DISS. ETH NO. 29124

TOWARDS UNDERSTANDING BIO-INSPIRED COMPUTING FOR OPTIMIZATION AND LEARNING

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

XUN ZOU
M. sc. ETH

born on 20.04.1993

accepted on the recommendation of

Prof. Dr. Angelika Steger, examiner
Dr. Johannes Lengler, co-examiner
Prof. Dr. Dirk Sudholt, co-examiner

2023

Xun Zou: *TOWARDS UNDERSTANDING BIO-INSPIRED COMPUTING FOR OPTIMIZATION AND LEARNING*, © 2023

DOI: 10.3929/ethz-b-000614107

ABSTRACT

The human brain and the evolutionary process (that created the human brain) are two powerful general problem solvers from nature. The attempt to understand and design algorithms based on them has led to the fields of neuroscience and evolutionary computing. This thesis contributes to these two areas by analyzing evolutionary algorithms and bio-inspired neural networks from a theoretical and empirical perspective.

Concretely, we contribute to the understanding of the two bio-inspired methods with three separate projects. First, we study the effect of population size in a mutation-based evolutionary algorithm on optimizing pseudo-Boolean functions. We find that for some monotone functions, increasing the population size by just a constant can have devastating impacts on performance. This is in stark contrast to many other benchmark functions on which a larger population size leads to either positive or neutral effects. Moreover, we show that large population sizes only cause troubles far away from the optimum, which is counter-intuitive since usually optimization gets harder as we approach the optimum.

Next, we consider a dynamic parameter control mechanism in controlling the offspring population size of a non-elitist evolutionary algorithm. Previous work has shown that the mechanism can run into problems if the fitness landscape is too easy, and it is conjectured that the easiest benchmark function, OneMax, suffers the most from this issue. However, we show that there are other functions for which the problem is more severe than for OneMax, thus disproving the conjecture.

Lastly, we model a sparse neural network with unstructured connections in the olfactory system of Drosophila. We show that despite the constraints imposed by the biological system, with a bio-plausible mechanism of global inhibition, the network can be a close approximation to a powerful machine learning model. We further investigate the effect of sparsity and find that the network can achieve a good balance between learning and noise resistance by employing sparse connections.

ZUSAMMENFASSUNG

Das menschliche Gehirn und der Prozess der Evolution (, welcher das menschliche Gehirn erschaffen hat,) sind zwei mächtige Allzweck-Problemlöser aus der Natur. Der Versuch, auf diesen basierende Algorithmen zu verstehen und zu entwerfen, hat die Gebiete der Neurowissenschaften und des evolutionären Rechnens hervorgebracht. Die vorliegende Arbeit leistet einen Beitrag zu diesen Gebieten mit Analysen von evolutionären Algorithmen und biologisch inspirierten neuronalen Netzen aus theoretischer und empirischer Perspektive.

Konkret tragen wir zum Verständnis der beiden biologisch inspirierten Methoden mit drei separaten Projekten bei. Erstens untersuchen wir den Effekt der Populationsgröße in einem mutationsbasierten evolutionären Algorithmus auf die Optimierung pseudo-boolescher Funktionen. Als Hauptbefund stellen wir fest, dass für einige monotone Funktionen bereits die Erhöhung der Populationsgröße um eine Konstante verheerende Auswirkungen auf die Leistung haben kann. Dies steht in starkem Gegensatz zu vielen anderen Benchmark-Funktionen bei denen eine erhöhte Populationsgröße entweder positive oder neutrale Auswirkungen zeitigt. Weiter zeigen wir, dass große Populationsgrößen nur Probleme in grossem Abstand zum Optimum verursachen. Dieser Befund ist kontraintuitiv, da die Optimierung normalerweise schwieriger wird, je mehr wir uns dem Optimum nähern.

Als Nächstes betrachten wir einen dynamischen Parameterkontrollmechanismus eines nicht-elitären evolutionären Algorithmus bei der Steuerung der Populationsgröße der Nachkommen. Frühere Forschung hat gezeigt, dass der Mechanismus auf Probleme stoßen kann, wenn die Fitnesslandschaft zu einfach ist, und es wird vermutet, dass die einfachste Benchmark Funktion, OneMax, am meisten unter diesem Problem leidet. Wir zeigen jedoch, dass dass es andere Funktionen gibt, bei denen das Problem gravierender ist als bei OneMax, wodurch diese Vermutung widerlegt wird.

Schließlich modellieren wir ein spärlich verbundenes neuronales Netz mit unstrukturierten Verbindungen im olfaktorischen System von Drosophila. Wir zeigen, dass trotz der durch das biologische System auferlegten Beschränkungen mit einem biologisch plausiblen Mechanismus der globaler Inhibition das Netzwerk ein mächtiges Machine Learning Modell gut approximieren kann. Wir untersuchen außerdem die Auswirkungen der

geringen Verbindungsdichte und stellen fest, dass das Netzwerk ein gutes Gleichgewicht zwischen Lernen und Rauschresistenz erreichen kann indem es spärliche Verbindungen verwendet.

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Angelika Steger. As the group leader, you brought together such a nice group of talented individuals from various backgrounds. It was my great pleasure to be part of it, and I appreciated the freedom to explore a wide range of research topics. As a scientist, you never stop being curious and coming up with new ideas. I enjoyed a lot the many discussions that we had on neuroscience and machine learning.

Next, I would like to thank my second supervisor, Johannes Lengler. You brought me into the research of evolutionary algorithms, and it was such a nice experience to collaborate with you. I was always impressed by the depth of your domain knowledge, the clearness of your reasoning, and the elegance in your arguments.

I would also like to thank Dirk Sudholt for devoting his time to examining and giving feedback on the thesis.

Moreover, I would like to thank all my colleagues at ETH, especially my co-authors. I will not forget the fun time that we spent together at work and in life.

Last but not least, I thank my family and friends for their constant care and support.

CONTENTS

1 INTRODUCTION

The richness I achieve comes from nature, the source of my inspiration.

— Claude Monet

1.1 THE TWO MOST POWERFUL NATURAL PROBLEM SOLVERS

Nature has always been a great source of inspiration, especially in the field of bionics, i. e. the application of biological methods and systems to the development of engineering systems and technology. In addition to its success in areas such as engineering and material science, bionics has also motivated the design of novel algorithms in computer science. In particular, nature has created two algorithms, which are arguably the two most powerful solvers for general problems. One of them is the human brain, and the other is the evolutionary process that created the human brain [1].

During billions of years, the power of evolution is unquestionable as it has produced a huge variety of life forms that have well adapted to a wide range of environments. The primary forces behind this progress are, by Darwinian theory [2], phenotypic variations maintained by the population of species through mechanisms like mutations, and the survival of individuals with the fittest combination of phenotypic traits, thus passing their genes to the next generation. In the field of evolutionary computing, this view has been translated into a stochastic trial-and-error style problem-solving process, which gives rise to a rich family of randomized search heuristics and applications in various optimization problems. What makes these algorithms particularly attractive is that they often come as off-the-shelf tools and can be applied with little domain expertise. Moreover, with the help of the powerful processing capabilities of modern computing architectures, they come also with the promise to find reasonably good solutions within an acceptable time [3].

On the other hand, the human brain is arguably the best solution to an ever-changing environment found by evolution so far. The desire to understand our brain and reproduce its intelligent behaviors has led to the fields of neuroscience and artificial intelligence. One of the most significant

discoveries in neuroscience is that all brains (including the human brain) consist of the same building blocks: neurons and synapses. Therefore, there has also been a large body of work conducted on simpler and smaller brains like the ones of rodents and insects, since they are much easier to study and understand from an experimental point of view. Recent advances in this direction include the publication of a connectome of the Drosophila central brain [4] and how it tracks allocentric traveling directions by performing vector arithmetic [5]. On the other hand, with the dramatic development of machine learning methods in the last decade, there has been a growing interest to model brain circuits and to explain neural recordings with artificial neural networks [6]. The fact that certain brain activities can be replicated from machine learning models confirms the efficacy of the brain, providing a deeper understanding of the working principles of the respective brain areas as well as the nature of the computational models.

This thesis contributes to the fields of evolutionary computation and computational neuroscience by analyzing and simulating evolutionary algorithms for optimization and neural circuits for learning. The rest of this section is organized as follows. In the next two subsections, 1.2 and 1.3, we introduce background information and motivation for the projects included in this thesis. Subsection 1.4 includes a summary for each individual project. In the end, we discuss common themes and methods that appeared in all the projects in Subsection 1.5.

1.2 ON THE THEORY OF EVOLUTIONARY COMPUTATION

The theoretical analysis of evolutionary algorithms is much younger than their use. In fact, the usage of natural-inspired search heuristics dates back to the 1960s, while rigorous analysis with proven performance guarantees only started in the late 1990s [7]. This is not surprising, as a typical application scenario is to apply them to a complex optimization problem in the hope of getting a satisfactory solution within an acceptable time. That being said, the formal analysis of the algorithm is typically not part of the process, while being easy to use and applicable to a wide range of problems is more important. However, despite their popularity in practice, the lack of sound theoretical guarantees for evolutionary algorithms has encumbered their acceptance to certain scientific communities [8].

The theory of evolutionary computation aims at understanding when these algorithms work, as well as when they do not work and why. To be able to find excellent solutions to a wide range of problems, evolutionary

algorithms usually come with a set of parameters. Therefore, a large part of studies has been devoted to questions such as [9]: How do changes in parameters affect performance? Are there optimal parameter settings for a particular class of fitness landscapes? Is it desirable to change parameter values dynamically during the optimization process? The answers to these questions can help to guide parameter choices in practice, and even suggest new building blocks and mechanisms to improve performance.

1.2.1 *Benchmark Functions and Evolutionary Algorithms*

Before coming to how the study of parameters can advance our understanding of evolutionary algorithms, it is worth mentioning a feature that distinguishes them from classical randomized algorithms. Since evolutionary algorithms are general randomized search heuristics that neither make explicit assumptions nor exploit the properties of the target problem, there is no doubt that they tend to be extremely difficult to analyze [3]. As a result, it is common practice to study them in a slightly simplified setting. This often allows us to have an intuitive idea of how the interested algorithm will perform on the objective, which is instructive to either prove its efficiency or to find out why and where it fails.

In the context of this thesis, we study the performance of evolutionary algorithms on monotone pseudo-Boolean functions. A pseudo-Boolean function is a function of the form $f : \{0,1\}^n \to \mathbb{R}$, i. e. it maps any bitstring of length n to a real value. If flipping a zero-bit into a one-bit always improves the fitness for any bitstring, i. e. the value of f evaluated on the bitstring, we call f a monotone function. From now on, we will refer to monotone Pseudo-Boolean functions by monotone functions for simplicity.

Monotone functions are relatively easy benchmark functions, as they always have a unique and global optimum at the all-ones bitstring. Moreover, from every search point (bitstring) there are short, fitness-increasing paths to the optimum, by flipping zero-bits to one-bits. At this point, one might doubt whether it makes sense to study monotone functions due to their easiness. Actually, despite that many algorithms are able to optimize every monotone function in polynomial time with respect to n, there are a surprising number of evolutionary algorithms that need exponential time to optimize some monotone functions. Furthermore, we will see later that, when a dynamic parameter control mechanism is introduced to evolutionary algorithms, easy fitness landscapes may cause problems while harder ones do not. We will discuss this in more detail in the following subsections.

Having specified the benchmark functions, we are ready to introduce the evolutionary functions (EAs) of interest. We will work with two classical evolutionary algorithms, i. e. the $(\mu+1)$-EA and the $(1,\lambda)$-EA. The former maintains a population of μ randomly initialized search points. Each iteration of it consists of two phases, mutation and selection. In each round, an offspring is generated by flipping each bit of a randomly chosen search point independently with probability c/n, where c is the mutation parameter. The offspring is then added to the population, and a search point with the least fitness value (breaking ties randomly) is excluded such that the population size remains μ. In contrast to the $(\mu+1)$-EA, the $(1,\lambda)$-EA keeps only one search point. In each generation, it generates λ offspring by flipping each bit of this search point independently with probability c/n, and it replaces the current search point with the fittest offspring, breaking ties randomly. Note that there is no selection between the parent and the fittest offspring, i. e. the parent is replaced even if it has a larger fitness value. As a result, $(1,\lambda)$-EA is a non-elitist algorithm, thus a comma is used in its notation instead of a plus. The pseudo-codes of the $(\mu+1)$-EA and the $(1,\lambda)$-EA can be found in Section 2.2.2 and 3.2.1 respectively.

1.2.2 *The impact of parameters*

As we can see, the basic parameters of evolutionary algorithms include the mutation rate c, the population size μ, and the offspring population size λ. The parameters can have a huge impact on their performance and finding appropriate parameter configurations is a non-trivial task. Due to their dependencies on both the problem and the state of optimization, determining optimal parameter values can be already very complex for a single parameter [10]. The mutation rate c is such an example. It was shown that when $c < 1$ the $(1+1)$-EA[1] finds the optimum of every monotone function in polynomial time, but when $c > 16$ there are monotone functions for which the $(1+1)$-EA needs exponential time [11, 12]. That is, a constant factor change in the parameters changes the run-time from polynomial to exponential. The inefficient range of c is improved from $c > 16$ to $c > c_0 \approx 2.13$ in [13] by the construction of a class of hard monotone functions termed HotTopic[2]. Furthermore, it was shown that for HotTopic functions, c_0 is a sharp performance threshold for a manifold of evolutionary

1 This is equivalent to the $(\mu+1)$-EA with $\mu = 1$.
2 The definition can be found in Section 2.2.3.

functions including the $(1+1)$-EA, the $(1+\lambda)$-EA[3], and the $(\mu+1)$-EA[4], i. e. $c < c_0$ leads to polynomial complexity while $c > c_0$ result in exponential run-time [14, 15]. Therefore, it is beneficial for many evolutionary algorithms to set the mutation rate c conservatively.

When there are more parameters in the algorithm, the interactions between the parameters can make the optimization process more complicated. For example, both theoretical and practical studies have shown that introducing population size to randomized search heuristics tends to bring positive or neutral effects [16, 17][5]. However, we will show in Chapter 2 that a large constant population can result in an exponential run-time for the $(\mu+1)$-EA on HotTopic functions even when the mutation rate is set very conservatively, i. e. $c < 1$. Moreover, contrary to the intuition that the hardest region for optimization is close to the optimum, we find that large populations only cause trouble before coming close to the optimum in our analysis. Our work thus complements the study on the $(\mu+1)$-EA in [14, 15], which focuses on the optimization progress close to the optimum.

1.2.3 *Dynamic parameter control*

The optimal parameter values may change as the optimization process proceeds, thus static parameter settings often lead to sub-optimal performance [10]. The goal of parameter control is to find suitable parameter settings throughout the optimization process. In continuous optimization, parameter control is essential for ensuring good convergence and obtaining high-quality solutions. One of the most famous examples is the $(1:s+1)$-rule for step size adaptation [20–23], where setting $s = 4$ lead to the widely used one-fifth rule. By dynamically updating the step size, the rule aims to maintain a constant probability $1/(s+1)$ of finding improvements throughout the optimization process. Recently, the $(1:s+1)$-rule has been extended to parameters in discrete domains, such as to the mutation rate [24, 25] and to offspring population size [26, 27] in evolutionary algorithms. In this thesis we will focus on the second case, i. e. using the $(1:s+1)$-rule to control the offspring population size λ in the $(1,\lambda)$-EA.

Since improving on the fitness function is usually much harder close to the optimum, the $(1,\lambda)$-EA can benefit from having a small λ at early stages of optimization to save computation, and having larger values of λ to

3 The elitist version of $(1,\lambda)$-EA, i. e. the parent is not replaced if it is fitter than all offspring.
4 Only for HotTopic functions that are hard to optimize close to the optimum, see Section 2.1.2 for a detailed discussion.
5 However, artificial counterexamples do exist [18, 19].

find fitness improvements more efficiently when approaching the optimum. The $(1 : s + 1)$-rule comes with two meta parameters, the success rate s and the update strength F. When the fittest offspring is fitter than the parent, λ is divided by F. When it is the other way around, λ is multiplied by $F^{1/s}$ to increase the chance of getting fitter offspring. We refer to the resulting algorithm by SA-$(1, \lambda)$-EA for short. By adjusting the value of λ, the SA-$(1, \lambda)$-EA tries to maintain a constant probability $1/(1 + s)$ of finding a fitter offspring than the parent in each generation.

