n}{\log\log\log n}$) or a search point with at least $3n/4$ ones in $O(\log(\lambda_{\max})\log n)$ expected generations and $O(\lambda_{\max}\log n)$ expected evaluations.

Together, along with $\lambda_{\max} = \Omega(n)$ and $\lambda_{\max} = \text{poly}(n)$, the total time to reach a κ-safe state or a search point with at least $3n/4$ ones from an arbitrary initial state is $O(\log(\lambda_{\max})\log n + n) = O(n)$ generations and $O(\lambda_{\max}\log n)$ evaluations.

By Lemma 6.4.16 with probability $\Omega(1)$ we reach a search point with at least $3n/4$ ones within $O(n)$ generations, without going back to the first slope or resetting λ. During this time, the algorithm behaves like the self-adjusting $(1, \lambda)$ EA without resetting on OneMax and we obtain an upper bound of $O(n \log n)$ evaluations from Theorem 5.3.7. Hence, the expected number of evaluations until we return to the first slope, reset λ or reach a search point with $3n/4$ ones is $O(n \log n)$. In expectation, a constant number of trials suffices to find a search point with at least $3n/4$ ones.

Hence, from any initial state, in expectation in $O(n)$ generations and $O(\lambda_{\max} \log n)$ evaluations we reach a search point with at least $3n/4$ ones.

Likewise, from a search point with at least $3n/4$ ones, by Lemma 6.4.17 with probability $\Omega(1)$ we find the optimum without resetting λ or returning to the first slope, and hence the analysis from Theorem 5.3.7 still applies. Thus, in expected $O(n)$ generations and $O(n \log n)$ evaluations we either reach the global optimum, return to the first slope or reset λ, and the probability of reaching the optimum is $\Omega(1)$. Iterating this argument an expected constant number of times proves the claimed bound. $\square$

6.5 Experimental Analysis

In this section we conduct an experimental analysis to improve our understanding of the $(1, \lambda)$ EA with static λ and the self-adjusting $(1, \lambda)$ EA resetting λ on Cliff and other functions. All the experiments were performed using the IOHProfiler [72].

For algorithm comparisons we performed Mann-Whitney U Test for all pairs of algorithms compared in the text. We performed two-sided tests to check whether the two input distributions differ or not, followed by one-sided tests both ways to confirm which algorithm is stochastically faster than the other. We report a comparison as statistically significant if the p-values of the two-sided test and that of the respective one-sided test both satisfied $p \leq 0.01$, that is, a confidence level of 0.01.

6.5.1 Cliff

We begin our analysis with a runtime comparison between the self-adjusting $(1, \lambda)$ EA resetting λ and the $(1, \lambda)$ EA with static λ on Cliff. For the $(1, \lambda)$ EA with static λ we use two values of λ. The first one is $\lambda = \left\lceil \log_{\frac{e}{e-1}}(e^2 n/\lambda) \right\rceil$ which is the smallest λ-value that theoretically guarantees an efficient runtime on OneMax [20]. The second value of λ used was manually tuned to the λ-value that yielded the best performance in terms of number of evaluations for each problem size n. For the self-adjusting $(1, \lambda)$ EA we used as parameters $s = 1$, $F = 1.5$ and $\lambda_{\max} = enF^{1/s}$. Figure 6.1 displays box plots of the number of evaluations used by the three algorithms over 100 runs for different problem sizes on Cliff.

There are two main things we can appreciate from Figure 6.1. Our first observation is that for the $(1, \lambda)$ EA with static λ, there are smaller λ-values than the theoretical best parameter values which perform better for the problem sizes tested here. This is because the exponential lower bound (from [166, Theorem 10]) for the runtime of the $(1, \lambda)$ EA when using $\lambda \leq (1 - \varepsilon) \log_{\frac{e}{e-1}} n$ only becomes important for large values of n. This observation was also made by Rowe and Sudholt [166] for OneMax with $n = 10000$. Our second observation is that even with the best (tuned) static λ-value the self-adjusting $(1, \lambda)$ EA shows a slightly better performance, although the self-adjusting $(1, \lambda)$ EA is statistically faster only for $n \geq 87$.

Because of this last observation we ran more experiments for larger problem sizes comparing only the $(1, \lambda)$ EA with tuned λ and the self-adjusting $(1, \lambda)$ EA. The results of these experiments, shown in Figure 6.2, demonstrate that for larger values of n the $(1, \lambda)$ EA with static λ is statistically slower than the self-adjusting $(1, \lambda)$ EA for all $n \geq 120$. In addition since the number of evaluations is normalised by $n \log n$ we can clearly see that the number

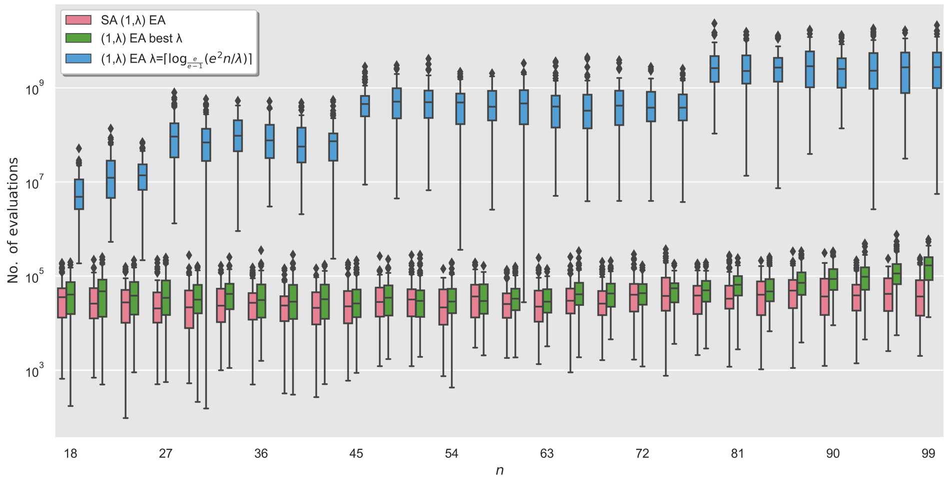

Figure 6.1: Box plots of the number of evaluations used by the $(1, \lambda)$ EA with $\lambda = \left\lceil \log_{\frac{e}{e-1}}(e^2 n/\lambda) \right\rceil$ and the best λ (manually tuned), and the self-adjusting $(1, \lambda)$ EA with $s = 1$, $F = 1.5$, $\lambda_{\max} = enF^{1/s}$ on CLIFF with $n \in \{18, 21, \ldots, 99\}$ over 100 runs. The y-axis (number of evaluations) is log-scaled.

of evaluations of the $(1, \lambda)$ EA grows faster than $n \log n$. This means that the larger the problem size n, the worse the performance of the $(1, \lambda)$ EA with static λ is, compared to the self-adjusting $(1, \lambda)$ EA.

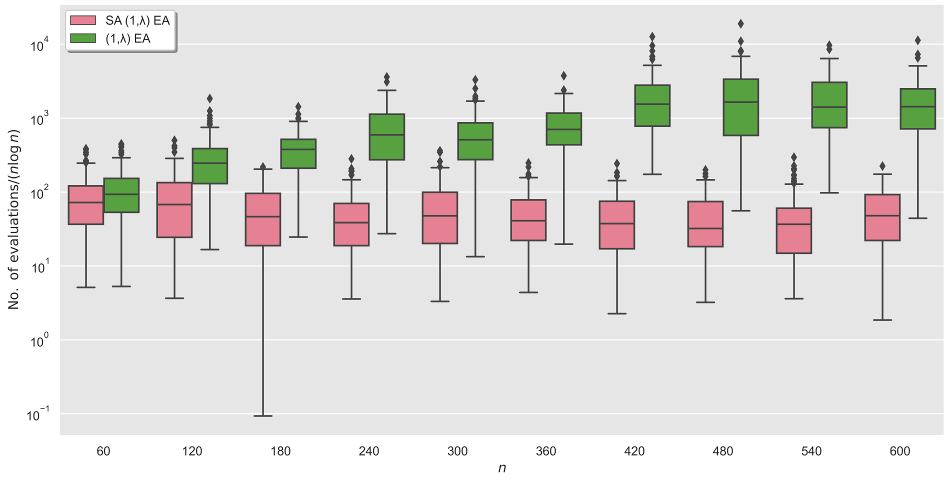

Figure 6.2: Box plots of the number of evaluations used by the $(1, \lambda)$ EA with the best λ (manually tuned), and the self-adjusting $(1, \lambda)$ EA with $s = 1$, $F = 1.5$, $\lambda_{\max} = enF^{1/s}$ on CLIFF over 100 runs. The y-axis (number of evaluations) is normalised by $n \log n$ and log-scaled.