Since the offspring population size is controlled by the $(1 : s + 1)$-rule, the parameter of interest is therefore shifted to the meta parameter s. It was shown in [26, 27] that optimizing OneMax, a function that counts the number of one-bits in the input bitstring, with the SA-$(1, \lambda)$-EA is efficient for $s < 1$, while for $s > 18$ it fails completely. In addition, simulations showed that the phase transition actually happens around $s \approx 3.4$, which suggests that the one-fifth rule is not a desirable choice in this case. This result on OneMax is then extended to all monotone functions in [28, 29], i. e. there are universal thresholds $s_1 > s_0 > 0$ such that the SA-$(1, \lambda)$-EA is efficient on every monotone functions when $s < s_0$, while it fails to find the optimum of every monotone functions in polynomial time when $s > s_1$. Counter-intuitively, the issue brought by large values of s only happens at places where the fitness landscape is too easy. The reason is that successful generations decrease λ significantly, while unsuccessful generations increase λ only mildly. Therefore, it was conjectured in [26, 27] that the SA-$(1, \lambda)$-EA suffers most from this issue on OneMax as it is the easiest function with a unique optimum for the $(1 + 1)$-EA [30–32].

In Chapter 3 we show that there are other functions that are more problematic than OneMax for the SA-$(1, \lambda)$-EA. Moreover, we try to distinguish two types of "easiness" of a benchmark function. The first type is related to how much progress an elitist hillclimber like $(1 + 1)$-EA can make on that function, and the second type concerns how likely a mutation produces an offspring that is fitter than the parent. OneMax is the easiest benchmark with respect to the first type, while for the SA-$(1, \lambda)$-EA the second type is relevant.

1.3 LEARNING FROM NEUROSCIENCE

The brain is so far the only proof of concept that intelligence can emerge from the interaction among an ensemble of simple computing units, i. e. neurons. Artificial neural networks, which are originally inspired by the

brain, have achieved significant breakthroughs in the last decade. The most recent examples include DALL·E 2, an AI system that can generate realistic or artistic images from descriptions in natural language [33], and ChatGPT, a language model that supports conversational interactions. It is worth noting that, although these models can be accessed with natural languages, a brain-friendly way of interaction, their working principles are already quite different from our brains in a number of ways. In this section, we provide two examples to illustrate that the two fields can benefit by learning from each other despite their differences.

The solutions to certain problems discovered by the brain and neural networks share a surprising similarity. As an example, grid cells were discovered in the entorhinal cortex of rats in 2005, and they were believed to play a crucial role in spatial navigation [34]. Their most notable feature is that they fire when the rat traverses locations corresponding to the vertices of a triangular grid embedded in the environment. In 2018, two groups of researchers discovered independently similar grid-firing patterns from units in recurrent neural networks that were optimized to perform navigation tasks in a range of 2D environments [35, 36]. Moreover, the state-of-the-art neural network architecture, Transformers [37], which is designed without the brain in mind, can also replicate the same spatial representations [6]. These studies indicate that the brain and neural networks converge in their solutions when trying to solve the same task, even though their implementations are rather different from each other.

A more relevant example to this thesis is the Bloom filter. Invented in 1970, a Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. It consists of an array of m zero bits and k distinct hash functions that map any element to one of the m positions. When adding an element, the bits at all its k hashed positions are set to one. Clearly, for an element, if any of its k positions is zero, we know it has not been added before. However, if all has been set to one, we might get a false positive, whose probability can be controlled by picking proper parameters m and k. In 2017, a neural circuit that is supposed to detect novel and familiar odorants (can be thought of as non-members and members) was identified in Drosophila (small fruit flies) [38], and its connection to Bloom filters was established a year later [39]. Drosophila implements its "hash functions" by sparse random projections, which has inspired a family of efficient hashing implementations.

1.3.1 *The olfactory pathway in Drosophila*

The aforementioned sparse random projections are present in the mushroom body of Drosophila, a center for olfactory memory and associative learning [40, 41]. The olfactory signal is first generated by the olfactory receptor neurons, which are activated when odor molecules bind to their olfactory receptors [42]. Distinct types of projection neurons (PNs) then integrate signals from stereotyped sets of receptor neurons that express distinct types of olfactory receptors [43]. In the end, PNs form random synaptic connections to the densely populated Kenyons cells (KCs) [44].

The connections between PNs and KCs are interesting in two ways. First, KCs outnumber PNs by a large margin. Based on anatomical estimates, the ratio of PNs to KCs in the mushroom body of Drosophila is roughly $1:40$ [45, 46]. Secondly, the projection between them is random and sparse, with each KCs receiving input from 7 PNs on average [44]. These two characteristics make the PN-KC pathway resemble a two-layer artificial neural network that extracts random features [47, 48], which has a wide second layer and randomly initialized Gaussian weights between the two layers. The difference is, the neural network is fully connected and its weights can be negative, while the projection from PNs to KCs is sparse and consists of only excitatory connections. This raises the question of whether the biologically constrained pathway can be as powerful as the artificial neural network in terms of feature extraction.

In Chapter 4, we build a computational model for the PN-KC pathway, by integrating an inhibitory neuron, the anterior paired lateral (APL) neuron. The APL neuron receives input from all KCs and provides feedback inhibition to all of them [49–51]. We show that the APL neuron plays a key role in enabling the sparse and sign-constrained PN-KC network to extract random features, as is done in a fully connected neural network.

1.4 SUMMARY OF RESULTS

In this section, we provide a brief summary of each individual chapter.

1.4.1 *Exponential slowdown for larger populations: the $(\mu+1)$-EA on monotone functions*

Pseudo-Boolean monotone functions are unimodal functions that are trivial to optimize for some hillclimbers, but are challenging for a surprising

number of evolutionary algorithms. A general trend is that evolutionary algorithms are efficient if parameters like the mutation rate are set conservatively, but may need exponential time otherwise. In particular, it was known that the $(1+1)$-EA can optimize every monotone function in pseudolinear time if the mutation rate is c/n for some $c < 1$, but that they need exponential time for some monotone functions for $c > c_0 \approx 2.13$. The second part of the statement was also known for the $(\mu+1)$-EA.

In Chapter 2 we show that the first statement does *not* apply to the $(\mu+1)$-EA. More precisely, we prove that for every constant $c > 0$ there is a constant $\mu_0 \in \mathbb{N}$ such that the $(\mu+1)$-EA with mutation rate c/n and population size $\mu_0 \leq \mu \leq n$ needs superpolynomial time to optimize some monotone functions. Thus, increasing the population size by just a constant has devastating effects on performance. This is in stark contrast to many other benchmark functions on which increasing the population size either increases the performance significantly or affects performance only mildly.

The reason why larger populations are harmful lies in the fact that larger populations may temporarily decrease the selective pressure on parts of the population. This allows unfavorable mutations to accumulate in single individuals and their descendants. If the population moves sufficiently fast through the search space, then such unfavorable descendants can become ancestors of future generations, and the bad mutations are preserved. Remarkably, this effect only occurs if the population renews itself sufficiently fast, which can only happen far away from the optimum. This is counterintuitive since usually optimization becomes harder as we approach the optimum. Previous work missed the effect because it focused on monotone functions that are only deceptively close to the optimum.

1.4.2 *OneMax is not the easiest function for fitness improvements*

Next, we study the $(1:s+1)$ success rule for controlling the offspring population size of the $(1,\lambda)$-EA. It was shown by Hevia Fajardo and Sudholt [26, 27] that this parameter control mechanism can run into problems for large s if the fitness landscape is too easy. They conjectured that this problem is worst for the OneMax benchmark, since in some well-established sense OneMax is known to be the easiest fitness landscape. In Chapter 3 we disprove this conjecture and show that OneMax is not the easiest fitness landscape with respect to finding improving steps.

As a consequence, we show that there exists s and ε such that the self-adjusting $(1,\lambda)$-EA with $(1:s+1)$-rule optimizes OneMax efficiently when

started with εn zero-bits, but does not find the optimum in polynomial time on DYNAMIC BINVAL[6]. Hence, we show that there are landscapes where the problem of the $(1 : s + 1)$-rule for controlling the population size of the $(1, \lambda)$-EA is more severe than for ONEMAX.

1.4.3 *Global inhibition enables sparse Dale networks to approximate kernel functions*

The olfactory pathway in the mushroom body of Drosophila features a random sparse expansion layer, which amplifies the dimension of the input signal and facilitates downstream associative learning. In machine learning, it is known that a random dense layer with an infinite width and certain activation functions implements feature maps that correspond to arc-cosine kernel functions[7]. In Chapter 4 we show that despite being sparse and obeying Dale's law (i.e., a neuron can either excite or inhibit), the biologically constrained network in the mushroom body can be a good approximation of the arc-cosine kernels thanks to the inhibitory anterior paired lateral neuron. By studying the parameterization of the random network, we find that excellent kernel approximation and performance on downstream tasks can be expected for decently large networks with a large range of sparsity levels and weight initialization. Moreover, sparse connections are favored in terms of noise resistance, i.e. the ability of the network to recognize familiar odorants based on their noisy representations.

1.5 COMMON THEMES THROUGHOUT THE THESIS

At first glance, this thesis is a compilation of three works in parallel directions. Actually, although the three chapters do not build on top of each other, there are common themes shared by all of them. This section is devoted to discussing these topics.

First of all, randomness is an inherent feature of the two bio-inspired computing models that we study in this thesis. Both evolutionary algorithms and the neural network we study in Chapter 4 make use of randomness to maintain variety in populations, i.e. populations of search points, offspring, and neurons. In evolutionary algorithms, randomness is involved in parent selection, mutation, and tie-breaking. While in the neural network, the connection between neurons is randomly decided, resulting in each neuron

6 See Section 3.2.2 for its definition.
7 See Section 4.2.1 or [48] for the definitions.

extracting features from a distinct combination of input channels. With the help of randomness, the efforts of designing specific rules and implementing such rules are relieved.

Furthermore, randomness is also an important tool for theoretical analysis, especially in Chapter 2 and 3. To be more specific, the functions that we study analytically in this thesis are symmetric functions like OneMax, where all bits are weighted equally, and functions constructed with randomness, including HotTopic and Dynamic BinVal. The reason is that we often need to estimate the improvement probability in different phases of optimization, and this means different numbers of one-bits for monotone functions. To ease analysis, we would like this probability to be dependent only on the number of one-bits in the search point, but not on the locations of these bits. That requires exactly each bit to be treated identically by the function, and randomness achieves this goal naturally.

Content-wise, the aim of all chapters is to understand the corresponding methods better by studying their parameters. In Chapter 2 we study the interaction of two parameters in $(\mu + 1)$-EA, the mutation rate c and population size μ. In Chapter 3, due to the introduction of the $(1 : s + 1)$-rule in $(1, \lambda)$-EA, our focus is shifted more towards the meta parameters, success rate s and update strength F. In the last chapter, the network model comes with three hyperparameters, the number of random features m, the sparsity of connection p, and the weight distribution. We have already discussed the importance of parameters to evolutionary algorithms in Section 1.2. For neural networks, it is common practice to conduct extensive hyperparameter search on benchmarks as the choice of hyperparameters can significantly affect the resulting model's performance [52].

2

EXPONENTIAL SLOWDOWN FOR LARGER POPULATIONS IN $(\mu+1)$-EA

This chapter is based on joint work with Johannes Lengler, which was presented in the *ACM/SIGEVO Workshop on Foundations of Genetic Algorithms (FOGA 2019)* [53] and published in *Theoretical Computer Science (2021)* [54].

2.1 INTRODUCTION

Population-based *evolutionary algorithms* (EAs) are general-purpose heuristics for optimization. Having a population may be helpful, because it allows for diversity in the algorithm's states. Such diversity may be helpful for escaping local minima, and it is a necessary ingredient for crossover operations as they are used in *genetic algorithms* (GAs). Theoretical and practical analysis of population-based algorithms have indeed mostly found positive or neutral effects, and showed a general trend that larger populations are better [16], or at least not worse than a population size of one [55]. The only (mild) observed negative effect is, intuitively speaking, that maintaining a population of size μ may slow down the optimization time by a factor of at most μ. Only few, highly artificial examples are known [18, 19] in which a $(\mu+1)$-EA or $(\mu+1)$-GA with time budget μt performs significantly worse than a $(1+1)$-EA with time budget t. In this sense, it is easy to believe that a $(\mu+1)$ algorithm is at least as good as a $(1+1)$ algorithm, except for the runtime increase that comes from each individual only having probability $1/\mu$ per round of creating an offspring.

Our results challenge this belief, and show that it is highly wrong for some monotone functions. Our main results show that increasing μ from 1 to a larger constant can increase the runtime from quasilinear to exponential.

A monotone[1] pseudo-Boolean function is a function $f : \{0,1\}^n \to \mathbb{R}$ such that for every $x, y \in \{0,1\}^n$ with $x \neq y$ and $x_i \geq y_i$ for all $1 \leq i \leq n$ it holds $f(x) > f(y)$. Monotone functions are easy benchmark functions for optimization techniques, since they always have a unique local and global optimum at the all-ones string. Moreover, from every search point there

1 Following [13, 14], we call them monotone functions, although *strictly monotone functions* would be slightly more accurate.

are short, fitness-increasing paths to the optimum, by flipping zero-bits into one-bits. Consequently, there are many algorithms which can easily optimize every monotone function. A particular example is *random local search* (RLS), which is the $(1+1)$ algorithm that flips in each round exactly one bit, uniformly at random. RLS can never increase the distance from the optimum for a monotone function, and it optimizes any such function in time $O(n \log n)$ by a coupon collector argument. Thus monotone functions are regarded as an easy benchmark for evolutionary algorithms. Nevertheless it was shown in [11–14] that a surprising number of evolutionary algorithms need exponential time to optimize some monotone functions, especially if they mutate too aggressively, i.e., the mutation parameter c is too large (see Section 2.1.2 for a detailed discussion). However, in all considered cases the algorithms were efficient if the mutation parameter satisfied $c < 1$.

2.1.1 *Our results*

We show that the $(\mu+1)$-Evolutionary Algorithm, $(\mu+1)$-EA, becomes inefficient even if the mutation strength is smaller than 1. More precisely, we show that for every $c > 0$ there is a $\mu_0 = \mu_0(c) \in \mathbb{N}$ such that for all $\mu_0 \leq \mu \leq n$ there are some monotone functions for which the $(\mu+1)$-EA with mutation rate c/n needs superpolynomial time to find the optimum. If μ is $O(1)$ then this time is even exponential in n. Note that for $0 < c \leq 1$, it is known that the $(1+1)$-EA finds the optimum in quasilinear time for any monotone functions [12, 56, 57]. Thus when we increase the population size only slightly (from 1 to μ_0), the optimization time explodes, from quasilinear to exponential.

The monotone functions that are hard to optimize are due to Lengler and Steger [13], and were dubbed HotTopic functions in [14]. These functions look locally like linear functions in which all bits have some positive weights. However, in each region of the search space there is a specific subset of bits (the 'hot topic'), which have very large weights, while all other bits have only small weights. If an algorithm improves in the hot topic, then it will accept the offspring regardless of whether the other bits deteriorate. In [13–15] it was shown that an algorithm like the $(1+1)$-EA with mutation rate $c > 2.13..$ will mutate too many of these bits outside of the hot topic, and will thus not make progress towards the global optimum.