In Figure 6.3 we explore how the $(1, \lambda)$ EA and self-adjusting $(1, \lambda)$ EA behave on CLIFF with $n = 30$ and different distance d from the *cliff* to the optimum. For the $(1, \lambda)$ EA we use $\lambda = \left\lceil \log_{\frac{e}{e-1}}(e^2 n/\lambda) \right\rceil$ and $\lambda = 3$ which is the best λ found for the standard CLIFF function. Both plots in Figure 6.3 show the same experiments, but the number of evaluations in the top box plots of Figure 6.3 is normalised by $n^d(1-1/n)^{d-n}$, which is the runtime of the $(1+1)$ EA on Cliff with distance d from the cliff to the optimum [153].

From Figure 6.3 we can see that when the distance d from the cliff to the optimum is reduced, the problem becomes harder for all algorithms up to a maximum from which reducing d more reduces their runtime. We also see that for small d the algorithms' runtime seem to grow at almost $\Theta(n^d(1-1/n)^{n-d})$. Because of this, we conjecture that for small distances d from the cliff to the optimum most of the time the algorithms find the global optimum without jumping down the cliff, therefore the algorithms do not benefit from their non-elitist selection in these cases. On the other hand for larger values of d the algorithms benefit greatly from their non-elitist selection because they are able to jump down the cliff more easily. This leaves a goldilocks d-value in the middle for which both jumping directly to the optimum and jumping down the cliff is difficult.

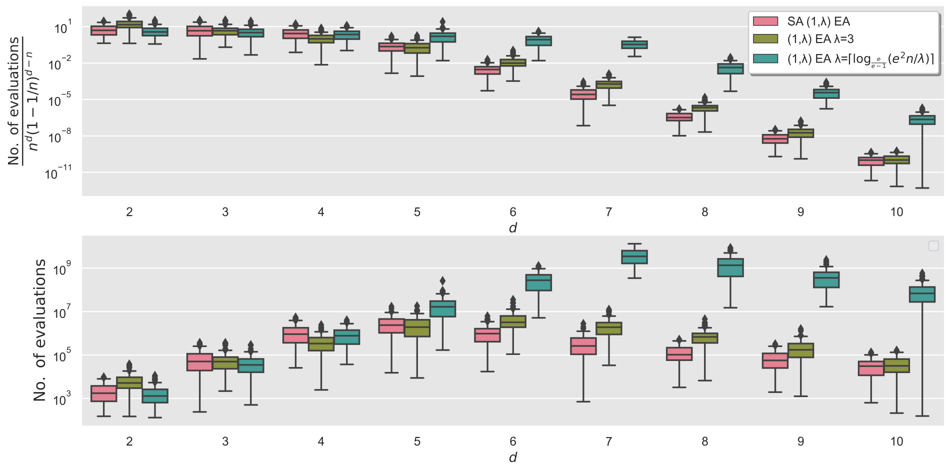

Figure 6.3: Box plots of the number of evaluations used by the $(1,\lambda)$ EA with $\lambda = \left\lceil \log_{\frac{e}{e-1}}(e^2n/\lambda) \right\rceil$ and $\lambda = 3$ (best λ for the standard Cliff function), and the self-adjusting $(1,\lambda)$ EA with $s=1$, $F = 1.5$, $\lambda_{\max} = enF^{1/s}$ on Cliff with $n = 30$, varying the distance d of the cliff from the optimum over 100 runs. The y-axis (number of evaluations) is log-scaled and in the top box plots also normalised by $n^d(1-1/n)^{d-n}$.

6.5.2 Varying $\lambda_{\max}$

We now explore how the hyper-parameter $\lambda_{\max}$ affects the performance of the self-adjusting $(1,\lambda)$ EA. Our theoretical analysis suggests that increasing the value of $\lambda_{\max}$ will increase the runtime of the algorithm because the algorithm spends more evaluations on the local optimum before resetting λ and therefore on jumping down the cliff. On the other hand decreasing the value of $\lambda_{\max}$ can decrease the runtime of the algorithm until a point were resets in λ does not allow the algorithm to optimise the second slope. In our theoretical analysis we used a value of $\lambda_{\max} = enF^{1/s}$ that allows the algorithm to reset relatively fast when encountering a local optima but also allows the algorithm to optimise the second slope with a small number of resets. We acknowledge that the factor $eF^{1/s}$ in our theoretical analysis was used to ease calculations; a smaller factor could improve performance but only by a constant factor.

In Figure 6.4 we show box plots of the number of evaluations used by the self-adjusting $(1,\lambda)$ EA with $s=1$, $F = 1.5$ over 100 runs for different $\lambda_{\max}$ on Cliff with $n = 1500$. As expected larger values of $\lambda_{\max}$ increase the runtime of the algorithm.

We can also see that there are smaller values than $\lambda_{\max} = enF^{1/s} \approx 4.08n$ that are better on Cliff. Values between $\frac{n\sqrt{2}}{4}$ and $\frac{n\sqrt{2}}{2}$ obtain the best performance and reducing $\lambda_{\max}$ further than this deteriorates the performance of the algorithm substantially.

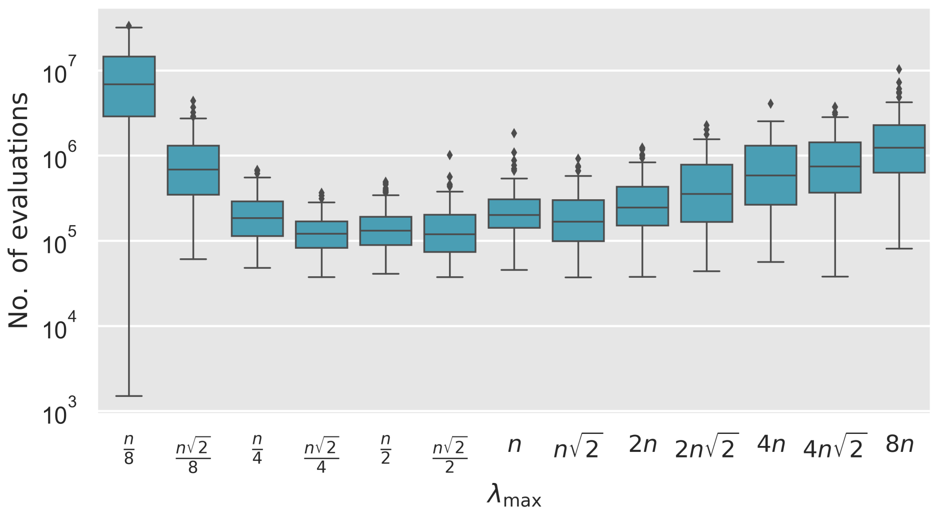

Figure 6.4: Box plots of the number of evaluations used by the self-adjusting $(1, \lambda)$ EA with $s = 1$, $F = 1.5$ over 100 runs for different $\lambda_{\max}$ on Cliff with $n = 1500$. The y-axis (number of evaluations) is log-scaled.

To explore further in Figure 6.5 we show box plots of the number of evaluations used by the self-adjusting $(1, \lambda)$ EA with $s = 1$, $F = 1.5$ over 100 runs for $\lambda_{\max} \in \{enF^{1/s}, n/2\}$ on Cliff and OneMax. On top of the plot we give the average number of resets per run rounded to the nearest integer. We can see that for all problem sizes n tested using $\lambda_{\max} = n/2$ gives an advantage to the self-adjusting $(1, \lambda)$ EA on Cliff but a disadvantage on OneMax. This is because for $\lambda_{\max} = n/2$ the algorithm jumps down the cliff faster but there tends to be an increased number of resets on average. Since this happens in both OneMax and Cliff, we attribute this to resets near the optimum where λ grows fast. An interesting observation is that using $\lambda = enF^{1/s}$ for OneMax allows the algorithm to find the optimum with almost no resets. Out of the 500 runs for all n there were only 5 resets and 3 of them were with $n = 300$. This suggests that our theoretical analysis was too unforgiving, since we bounded the number of resets near the optimum in a typical run by a constant.

In Figure 6.6 we show a detailed view of four example runs. Here we can see that on Cliff both values of $\lambda_{\max}$ reset λ and jump down the cliff several times before starting to optimise the second slope. But, when using $\lambda_{\max} = n/2$ the fitness stagnates during the final generations before reaching the global optimum in both OneMax and Cliff. This is because λ resets and then there are fallbacks in fitness. It is important to note that these fallbacks are small and the algorithm is able to recover fast from them. In our theoretical analysis we assumed a much worse case were a reset near the optimum would cause the algorithm to go back to the first slope.

6.5.3 Other Problems

We finish our empirical analysis with a comparison between the self-adjusting $(1, \lambda)$ EA and the $(1, \lambda)$ EA with static λ for different problems. We select four problems from the Pseudo-Boolean Optimization (PBO) problem set proposed by [74].