The key insight of our work is that for such weighted linear functions with imbalanced weights, populations may also lead to an accumulation of

bad mutations, even if the mutation rate is small. Here is the intuition. For a search point x, we call the number of one-bits in the hot topic in x the *rank* of x. Consider a $(\mu+1)$-EA close to the optimum, and assume for simplicity that all search points in the population S_0 have the same rank i. At some point one of them will improve in the hot topic by flipping a zero-bit there. Let us call the offspring x, and let us assume that its rank is $i+1$. Then x is fitter than all other search points in the population because it has a higher rank. Moreover, every offspring or descendant of x will also be fitter than all the other points in the population, as long as they maintain rank $i+1$. Thus for a while the $(\mu+1)$-EA will accept all (or most) descendants of x, and remove search points of rank i from the population. This goes on until some time t_0 at which search points of rank i are completely eliminated from the population. Note that at time t_0, most descendants x' of x have considerably smaller fitness than x, since the algorithm accepts every type of mutation outside of the hot topic, and most mutations are detrimental. If some descendant x' of x creates an offspring y of even higher rank, then y is accepted and the cycle repeats with y instead of x. The crucial point is that y is an offspring of x', which has accumulated a lot of bad mutations compared to x. So typically, x' is considerably less fit than x, but still it passes on its bad genes.

The above effect needs that the probability of improving in the hot topic has the right order. If the probability is too large (close to one), then x will already spawn an offspring of rank $i+1$ before it has spawned many descendants with the same rank. On the other hand, if the probability is too small then there will be no rank-improving mutations until time t_0, and after time t_0 the algorithm starts to remove the worst individuals of rank $i+1$ from the population. We remark that this latter regime was already studied in [14], for the extreme case in which the improvement probability is so small that typically the population of rank $i+1$ collapses into copies of x before a further improvement is made. (In the terminology of [14], it was the assumption that the parameter ε of the HotTopic function was sufficiently small.) However, there is a rather large range of improvement probabilities that lead to the aforementioned effect, i.e., they typically yield an offspring y from some inferior search point x' of rank $i+1$.

2.1.2 *Related work*

The analysis of EAs on monotone functions started in 2010 by the work of Doerr, Jansen, Sudholt, Winzen and Zarges [11, 12]. Their contribution

was twofold: firstly, they showed that the $(1+1)$-EA, which flips each bit independently with static mutation rate c/n, needs time $O(n \log n)$ on all monotone functions if the mutation parameter c is a constant strictly smaller than one. This result was already implicit in [56].

On the other hand, it was also shown in [11, 12] that for large mutation rates, $c > 16$, there are monotone functions for which the $(1+1)$-EA needs exponential time. The construction of hard monotone functions in [11, 12] was later simplified by Lengler and Steger [13], who improved the range for c from $c > 16$ to $c > c_0 = 2.13...$ Their construction was later called HotTopic functions in [14], and it will also be the basis for the results in this work.

For a long time, it was an open question whether $c = 1$ is a threshold at which the runtime switches from polynomial to exponential. On the presumed threshold $c = 1$, a bound of $O(n^{3/2})$ was known due to Jansen [56], but it was unclear whether the runtime is quasilinear. Finally, Lengler, Martinsson and Steger [57] could show that $c = 1$ is not a threshold, showing by an information compression argument an $O(n \log^2 n)$ bound for all $c \in [1, 1+\varepsilon]$ for some $\varepsilon > 0$.

Recently, the limits of our understanding of monotone functions were pushed significantly by Lengler [14, 15], who analyzed monotone functions for a manifold of other evolutionary and genetic algorithms. In particular, he analyzed the algorithms on HotTopic functions, and found sharp thresholds in the parameters, such that on one side of the threshold the runtime on HotTopic was $O(n \log n)$, while on the other side of the threshold it was exponential. These algorithms include the $(1+1)$-EA, the $(1+\lambda)$-EA, the $(\mu+1)$-EA, for which the threshold condition was $c < c_0$, where $c_0 = 2.13..$, and it further included the $(1+(\lambda,\lambda))$-GA, and the so-called 'fast $(1+1)$-EA' and 'fast $(1+\lambda)$-EA'.[2] Surprisingly, for the genetic algorithms $(\mu+1)$-GA and the 'fast $(\mu+1)$-GA', any parameter range leads to runtime $O(n \log n)$ on HotTopic if the population size μ is large enough, showing that crossover is strongly beneficial in these cases.

For some of the algorithms, Lengler in [14, 15] also complemented the results on HotTopic functions by statements asserting that for less aggressive choices of the parameters the algorithms optimize *every* monotone function efficiently. For example, he proved that for mutation parameter $c < 1$ and for every constant $\lambda \in \mathbb{N}$, with high probability the $(1+\lambda)$-EA optimizes every monotone function in $O(n \log n)$ steps. Analogous statements were proven

2 The so-called "fast" versions draw the parameter c randomly in each iteration from a heavy-tailed distribution. This avoids that the probability of flipping k bits drops exponentially in k [58].

for the 'fast $(1+1)$-EA' and 'fast $(1+\lambda)$-EA', and for the $(1+(\lambda,\lambda))$-GA, but the condition $c < 1$ needs to be replaced by analogous conditions on the parameters of the respective algorithms. Moreover, in the case of the 'fast $(1+\lambda)$-EA', the result was only proven if the algorithm starts sufficiently close to the optimum. Lengler did not prove any results for general monotone functions for the population-based algorithms $(\mu+1)$-EA and $(\mu+1)$-GA, and for their 'fast' counterparts. Our result shows that at least for the $(\mu+1)$-EA, this gap had a good reason. As mentioned before, we will show that for every (constant) mutation parameter $c > 0$, there are monotone functions on which the $(\mu+1)$-EA needs superpolynomial time if the population size μ is larger than some constant $\mu_0 = \mu_0(c)$. It also shows that the $(\mu+1)$-EA and the $(1+\lambda)$-EA behave completely differently on the class of monotone functions, since the $(1+\lambda)$-EA is efficient for all constant λ whenever $c < 1$.

Surprisingly, our instance of a hard monotone function is again a HotTopic function. This may appear contradictory to the result in [14, 15] that the $(\mu+1)$-EA is efficient on HotTopic functions if $c < c_0$. The reason why there is no contradiction is that all the results in [14, 15] on HotTopic come with an important catch. The HotTopic functions come with several parameters, and we will give the formal definition and a more detailed discussion in Section 2.2.3. For now it suffices to know that one of the parameters, ε, essentially determines how close the algorithm needs to come to the optimum before the fitness function starts switching between different hot topics. In [14, 15], only small values of ε were considered. More precisely, it was shown that for every $\mu \in \mathbb{N}$ there is an $\varepsilon_0 > 0$ such that the results for the $(\mu+1)$-EA hold for all HotTopic functions with parameter $\varepsilon \leq \varepsilon_0$, and there were similar restrictions for other parameters of the HotTopic function. In a nutshell, *the effect of switching hot topics was only studied close to the optimum*. Arguably, this was a natural approach since usually the hardest region for optimization is close to the optimum. In this work, we consider HotTopic functions in a different parameter regime: we study relatively large values of the parameter ε, which is a regime of the HotTopic functions in which the action happens far away from the optimum. *Consequently, the results from [14, 15] on the $(\mu+1)$-EA on* HotTopic *do not carry over to the version of* HotTopic *functions that we consider in this work.* We stress this point to resolve the apparent contradiction between our results and the results in [14, 15].

The above discussion also shows a rather uncommon phenomenon. Consider a small mutation parameter, e.g., $c = 1/2$. Our results show that the

$(\mu+1)$-EA fails to make progress if the HotTopic function starts switching hot topics far away from the optimum. On the other hand, by the results in [14], the $(\mu+1)$-EA is not deceived if the HotTopic function starts switching hot topics close to the optimum. Thus, we have found an example where optimization close to the optimum is easier than optimization far away from the optimum, quite the opposite of the usual behavior of algorithms. This strange effect occurs because the problem of the $(\mu+1)$-EA arises from having a non-trivial population. However, close to the optimum, progress is so hard that the population tends to degenerate into multiple copies of a single search point, which effectively decreases the population size to one and thus eliminates the problem (see also the discussion in Section 2.1.1 above).

Most other work on population-based algorithms has shown benefits of larger population sizes, especially when crossover is used [59–62]. Without crossover, the effect is often rather small [55]. The only exception in which a population has theoretically been proven to be severely disadvantageous is on Ignoble Trails. This rather specific function has been carefully designed to lead into a trap for crossover operators [18], and it is deceptive for $\mu = 2$ if crossover is used, but not for $\mu = 1$. Arguably, the HotTopic functions are also rather artificial, although they were not specifically designed to be deceptive for populations. However, regarding the larger and more natural framework of monotone functions, our results imply that a $(\mu+1)$-EA with mutation parameter $c = 1$ does not optimize all monotone functions efficiently if μ is too large, while the corresponding $(1+1)$-EA is efficient.

Moreover, Lengler and Schaller pointed out an interesting connection between HotTopic functions and a dynamic optimization problem in [63], which is arguably more natural. In that paper, the algorithm should optimize a linear function with positive weights, but the weights of the objective function are re-drawn each round (independently and identically distributed). This setting is similar to monotone functions, since a one-bit is always preferable over a zero-bit, and the all-one string is always the global optimum. However, the weight of each bit changes from round to round, which somewhat resembles that the HotTopic function switches between different hot topics as the algorithm progresses. In [63] the $(1+1)$-EA was studied, and the behavior in the dynamic setting is very similar to the behavior on HotTopic functions. It remains open whether the effects observed in our work carry over to this dynamic setting.

2.2 PRELIMINARIES AND DEFINITIONS

2.2.1 *Notation*

Throughout the chapter, we will assume that $f : \{0,1\}^n \to \mathbb{R}$ is a monotone function, i.e., for every $x, y \in \{0,1\}^n$ with $x \neq y$ and such that $x_i \geq y_i$ for all $1 \leq i \leq n$ it holds $f(x) > f(y)$. We will consider algorithms that try to maximize f, and we will mostly focus on the *runtime* of an algorithm, which we define as the number of function evaluations before the first evaluation of the global maximum of f.

For $n \in \mathbb{N}$, we denote $[n] := \{1, \dots, n\}$. For a search point x, we write $\textsc{Om}(x)$ for the OneMax-value of x, i.e., the number of one-bits in x. For $x \in \{0,1\}^n$ and $\emptyset \neq I \subseteq [n]$, we denote by $d(I, x) := |\{i \in I \mid x_i = 0\}|/|I|$ the *density* of zero-bits in I. In particular, $d([n], x) = 1 - \textsc{Om}(x)/n$. Landau notation like $O(n), o(n), \dots$ is with respect to $n \to \infty$. An event $\mathcal{E} = \mathcal{E}(n)$ holds *with high probability* or *whp* if $\Pr[\mathcal{E}(n)] \to 1$ for $n \to \infty$. A function $f : \mathbb{N} \to \mathbb{R}$ grows *stretched-exponentially* if there is $\delta > 0$ such that $f(x) = \exp\{\Omega(n^\delta)\}$, and it grows *quasilinearly* if there is $C > 0$ such that $f(x) = O(n \log^C n)$.

Throughout the chapter, we will use n for the dimension of the search space, μ for the population size, and c for the mutation parameter. We will always assume that the mutation parameter c is a constant independent of n, but the population size $\mu = \mu(n)$ may depend on n.

2.2.2 *Algorithm*

We will consider the $(\mu + 1)$-EA with population size $\mu \in \mathbb{N}$ and mutation parameter $c > 0$ for maximizing a pseudo-boolean fitness function $f : \{0,1\}^n \to \mathbb{R}$. This algorithm maintains a population of μ search points. In each round, it picks one of these search points uniformly at random, the *parent* x^t for this round. From this parent it creates an *offspring* y^t by flipping each bit of x^t independently with probability c/n, and adds it to the population. From the $\mu + 1$ search points, it then discards the one with lowest fitness from the population, breaking ties randomly[3].

3 We break ties randomly for simplicity. Other selection schemes may give preference to offspring, or generally to more recent search points in case of ties. However, the tie-breaking scheme does not have an impact on our analysis.

Algorithm 1: The $(\mu+1)$-EA with mutation parameter c for maximizing an unknown fitness function $f : \{0,1\}^n \to \mathbb{R}$. The population S is a multiset, i.e., it may contain some search points several times.

Initialization:
  $S_0 \leftarrow \emptyset$;
  for $i = 1, \dots, \mu$ **do**
    Sample $x^{(0,i)}$ uniformly at random from $\{0,1\}^n$;
    $S_0 \leftarrow S_0 \cup \{x^{(0,i)}\}$;
Optimization:
  for $t = 1, 2, 3, \dots$ **do**
    Mutation:
    Choose $x^t \in S_{t-1}$ uniformly at random;
    Create y^t by flipping each bit in x^t independently with probability c/n;
    Selection:
    Set $S_t \leftarrow S_{t-1} \cup \{y^t\}$;
    Select $x \in \arg\min\{f(x) \mid x \in S_t\}$ (break ties randomly) and update $S_t \leftarrow S_t \setminus \{x\}$;

2.2.3 *HotTopic functions*

In this section we give the construction of hard monotone functions by Lengler and Steger [13], following closely the exposition in [14]. The functions come with five parameters $n \in \mathbb{N}$, $0 < \beta < \alpha < 1$, $0 < \varepsilon < 1$, and $L \in \mathbb{N}$, and they are given by a randomized construction. Following [14], we call the corresponding function $\textsc{HotTopic}_{n,\alpha,\beta,\varepsilon,L} = \mathrm{HT}_{n,\alpha,\beta,\varepsilon,L} = \mathrm{HT}$.

For $1 \leq i \leq L$ we choose sets $A_i \subseteq [n]$ of size αn independently and uniformly at random, and we choose subsets $B_i \subseteq A_i$ of size βn uniformly at random. We define the *level* $\ell(x)$ of a search point $x \in \{0,1\}^n$ by

$$\ell(x) := \max\left\{\ell' \in [L] : d(B_{\ell'}, x) \leq \varepsilon\right\}, \tag{2.1}$$

where we set $\ell(x) = 0$, if no such ℓ' exists. Then we define $f : \{0,1\}^n \to \mathbb{R}$ as follows:

$$\mathrm{HT}(x) := \ell(x) \cdot n^2 + \sum_{i \in A_{\ell(x)+1}} x_i \cdot n + \sum_{i \in R_{\ell(x)+1}} x_i, \tag{2.2}$$

where $R_{\ell(x)+1} := [n] \setminus A_{\ell(x)+1}$, and where we set $A_{L+1} := B_{L+1} := \emptyset$. One easily checks that this function is monotone [14].

So the set $A_{\ell+1}$ defines the hot topic while the algorithm is at level ℓ, where the level is determined by the sets B_i. Following up on the discussion in the introduction, observe that the level ℓ increases if the density of zero-bits in $B_{\ell'}$ drops below ε for some $\ell' > \ell$. From the analysis we will see that with high probability this only happens if the density of zero-bits in $A_{\ell+1}$ and in the whole string is also roughly ε, up to some constant factors. Hence, the parameter ε determines how far away the algorithm is from the optimum when the level changes.

Throughout the chapter, we will assume that α and β are independent of n, whereas we will choose small constants $\eta, \rho > 0$ and set $\varepsilon = \mu^{-1+\eta}$ and $L = \exp\{\rho\varepsilon n / \log^2 \mu\}$, i.e., ε and L may depend of n, since we also allow μ to depend on n.[4]

2.2.4 *Tools*

To obtain good tail bounds, we often apply Chernoff's inequality.

4 In the papers [13–15] the parameter L was replaced by a constant parameter ρ such that $L = e^{\rho n}$. This had the advantage that their parameters were all independent of n, but since our parameters depend on n anyway, it is more convenient to use the parameter L. However, both versions are equivalent.

Theorem 1 (Chernoff Bound [64]). *Let $Y_1, \dots, Y_m$ be independent random variables (not necessarily i.i.d.) that take values in $[0,1]$. Let $S := \sum_{i=1}^{m} Y_i$, then for all $0 \leq \delta \leq 1$,*

$$\Pr[S \leq (1-\delta)\,\mathbb{E}[S]] \leq e^{-\delta^2\,\mathbb{E}[S]/2}$$

and for all $\delta \geq 0$,

$$\Pr[S \geq (1+\delta)\,\mathbb{E}[S]] \leq e^{-\min\{\delta^2,\delta\}\,\mathbb{E}[S]/3}.$$

Finally, for all $k \geq 2e\,\mathbb{E}[S]$,

$$\Pr[S \geq k] \leq 2^{-k}.$$

In addition, we will need the following theorem to bound the sum of geometrically distributed random variables.