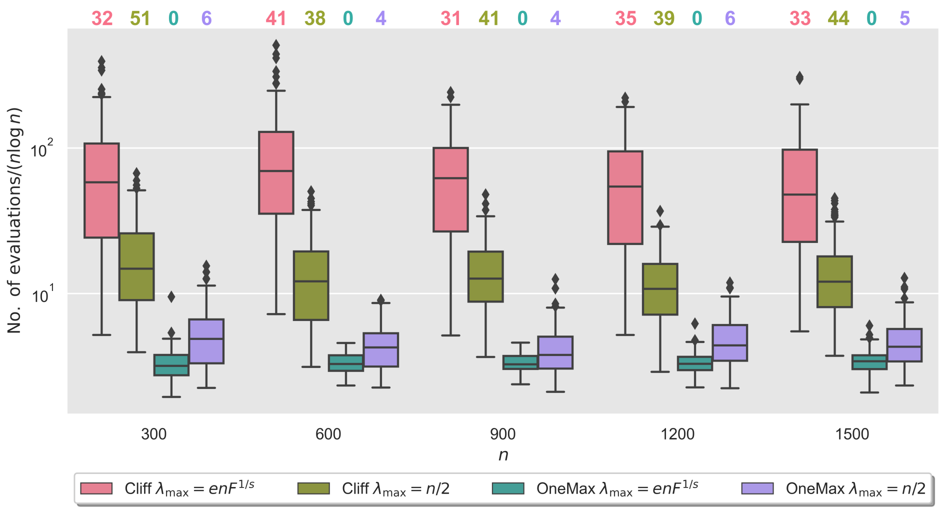

Figure 6.5: Box plots of the number of evaluations used by the self-adjusting $(1, \lambda)$ EA with $s = 1$, $F = 1.5$ over 100 runs for $\lambda_{\max} \in \{enF^{1/s}, n/2\}$ on Cliff and OneMax. The rounded average number of resets per run is shown on the top. The y-axis (number of evaluations) is normalised by $n \log n$ and log-scaled.

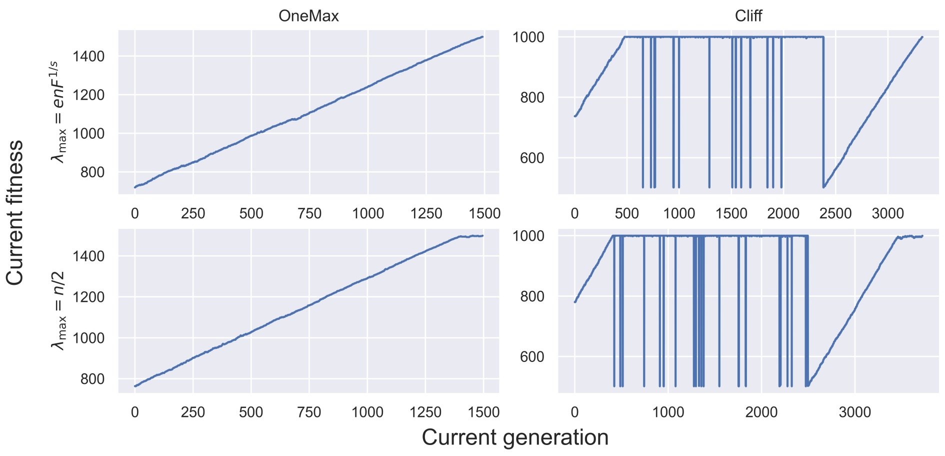

Figure 6.6: Current fitness per generation of four example runs of the self-adjusting $(1, \lambda)$ EA with $\lambda_{\max} = enF^{1/s}$ (top) and $\lambda_{\max} = n/2$ (down) on OneMax (left) and Cliff (right) with $n = 1500$.

- Low Autocorrelation Binary Sequences (LABS): A non-linear combinatorial problem. The problem is to maximise the merit factor of a binary sequence. The merit factor is proportional to the reciprocal of the sequence's autocorrelation (also called serial correlation). Its equivalent pseudo-Boolean function is:

$$\mathrm{LABS}(x) = \frac{n^2}{2\sum_{k=1}^{n-1}\left(\sum_{i=1}^{n-k} x'_i \cdot x'_{i+k}\right)^2} \quad \text{where } x'_i = 2x_i - 1.$$

- Ising Model problem: consists of discrete variables that represent magnetic dipole moments of atomic "spins" that can be in one of two states (+1 or -1). The spins are arranged in a graph describing the interactions strengths (edges) between spins (vertices), here we use a two-dimensional torus lattice. Neighboring spins with the same state have a lower interaction than those with a different state. The ISG problem is to set the signs of all spins to minimise interactions.

- Maximum Independent Vertex Set (MIS): An independent vertex set is a subset of vertices where no vertex in the subset is adjacent to any other vertex within the subset. The MIS problem is to find the largest independent vertex set for a given graph.
- N-Queens problem: Is the task to place N queens in a chess board of size $N \times N$ in such a way that the N queens cannot "attack" each other.

For more information on the problems we refer the reader to [74] and the IOHProfiler project [72]. We performed 100 runs of the self-adjusting $(1, \lambda)$ EA with $\lambda_{\max} \in \{enF^{1/s}, n\}$ and the $(1, \lambda)$ EA with $\lambda = \lceil \log_{\frac{e}{e-1}}(n) \rceil$ across the four problems with different problem sizes. Figure 6.7 shows the results of these runs. We included the self-adjusting $(1, \lambda)$ EA with $\lambda_{\max} = n$ because the results from Section 6.5.2 suggest that the factor $eF^{1/s}$ used in our theoretical analysis is not needed, but a smaller value of $\lambda_{\max} = n/2$ already results in a substantial number of resets.

Overall we can see that the self-adjusting $(1, \lambda)$ EA with $\lambda_{\max} = n$ performs slightly better than its counterpart with $\lambda_{\max} = enF^{1/s}$ for all problems and most problem sizes tested. When comparing the self-adjusting $(1, \lambda)$ EA with $\lambda_{\max} = n$ and the $(1, \lambda)$ EA with static λ we found that the self-adjusting $(1, \lambda)$ EA is statistically significantly faster than the $(1, \lambda)$ EA for most problem sizes on LABS, MIS and the N-Queens problem, but statistically significantly slower on Ising models. The main conclusion that we draw from Figure 6.7 is that the self-adjusting $(1, \lambda)$ EA is able to perform better or similar to the $(1, \lambda)$ EA on common multimodal benchmark problems with the added advantage of easier parameter selection.

6.6 Conclusions

The usefulness of parameter control has so far mainly been demonstrated for elitist evolutionary algorithms on relatively easy problems. For the more difficult multimodal problem Cliff we showed that the self-adjusting $(1, \lambda)$ EA using success-based rules and a reset mechanism can find the global optimum in $O(n)$ expected generations and $O(n \log n)$ expected evaluations. This is a speedup of order $\Omega(n^{2.9767})$ over the expected optimisation time with the best fixed value of λ.

The latter conclusion was obtained by refining the previous bounds on the expected optimisation time of the $(1, \lambda)$ EA on Cliff from [106], $O(e^{5\lambda}) = O(148.413^{\lambda})$ and $\min\{n^{n/4}, e^{\lambda/4}\}/3 = \min\{n^{n/4}, 1.284^{\lambda}\}/3$, towards bounds of $\Omega(\xi^{\lambda})$ and $O(\xi^{\lambda} \log n)$, for $\xi \approx 6.196878$, revealing the degree of the polynomial in the expected runtime of the $(1, \lambda)$ EA with the best fixed λ as $\eta \approx 3.976770136$.

Our theoretical results demonstrate the power of parameter control for the multimodal Cliff problem and that drastic performance improvement can be obtained. We extended our theoretical results with an extensive empirical analysis. We showed how the $(1, \lambda)$ EA and the self-adjusting $(1, \lambda)$ EA behave on Cliff with realistic problem sizes and Cliff functions where the position of the cliff is chosen differently from $2n/3$ ones. We also analysed how the hyper-parameter $\lambda_{\max}$ affects the performance of the algorithm, showing that too small values of $\lambda_{\max}$ can deteriorate the performance of the algorithm but $\lambda_{\max} = n$ and $\lambda_{\max} = enF^{1/s}$ achieve a good performance on all problems tested. Finally we considered other problems and showed that the self-adjusting $(1, \lambda)$ EA performs better or similar to the $(1, \lambda)$ EA with the added advantage of easier parameter selection.