Theorem 2 (Theorem 1 in [65]). *Let Y_j, $1 \leq j \leq m$, be independent random variables following the geometric distribution with success probability p_j, and let $S := \sum_{j=1}^{m} Y_j$. If $\sum_{j=1}^{m} p_j^{-2} \leq s < \infty$ then for any $\delta > 0$,*

$$\Pr[S \leq \mathbb{E}[S] - \delta] \leq \exp\Big(-\frac{\delta^2}{2s}\Big).$$

For $h := \min\{p_j \mid j \in [m]\}$,

$$\Pr[S \geq \mathbb{E}[S] + \delta] \leq \exp\Big(-\frac{\delta}{4}\min\Big\{\frac{\delta}{s}, h\Big\}\Big).$$

The following lemma estimates useful probabilities, e.g. the probability to improve on the current hot topic.

Lemma 3. *Let $\alpha, c > 0$ be constants. Consider a set $A \subseteq [n]$ of size αn where n is large enough, and consider a search point $x \in \{0,1\}^n$.*

1. *The probability that the number of one-bits in A does not decrease after a standard bit mutation with rate c/n on x can be bounded by $p_R = e^{-\alpha c}/2$ from below.*
2. *The probability that a standard bit mutation with rate c/n strictly increases the number of one-bits in A has a lower bound $p_L = \varepsilon(x)\alpha c e^{-\alpha c}/2$ and an upper bound $p_U = \varepsilon(x)\alpha c$, where $\varepsilon(x) = d(A,x)$.*
3. *Let $(1-\varepsilon')\alpha n \leq i \leq \alpha n$ where $0 < \varepsilon' < 1$ and $\varepsilon' n \geq 2ec$. Assume $\mathrm{rk}(x) < i$, and let y be an offspring of x. Then at least one of the following inequalities holds.*

$$\Pr[\mathrm{rk}(y) \geq i] \leq 2^{-\varepsilon'\alpha n} \qquad \text{or} \qquad \frac{\Pr[\mathrm{rk}(y) \geq i+1]}{\Pr[\mathrm{rk}(y) \geq i]} \leq 2\varepsilon'\alpha c.$$

Proof of Lemma 3. We show the statements one by one.

1. One way of creating an offspring with the same number of onebits in A is to flip no bits at all in A, which is $(1-c/n)^{\alpha n} = e^{-\alpha c} - O(1/n) \geq e^{-\alpha c}/2$ when n is large enough.

2. We observe that the probability we consider is at least

$$\begin{aligned}\Pr[\text{flip 1 zero-bit and 0 one-bits in } A] &= \varepsilon(x)\alpha n \cdot \frac{c}{n}\Big(1-\frac{c}{n}\Big)^{\alpha n-1} \\ &= \varepsilon(x)\alpha c\Big(e^{-\alpha c} - O\Big(\frac{1}{n}\Big)\Big) \\ &\geq \frac{1}{2}\varepsilon(x)\alpha c e^{-\alpha c}.\end{aligned}$$

   And it is at most

$$\Pr[\text{flip at least 1 zero-bit}] \leq \sum_{i=1}^{\varepsilon(x)\alpha n} \Pr[\text{flip the } i\text{-th zero-bit}] = \varepsilon(x)\alpha c,$$

   where the second inequality follows a union bound over all zero-bits in A.

3. Assume first that $\mathrm{rk}(x) < (1-2\varepsilon')\alpha n$. Then for $\mathrm{rk}(y) \geq i$, at least $\varepsilon'\alpha n$ zero-bits must be flipped in one mutation. The expected number of flipped zero-bits is at most $\alpha n \cdot c/n = \alpha c$, so that happens with probability $2^{-\varepsilon'\alpha n}$ by the Chernoff bound. So let us consider the other case, $\mathrm{rk}(x) \geq (1-2\varepsilon')\alpha n$. Let P be a permutation on the αn bits in A such that $P(j) < P(j')$ for all $x_j = 1$ and $x_{j'} = 0$. Consider mutating the bits in x in the permuted order, and we track the number $G := G_0 - G_1$ during that process, where G_0 (G_1) is the flipped zero-bits (one-bits). Clearly, G will be decreasing while we are at the one-bits and increasing afterwards. Then $\mathrm{rk}(y) \geq i$ if and only if $G \geq i - \mathrm{rk}(x)$ after flipping some zero-bit j, and $\mathrm{rk}(y) \geq i+1$ if and only if at least one more zero-bit is flipped after bit j. The number of remaining zero-bits is at most $\alpha n - \mathrm{rk}(x) - 1 < 2\varepsilon'\alpha n$, so the probability of flipped at least one remaining zero-bit is at most $2\varepsilon'\alpha c$ by a union bound. Therefore,

$$\begin{aligned}\Pr[\mathrm{rk}(y) \geq i+1] &\leq 2\varepsilon'\alpha c \cdot \Pr[G \geq i - \mathrm{rk}(x) \text{ at some zero-bit } j] \\ &= 2\varepsilon'\alpha c \cdot \Pr[\mathrm{rk}(y) \geq i].\end{aligned}$$

□

We will use the following two theorems to bound the running time of the $(\mu+1)$-EA. The first one states that a sequence of random variables whose differences are small with exponentially decaying tail bound are *sub-Gaussian.*[5]

Theorem 4 (Timo Kötzing, Theorem 10 in [66]). *Let $(Y_i)_{i\geq 0}$ be a supermartingale such that there are $c' > 0$ and δ' with $0 < \delta' < 1$ and, for all $i \geq 0$ and for all $y \geq 0$,*

$$\Pr[|Y_{i+1} - Y_i| \geq y \mid Y_0, \dots, Y_i] \leq c'(1+\delta')^{-y}.$$

Then $(Y_i)_{i\geq 0}$ is $(128c'\delta'^{-3}, \delta'/4)$-sub-Gaussian.

The other theorem bounds first hitting times of sub-Gaussian supermartingales.

Theorem 5 (Timo Kötzing, Theorem 12 in [66]). *Let $(Y_i)_{i\geq 0}$ be a sequence of random variables and let $r \in \mathbb{R}$. If, for all $i \geq 0$,*

$$\mathbb{E}[Y_{i+1} - Y_i \mid Y_0, \dots, Y_i] \leq r,$$

then $(Y_i - ri)_{i\geq 0}$ is a supermartingale. If further $(Y_i - ri)_{i\geq 0}$ is (c'', δ'')-sub-Gaussian, then, for all $i \geq 0$ and all $y > 0$,

$$\Pr\left[\max_{0\leq j\leq i}(Y_j - Y_0) \geq ri + y\right] \leq \exp\left(-\frac{y}{2}\min\left(\delta'', \frac{y}{c''i}\right)\right).$$

2.3 FORMAL STATEMENT OF THE RESULT

The main result of this chapter is the following.

Theorem 6. *For every constant $c > 0$ and $0 < \beta < \alpha < 1$ there exist constants $\mu_0 = \mu_0(c) \in \mathbb{N}$ and $\eta, \rho > 0$ such that the following holds for all $\mu_0 \leq \mu \leq n$ where n is sufficiently large. Consider the $(\mu+1)$-EA with population size μ and mutation rate c/n on the n-bit* HotTopic *function $\mathrm{HT}_{n,\alpha,\beta,\varepsilon,L}$, where $\varepsilon = \mu^{-1+\eta}$ and $L = \lfloor \exp\{\rho\varepsilon n/\log^2 \mu\}\rfloor$. Then with high probability the $(\mu+1)$-EA visits every level of the* HT *function at least once. In particular, it needs at least L steps to find the optimum, with high probability and in expectation.*

That is, if $\mu \geq \mu_0$ is a constant (independent of n) then with high probability the optimization time is exponential.

5 The reader can take the concept of being sub-Gaussian as a black box. Theorem 4 asserts that exponential tail bounds guarantee the property, Theorem 5 describes the consequences. For completeness, we also give the definition: a sequence of random variables $(Y_i)_{i\geq 0}$ is (c,δ)-sub-Gaussian if and only if $\mathbb{E}[\exp(z(Y_{i+1} - Y_i)) \mid Y_0, \dots, Y_i] \leq \exp(z^2c/2)$holds for all $i \geq 0$ and $z \in [0, \delta]$.

We remark that the requirement $\mu \leq n$ is not tight, and we conjecture that the runtime is always superpolynomial for $\mu \geq \mu_0$, also for much larger values of μ. However, we did not undertake big efforts to extend the range of μ since we do not feel that it adds much to the statement. For larger values of μ, e.g., $\mu = n^2$, our proof does not go through unmodified. With our definition of $\varepsilon = \mu^{-1+\eta}$, we only get error probabilities of the form $\exp\{-\Omega(\varepsilon n/\log^2 \mu)\}$, which are not $o(1)$ if e.g. $\mu = n^2$. Hence we would need to choose larger values of ε, and then we lose a very convenient property, namely that for every fixed i, with high probability no individual of rank at most $i-1$ creates an individual of rank at least $i+1$. To avoid these complications, we only consider $\mu \leq n$.

2.4 PROOF OVERVIEW

The next three sections are devoted to proving Theorem 6. The key ingredient is to analyze the drift of the density $d([n], x)$ for search points x which have roughly density ε. We start by giving an informal overview, and by discussing similarities and differences to the situation in [13] and [14].

We will analyze the algorithm in the regime where the fittest search point x^* in the population satisfies

$$d(A_{\ell+1}, x^*) \in [\varepsilon/2, 2\varepsilon] \quad \text{and} \quad d(R_{\ell+1}, x^*) \in [\varepsilon/2, 2\varepsilon], \tag{2.3}$$

where $\ell = \ell(x^*)$ is the current level and $\varepsilon = \mu^{-1+\eta}$ is the parameter of the HotTopic function. It will turn out that for large μ, the algorithm already needs stretched-exponential time to escape this situation.

The main idea is similar to [13, 14], in which the $(1+1)$-EA and other algorithms were analyzed. We first sketch the main argument for the $(1+1)$-EA, and explain afterwards which parts must be replaced by new arguments. The crucial ingredient is that while the density $d(A_{\ell+1}, x)$ of zero-bits on the hot topic decreases from 2ε to ε, the total density $d([n], x)$ has a positive drift, i.e., a drift away from the optimum. Moreover, the probability to change k bits in one step has a tail that decays exponentially with k. Therefore, it was shown that with high probability $d([n], x)$ stays above $\varepsilon + \gamma$ for an exponential number of steps, where γ is a small constant. Then it was argued that as long as $d([n], x)$ stays bounded away from ε, it is exponentially unlikely that the level ever increases by more than one. Since there are an exponential number of levels, this implies an exponential runtime.

The analysis of $(\mu+1)$-EA and $(\mu+1)$-GA for constant μ in [14] was obtained by reducing it to the analysis of a related $(1+1)$ algorithm. This was possible since the choice of parameters in [14] (choosing the parameter $\varepsilon = \varepsilon(\mu)$ sufficiently small) made the algorithm operate close to the optimum. In this range, there are only few zero-bits, and thus it is rather unlikely that a mutation improves the fitness. On the other hand, there is always a constant probability (if μ is constant) to create a copy of the fittest individual. In such a situation, the population degenerates frequently into a collection of copies of a single search point. Thus, the population-based algorithms behave similarly to a $(1+1)$ algorithm. This $(1+1)$ algorithm has essentially the same mutation parameter as the $(\mu+1)$-EA, while for the $(\mu+1)$-GA it has a much smaller mutation parameter (less than one), which is the reason why the $(\mu+1)$-GA is efficient on all HotTopic instances with small parameter ε. For us, the situation is more complex since we consider larger values of ε. As a consequence, it is easier to find a search point with better fitness, and the population does not collapse. Hence, it is not possible to represent the population by a single point.

Instead, we proceed as follows. Fix a fitness level ℓ, and consider the auxiliary fitness function

$$f_\ell(x) := n\sum_{j\in A_{\ell+1}} x_j + \sum_{j\in R_{\ell+1}} x_j. \tag{2.4}$$

We will first study the behavior of the $(\mu+1)$-EA on f_ℓ. Considering this fitness function is essentially the same as assuming that the level remains the same. We will see in the end that this assumption is justified, by the same arguments as in [13, 14]. For a search point x, we define the *rank* $\mathrm{rk}(x) := \big|\{j \in A_{\ell+1} \mid x_j = 1\}\big|$ of x as the number of correct bits in the current hot topic. Note that by construction of f_ℓ, a search point with higher rank is always fitter than a search point with smaller rank.

Now we define $\mathcal{X}_i$ to be the set of search points of rank i that are visited by the $(\mu+1)$-EA, and we define Z_i to be the OneMax-value (the number of one-bits) of the last search point in $\mathcal{X}_i$ that the algorithm deletes from its population. Note that due to elitist selection, this search point is also (one of) the fittest search point(s) in $\mathcal{X}_i$ that the algorithm ever visits, and hence it has the largest OneMax-value among all search points in $\mathcal{X}_i$ that the algorithm ever visits. Then our goal is to show that $\mathbb{E}[Z_{i+1} - Z_i] = -\Omega(1)$, under the assumption that the population satisfies (2.3), i.e., that the density of the fittest search point is close to ε. This assumption can be justified by a coupling argument as in [13, 14]. Computing the drift of Z_i is the heart of our proof, and the main technical contribution of this chapter. In fact,

to simplify the analysis we only prove the slightly weaker statement that $\mathbb{E}[Z_{i+K} - Z_i] = -\Omega(1)$ for a suitable constant K, which is equally suited. Once we have established this negative drift, the remainder of the proof as in [13, 14] carries over almost unchanged.

To estimate the drift $\Delta := \mathbb{E}[Z_{i+K} - Z_i]$, we will assume for this exposition that $\mu = \omega(1)$, so that we may use O-notation. (In the formal proof we will use the weaker assumption $\mu \geq \mu_0$ for a sufficiently large constant $\mu_0 = \mu_0(c)$.) We distinguish between *good* and *bad* events. Good events will represent the typical situation; they will occur with high probability, and if they occur K times in a row, then it will deterministically follow that $Z_{i+K} - Z_i \leq -\log \mu$. On the other hand, bad events may lead to a positive difference, but they are unlikely and thus they contribute only a lower order term to the drift. We will discriminate two types of bad events. Firstly, we will show that the probability $\Pr[Z_{i+K} - Z_i > \lambda \log \mu]$ drops exponentially in λ. This implies that the events in which $Z_{i+K} - Z_i > \log^2 \mu$ contribute at most a term $o(1)$ to the drift. Hence, we can restrict ourselves to the case that $Z_{i+K} - Z_i \leq \log^2 \mu$. Now assume that we have any event of probability $o(\log^{-2} \mu)$. In the case $Z_{i+K} - Z_i \leq \log^2 \mu$, this event can contribute at most a $o(1)$ term to the drift. Hence, we may declare any such event as a bad event, and conclude that all bad events together only contribute a $o(1)$ term to the drift.

As we have argued, we may neglect any event with probability $o(\log^{-2} \mu)$. This is a rather large error probability, which allows us to dub many events as 'bad', and to use rather coarse estimates on the error probability. We conclude this overview by describing how a good event, and thus a typical situation, looks like. In what follows, all claims hold with probability at least $1 - o(\log^{-2} \mu)$.

Let us call t_i the first round in which an individual of rank at least i is created, and T_i the round in which the last individual of rank at most i is eliminated. Then typically $T_i - t_i = O(\mu \log \mu) \cap \Omega(\mu)$. Let $|X_i| = |X_i(t)|$ denote the number of search points in the population of rank i at time t. We want to study the *family forest* F_i of $X_{\geq i}$, which is closely related to the family trees and family graphs that have been used in other work on population-based EAs, e.g. [16, 55, 67, 68]. The vertices of this forest are all individuals of rank at least i that are ever included into the population. A vertex is called a *root* if its parent has rank less than i. Otherwise, the forest structure reflects the creation of the search points, i.e., vertex u is a child of vertex v if the individual u was created by a mutation of v.