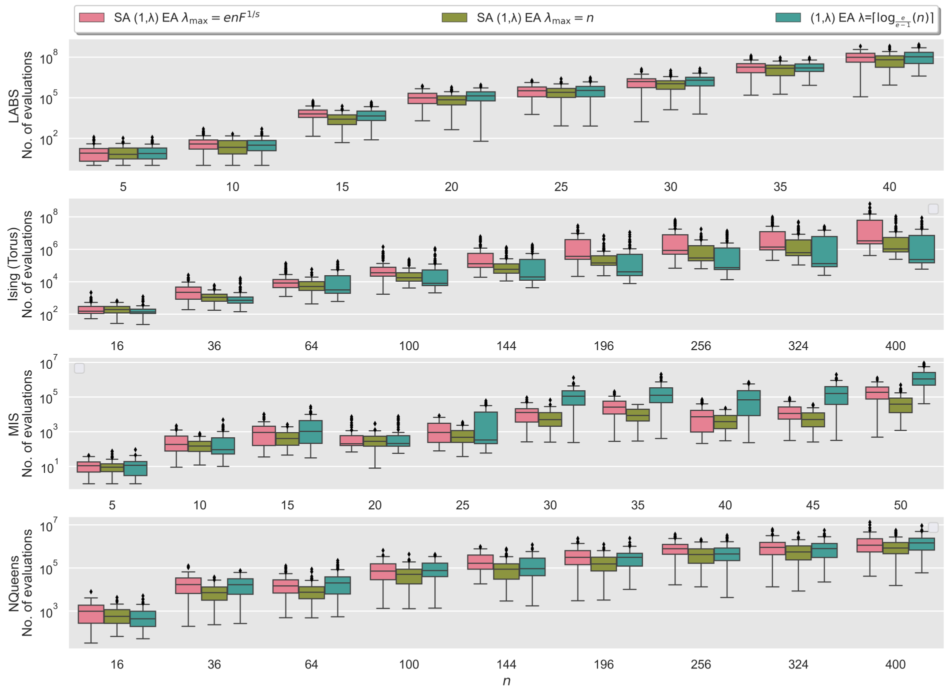

Figure 6.7: Box plots of the number of evaluations used by the self-adjusting $(1, \lambda)$ EA with $s = 1$, $F = 1.5$ over 100 runs for $\lambda_{\max} \in \{enF^{1/s}, n\}$ on LABS, Ising, MIS and N-Queens problems. The y-axis (number of evaluations) is log-scaled.

For the $(1, \lambda)$ EA the parameter λ has a huge impact in performance. If λ is too small we obtain exponential runtimes in the problem size n [166] and λ needs to be $\Omega(\log n)$ to have efficient runtimes. But the probability of accepting a worsening in fitness and leaving a local optima is related exponentially to the parameter λ, which means that increases in λ by a constant factor can increase the runtime by a polynomial factor in the problem size n, making parameter tuning a hard task. On the other hand for the self-adjusting $(1, \lambda)$ EA the parameter $\lambda_{\max}$ is less critical. Although small $\lambda_{\max}$ can deteriorate the performance of the algorithm, our theoretical and empirical analyses suggest that the time to escape a local optima depends only linearly on $\lambda_{\max}$ (instead of exponentially), therefore a wide range of $\lambda_{\max}$-values obtain the same asymptotic runtime. This eases the task of parameter tuning for the self-adjusting $(1, \lambda)$ EA.

Chapter 7

Conclusions

The analysis of parameter control mechanisms is a relatively new but very fruitful research area. We have seen a staggering amount of progress in the last decade, with a plethora of studies showing the advantages of parameter control mechanisms. In spite of this progress there are still plenty of open questions regarding the use of parameter control mechanisms for multimodal optimisation. In this thesis we have contributed to answer some of these questions for success-based parameter control mechanisms. More specifically, we improved our understanding of how success-based parameter control mechanisms for evolutionary algorithms can be efficiently implemented for multimodal optimisation.

We began by analysing one of the most successful implementations of success-based parameter control mechanisms: the self-adjusting $(1+(\lambda,\lambda))$ GA [50, 60]. We showed that parameter control mechanism in the original self-adjusting $(1+(\lambda,\lambda))$ GA tends to diverge λ to its maximum value $\lambda_{\max} = n$ on multimodal problems. Then the self-adjusting $(1+(\lambda,\lambda))$ GA behaves as the (1+n) EA with mutation probability $p = 1/n$. Using this insight we gave an easy to use general method for obtaining upper bounds on the expected number of fitness evaluations to find a global optimum of the original self-adjusting $(1+(\lambda,\lambda))$ GA, based on the fitness-level method. With it we showed novel runtime upper bounds for unimodal functions and the multimodal benchmark problem class Jump_k. The latter bound is asymptotically the same as the runtime of the $(1+1)$ EA. Afterwards, we show that this bound is tight up to lower-order terms by giving a matching lower bound. This demonstrates that the self-adjusting $(1+(\lambda,\lambda))$ GA does not benefit from the use of crossover on Jump_k.

Capping λ to a value $\lambda_{\max}$ different than n can be beneficial, but choosing an appropriate value for $\lambda_{\max}$ requires problem-specific knowledge. As a generic choice we suggest using $\lambda_{\max} = n/2$. In this case, when λ reaches its maximum it performs random search steps during the mutation phase. This not only guards the algorithm against hard and deceptive problems it can also improve the ability of the algorithm to escape "small" local optima. The self-adjusting $(1+(\lambda,\lambda))$ GA with $\lambda_{\max} = n/2$ is faster than the self-adjusting $(1+(\lambda,\lambda))$ GA with $\lambda_{\max} = n$ and the $(1+1)$ EA with the default mutation rate $1/n$ for all Jump_k functions with $k \leq \log n$ and $k > n/\log n$. Furthermore, the algorithm still retains its original exploitation capabilities in the crossover phase.

Our analysis also provided insights into the parameter landscape, which describes how parameter values relate to performance. We showed that the parameter landscape with respect to the impact of $\lambda_{\max}$ on the runtime of the self-adjusting $(1+(\lambda,\lambda))$ GA on the multimodal benchmark problem class Jump_k is bimodal for appropriate problem sizes n and jump sizes k. For most common values of n and k the two local optima in the bimodal landscape are far apart from each other. Additionally, for different values of n and k the optimal parameter choice for $\lambda_{\max}$ switches between the two local optima. This fluctuating parameter landscape combined with it being bimodal makes parameter selection difficult.

If instead of capping λ to $\lambda_{\max}$ we reset the parameter to 1 whenever there is an unsuccessful generation at the maximum value, we avoid the parameter selection problem for $\lambda_{\max}$. This makes the self-adjusting $(1 + (\lambda, \lambda))$ GA cycle through the parameter space whenever a local optima is encountered, approaching optimal or near-optimal parameter values in every cycle in spite of the parameter landscape. For $\textsc{Jump}_k$, this strategy gives the same expected runtime as that of the $(1 + 1)$ EA with the optimal mutation rate and fast mutation operators, up to small polynomial factors.

After analysing the elitist self-adjusting $(1 + (\lambda, \lambda))$ GA, we then presented in Chapter 5 the first runtime analysis of a success-based parameter control mechanism adjusting λ on a non-elitist algorithm. We proved that a success-based parameter control mechanism adjusting the parameter λ in a $(1, \lambda)$ EA, is able to optimise OneMax in $O(n)$ generations and $O(n \log n)$ evaluations. The latter is best possible for any unary unbiased black-box algorithm [67, 128]. This result required the correct selection of the success rate s. The above holds for constant $0 < s < 1$, however the runtime on OneMax (and other common benchmark problems) becomes exponential with overwhelming probability if $s \geq 18$.

In sharp contrast, the self-adjusting $(1, \lambda)$ EA is not affected by the choice of the success rate (from positive constants) if the problem in hand is everywhere hard, that is, improvements are always found with a probability of at most $n^{-\varepsilon}$ for a constant $\varepsilon > 0$. In this case self-adjusting the offspring population size drastically reduces the number of generations to just $O(d + \log(1/p^{+}_{\min}))$, that is, roughly to the number of fitness values, improving and generalising previous results [120]. Nevertheless, we show that the expected number of evaluations is bounded by the same fitness-level upper bound as known for the $(1 + 1)$ EA.

As a byproduct of our analysis, we have also shown an upper bound for the expected number of evaluations of the elitist self-adjusting $(1 + \lambda)$ EA on arbitrary fitness functions.

We brought everything together in Chapter 6 by analysing the self-adjusting $(1, \lambda)$ EA studied in Chapter 5 on the multimodal benchmark problem Cliff. Based on our results from Chapter 4 we implemented a reset mechanism for the adjustment of the offspring population size to help the self-adjusting $(1, \lambda)$ EA cope with local optima. We showed that the self-adjusting $(1, \lambda)$ EA using success-based rules and a reset mechanism can find the global optimum in $O(n)$ expected generations and $O(n \log n)$ expected evaluations. This is a speedup of order $\Omega(n^{2.9767})$ over the expected optimisation time with the best fixed value of λ.

The latter conclusion was obtained by refining the previous bounds on the expected optimisation time of the $(1, \lambda)$ EA on Cliff from [106], $O(e^{5\lambda}) = O(148.413^{\lambda})$ and $\min\{n^{n/4}, e^{\lambda/4}\}/3 = \min\{n^{n/4}, 1.284^{\lambda}\}/3$, towards bounds of $\Omega(\xi^{\lambda})$ and $O(\xi^{\lambda} \log n)$, for $\xi \approx 6.196878$, revealing the degree of the polynomial in the expected runtime of the $(1, \lambda)$ EA with the best fixed λ as $\eta \approx 3.976770136$.

Throughout this thesis we have implemented extensive empirical analyses complementing our theoretical results. Our empirical analyses show precise runtime results for the algorithms studied on concrete problem instances, detailed investigation of how the hyper-parameters introduced by the parameter control mechanisms affect the runtime of the algorithms and showcase how the mathematical results obtained translate to other more complex optimisation problems.

7.1 Future work

In this thesis, we have focused on the analysis of two algorithms that self-adjust the offspring population size: the self-adjusting $(1 + (\lambda, \lambda))$ GA and the self-adjusting $(1, \lambda)$ EA. Although this has resulted in a substantial amount of knowledge of how self-adjusting mechanisms work for multimodal optimisation, we study a limited number of algorithms. A possible target for future work would be analysing other algorithms using a parameter control mechanism to adjust the offspring population size λ. The perfect candidates are so-called population-based non-elitist EAs, because similar to the $(1, \lambda)$ EA, they require large values of offspring population size [33, 47, 123]. This requirement comes from the need to avoid losing fitness during *hard* parts of the optimisation, but is not always needed in *easy* parts of the optimisation. Therefore using a self-adjusting mechanism to adjust λ could make these algorithms more efficient.

From all population-based non-elitist EAs a natural extension to our work would be to study the self-adjusting (μ, λ) EA with $\mu > 1$. In this case we hypothesise that using a parent population might allow the algorithm to retain good solutions in the parent population resulting in an algorithm that is more robust with respect to the hyper-parameters of the self-adjusting mechanism. Although the final goal would be to theoretically analyse this algorithm, we believe that an exploratory experimental analysis beforehand would be helpful to guide such theoretical studies.

Another open problem is to find ways to make the self-adjusting $(1, \lambda)$ EA more robust with respect to to its hyper-parameters. In this thesis we have already shown a possible solution: using a mutation probability small enough to make every optimisation problem an every-where hard problem with respect to the self-adjusting $(1, \lambda)$ EA. Although this solves the issue, it also results in a loss of performance with respect to the standard mutation probability $1/n$ (with a proper selection of the hyper-parameters). As discussed above, using a parent population could also result in a more robust algorithm. Another possible solution related to this is to impose a minimum λ. This would prevent the algorithm from reducing the offspring population size to harmful values. In particular, we conjecture that for OneMax, a minimum $\lambda > 1$ (but constant with respect to the problem size n) would allow the algorithm to find a solution at a distance εn from the optimum and then the self-adjusting $(1, \lambda)$ EA would be able to optimise OneMax efficiently, for any constant value of its hyper-parameters.

This thesis has helped to narrow the gap between theory and practice. An important direction for future work that can further shrink this gap, is to study parameter control mechanisms on functions beyond the ones analysed here. Kaufmann, Larcher, Lengler, and Zou [115] have already expanded our analyses of the self-adjusting $(1, \lambda)$ EA to the class of monotone functions. Due to the range and importance of their associated applications, classical combinatorial optimisation problems are another interesting class of problems to study. Additionally, experimental studies could be implemented on real-world applications comparing the performance of the algorithms studied here against other commonly used randomised search heuristics.

Further down the line, we would like to explore the possibility of combining multiple parameter control mechanisms into one algorithm. For example, our promising results for the $(1, \lambda)$ EA self-adjusting λ can be combined with a self-adaptive mutation rate, which by itself has been shown to be helpful for non-elitist algorithms [25, 69]. Although analysing a combination of parameter control mechanisms is significantly more difficult, it is a promising direction for research, and we believe that more research effort should be put into this.

Bibliography

[1] Youhei Akimoto, Anne Auger, and Tobias Glasmachers. Drift theory in continuous search spaces: Expected hitting time of the (1 + 1)-ES with 1/5 success rule. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, page 801–808. ACM, 2018.

[2] Fawaz Alanazi and Per Kristian Lehre. Runtime analysis of selection hyper-heuristics with classical learning mechanisms. In *Proceedings of Congress on Evolutionary Computation*, CEC '14, pages 2515–2523. IEEE, 2014.

[3] Denis Antipov and Benjamin Doerr. Runtime analysis of a heavy-tailed $(1 + (\lambda, \lambda))$ genetic algorithm on Jump functions. In *Proceedings of Parallel Problem Solving from Nature – PPSN XVI*, pages 545–559. Springer, 2020.

[4] Denis Antipov and Benjamin Doerr. A tight runtime analysis for the $(\mu + \lambda)$ EA. *Algorithmica*, 83(4):1054–1095, 2021.

[5] Denis Antipov and Benjamin Doerr. Precise runtime analysis for plateau functions. *ACM Trans. Evol. Learn. Optim.*, 1(4), oct 2021.

[6] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. A tight runtime analysis for the $(1 + (\lambda, \lambda))$ GA on LeadingOnes. In *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, FOGA '19, page 169–182. ACM, 2019.

[7] Denis Antipov, Benjamin Doerr, and Quentin Yang. The efficiency threshold for the offspring population size of the (μ, λ) EA. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 1461–1469. ACM, 2019.

[8] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. Fast mutation in crossover-based algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '20, page 1268–1276. ACM, 2020.

[9] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. The $(1 + (\lambda, \lambda))$ GA is even faster on multimodal problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '20, page 1259–1267. ACM, 2020.

[10] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. Lazy parameter tuning and control: Choosing all parameters randomly from a power-law distribution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, page 1115–1123. ACM, 2021.

[11] Thomas Bäck. Self-adaptation in genetic algorithms. In *Proceedings of the First European Conference on Artificial Life*, pages 263–271, 1991.

[12] Thomas Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In *Proceedings of Parallel Problem Solving from Nature – PPSN II*, pages 85–94, 1992.

[13] Thomas Bäck and Martin Schütz. Intelligent mutation rate control in canonical genetic algorithms. In *International Symposium on Foundations of Intelligent Systems*, pages 158–167. Springer, 1996.

[14] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Handbook of Evolutionary Computation.* IOP Publishing Ltd., 1st edition, 1997.

[15] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. Unbiased black-box complexity of parallel search. In *Proceedings of Parallel Problem Solving from Nature – PPSN XIII*, pages 892–901. Springer, 2014.

[16] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. Black-box complexity of parallel search with distributed populations. In *Proc. of FOGA*, pages 3–15. ACM, 2015.

[17] Henry Bambury, Antoine Bultel, and Benjamin Doerr. Generalized jump functions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, page 1124–1132. ACM, 2021.

[18] Anton Bassin and Maxim Buzdalov. The 1/5-th rule with rollbacks: On self-adjustment of the population size in the $(1 + (\lambda, \lambda))$ GA. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, page 277–278. ACM, 2019.

[19] Riade Benbaki, Ziyad Benomar, and Benjamin Doerr. A rigorous runtime analysis of the 2-mmas$_{ib}$ on jump functions: Ant colony optimizers can cope well with local optima. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, page 4–13. ACM, 2021.

[20] Jakob Bossek and Dirk Sudholt. Do additional optima speed up evolutionary algorithms? In *Proceedings of the 16th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, FOGA '21. ACM, 2021.

[21] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Proceedings of Parallel Problem Solving from Nature – PPSN XI*, volume 6238, pages 1–10. Springer, 2010.

[22] Maxim Buzdalov and Benjamin Doerr. Runtime analysis of the $(1 + (\lambda, \lambda))$ genetic algorithm on random satisfiable 3-CNF formulas. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 1343–1350. ACM, 2017.

[23] Maxim Buzdalov and Carola Doerr. Optimal static mutation strength distributions for the $(1 + \lambda)$ evolutionary algorithm on OneMax. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, page 660–668. ACM, 2021.

[24] Maxim Buzdalov, Mikhail Kever, and Benjamin Doerr. Upper and lower bounds on unrestricted black-box complexity of $\text{Jump}_{n,\ell}$. In *Evolutionary Computation in Combinatorial Optimization*, pages 209–221. Springer, 2015.

[25] Brendan Case and Per Kristian Lehre. Self-adaptation in nonelitist evolutionary algorithms on discrete problems with unknown structure. *IEEE Transactions on Evolutionary Computation*, 24(4):650–663, 2020.

[26] J. Cervantes and Christopher R. Stephens. Limitations of existing mutation rate heuristics and how a rank GA overcomes them. *IEEE Transactions on Evolutionary Computation*, 13(2):369–397, 2009.

[27] T. Chen, J. He, G. Sun, G. Chen, and X. Yao. A new approach for analyzing average time complexity of population-based evolutionary algorithms on unimodal problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(5): 1092–1106, 2009.