As X_i grows, eventually the first few search points of rank $i+1$ are created, and form the first roots of the family forest. Then the forest starts growing, both because new roots may appear and because the vertices in the forest may create offspring. At some point we have $|X_{i+1}| = \mu^\delta$ for some (suitably small) $\delta > 0$. At this point, we still have typically $|X_i| = O(\mu^\delta/\varepsilon) = O(\mu^{1+\delta-\eta}) = o(\mu)$, where the latter holds if δ is small enough. Moreover, at this point there are no search points of rank strictly larger than $i+1$. The sets X_i and X_{i+1} both continue to grow with roughly the same speed until the search points of rank at most $i-1$ are eliminated from the population. Afterwards, the search points of rank i are eliminated from the population, until only search points of rank at least $i+1$ remain. Crucially, up to this point every search point of rank at least $i+1$ is accepted into the population. In other words, there is no selective pressure on the search points of rank $i+1$, and every mutation of a search point of rank $i+1$ enters the family tree, as long as the rank $i+1$ is preserved. Therefore, we can contain the family forest F_{i+1} of rank $i+1$ up to this point in a random forests F' which is obtained by certain forest growth processes in which no vertex is ever eliminated and all vertices continue to spawn offspring with a fixed rate.

We want to understand the set of individuals in X_{i+1} that spawn offspring in X_{i+2}, and thus spawn the roots for the family forest F_{i+2}. As before we can argue that no individuals of rank at least $i+2$ are created before the family forest of rank $i+1$ reaches size μ^δ. Moreover, we can show that the time T_{i+1} at which all individuals of rank $i+1$ are eliminated from the population satisfies $T_{i+1} - t_{i+1} \leq C\mu\log\mu$ for a suitable constant $C > 0$. Hence, F_{i+1} is bounded from above by the random forest F' at time $t_{i+1} + C\mu\log\mu$. This forest is only polynomially large in μ.

The recursive trees that we use to bound F_{i+1} are well understood, see also Figure 2.1. In particular, it is known that even in F' only a small fraction μ^δ of the vertices are in depth at most $\phi\log\mu$, where $\delta, \phi > 0$ are suitable constants. Since each such vertex creates an offspring of strictly larger rank with probability ε/μ per round, the expected number of offspring of rank $i+2$ of these vertices is at most $O(\mu^\delta\varepsilon/\mu \cdot (T_{i+1} - t_{i+1}))$. With the right choice of parameters, this is $\mu^{-\Omega(1)}$, and we may conclude that no vertices of depth at most $\phi\log\mu$ create roots of rank $i+2$. On the other hand, since we do not truncate any vertices in the creation of F', they are obtained from their parents by unbiased mutations of $[n] \setminus A_\ell$, and we can show that most (all but at most μ^δ) vertices of depth at least $\phi\log\mu$ in F' have accumulated $c'\log\mu$ more bad than good bit-flips when compared to their roots, for a

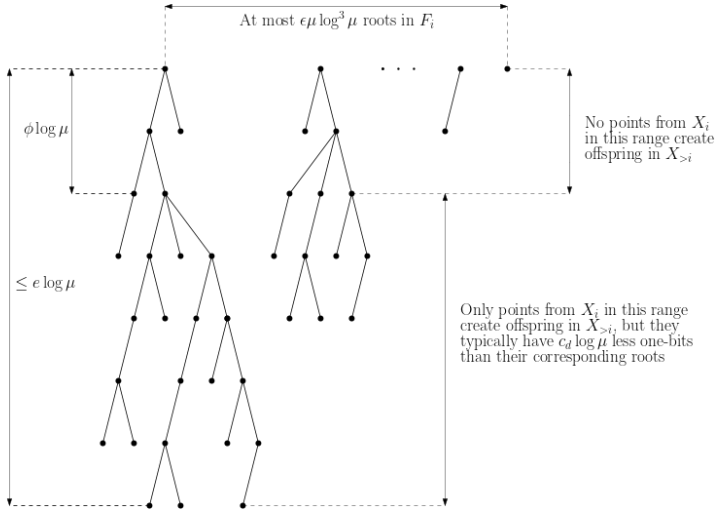

FIGURE 2.1: A depiction of the family forest F_i, where ϕ and c_d are constants to be introduced in Section 2.5.3. The same picture also applies to its upper bound F'.

suitable $c' > 0$. For the μ^δ exceptional vertices, none of them will create a root of rank $i+2$ in $T_{i+1} - t_{i+1}$ rounds, even if they are in F_{i+1}.

To summarize, good events consist of the following four main points. Firstly, no vertex of rank at most i creates an offspring of rank at least $i+2$. Secondly, every vertex in X_{i+1} that creates an offspring in X_{i+2} has at least depth $\phi \log \mu$ in the family forest. Thirdly, every vertex in X_{i+1} of depth at least $\phi \log \mu$ that creates an offspring in X_{i+2} has a OneMax value that is at least $c' \log \mu$ smaller than that of its root. Finally, we also require that no vertex in F' exceeds the OneMax value of its root by more than $C \log \mu$, for some $C > 0$. The complete list in the proof contains even more requirements, but these four already imply a decline in Z_i if they hold over K consecutive steps. In this case, inductively the OneMax values of all roots in F_{i+K} are at most $Z_i - Kc' \log \mu$. Moreover, Z_{i+K} exceeds the OneMax value of the corresponding root in X_{i+K} by at most $C \log \mu$, so we have $Z_{i+K} \leq Z_i - Kc' \log \mu + C \log \mu$. Choosing K sufficiently large shows that Z_i must decrease in these typical situations.

2.5 DRIFT ANALYSIS

In this main section of the proof, we show that the random variable Z_i has negative drift. We will use the same notation as in the proof outline. In particular, X_i denotes the set of all search points of rank i that the algorithm visits, and Z_i denotes the OneMax-value of the last search point from X_i that the algorithm keeps in its population. If X_i is empty (which, as we will see, is very unlikely), then we set $Z_i := Z_{i-1}$. Moreover, we define $X_{\geq i} := \bigcup_{i' \geq i} X_{i'}$, and the definition of terms like $X_{>i}$ is analogous. For a given parent individual x, we denote by p_I (by p_R) the probability that an offspring of x has rank which is strictly larger than (at least as large as) the rank of x.

Throughout this section, we fix a level ℓ and consider the $(\mu+1)$-EA on the linear function f_ℓ defined in (2.4). In this section, we will study the case that $i \in [(1-2\varepsilon)\alpha n, (1-\varepsilon/2)\alpha n]$, where $\varepsilon = \mu^{-1+\eta}$. Note that this is a weaker form of Condition (2.3), i.e., we consider search points for which the density in A is close to ε.

2.5.1 *Preliminaries*

In this section we first give bounds on the time that the set $X_{\geq i}$ needs to grow from size 1 to size μ^κ, and we will conclude that $X_{\geq i}$ is large at the latter point in time. We start by bounding the time.

Lemma 7. *For all $0 < \alpha < 1$, $c > 0$, $0 < \eta < \kappa \leq 1$, there exists a constant μ_0 such that the following holds for all $\mu_0 \leq \mu \leq n$. Let $i > (1-2\varepsilon)\alpha n$, where $\varepsilon = \mu^{-1+\eta}$. Consider the $(\mu+1)$-EA with mutation rate c/n on the linear function f_ℓ. Denote by $T_i^\kappa = T^\kappa$ the number of rounds until $|X_{\geq i}|$ reaches μ^κ after the algorithm visits the first point x^i in $X_{\geq i}$. With probability $1 - 2\mu^{-\Omega(1)}$,*

$$\frac{1}{2}(\kappa-\eta)\mu\log\mu \leq T^\kappa \leq 4\kappa e^{\alpha c}\mu\log\mu.$$

Moreover,

$$\mathbb{E}[T^\kappa] \leq 3\kappa e^{\alpha c}\mu\log\mu.$$

Proof. By the definition of f_ℓ, all individuals in $X_{\geq i}$ are fitter than those in $X_{<i}$. So no points in $X_{\geq i}$ will be discarded until $X_{<i}$ becomes empty, and we are interested in the growth of $|X_{\geq i}|$ during this period. Let T_j be the time needed for $|X_{\geq i}|$ to grow from j to $j+1$. By definition we have

$T^\kappa = \sum_{j=1}^{\mu^\kappa - 1} T_j$. Denote by x^t the point selected as parent by the algorithm in round t and denote by y^t its offspring. The probability that both x^t and y^t belong to $X_{\geq i}$ is at least $p_j = j/\mu \cdot p_R$, where j is the size of $X_{\geq i}$ at the beginning of round t and $p_R = e^{-\alpha c}/2$ is defined in Lemma 3.1. It is clear that we can dominate T_j by random variable $\bar{T}_j$ that follows a geometric distribution with parameter p_j. By Lemma 1.8.8 in [64], T^κ is dominated by $\bar{T}^\kappa := \sum_{j=1}^{\mu^\kappa - 1} \bar{T}_j$. Next we apply Theorem 2 to bound $\bar{T}^\kappa$ from above.

The expectation of $\bar{T}^\kappa$ is

$$\mathbb{E}[\bar{T}^\kappa] = \sum_{j=1}^{\mu^\kappa - 1} \mathbb{E}[\bar{T}_j] \leq 2e^{\alpha c}\mu \sum_{j=1}^{\mu^\kappa} \frac{1}{j}.$$

For Harmonic series, we have $\log(m+1) < \sum_{j=1}^{m} 1/j \leq \log m + 1$, where log denotes the natural logarithm. Therefore, for large enough μ,

$$\mathbb{E}[T^\kappa] \leq \mathbb{E}[\bar{T}^\kappa] \leq 2e^{\alpha c}\mu(\log(\mu^\kappa) + 1) \leq 3\kappa e^{\alpha c}\mu \log \mu. \tag{2.5}$$

Let $h := \min\{p_j \mid j = 1, \ldots, \mu^\kappa - 1\}$, clearly $h = p_1 = e^{-\alpha c}/(2\mu)$. Let $s := \sum_{j=1}^{\mu^\kappa - 1} p_j^{-2}$, we have

$$s \leq 4e^{2\alpha c}\mu^2 \sum_{j=1}^{\mu^\kappa} \frac{1}{j^2} \leq \frac{2e^{2\alpha c}\pi^2}{3}\mu^2,$$

where the last step follows from $\sum_{j=1}^{\infty} 1/j^2 = \pi^2/6$. Given h and the bound on s, by Theorem 2 it holds for $\delta = \kappa e^{\alpha c}\mu \log \mu$ that

$$\Pr\left[\bar{T}^\kappa \geq \mathbb{E}[\bar{T}^\kappa] + \delta\right] \leq e^{-\Omega(\log \mu)} = \mu^{-\Omega(1)}.$$

Since $T^\kappa \preceq \bar{T}^\kappa$, together with equation (2.5) we conclude that $T^\kappa \leq 4\kappa e^{\alpha c}\mu \log \mu$ with probability $1 - \mu^{-\Omega(1)}$.

We still need a lower bound of T^κ. Consider the probability that $X_{\geq i}$ gets a new offspring y^t in a round where $|X_{\geq i}| = j$:

$$\begin{aligned}\Pr\left[y^t \in X_{\geq i}\right] &= \Pr\left[x^t \notin X_{\geq i} \wedge y^t \in X_{\geq i}\right] + \Pr\left[x^t \in X_{\geq i} \wedge y^t \in X_{\geq i}\right] \\ &\leq (\mu - j)/\mu \cdot p_U + j/\mu \cdot 1 \leq j/\mu + p_U,\end{aligned}$$

where p_U is defined in Lemma 3.2. Let $p'_j = j/\mu + p_U$, similarly as for the upper bound on T^κ, we can subdominate T^κ with a random variable

$\hat{T}^\kappa = \sum_{j=1}^{\mu^\kappa-1} \hat{T}_j$ (Lemma 1.8.8 in [64]), where the $\hat{T}_j$ are independent and geometrically distributed with parameter p'_j, respectively. Then

$$\begin{aligned}\mathbb{E}[\hat{T}^\kappa] &\geq \sum_{j=1}^{\mu^\kappa-1} \frac{1}{p'_j} = \sum_{j=1}^{\mu^\kappa-1} \frac{\mu}{j+\mu p_U} \\ &\geq \sum_{j=1}^{\mu^\kappa-1} \frac{\mu}{j+\lceil \mu p_U\rceil} = \sum_{j=1}^{\mu^\kappa-1+\lceil \mu p_U\rceil} \frac{\mu}{j} - \sum_{j'=1}^{\lceil \mu p_U\rceil} \frac{\mu}{j'} \\ &> \mu \log\left(\mu^\kappa + \lceil \mu p_U\rceil\right) - \mu\big(\log\lceil \mu p_U\rceil + 1\big).\end{aligned}$$

Since $i \geq (1-2\varepsilon)\alpha n$, $P_U = O(\varepsilon) = O(\mu^{-1+\eta})$ for $0 < \eta < \kappa$. So $\lceil \mu p_U\rceil = O(\mu^\eta)$. Hence,

$$\mathbb{E}[T^\kappa] \geq \mathbb{E}[\hat{T}^\kappa] \geq \big(1 - O\big(\log^{-1}\mu\big)\big)(\kappa-\eta)\mu\log\mu.$$

Let $s' := \sum_{j=1}^{\mu^\kappa} p_j'^{-2}$. As $p'_j > p_j$, it holds $s' < s$ that. Applying Theorem 2 with s' and $\delta' = \varepsilon'\mu\log\mu$, we obtain

$$\Pr\left[\hat{T}^\kappa \leq \mathbb{E}[\hat{T}^\kappa] - \delta'\right] \leq e^{-\Omega(\log^2\mu)} = \mu^{\Omega(1)}.$$

Similarly, we have $\hat{T}^\kappa \preceq T^\kappa$, by picking a sufficiently small ε' we conclude that

$$T^\kappa \geq \frac{1}{2}(\kappa-\eta)\mu\log\mu$$

with probability $1-\mu^{-\Omega(1)}$. □

In the following lemma, we give a lower bound on $|X_{\geq i+1}|$ when $X_{\geq i}$ reaches a certain size.

Lemma 8. *Let $\alpha, \kappa \in (0,1)$, $c > 0$, $\eta < 1$ be constants such that $\kappa > 1 - \eta/2$. Consider the $(\mu+1)$-EA with $\mu \leq n$ and mutation rate c/n on the linear function f_ℓ. Let $\varepsilon = \mu^{-1+\eta}$ and let $i \leq (1-\varepsilon/2)\alpha n$. Denote by $Y^\kappa_{i+1} = Y^\kappa$ the size of $X_{\geq i+1}$ when $|X_{\geq i}|$ reaches μ^κ. Then with probability $1 - \exp\left(-\Omega\big(\mu^{2(\kappa-1)+\eta}\big)\right)$,*

$$Y^\kappa = \Omega\big(\varepsilon\mu^{2\kappa-1}\big) = \Omega\big(\mu^{2(\kappa-1)+\eta}\big) = \mu^{\Omega(1)}.$$

Proof. Note that we may assume that $\mu \geq \mu_0$ for a constant μ_0 of our choice, since otherwise the probability may be zero and thus the statement is vacuous. In each round $|X_{\geq i}|$ increases by either 0 or 1, so after $X_{\geq i}$ reaches size $R := \lfloor \mu^\kappa/2\rfloor$ there are at least R more rounds until $|X_{\geq i}| = \mu^\kappa$. In

each of the remaining R rounds, the probability of a parent $x \in X_{\geq i}$ being selected and its offspring y belonging to $X_{>i}$ is at least

$$\Pr[y \in X_{>i}] > \Pr[x \in X_{\geq i} \wedge y \in X_{>i}] \geq R/\mu \cdot p_L,$$

where P_L is defined in Lemma 3. Let Y_j be independent Bernoulli variables with parameters $R/\mu \cdot p_L$ for $j \in [R]$. Then Y^κ dominates the sum of Y_j, i.e. $Y^\kappa \succeq \bar{Y}^\kappa := \sum_{j=1}^{R} Y_j$. It holds that

$$\mathbb{E}[\bar{Y}^\kappa] = R \cdot R/\mu \cdot p_L = \Theta\big(\varepsilon\mu^{2\kappa-1}\big) = \Theta\big(\mu^{2(\kappa-1)+\eta}\big).$$

By Chernoff's inequality (Theorem 1), we have for any constant $0 < \delta < 1$,

$$\Pr\left[\bar{Y}^\kappa < (1-\delta)\,\mathbb{E}[\bar{Y}^\kappa]\right] \leq \exp\left(-\Omega\big(\mu^{2(\kappa-1)+\eta}\big)\right).$$

The claim follows from $Y^\kappa \succeq \bar{Y}^\kappa$. □

2.5.2 *Tail bounds*

In this section, we will give rather loose tail bounds to show that it is unlikely that Z_i is much larger than Z_{i-1}. All constants in this section are independent of μ. This includes all hidden constants in the O-notation.