[28] Francisco Chicano, Andrew M. Sutton, L. Darrell Whitley, and Enrique Alba. Fitness Probability Distribution of Bit-Flip Mutation. *Evolutionary Computation*, 23(2):217–248, 06 2015.

[29] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. The MIT Press, 3rd edition, 2009.

[30] Dogan Corus and Pietro S. Oliveto. Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 22(5):720–732, 2018.

[31] Dogan Corus and Pietro S. Oliveto. On the benefits of populations for the exploitation speed of standard steady-state genetic algorithms. *Algorithmica*, 82(12):3676–3706, 2020.

[32] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. On the runtime analysis of the Opt-IA artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 83–90. ACM, 2017.

[33] Dogan Corus, Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. Level-based analysis of genetic algorithms and other search processes. *IEEE Transactions on Evolutionary Computation*, 22(5):707–719, 2018.

[34] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. Fast artificial immune systems. In *Parallel Problem Solving from Nature – PPSN XV*, pages 67–78. Springer, 2018.

[35] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. On inversely proportional hypermutations with mutation potential. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 215–223. ACM, 2019.

[36] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. When hypermutations and ageing enable artificial immune systems to outperform evolutionary algorithms. *Theoretical Computer Science*, 832:166–185, 2020.

[37] Dogan Corus, Andrei Lissovoi, Pietro S. Oliveto, and Carsten Witt. On steady-state evolutionary algorithms and selective pressure: Why inverse rank-based allocation of reproductive trials is best. *ACM Trans. Evol. Learn. Optim.*, 1(1), apr 2021.

[38] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. Fast immune system-inspired hypermutation operators for combinatorial optimization. *IEEE Transactions on Evolutionary Computation*, 25(5):956–970, 2021.

[39] Duc-Cuong Dang and Per Kristian Lehre. Self-adaptation of mutation rates in non-elitist populations. In *Proceedings of Parallel Problem Solving from Nature – PPSN XIV*, pages 803–813. Springer, 2016.

[40] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima with diversity mechanisms and crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO' 16, pages 645–652. ACM, 2016.

[41] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation*, 22(3):484–497, 2018.

[42] Duc-Cuong Dang, Anton Eremeev, and Per Kristian Lehre. Escaping local optima with non-elitist evolutionary algorithms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12275–12283, 2021.

[43] Duc-Cuong Dang, Anton Eremeev, and Per Kristian Lehre. Non-elitist evolutionary algorithms excel in fitness landscapes with sparse deceptive regions and dense valleys. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, page 1133–1141. ACM, 2021.

[44] Nguyen Dang and Carola Doerr. Hyper-parameter tuning for the $(1 + (\lambda, \lambda))$ GA. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19. ACM, 2019.

[45] Kenneth De Jong. Parameter setting in EAs: a 30 year perspective. In Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 1–18. Springer, 2007.

[46] Luc Devroye. *The compound random search.* PhD thesis, Purdue University, 1972.

[47] Benjamin Doerr. Does comma selection help to cope with local optima? In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '20, page 1304–1313. ACM, 2020.

[48] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, Natural Computing Series, pages 1–87. Springer, 2020.

[49] Benjamin Doerr. The runtime of the compact genetic algorithm on jump functions. *Algorithmica*, 83(10):3059–3107, 2021.

[50] Benjamin Doerr and Carola Doerr. Optimal static and self-adjusting parameter choices for the $(1+(\lambda,\lambda))$ Genetic Algorithm. *Algorithmica*, 80(5):1658–1709, 2018.

[51] Benjamin Doerr and Carola Doerr. Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, Natural Computing Series, pages 271–321. Springer, 2020.

[52] Benjamin Doerr and Leslie Ann Goldberg. Adaptive drift analysis. *Algorithmica*, 65 (1):224–250, 2013.

[53] Benjamin Doerr and Timo Kötzing. Lower bounds from fitness levels made easy. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, page 1142–1150. ACM, 2021.

[54] Benjamin Doerr and Timo Kötzing. Multiplicative up-drift. *Algorithmica*, 83(10): 3017–3058, 2021.

[55] Benjamin Doerr and Marvin Künnemann. Optimizing linear functions with the (1+λ) evolutionary algorithm–different asymptotic runtimes for different instances. *Theoretical Computer Science*, 561:3–23, 2015.

[56] Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. Optimizing monotone functions can be difficult. In *Proceedings of Parallel Problem Solving from Nature – PPSN XI*, pages 42–51. Springer, 2010.

[57] Benjamin Doerr, Mahmoud Fouz, and Carsten Witt. Sharp bounds by probability-generating functions and variable drift. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '11, pages 2083–2090. ACM, 2011.

[58] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. *Algorithmica*, 64(4):673–697, 2012.

[59] Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. Mutation Rate Matters Even When Optimizing Monotonic Functions. *Evolutionary Computation*, 21(1):1–27, 03 2013.

[60] Benjamin Doerr, Carola Doerr, and Franziska Ebel. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87–104, 2015.

[61] Benjamin Doerr, Carola Doerr, and Jing Yang. k-bit mutation with self-adjusting k outperforms standard bit mutation. In *Proceedings of Parallel Problem Solving from Nature – PPSN XIV*, pages 824–834. Springer, 2016.

[62] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 777–784. ACM, 2017.

[63] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica*, 80(5):1732–1768, 2018.

[64] Benjamin Doerr, Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. On the runtime analysis of selection Hyper-Heuristics with adaptive learning periods. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, page 1015–1022. ACM, 2018.

[65] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Solving problems with unknown solution length at almost no extra cost. *Algorithmica*, 81(2):703–748, 2019.

[66] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. The (1+ λ) Evolutionary Algorithm with self-adjusting mutation rate. *Algorithmica*, 81(2):593–631, 2019.

[67] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. *Theoretical Computer Science*, 801:1–34, 2020.

[68] Benjamin Doerr, Carola Doerr, and Johannes Lengler. Self-adjusting mutation rates with provably optimal success rules. *Algorithmica*, 83(10):3108–3147, 2021.

[69] Benjamin Doerr, Carsten Witt, and Jing Yang. Runtime analysis for self-adaptive mutation rates. *Algorithmica*, 83(4):1012–1053, 2021.

[70] Carola Doerr. Complexity theory for discrete black-box optimization heuristics. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, Natural Computing Series, pages 271–321. Springer, 2020.

[71] Carola Doerr and Markus Wagner. On the effectiveness of simple success-based parameter selection mechanisms for two classical discrete black-box optimization benchmark problems. *CoRR*, abs/1803.01425, 2018. URL http://arxiv.org/abs/1803.01425.

[72] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *CoRR*, abs/1810.05281, 2018. URL http://arxiv.org/abs/1810.05281.

[73] Carola Doerr, Furong Ye, Sander van Rijn, Hao Wang, and Thomas Bäck. Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: Profiling $(1 + \lambda)$ EA variants on OneMax and LeadingOnes. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 951–958. ACM, 2018.

[74] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M. Shir, and Thomas Bäck. Benchmarking discrete optimization heuristics with IOHprofiler. *Applied Soft Computing*, 88:106027, 2020.

[75] Stefan Droste, Thomas Jansen, and Ingo Wegener. Dynamic parameter control in simple evolutionary algorithms. In *Proceedings of the 6th ACM/SIGEVO Workshop on Foundations of Genetic Algorithms*, pages 275–294. Morgan Kaufmann, 2000.

[76] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1):51–81, 2002.

[77] A. E. Eiben and J. Smith. *Introduction to evolutionary computing.* Springer, 2nd edition, 2015.

[78] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, 1999.

[79] A.E. Eiben and S.K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[80] Joseph Felsenstein. The evolutionary advantage of recombination. *Genetics*, 78:737–756, 1974.

[81] Terence C. Fogarty. Varying the probability of mutation in the genetic algorithm. In *ICGA*, 1989.

[82] Michael Foster, Matthew Hughes, George O'Brien, Pietro S. Oliveto, James Pyle, Dirk Sudholt, and James Williams. Do sophisticated evolutionary algorithms perform better than simple ones? In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '20, pages 184–192. ACM, 2020.

[83] Tobias Friedrich, Nils Hebbinghaus, and Frank Neumann. Comparison of simple diversity mechanisms on plateau functions. *Theoretical Computer Science*, 410(26): 2455–2462, 2009.

[84] Tobias Friedrich, Pietro S. Oliveto, Dirk Sudholt, and Carsten Witt. Analysis of diversity-preserving mechanisms for global exploration. *Evol. Comput.*, 17(4):455–476, 2009.

[85] Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Samadhi Nallaperuma, Frank Neumann, and Martin Schirneck. Fast building block assembly by majority vote crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '16, page 661–668. ACM, 2016.

[86] Tobias Friedrich, Francesco Quinzan, and Markus Wagner. Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, page 293–300. ACM, 2018.

[87] Josselin Garnier, Leila Kallel, and Marc Schoenauer. Rigorous Hitting Times for Binary Mutations. *Evolutionary Computation*, 7(2):173–203, 06 1999.

[88] Christian Gießen and Timo Kötzing. Robustness of populations in stochastic environments. *Algorithmica*, 75(3):462–489, 2016.

[89] Christian Gießen and Carsten Witt. The interplay of population size and mutation probability in the $(1 + \lambda)$ EA on OneMax. *Algorithmica*, 78(2):587–609, 2017.

[90] Christian Gießen and Carsten Witt. Optimal mutation rates for the $(1+\lambda)$ EA on OneMax through asymptotically tight drift analysis. *Algorithmica*, 80(5):1710–1731, 2018.

[91] Brian W. Goldman and William F. Punch. Fast and efficient black box optimization using the parameter-less population pyramid. *Evolutionary Computation*, 23(3):451–479, 2015.

[92] Walter J. Gutjahr. First steps to the runtime complexity analysis of ant colony optimization. *Computers and Operations Research*, 35(9):2711–2727, 2008.

[93] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. On the impact of the cutoff time on the performance of algorithm configurators. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, pages 907–915. ACM, 2019.

[94] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. Fast perturbative algorithm configurators. In *Proceedings of Parallel Problem Solving from Nature – PPSN XVI*, volume 12269, pages 19–32. Springer, 2020.

[95] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. Analysis of the performance of algorithm configurators for search heuristics with global mutation operators. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '20, pages 823–831. ACM, 2020.

[96] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. On the impact of the performance metric on efficient algorithm configuration. *Artificial Intelligence*, 303:103629, 2022.

[97] Václav Hasenöhrl and Andrew M. Sutton. On the runtime dynamics of the compact genetic algorithm on Jump functions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, page 967–974. ACM, 2018.

[98] Jun He and Xin Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, 2004.

[99] Mario A. Hevia Fajardo. An empirical evaluation of success-based parameter control mechanisms for evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO' 19, pages 787–795. ACM, 2019.

[100] Mario Alejandro Hevia Fajardo and Dirk Sudholt. On the choice of the parameter control mechanism in the $(1 + (\lambda, \lambda))$ genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO' 20, page 832–840. ACM, 2020.

[101] Mario Alejandro Hevia Fajardo and Dirk Sudholt. Self-adjusting population sizes for non-elitist Evolutionary Algorithms: Why success rates matter. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, page 1151–1159. ACM, 2021.

[102] Mario Alejandro Hevia Fajardo and Dirk Sudholt. Self-adjusting offspring population sizes outperform fixed parameters on the Cliff function. In *Proceedings of the 16th Workshop on Foundations of Genetic Algorithms*, FOGA '21, pages 5:1–5:15. ACM, 2021.

[103] Mario Alejandro Hevia Fajardo and Dirk Sudholt. Hard problems are easier for success-based parameter control. Submitted to: *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, ACM, 2022.

[104] Mario Alejandro Hevia Fajardo and Dirk Sudholt. On the choice of the parameter control mechanism in the $(1 + (\lambda, \lambda))$ Genetic Algorithm. Submitted to: *ACM Transactions on Evolutionary Learning and Optimization*, 2022.

[105] Mario Alejandro Hevia Fajardo and Dirk Sudholt. Self-adjusting population sizes for non-elitist Evolutionary Algorithms. Submitted to: *Algorithmica*, 2022.

[106] Jens Jägersküpper and Tobias Storch. When the plus strategy outperforms the comma strategy and when not. In *Proceedings of the IEEE Symposium on Foundations of Computational Intelligence, FOCI 2007*, pages 25–32. IEEE, 2007.

[107] Thomas Jansen. *Analyzing evolutionary algorithms: the computer science perspective.* Natural computing series. Springer, 2013.

[108] Thomas Jansen and Dirk Sudholt. Analysis of an asymmetric mutation operator. *Evolutionary Computation*, 18(1):1–26, 2010.

[109] Thomas Jansen and Ingo Wegener. On the choice of the mutation probability for the (1+1) EA. In *Proceedings of Parallel Problem Solving from Nature – PPSN VI*, pages 89–98. Springer, 2000.

[110] Thomas Jansen and Ingo Wegener. On the utility of populations in evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'01)*, pages 1034–1041. Morgan Kaufmann Publishers Inc., 2001.

[111] Thomas Jansen and Ingo Wegener. The analysis of evolutionary algorithms—a proof that crossover really can help. *Algorithmica*, 34(1):47–66, 2002.

[112] Thomas Jansen and Ingo Wegener. On the analysis of a dynamic evolutionary algorithm. *Journal of Discrete Algorithms*, 4(1):181–199, 2006.

[113] Thomas Jansen and Christine Zarges. Example landscapes to support analysis of multimodal optimisation. In *Proceedings of Parallel Problem Solving from Nature – PPSN XIV*, pages 792–802. Springer, 2016.

[114] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13(4):413–440, 2005.

[115] Marc Kaufmann, Maxime Larcher, Johannes Lengler, and Xun Zou. Self-adjusting population sizes for the (1, (λ)-EA on monotone functions. In Günter Rudolph, Anna V. Kononova, Hernán E. Aguirre, Pascal Kerschke, Gabriela Ochoa, and Tea Tusar, editors, *Proceedings of Parallel Problem Solving from Nature – PPSN XVII*, volume 13399 of *Lecture Notes in Computer Science*, pages 569–585. Springer, 2022.

[116] Stefan Kern, Sibylle D. Müller, Nikolaus Hansen, Dirk Büche, Jiri Ocenasek, and Petros Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms – a comparative review. *Natural Computing*, 3(1):77–112, 2004.

[117] Timo Kötzing and Martin S. Krejca. First-hitting times under drift. *Theoretical Computer Science*, 796:51–69, 2019.

[118] Timo Kötzing, Dirk Sudholt, and Madeleine Theile. How crossover helps in pseudo-boolean optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '11, pages 989–996. ACM, 2011.

[119] Lázló Kozma. Useful inequalities, 2019. http://www.lkozma.net/inequalities_cheat_sheet/.

[120] Jörg Lässig and Dirk Sudholt. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proceedings of the 11th ACM/SIGEVO Workshop on Foundations of Genetic Algorithms*, FOGA '11, pages 181–192. ACM, 2011.

[121] Jörg Lässig and Dirk Sudholt. General upper bounds on the running time of parallel evolutionary algorithms. *Evolutionary Computation*, 22(3):405–437, 2014.

[122] Jörg Lässig and Dirk Sudholt. Analysis of speedups in parallel evolutionary algorithms and (1+λ) EAs for combinatorial optimization. *Theoretical Computer Science*, 551: 66–83, 2014.

[123] Per Kristian Lehre. Negative drift in populations. In *Parallel Problem Solving from Nature – PPSN XI*, pages 244–253. Springer, 2010.

[124] Per Kristian Lehre. Fitness-levels for non-elitist populations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '11, page 2075–2082. ACM, 2011.

[125] Per Kristian Lehre and Pietro S. Oliveto. Theoretical analysis of stochastic search algorithms. In Rafael Martí, Pardalos Panos, and Mauricio G. C. Resende, editors, *Handbook of Heuristics*, pages 1–36. Springer, 2018.

[126] Per Kristian Lehre and Ender Özcan. A runtime analysis of simple hyper-heuristics: To mix or not to mix operators. In *Proceedings of the 12th ACM/SIGEVO Workshop on Foundations of Genetic Algorithms*, FOGA '13, pages 97–104. ACM, 2013.

[127] Per Kristian Lehre and Dirk Sudholt. Parallel black-box complexity with tail bounds. *IEEE Transactions on Evolutionary Computation*, 24(6):1010–1024, 2020.

[128] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. *Algorithmica*, 64(4):623–642, 2012.

[129] Per Kristian Lehre and Xin Yao. On the impact of the mutation-selection balance on the runtime of evolutionary algorithms. In *Proceedings of the 10th ACM/SIGEVO Workshop on Foundations of Genetic Algorithms*, FOGA '09, pages 47–58. ACM, 2009.

[130] Johannes Lengler. A general dichotomy of evolutionary algorithms on monotone functions. *IEEE Transactions on Evolutionary Computation*, 24(6):995–1009, 2020.

[131] Johannes Lengler. Drift analysis. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, Natural Computing Series, pages 89–131. Springer, 2020.

[132] Johannes Lengler and Simone Riedi. Runtime analysis of the (μ+1)-EA on the dynamic BinVal function. In *Evolutionary Computation in Combinatorial Optimization*, pages 84–99, Cham, 2021. Springer.

[133] Johannes Lengler and Angelika Steger. Drift analysis and evolutionary algorithms revisited. *Combinatorics, Probability and Computing*, 27(4):643–666, 2018.

[134] Johannes Lengler and Xun Zou. Exponential slowdown for larger populations: The (μ+1)-EA on monotone functions. *Theoretical Computer Science*, 875:28–51, 2021.