2.5.2.1 *Tail bound on the lifetime of X_i*

As before, let t_i be the first round in which an individual of rank at least i is created, and let T_i be the round in which the last individual of rank at most i is eliminated.

Lemma 9. *For all $0 < \alpha, \eta < 1$, $c > 0$, there is a constant $\mu_0 \in \mathbb{N}$ such that the following holds for all $\mu_0 \leq \mu \leq n$. Let $i \in [(1-2\varepsilon)\alpha n, (1-\varepsilon/2)\alpha n]$, where $\varepsilon = \mu^{-1+\eta}$. Consider the $(\mu+1)$-EA with mutation rate c/n on the linear function f_ℓ. Then with probability at least $1-\mu^{-\Omega(1)}$, $T_i - t_i \leq 8e^{\alpha c}\mu\log\mu$. Moreover, for all $\beta \geq 1$ and $C = 16e^{\alpha c}$,*

$$\Pr[T_i - t_i \geq \beta \cdot C\mu\log\mu] \leq 2^{-\beta}.$$

Proof. We first show that $\Pr[T_i - t_i \geq C'\mu\log\mu] \leq 1/2$ for a suitable constant $C' > 0$. Let $x^{\geq i}$ be the first individual of rank at least i and let x^j with rank j be the first individual of rank strictly larger than i. We can divide the process from t_i to T_i into two parts. The first part ends when x^j is created, and we denote by t_j the round when this happens. The second part starts

after t_j and ends when $X_{>i}$ reaches size μ. Since we are proving an upper bound of the tail, we can consider the second part ends when $X_{\geq j}$ reaches μ for simplicity.

If $x^{\geq i} = x^j$, then we have $t_j = t_i$, namely the first part does not exist. So for the tail bound of the first part, we may assume that $x^{\geq i} \in X_i$. By Lemma 7, for some $1 - \eta/2 < \kappa < 1$, we have $|X_{\geq i}| \geq \mu^\kappa$ at time $T := t_i + 4\kappa e^{\alpha c}\mu \log \mu$. By Lemma 8 we have $|X_{>i}| > 0$ at this point, so x_j must have been created before time T. For the second part, we apply Lemma 7 again for $X_{\geq j}$. By time $t_j + 4e^{\alpha c}\mu \log \mu$, $X_{\geq j}$ reaches size μ.

To summarize, we have applied Lemma 7 twice and Lemma 8 once. Therefore, with probability at least $1 - 5\mu^{-\Omega(1)}$, $T_i - t_i \leq 8e^{\alpha c}\mu \log \mu$. Since $\mu \geq \mu_0$, for large enough μ_0 we obtain $\Pr[T_i - t_i \geq C'\mu \log \mu] \leq 1/2$ for $C' = 8e^{\alpha c}$.

To conclude the proof, we set $C := 2C'$. Then for all integral $\beta' \in \mathbb{N}$ we consider β' phases and repeat the same argument. This shows $\Pr[T_i - t_i \geq \beta' \cdot C'\mu \log \mu] \leq 2^{-\beta'}$. Hence, for $C = 16e^{\alpha c}$ it holds for all $\beta \geq 1$,

$$\Pr[T_i - t_i \geq \beta \cdot C\mu \log \mu] \leq \Pr[T_i - t_i \geq \lceil \beta \rceil C'\mu \log \mu] \leq 2^{-\lceil \beta \rceil} \leq 2^{-\beta}. \square$$

2.5.2.2 *Family forests*

From now on we will be mostly working on family forests, so we introduce the definition and several related lemmas here. The main idea is to couple the algorithm with a process that is not subject to selection. This idea has been used before to analyze population-based algorithms [16, 55, 67, 68].

We denote the *family forest* for search points with rank at least i by F_i. The vertex set of F_i are the vertices in $X_{\geq i}$ that are (once) in the population, while the roots of the trees are vertices whose parents are in $X_{<i}$. Moreover, any path connecting a root and a vertex in F_i corresponds to a series of mutations that create this vertex. Note that the size of F_i increase over time.

As analysing F_i directly can be complicated, we couple it with a simpler random forest F', which is generated by the following process. In round 0 there is a single root in F'. In each subsequent round, each vertex in F' creates a new child with probability $1/\mu$ and a new root is added. Lemma 10 shows that F_i can be coupled to a subgraph in F'.

Lemma 10. *The family forest F_i can be coupled to F' such that F' contains F_i as a subgraph at any round.*

Proof. Throughout the coupling process we maintain that F_i is a subgraph of F'. The first point $x^{\geq i}$ that the algorithm visits in $X_{\geq i}$ (in round t_i)

corresponds to the only root r_0 in round 0 in F'. In every round $t > t_i$, a point x^t in the current population is selected to create an offspring y^t. For each $x \in F_i$, if $x^t = x$ (which happens with probability $1/\mu$ if x is still in the current population, and with probability zero otherwise) then we attach a child to x in F': if $y^t \in X_{\geq i}$ then we attach y^t to x in F', otherwise we attach a dummy child to x in F'. In this case, we still associate the offspring with the dummy child, and in our upcoming considerations we will ignore that this search point does belong to $X_{\geq i}$. If x^t is not in $X_{\geq i}$ while y^t is, we add y^t as a new root r_t to F', otherwise we add a new dummy root to F'. For every node $x \in F'$ that is a dummy node (that has no corresponding node in F) or whose copy in F has been removed from the population, we add another dummy node as its child with probability $1/\mu$. In this way, for each vertex in F' we create a new child with probability $1/\mu$ and a root is added in each round. On the other hand, by construction, F_i is a subgraph of F' at all times. □

Note that the search points associated with the vertices in F' are obtained from the root by mutation only, without any interfering selection step. This makes the process easy to analyze. Such a selection-free mutation process has been analyzed before, e.g. [69]. In Lemma 11 we show several useful properties of F'. Due to the coupling from F_i to F', the properties will also hold for F_i as well.

Lemma 11. *F' satisfies the following properties:*

1. *Let s_t denote the number of vertices in F' in round t, then $\Pr[s_t \geq S] \leq te^{t/\mu}/S$ for all $S > 0$.*
2. *Let x be a search point that corresponds to a vertex in F' of depth at most d with root y. Then for $k \geq 2edc$,*
$$\Pr[x \text{ and } y \text{ differ in more than } k \text{ bits}] \leq 2^{-k}.$$
3. *Let x be a search point that corresponds to a vertex in F' of depth larger than d with root y. If n is sufficiently large and $\textsc{Om}(y) \geq (1-8\varepsilon)n$ then*
$$\Pr[x \text{ has more one-bits than } y] \leq 2e^{-dc/32}$$
and
$$\Pr[y \text{ has less than } dc/16 \text{ more one-bits than } x] \leq 2e^{-dc/128}. \tag{2.6}$$
If n is sufficiently large and $\textsc{Om}(y) \leq (1-8\varepsilon)n$ then $\Pr[\textsc{Om}(x) \geq (1-4\varepsilon)n] \leq 2 \cdot 2^{-\varepsilon n}$.

4. *Let s_t^d denote the number of vertices of depth d in round t for an arbitrary tree from F'. Then*

$$\mathbb{E}[s_t^d] \leq \frac{t^d}{d!\mu^d}.$$

In particular, for $t = O(\mu \log \mu)$ the depth of the tree is at most $e \log \mu$ with probability $1 - \mu^{-\Omega(1)}$. Moreover, if $t \geq 2d\mu$, $\sum_{i=0}^{d} \mathbb{E}[s_t^i] \leq 2t^d/(d!\mu^d)$.

Proof. We prove the statements one by one.

1. In t rounds we have added t roots to the forest, and we will give a uniform bound for all of them. So we fix a root and denote by σ_τ the number of vertices in this tree in round τ, where $0 \leq \tau \leq t$. We assume pessimistically that the root is introduced in round 0. Then we have $\sigma_0 = 1$ and $\mathbb{E}[\sigma_{\tau+1} \mid \sigma_\tau] = (1 + 1/\mu)\sigma_\tau$ for $0 \leq \tau \leq t-1$. By linearity of expectation, we have $\mathbb{E}[\sigma_t] \leq (1+1/\mu)^t$. Since there are t roots, and using that $(1+1/\mu)^\mu \leq e$, we obtain

$$\mathbb{E}[s_t] \leq t\,\mathbb{E}[\sigma_t] \leq t(1+1/\mu)^t \leq te^{t/\mu}.$$

By Markov's inequality, it holds that

$$\Pr[s_t \geq S] \leq \frac{\mathbb{E}[s_t]}{S} \leq \frac{te^{t/\mu}}{S}.$$

2. Let y_i be the i-th bit in y, the event $y_i \neq x_i$ implies that the i-th bit is flipped at least once. Denote by $d' \leq d$ the distance between x and y. By a union bound

$$\begin{aligned}\Pr[y_i \neq x_i] &\leq \Pr[\text{bit } i \text{ is flipped at least once}] \\ &\leq d' \Pr[\text{bit } i \text{ is flipped in one mutation}] \leq dc/n.\end{aligned}$$

Let $D = \{i \in [n] \mid y_i \neq x_i\}$ be the set of bits that y and x disagree. Then its expected size is $\mathbb{E}[|D|] \leq dc$. Since the bits are modified independently, we can apply Chernoff's inequality for $k \geq 2edc \geq 2e\,\mathbb{E}[|D|]$,

$$\Pr[|D| \geq k] \leq 2^{-k}.$$

3. Let the depth of x be $d' \geq d$. First we argue that we may assume $d' \leq n/(16ec)$. If $d' \geq n/(16ec)$, then consider just the last $n/(16ec)$

steps. In these, every bit has a constant probability to be touched exactly once, and a constant probability not to be touched at all. If the number of one-bits before the last $n/(16ec)$ steps was at least $n/2$, then with probability $1 - e^{-\Omega(n)}$, x has at least $8\varepsilon n$ zero-bits due to the first case, and if the number of one-bits was at most $n/2$ then the second case gives at least $8\varepsilon n$ zero-bits. In either case, x has more zero-bits than y with sufficiently large probability. So we may assume $d' \leq n/(16ec)$.

We then consider the case $\textsc{Om}(y) \geq (1 - 8\varepsilon)n$. Let B_{01} be the number of bits flipped from 0 to 1. Then similarly as for Property 2 we bound $\mathbb{E}[B_{01}]$ by

$$\begin{aligned}
&|\{i \mid y_i = 0\}| \cdot \Pr[x_i = 1 \mid y_i = 0] \\
&\leq |\{i \mid y_i = 0\}| \cdot \Pr[\text{bit } i \text{ flipped at least once in } d' \text{ mutations}] \\
&\leq 8\varepsilon n \cdot (d'c/n) = 8\varepsilon c d',
\end{aligned}$$

where the second inequality follows from a union bound. Similarly, let B_{10} be the number of bits flipped from 1 to 0 in d' mutations, its expectation $\mathbb{E}[B_{10}]$ is

$$\begin{aligned}
&|\{i \mid y_i = 1\}| \cdot \Pr[x_i = 0 \mid y_i = 1] \\
&\geq |\{i \mid y_i = 1\}| \cdot \Pr[\text{bit } i \text{ flipped exactly once in } d' \text{ mutations}] \\
&\geq \frac{n}{2}\binom{d'}{1}\frac{c}{n}\left(1 - \frac{c}{n}\right)^{d'-1} \geq \frac{d'c}{2}\left(1 - \frac{c}{n}d'\right) \geq \frac{d'c}{4}.
\end{aligned}$$

Since all bits contribute independently, we may apply the Chernoff bound. With probability at least $1 - e^{-d'c/32}$ each, we have $B_{01} \leq cd'/8$ and $B_{10} \geq cd'/8$. Both inequalities together imply that $\textsc{Om}(x) \leq \textsc{Om}(y)$ as desired, and the probability that at least one of the inequalities is violated is at most $2e^{-d'c/32} \leq 2e^{-dc/32}$.

Similarly, the probability that B_{01} (B_{10}) overshoot (undershoot) its expectation by more than $d'c/16$ is at most $e^{-d'c/128}$. Therefore, the probability that $B_{01} \geq B_{10} - d'c/16$ is at most $2e^{-d'c/128} \leq 2e^{-dc/128}$.

For the second statement, assume $\textsc{Om}(y) \leq (1 - 8\varepsilon)n$, and consider the first vertex x' on the path from y to x such that $\textsc{Om}(x') \geq (1 - 6\varepsilon)n$. The probability that more than εn bits were flipped in the creation of x' is at most $2^{-\varepsilon n}$ by the Chernoff bound, since by definition of x' the parent of x' has an Om-value smaller than $(1 - 6\varepsilon)n$, we may assume that $\textsc{Om}(x') \leq (1 - 5\varepsilon)n$. Then, starting from x' we may

use the same calculation as above, only that we need to bound the probability that εn more zero-bits than one-bits are flipped. This is bounded by the probability that $B_{01} \geq \varepsilon n$. Since $d' \leq n/(16ec)$ we have $\varepsilon n \geq 16\varepsilon c d' \geq 2e\,\mathbb{E}[B_{01}]$, by the Chernoff bound, this probability is at most $2^{-\varepsilon n}$.

4. There can only be one root in a tree, so $s_t^0 = 1$ for all $t \geq 0$. For $d \geq 1$ and $t \geq 1$, it holds that

$$s_t^d = s_{t-1}^d + \sum_{i=1}^{s_{t-1}^{d-1}} Y_i,$$

where Y_i is an indicator variable that takes value 1 if the i-th vertex of depth $d-1$ creates a offspring in round t. By Wald's equation, we obtain

$$\mathbb{E}[s_t^d] = \mathbb{E}[s_{t-1}^d] + \mathbb{E}[s_{t-1}^{d-1}]/\mu.$$

Plugging in $\mathbb{E}[s_t^d] = 0$ for all $t < d$, we can derive that

$$\mathbb{E}[s_t^d] = \sum_{i=d-1}^{t-1} \mathbb{E}[s_i^{d-1}]/\mu \tag{2.7}$$

and for all $t \geq d \geq 1$.

We show that the result by induction. For $d = 1$, by equation (2.7) we have $\mathbb{E}[s_t^1] = t/\mu$ for all $t \geq 1$. Now assume that $\mathbb{E}[s_t^d] \leq t^d/(d!\mu^d)$ for all $t \geq d$ where $d \geq 1$, again by equation (2.7) it holds that

$$\mathbb{E}[s_t^{d+1}] \leq \sum_{i=d}^{t-1} \frac{i^d}{d!\mu^d}\frac{1}{\mu} = \frac{1}{d!\mu^{d+1}} \sum_{i=0}^{t-1} i^d \leq \frac{1}{d!\mu^{d+1}} \sum_{i=0}^{t-1} t^d = \frac{t^{d+1}}{(d+1)!\mu^{d+1}}$$

for all $t \geq d+1$.

Now consider $t = O(\mu \log \mu)$ and $d = k \log \mu$ for some constant $k > e$. With Stirling's approximation $d! = \sqrt{2\pi d}(d/e)^d$, $\mathbb{E}[s_t^d] = O(\mu^{k(1-\log k)})$. By Markov's inequality, $\Pr[s_t^d \geq 1] = O(\mu^{k(1-\log k)}\sqrt{\log \mu})$ $= \mu^{-\Omega(1)}$ as $k(1-\log k) < 0$. The conclusion follows from that $s_t^d = 0$ implies a depth smaller than d.

For the last statement, let $a_t^d := t^d/(d!\mu^d)$. If $t \geq 2d\mu$, $a_t^{d-1}/a_t^d = d\mu/t \leq 1/2$ for $d \geq 1$. Therefore,

$$\sum_{i=0}^{d} \mathbb{E}[s_t^i] \leq \sum_{i=0}^{d} a_t^i \leq \sum_{i=0}^{d} 2^{-(d-i)} a_t^d < 2a_t^d = \frac{2t^d}{d!\mu^d}. \qquad \square$$

2.5.2.3 *Tail bound on steps of Z_i*

The first consequence of the coupling is an exponential tail bound on the difference $Z_i - Z_{i-1}$. Note that the tail bound only holds in one direction. There is no comparable tail bound for $Z_{i-1} - Z_i$, at least not without further knowledge on X_{i-1}: if there is a single search point $x \in X_{i-1}$ that has k more one-bits than all other search points in X_{i-1}, then x might not spawn an offspring and Z_i could drop by k or more, and k could be as large as $\Omega(n)$ without assumptions on X_{i-1}.

Lemma 12. *For all $0 < \alpha, \eta < 1$, $c > 0$ there is a constant $\mu_0 \in \mathbb{N}$ such that the following holds for all $\mu_0 \le \mu \le n$, where n is sufficiently large. Let $i \in [(1-2\varepsilon)\alpha n, (1-\varepsilon/2)\alpha n]$, where $\varepsilon = \mu^{-1+\eta}$. Assume that the $(\mu+1)$-EA with mutation rate c/n on the linear function f_ℓ satisfies $Z_{i-1} \ge (1-4\varepsilon)n$. Then for all $1 \le \beta \le \varepsilon n / \log^2 \mu$ and $C_2 = 6400e^{\alpha c+1}$,*

$$\Pr[Z_i - Z_{i-1} \ge \beta \cdot C_2 \log \mu] \le 2^{-\beta}.$$

If on the other hand $Z_{i-1} < (1-4\varepsilon)n$, then $Z_i < (1-2\varepsilon)n$ with probability $1 - e^{-\Omega(\varepsilon n/\log^2 \mu)}$.

Proof. By Lemma 9, there is $C = 16e^{\alpha c}$ such that for all $\beta \ge 1$,

$$\Pr[T_i - t_i \ge (\beta+2) \cdot C\mu \log \mu] \le \tfrac{1}{4}2^{-\beta}.$$

By Lemma 11.1, at round $t = (\beta+2) \cdot C\mu \log \mu$ we have

$$\Pr[s_t \ge \mu^{2(\beta+2)C}] \le \frac{t}{\mu^{2(\beta+2)C}} \le (\beta+2)C\mu^{1-2(\beta+2)C} \log \mu < \tfrac{1}{4}2^{-\beta},$$

where the last step holds for all $\mu \ge \mu_0$ if μ_0 is sufficiently large. That is, the probability that the algorithm visits at least $\mu^{2(\beta+2)C}$ vertices in $X_{\ge i}$ is at most $\frac{1}{4}2^{-\beta}$.

From now on, we consider F' at a time when it has at most $\mu^{2(\beta+2)C}$ vertices. Let x be a search point that corresponds to a vertex in F' of depth at most $d = \beta C_2' \log \mu$ with root r, where $C_2' = 200C/c$. By Lemma 11.2, for $C_2 = 400eC$ it holds for large enough μ_0 that

$$\Pr[x \text{ and } r \text{ differ in more than } \beta C_2 \log \mu \text{ bits}] \le 2^{-\beta C_2 \log \mu} \le \tfrac{1}{4}2^{-\beta} \cdot \mu^{-2(\beta+2)C}.$$

By a union bound over all vertices in F', the probability that there exists such a vertex x among them is at most $\frac{1}{4}2^{-\beta}$.

Now let x be a search point that corresponds to a vertex in F' of depth larger than d with root r. For large enough μ_0 by Lemma 11.3, if $\textsc{Om}(r) \geq (1-8\varepsilon)n$ then

$$\Pr[x \text{ has more one-bits than } r] \leq 2e^{-dc/32} \leq \tfrac{1}{8}2^{-\beta} \cdot \mu^{-2(\beta+2)C}.$$

The probability that there exists such a vertex x in F' is at most $\frac{1}{8}2^{-\beta}$ by a union bound. On the other hand, if n is sufficiently large and $\textsc{Om}(r) \leq (1-8\varepsilon)n$ then for $\beta \leq \varepsilon n/\log^2 \mu$,

$$\Pr[\textsc{Om}(x) \geq (1-4\varepsilon)n] \leq 2 \cdot 2^{-\varepsilon n} \leq \tfrac{1}{8}2^{-\beta} \cdot \mu^{-2(\beta+2)C},$$

Similarly, the probability that such a vertex x exists in F' is at most $\frac{1}{8}2^{-\beta}$.

To summarize, we have shown that each of the following four events happens with probability at least $1 - 1/4 \cdot 2^{-\beta}$.

- $\mathcal{E}_1$: $T_i - t_i < (\beta+2)C\mu\log\mu$.
- $\mathcal{E}_2$: $s_t < \mu^{2(\beta+2)C}$ at time $t = (\beta+2)C\mu\log\mu$.
- $\mathcal{E}_3$: Among the first $\mu^{2(\beta+2)C}$ vertices in F', there is no search point x with a distance at most $\beta C_2' \log\mu$ to its root r such that $|\{i \in [n] \mid r_i \neq x_i\}| > \beta C_2 \log\mu$.
- $\mathcal{E}_4$: Among the first $\mu^{2(\beta+2)C}$ vertices in F', there is no search point x with a distance larger than $\beta C_2' \log\mu$ to its root r such that either $\textsc{Om}(r) \geq (1-8\varepsilon)n$ and $\textsc{Om}(x) > \textsc{Om}(r)$ or $\textsc{Om}(r) \leq (1-8\varepsilon)n$ and $\textsc{Om}(x) \geq (1-4\varepsilon)n$.

Now we argue how the bounds for these events imply the lemma. By $\mathcal{E}_1$ and $\mathcal{E}_2$, we may restrict ourselves to the first $\mu^{2(\beta+2)C}$ vertices in F'. We claim that there are no offspring x in distance at most $\beta C_2' \log\mu - 1$ from their root r that have Om-value larger than $Z_{i-1} + \beta C_2 \log\mu$. To see this, we add the parent of r, $r' \in X_{<i}$, and the edge between r' and r to F'. Now r' is the root of x and it can act as a reference point: by the definition of Z_{i-1} we have $Z_{i-1} \geq \textsc{Om}(r')$. If the distance from r' to x is at most $\beta C_2' \log\mu$, by $\mathcal{E}_3$ we have $\textsc{Om}(x) \leq \textsc{Om}(r') + \beta C_2 \log\mu$. If x is of larger distance from the added root r', we need to discriminate two cases. Either r' has Om-value at least $(1-8\varepsilon)n$ in which case $\textsc{Om}(x)$ do not exceed $\textsc{Om}(r')$ by the first part of $\mathcal{E}_4$. Or r' has Om-value at most $(1-8\varepsilon)n$, in which case x do not exceed a Om-value of $(1-4\varepsilon)n$ by the second part of $\mathcal{E}_4$. Therefore, if $Z_{i-1} \geq (1-4\varepsilon)n$, we can conclude that Om-value of x do not exceed Z_{i-1}

in both cases. Hence, we have shown that $\textsc{Om}(x) - Z_{i-1} > \beta \cdot C_2 \log \mu$ is only possible if at least one of the events $\mathcal{E}_1$ - $\mathcal{E}_4$ does not occur, and thus

$$\Pr[Z_i - Z_{i-1} > \beta \cdot C_2 \log \mu] \leq \sum_{j=1}^{4} (1 - \Pr[\mathcal{E}_j]) \leq 2^{-\beta}.$$

If $Z_{i-1} < (1-4\varepsilon)n$, with the same arguments and letting $\beta \geq \varepsilon n/(\log^2 \mu)$ we have $Z_i < (1-2\varepsilon)n$ with probability $1 - 2^{-\Omega(\varepsilon n/\log^2 \mu)}$. □

2.5.3 *Typical situations*

As outlined in the overview, our analysis of the drift will be based on studying what happens in 'typical' situations. To characterize these, we use the following definition of 'good' events. Again we consider the $(\mu+1)$-EA on the linear function f_ℓ. For parameters $\phi, c_d, c_e > 0$ we define the event $\mathcal{E}_{\text{good}}(i) := \mathcal{E}_a \cap \mathcal{E}_b \cap \ldots \cap \mathcal{E}_e$, where $\mathcal{E}_a$ etc. are the following events about the family forest F_i of rank i. Recall the family forest consists of all $x \in X_{\geq i}$, and a vertex u is a child of v if u was created as an offspring of v. We will be concerned about those vertices in the family forest in X_i, i.e., vertices of rank exactly i.

- $\mathcal{E}_a$: No vertex in $X_{\leq i-1}$ creates offspring in $X_{\geq i+1}$.
- $\mathcal{E}_b$: There are at most $\varepsilon\mu \log^3 \mu$ roots in F_i.
- $\mathcal{E}_c$: No vertex in X_i of depth at most $\phi \log \mu$ in F_i creates offspring in $X_{>i}$.
- $\mathcal{E}_d$: For every vertex $x \in X_i$ that creates an offspring in $X_{\geq i+1}$, if the root r of x has $\textsc{Om}(r) \geq (1-8\varepsilon)n$ then $\textsc{Om}(x) \leq \textsc{Om}(r) - c_d \log \mu$, and if $\textsc{Om}(r) \leq (1-8\varepsilon)n$ then $\textsc{Om}(x) \leq (1-4\varepsilon)n$. Moreover, the mutation changes at most $c_d/2 \cdot \log \mu$ bits.
- $\mathcal{E}_e$: No vertex in X_i has an Om-value which exceeds the Om-value of its root in F_i by more than $c_e \log \mu$.

Lemma 13. *For every $0 < \alpha < 1$, $c > 0$ there are $c_d, c_e > 0$ such that the following holds. For any constant parameters $0 < \phi < 1$ and $\eta > 0$ that satisfy the following conditions, where $g(\phi) = \phi(\log(8e^{\alpha c+1}) - \log \phi)$,*

$$\eta < \min\left\{g(\phi),\ \frac{1}{2} - g(\phi),\ \frac{c\phi}{128},\ \frac{c_d}{6}\right\}, \tag{2.8}$$

there exists μ_0 such that for all $\mu_0 \le \mu \le n$ and all $i \ge (1-8\varepsilon)\alpha n$, the $(\mu+1)$-EA on f_ℓ satisfies

$$\Pr\left[\mathcal{E}_{\text{good}}(i)\right] \ge 1 - O(\log^{-2}\mu).$$

We remark that $g(\phi) > 0$ for $0 < \phi < 1$ and $g(\phi) < 1/2$ for small enough ϕ, so there exists $\eta > 0$ that satisfies (2.8).

Proof. We need to show that $\Pr[\mathcal{E}] = 1 - O(\log^{-2}\mu)$ holds for $\mathcal{E} = \mathcal{E}_a, \dots, \mathcal{E}_e$. Thus we split the proof into five parts. Note that we actually show the stronger statement $\Pr[\mathcal{E}] = 1 - \mu^{-\Omega(1)}$ for $\mathcal{E} = \mathcal{E}_a, \mathcal{E}_c, \mathcal{E}_d, \mathcal{E}_e$.

$\mathcal{E}_a$: No vertex in $X_{\le i-1}$ creates offspring in $X_{\ge i+1}$ for $\eta < 1/2$.

We consider the number of offspring that are created from points in $X_{\le i-1}$ and are members of $X_{\ge i+1}$ after the first point x in $X_{\ge i}$ is created.

We first argue that the probability that $x \in X_i$ is $1 - O(\varepsilon)$. Since we assume $i \ge (1-8\varepsilon)\alpha n$ and the rank of x is at least i, the density of zero-bits is $d(A_{\ell+1}, x) \le 8\varepsilon$. By Lemma 3,

$$\frac{\Pr[x \in X_{\ge i+1}]}{\Pr[x \in X_{\ge i}]} \le 16\varepsilon\alpha c,$$

which implies $\Pr[x \in X_i \mid x \in X_{\ge i}] = 1 - O(\varepsilon) = 1 - O(\mu^{\eta-1})$.

By Lemma 9, after the first search point in X_i is created, with probability $1 - O(\mu^{-\Omega(1)})$ it takes at most $T = 8e^{\alpha c}\mu\log\mu$ rounds until the set $X_{\le i}$ is completely deleted. If a search point in $X_{\le i-1}$ creates an offspring in $X_{\ge i+1}$, at least 2 zero-bits need to be flipped. This probability is $O(\varepsilon^2)$ by a union bound, and hence the expected number of offspring in $X_{\ge i+1}$ created from $X_{\le i-1}$ is at most $O(\varepsilon^2 T) = O(\mu^{-1+2\eta}\log\mu)$. Since $\eta < 1/2$, by Markov's inequality, the probability that the number of such offspring is at least 1 can be bounded by $O(\mu^{-1+2\eta}\log\mu) = O(\log^{-2}\mu)$, as required.

$\mathcal{E}_b$: There are at most $\varepsilon\mu\log^3\mu$ roots in F_i.

We know from $\mathcal{E}_a(i-1)$ that we may assume that no points in X_i are created from $X_{\le i-2}$. Hence, it suffices to count the number of roots in X_i that are created from X_{i-1}. As in the proof for $\mathcal{E}_a$, by Lemma 9, after the first search point in X_{i-1} is created, with probability $1 - \mu^{-\Omega(1)}$ it takes at most $T = 8e^{\alpha c}\mu\log\mu$ rounds until the set $X_{\le i-1}$ is completely deleted. In each round we have a probability of at most $p_U = O(\varepsilon)$ to create a new root in X_i (p_U defined in Lemma 3), so the expected number of roots in X_i is $O(\varepsilon T)$. By Markov's inequality, the number of roots is at most $\varepsilon\mu\log^3\mu$ with probability $O(\varepsilon T/(\varepsilon\mu\log^3\mu)) = O(\log^{-2}\mu)$.

$\mathcal{E}_c$: No vertex in X_i of depth at most $\phi \log \mu$ in F_i creates offspring in $X_{>i}$.

As a sketch for the proof, we first show that the number of vertices of depth at most $\phi \log \mu$ in F_i is at most $\mu^{2g(\phi)}$ with high probability. Then by a simple estimation, the expected number of offspring in $X_{>i}$ created by those vertices is $O(\mu^{-1+2\eta+2g(\phi)} \log^4 \mu)$. Since $g(\phi) < 1/2$ for small enough ϕ, for $\eta < 1/2 - g(\phi)$ with probability $1 - O(\mu^{-\Omega(1)})$ no such offspring is created.

By Lemma 10, we couple F_i with F'. Since by $\mathcal{E}_b$ there are at most $\varepsilon\mu \log^3 \mu$ roots in F_i, we only need to consider $\varepsilon\mu \log^3 \mu$ trees in F'.

Recall that by Lemma 9, the lifetime of X_i is at most $T := 8e^{\alpha c}\mu \log \mu$ with probability at least $1 - \mu^{-\Omega(1)}$, if $\mu \geq \mu_0$ for a sufficiently large μ_0. Hence, it suffices to study F' after T rounds. We want to bound the number of vertices with depth at most $\phi \log \mu$. We fix a root, and consider the tree attached to this root. By Property Lemma 11.4 and by the Stirling formula $k! = \Theta(\sqrt{k}(k/e)^k)$ int he second step, the expected number of vertices with depth at most $\phi \log \mu$ at round T is

$$\begin{aligned}\sum_{d=0}^{\phi \log \mu} \mathbb{E}[s_T^d] < 2\,\mathbb{E}[s_T^{\phi \log \mu}] &= \frac{2T^{\phi \log \mu}}{(\phi \log \mu)!\mu^{\phi \log \mu}} \\ &= \Theta\Big(\frac{(8e^{\alpha c}\mu \log \mu)^{\phi \log \mu}}{\sqrt{\log \mu}(\phi \log \mu / e)^{\phi \log \mu}\mu^{\phi \log \mu}}\Big) \\ &= o(\mu^{\phi(\log(8e^{\alpha c+1}) - \log \phi)}) = o(\mu^{g(\phi)}).\end{aligned}$$

Note that for $0 < \phi < 1$ we have $g(\phi) > 0$. By Markov's inequality,

$$\Pr\left[\sum_{d=0}^{\phi \log \mu} s_T^d \geq \mu^{2g(\phi)}\right] = o\big(\mu^{-g(\phi)}\big).$$

Since we consider $\varepsilon\mu \log^3 \mu = \mu^{\eta} \log^3 \mu$ trees in F', by a union bound over all trees, with probability at least $1 - o(\mu^{\eta - g(\phi)} \log^3 \mu)$ the number of vertices with depth at most $\phi \log \mu$ is at most $\mu^{\eta+2g(\phi)} \log^3 \mu$. Note that the error probability is $o(1)$ since we assumed that $\eta < g(\phi)$.