[135] Johannes Lengler, Dirk Sudholt, and Carsten Witt. The complex parameter landscape of the compact genetic algorithm. *Algorithmica*, 83(4):1096–1137, 2021.

[136] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. On the runtime analysis of generalised selection hyper-heuristics for pseudo-boolean optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 849–856. ACM, 2017.

[137] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. On the time complexity of algorithm selection hyper-heuristics for multimodal optimisation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2322–2329, 2019.

[138] Andrei Lissovoi, Pietro Oliveto, and John Alasdair Warwicker. How the duration of the learning period affects the performance of random gradient selection hyper-heuristics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2376–2383, Apr. 2020.

[139] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. Simple hyper-heuristics control the neighbourhood size of randomised local search optimally for LeadingOnes. *Evolutionary Computation*, 28(3):437–461, 2020.

[140] Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[141] Andrea Mambrini and Dirk Sudholt. Design and analysis of schemes for adapting migration intervals in parallel evolutionary algorithms. *Evolutionary Computation*, 23 (4):559–582, 2015.

[142] Daiki Morinaga and Youhei Akimoto. Generalized drift analysis in continuous domain: Linear convergence of (1+1)-ES on strongly convex functions with lipschitz continuous gradients. In *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, FOGA '19, page 13–24. ACM, 2019.

[143] Daiki Morinaga, Kazuto Fukuchi, Jun Sakuma, and Youhei Akimoto. Convergence rate of the (1+1)-evolution strategy with success-based step-size adaptation on convex quadratic functions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '21, page 1169–1177. ACM, 2021.

[144] Frank Neumann and Ingo Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '04, pages 713–724. Springer, 2004.

[145] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization.* Natural Computing Series. Springer, 2010.

[146] Frank Neumann, Dirk Sudholt, and Carsten Witt. Analysis of different MMAS ACO algorithms on unimodal functions and plateaus. *Swarm Intelligence*, 3(1):35–68, 2009.

[147] Pietro S. Oliveto and C. Witt. Erratum: Simplified drift analysis for proving lower bounds in evolutionary computation. *CoRR*, abs/1211.7184, 2012. URL http://arxiv.org/abs/1211.7184.

[148] Pietro S. Oliveto and Carsten Witt. Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica*, 59(3):369–386, 2011.

[149] Pietro S. Oliveto and Carsten Witt. Improved time complexity analysis of the simple genetic algorithm. *Theoretical Computer Science*, 605:21–41, 2015.

[150] Pietro S. Oliveto and Xin Yao. Runtime analysis of evolutionary algorithms for discrete optimization. In Anne Auger and Benjamin Doerr, editors, *Theory of Randomized Search Heuristics*, volume 1 of *Theoretical Computer Science*, pages 21–52. World Scientific, 2011.

[151] Pietro S. Oliveto, Jun He, and Xin Yao. Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4(3):281–293, 2007.

[152] Pietro S. Oliveto, Per Kristian Lehre, and Frank Neumann. Theoretical analysis of rank-based mutation - combining exploration and exploitation. In *Proceedings of the Congress on Evolutionary Computation*, CEC '09, pages 1455–1462. IEEE, 2009.

[153] Tiago Paixão, Jorge Pérez Heredia, Dirk Sudholt, and Barbora Trubenová. Towards a runtime comparison of natural and artificial evolution. *Algorithmica*, 78(2):681–713, 2017.

[154] Eduardo Carvalho Pinto and Carola Doerr. Towards a more practice-aware runtime analysis of evolutionary algorithms. *CoRR*, abs/1812.00493, 2018.

[155] Mike Preuss. *Multimodal Optimization by Means of Evolutionary Algorithms.* Natural Computing Series. Springer, 2015.

[156] Yasha Pushak and Holger Hoos. Algorithm configuration landscapes: More benign than expected? In *Parallel Problem Solving from Nature – PPSN XV*, pages 271–283. Springer, 2018.

[157] Yasha Pushak and Holger H. Hoos. Golden parameter search: Exploiting structure to quickly configure parameters in parallel. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '20, page 245–253. ACM, 2020.

[158] Chao Qian, Ke Tang, and Zhi-Hua Zhou. Selection hyper-heuristics can provably be helpful in evolutionary multi-objective optimization. In *Proceedings of Parallel Problem Solving from Nature – PPSN XIV*, pages 835–846. Springer, 2016.

[159] Francesco Quinzan, Andreas Göbel, Markus Wagner, and Tobias Friedrich. Evolutionary algorithms and submodular functions: benefits of heavy-tailed mutations. *Natural Computing*, 20(3):561–575, 2021.

[160] Amirhossein Rajabi and Carsten Witt. Evolutionary algorithms with self-adjusting asymmetric mutation. In *Proceedings of Parallel Problem Solving from Nature – PPSN XVI*, pages 664–677. Springer, 2020.

[161] Amirhossein Rajabi and Carsten Witt. Self-adjusting evolutionary algorithms for multimodal optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '20, page 1314–1322. ACM, 2020.

[162] Amirhossein Rajabi and Carsten Witt. Stagnation detection with randomized local search. In *Evolutionary Computation in Combinatorial Optimization*, pages 152–168. Springer, 2021.

[163] Ingo Rechenberg. *Evolutionsstrategie*. Frommann-Holzboog-Verlag, 1973.

[164] Anna Rodionova, Kirill Antonov, Arina Buzdalova, and Carola Doerr. Offspring population size matters when comparing evolutionary algorithms with self-adjusting mutation rates. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 855–863. ACM, 2019.

[165] Jonathan E. Rowe and Aishwaryaprajna. The benefits and limitations of voting mechanisms in evolutionary optimisation. In *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, FOGA '19, page 34–42. ACM, 2019.

[166] Jonathan E. Rowe and Dirk Sudholt. The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm. *Theoretical Computer Science*, 545:20–38, 2014.

[167] M. Schumer and K. Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13(3):270–276, 1968.

[168] Ramteen Sioshansi and Antonio J. Conejo. Optimization is ubiquitous. In *Optimization in Engineering: Models and Algorithms*, pages 1–16. Springer, 2017.

[169] Giovanni Squillero and Alberto Tonda. Divergence of character and premature convergence: A survey of methodologies for promoting diversity in evolutionary optimization. *Information Sciences*, 329:782–799, 2016. Special issue on Discovery Science.

[170] Tobias Storch and Ingo Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320(1):123–134, 2004.

[171] Dirk Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17(3):418–435, 2013.

[172] Dirk Sudholt. Parallel evolutionary algorithms. In Janusz Kacprzyk and Witold Pedrycz, editors, *Springer Handbook of Computational Intelligence*, pages 929–959. Springer, 2015.

[173] Dirk Sudholt. How crossover speeds up building block assembly in genetic algorithms. *Evolutionary Computation*, 25(2):237–274, 2017.

[174] Dirk Sudholt. The benefits of population diversity in evolutionary algorithms: A survey of rigorous runtime analyses. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, Natural Computing Series, pages 359–404. Springer, 2020.

[175] Dirk Sudholt and Carsten Witt. Runtime analysis of a binary particle swarm optimizer. *Theoretical Computer Science*, 411(21):2084–2100, 2010.

[176] Dirk Thierens. Adaptive mutation rate control schemes in genetic algorithms. In *Proceedings of the Congress on Evolutionary Computation*, volume 1 of *CEC '02*, pages 980–985 vol.1. IEEE, 2002.

[177] Abraham Wald. On cumulative sums of random variables. *The Annals of Mathematical Statistics*, 15(3):283 – 296, 1944.

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

[179] Darrell Whitley, Swetha Varadarajan, Rachel Hirsch, and Anirban Mukhopadhyay. Exploration and exploitation without mutation: Solving the jump function in $\theta(n)$ time. In *Parallel Problem Solving from Nature – PPSN XV*, pages 55–66. Springer, 2018.

[180] Carsten Witt. Runtime analysis of the $(\mu+1)$ EA on simple pseudo-boolean functions. *Evolutionary Computation*, 14(1):65–86, 2006.

[181] Carsten Witt. Population size versus runtime of a simple evolutionary algorithm. *Theoretical Computer Science*, 403(1):104–120, 2008.

[182] Carsten Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing*, 22(2):294–318, 2013.

[183] Carsten Witt. Fitness levels with tail bounds for the analysis of randomized search heuristics. *Information Processing Letters*, 114(1–2):38–41, 2014.

[184] Furong Ye, Carola Doerr, and Thomas Bäck. Interpolating local and global search by controlling the variance of standard bit mutation. In *Proceedings of the Congress on Evolutionary Computation*, CEC '19, pages 2292–2299. IEEE, 2019.

[185] Christine Zarges. Rigorous runtime analysis of inversely fitness proportional mutation rates. In Günter Rudolph, Thomas Jansen, Nicola Beume, Simon Lucas, and Carlo Poloni, editors, *Parallel Problem Solving from Nature – PPSN X*, pages 112–122. Springer, 2008.