In each round, every such vertex has a probability of at most $O(\varepsilon/\mu)$ to create an offspring of strictly larger rank: it must be selected as parent and its offspring must have strictly larger rank. Since the vertices in X_i are present for at most $T = 8e^{\alpha c}\mu \log \mu$ rounds, the expected number of offspring in $X_{\geq i+1}$ created by vertices in X_i of depth at most $\phi \log \mu$ is $O(T \cdot \varepsilon/\mu \cdot \mu^{\eta+2g(\phi)} \log^3 \mu) = O(\mu^{-1+2\eta+2g(\phi)} \log^4 \mu)$. By Markov's inequality, the probability that the number of such offspring is at least 1 is

$O(\mu^{-1+2\eta+2g(\phi)}\log^4\mu)$. Since $g(x)$ is monotonically increasing in $(0,1)$ and $g(0)=0$, $\eta < 1/2 - g(\phi)$ holds for small enough constant ϕ, making the error probability $\mu^{-\Omega(1)}$. Hence, we have shown that with sufficiently small probability the vertices in depth at most $\phi\log\mu$ do not create offspring in $X_{>i}$.

$\mathcal{E}_d$: For every vertex $x \in X_i$ that creates an offspring in $X_{\geq i+1}$, if the root r of x has $\textsc{Om}(r) \geq (1-8\varepsilon)n$ then $\textsc{Om}(x) \leq \textsc{Om}(r) - c_d\log\mu$, and if $\textsc{Om}(r) \leq (1-8\varepsilon)n$ then $\textsc{Om}(x) \leq (1-4\varepsilon)n$. Moreover, the mutation changes at most $c_d/2\cdot\log\mu$ bits.

If $\mathcal{E}_c$ holds, the vertices in X_i that create offspring in $X_{\geq i+1}$ must be of distance at least $d = \phi'\log\mu$ where $\phi' > \phi$ from their roots. Consider a root r with $\textsc{Om}(r) \geq (1-8\varepsilon)n$. By equation (2.6) in Lemma 11, for $c_d = c\phi/16$, $\Pr[\textsc{Om}(x) - \textsc{Om}(r) \geq -c_d\log\mu] \leq 2e^{-c_d\log\mu/8} = 2\mu^{-M}$, with $M := c\phi/128$. If $\mathcal{E}_b(i+1)$ holds, the number of offspring in $X_{\geq i+1}$ created by points in X_i is at most $\varepsilon\mu\log^3\mu$, which means the number of points in X_i that create offspring in $X_{\geq i+1}$ is at most $\varepsilon\mu\log^3\mu$. By a union bound, with probability at least $1 - O(\varepsilon\mu\log^3\mu\cdot 2\mu^{-M}) = 1 - O(\mu^{-M+\eta}\log^3\mu)$, a vertex in X_i that creates an offspring in $X_{\geq i+1}$ has a Om-value which is at least $c_d\log\mu$ smaller than that of its root. Since we assumed $\eta < M$, this probability is $1-\mu^{-\Omega(1)}$, and thus sufficiently large. This concludes the case that the root has Om-value at least $(1-8\varepsilon)n$.

If a vertex x has a root which has at most Om-value $(1-8\varepsilon)n$, we consider the first vertex x' of Om-value at least $(1-6\varepsilon)n$ on the path from the root to x. Then we know that x' has Om-value at most $(1-5\varepsilon)n$, since its direct parent has Om-value less than $(1-6\varepsilon)n$ and the probability to flip at least εn bits in one mutation is $2^{-\varepsilon n}$. Then by similar arguments as above, the probability that a descendant of x' has Om-value which is εn larger than x' is also $2^{-\varepsilon n} = \mu^{-\omega(1)}$, and thus we can easily apply a union bound over $\varepsilon\mu\log^3\mu$ vertices in X_i that create offspring in $X_{\geq i+1}$.

Finally, we come to the number of bit flips in the improving mutation. In one mutation the expected number of changed bits is c. Let $c_d/2\cdot\log\mu = (1+\delta')c$ for some $\delta' > 1$, by Chernoff bound, the probability that the number of changed bits is larger than $c_d/2\cdot\log\mu$ can be bounded by $e^{-\delta' c/3} = \Theta(\mu^{-c_d/6})$. Similarly, by a union bound, the error probability is at most $O(\varepsilon\mu\log^3\mu\cdot\mu^{-c_d/6}) = O(\mu^{\eta-c_d/6}\log^3\mu)$, which is $\mu^{-\Omega(1)}$ since $\eta < c_d/6$.

$\mathcal{E}_e$: No vertex in X_i has an Om-value which exceeds the Om-value of its root in F_i by more than $c_e\log\mu$.

We set $c_e := 2e^2ck$ where $k > 1$ is a positive constant to be chosen later and assume the distance between a vertex x and its root r is d. By Lemma 11.4, with probability $1 - \mu^{-\Omega(1)}$, $d \leq e \log \mu$, and thus $c_e \log \mu \geq 2edc$. By Lemma 11.2, the probability that x and r differ in more than $c_e \log \mu$ is at most $2^{-c_e \log \mu} = \mu^{-2e^2ck \log 2}$. Therefore, the probability that $\textsc{Om}(x)$ exceeds $\textsc{Om}(y)$ by more than $c_e \log \mu$ is at most $\mu^{-2e^2ck \log 2}$. Moreover, The lifetime of X_i is $8e^{\alpha c}\mu \log \mu$ with probability $1 - \mu^{-\Omega(1)}$ by Lemma 9. By Lemma 11.1, with probability $1 - O(\mu^{1-e^{\alpha c}} \log \mu)$ there are at most $\mu^{9e^{\alpha c}}$ vertices in F_i. By a union bound over all these vertices, the error probability is at most $O(\mu^{9e^{\alpha c} - 2e^2ck \log 2})$. By choosing $k > 9e^{\alpha c}/(2e^2c \log 2)$, this probability is $\mu^{-\Omega(1)}$. □

2.5.4 *Estimating the drift*

We are now ready to collect the information to prove negative drift of the Z_i. We first give a lemma that shows that $Z_{i+K} - Z_i$ is negative in case of good events. As outlined in the introduction, good events don't imply that $Z_{i+1} - Z_i$ is negative, we need to make K steps for some constant $K \in \mathbb{N}$.

Lemma 14. *Let $\ell \in [L]$ and $i \in \mathbb{N}$. Consider the $(\mu+1)$-EA on the linear auxiliary function $f_\ell(x) := n \sum_{j \in A_\ell} x_j + \sum_{j \in R_\ell} x_j$. Assume that in some step $t \geq 0$ the highest rank in the population is i, that $\mathcal{E}_{\text{good}}(i), \dots, \mathcal{E}_{\text{good}}(i+K)$ hold, where $K := \lceil 2(c_e+1)/c_d \rceil$, and that $\textsc{Om}(r) \geq (1-8\varepsilon)n$ holds for all roots r in $F_i, \dots, F_{i+K-1}$. Then $Z_{i+K} \leq Z_i - \log \mu$.*

Proof. Let $j \in \{i+1, \dots, i+K\}$, and let $r \in X_j$ be any root in F_j. By $\mathcal{E}_a(j-1)$, the parent individual x of r is in X_{j-1}. By $\mathcal{E}_d(j-1)$, the root r' of x in F_{j-1} satisfies $\textsc{Om}(r') \geq \textsc{Om}(x) + c_d \log \mu \geq \textsc{Om}(r) + c_d/2 \cdot \log \mu$. By induction, we obtain that for every root $r \in X_j$ there exists a root $\tilde{r} \in X_i$ such that $\textsc{Om}(r) \leq \textsc{Om}(\tilde{r}) - (j-i)c_d/2 \cdot \log \mu \leq Z_i - (j-i)c_d/2 \cdot \log \mu$, where the second step holds since $\textsc{Om}(\tilde{r}) \leq Z_i$ by definition of Z_i. Now consider any individual $\tilde{x} \in X_{i+K}$, and let $r \in X_j$ be its root. By $\mathcal{E}_e(i+K)$, we have

$$\begin{aligned} \textsc{Om}(\tilde{x}) &\leq \textsc{Om}(r) + c_e \log \mu \leq Z_i - K \cdot c_d/2 \cdot \log \mu + c_e \log \mu \\ &\leq Z_i - \log \mu, \end{aligned} \tag{2.9}$$

where the latter inequality follows from the definition of K. Since (2.9) holds for all $\tilde{x} \in X_{i+K}$, we obtain $Z_{i+K} \leq Z_i - \log \mu$, as required. □

We are now ready to prove the main theorem on the drift of Z_i. Recall that we have upper, but no lower tail bounds on $Z_i - Z_{i-1}$, cf. the comment

before Lemma 12. In order to still be able to apply the negative drift theorem later, we show that the drift is even negative if we truncate the difference $Z_{i+K} - Z_i$ at $-\log \mu$.

Theorem 15. *For every $c > 0$ there is a $\mu_0 \in \mathbb{N}$ and a $K \in \mathbb{N}$ such that for all $\mu_0 \leq \mu \leq n$ where n is sufficiently large the following holds for the $(\mu+1)$-EA with mutation parameter c on the auxiliary function f_ℓ. Assume that in some generation the fittest search point satisfies (2.3). Then*

$$\mathbb{E}[\max\{Z_{i+K} - Z_i, -\log \mu\}] \leq -1.$$

Proof. Let K be the constant from Lemma 14. Recall from Lemma 13 that the event $\mathcal{E}_{\text{good}}$ has probability $1 - O(\log^{-2} \mu)$, which is at least $1/2$ if μ is sufficiently large. By Lemma 14, the event $\mathcal{E}_{\text{good}}$ implies $Z_{i+K} - Z_i \leq -\log \mu$, so in this case the term $\max\{Z_{i+K} - Z_i, -\log \mu\}$ evaluates to $-\log \mu$. Hence, let $E_{\text{good}} := \mathbb{E}[\max\{Z_{i+K} - Z_i, -\log \mu\} \mid \mathcal{E}_{\text{good}}] \cdot \Pr[\mathcal{E}_{\text{good}}]$, it holds that

$$\begin{aligned} E_{\text{good}} &= \sum_{j=-\infty}^{\infty} \max\{j, -\log \mu\} \cdot \Pr[Z_{i+K} - Z_i = j \wedge \mathcal{E}_{\text{good}}] \\ &= (-\log \mu) \cdot \sum_{j=-\infty}^{-\lceil \log \mu \rceil} \Pr[Z_{i+K} - Z_i = j \wedge \mathcal{E}_{\text{good}}] \\ &= -\log \mu \cdot \Pr[\mathcal{E}_{\text{good}}] \leq -2, \end{aligned}$$

where the second equality holds because $\Pr[Z_{i+K} - Z_i = j \wedge \mathcal{E}_{\text{good}}] = 0$ for $j \geq -\lfloor \log \mu \rfloor$ and the last step follows from $\Pr[\mathcal{E}_{\text{good}}] \geq 1/2$ if μ is sufficiently large.

In the remainder, we will show that the term E_{good} is very close to $\mathbb{E}[\max\{Z_{i+1} - Z_i, -\log \mu\}]$. In fact, the difference is

$$\begin{aligned} &\mathbb{E}[\max\{Z_{i+K} - Z_i, -\log \mu\}] - E_{\text{good}} \\ &= \sum_{j=-\infty}^{\infty} \max\{j, -\log \mu\} \cdot \Pr[Z_{i+K} - Z_i = j \wedge \neg\mathcal{E}_{\text{good}}] \\ &\leq \sum_{j=1}^{\infty} j \cdot \Pr[Z_{i+K} - Z_i = j \wedge \neg\mathcal{E}_{\text{good}}]. \end{aligned} \tag{2.10}$$

For an arbitrary constant $C > 0$ we may define $j_0 := \lceil C \log \mu \log\log \mu \rceil$. Then we bound j by j_0 in the range $j \leq j_0$, and we bound $\Pr[Z_{i+K} - Z_i = j \wedge \neg \mathcal{E}_{\text{good}}]$ by $\Pr[Z_{i+K} - Z_i = j]$ for $j > j_0$. Since for $j > j_0$,

$$\begin{aligned}
\Pr[Z_{i+K} - Z_i = j] &\leq \Pr[Z_{i+K} - Z_i \geq j] \\
&= \Pr\left[\sum_{k=1}^{K} (Z_{i+k} - Z_{i+k-1}) \geq j\right] \\
&\leq \sum_{k=1}^{K} \underbrace{\Pr[Z_{i+K} - Z_{i+k-1} \geq j/K]}_{\leq 2^{-j/(KC_2 \log \mu)} \text{ by Lemma 12}} \\
&\leq K 2^{-j/(KC_2 \log \mu)}. \qquad (2.11)
\end{aligned}$$

We obtain

$$\begin{aligned}
(2.10) &\leq \sum_{j=1}^{j_0} j_0 \cdot \Pr[Z_{i+K} - Z_i = j \wedge \neg \mathcal{E}_{\text{good}}] + \sum_{j=j_0+1}^{\infty} j \cdot \Pr[Z_{i+K} - Z_i = j] \\
&\leq j_0 \cdot \underbrace{\Pr[\neg \mathcal{E}_{\text{good}}]}_{= O(\log^{-2} \mu)} + \sum_{j=j_0+1}^{\infty} j \cdot \underbrace{\Pr[Z_{i+1} - Z_i = j]}_{\leq K 2^{-j/(KC_2 \log \mu)} \text{ by (2.11)}} \\
&= O\big(j_0 \log^{-2} \mu\big) + O\big(\log \mu \cdot j_0 2^{-j_0/(KC_2 \log \mu)}\big),
\end{aligned}$$

where the factor $\log \mu$ in the second term appears because $\sum_{s=s_0}^{\infty} s 2^{-s/x} = O(x s_0 2^{-s_0/x})$ for $x \geq 1$, which can be seen by grouping the sum into batches of x summands. The second term is $O(\log^{-1} \mu)$ if we choose the constant $C > 0$ in the definition of $j_0 = \lceil C \log \mu \log\log \mu \rceil$ appropriately. The first term is $O(\log\log \mu \cdot \log^{-1} \mu)$. Hence, by choosing μ sufficiently large, we can make both terms smaller than $1/2$, and obtain that $\mathbb{E}[\max\{Z_{i+K} - Z_i, -\log \mu\}] \leq E_{\text{good}} + 1 \leq -1$, as desired. □

2.6 PROOF OF MAIN THEOREM

In the previous section we have analyzed the random variable Z_i, and in particular we have shown that it has negative drift. In this section we will show how our main result, the lower bound on the runtime for the $(\mu + 1)$-EA, follows from the negative drift of Z_i. The proof follows from similar ideas as in [13] and [14]. We start with a lemma that describes the behavior of the $(\mu + 1)$-EA on f_ℓ.

Lemma 16. *For every constant $0 < \delta < 2/7$ the following holds. Let $\ell \in [L]$ and consider the $(\mu+1)$-EA on f_ℓ under the assumption that $d([n], x) \geq \varepsilon(1+2\delta)$ and $d(A_{\ell+1}, x) \geq \varepsilon(1+\delta)$ hold for all x in the initial population. For $t \geq 0$, let x^t be the offspring in round t. Then with probability $1 - \exp\{-\Omega(\varepsilon n/\log^2 \mu)\}$, the following holds for all $t \leq L$.*

- $d([n], x^t) \geq \varepsilon(1+\delta)$.
- $d([n], x^t) \geq \varepsilon(1+2\delta)$ *or* $d(A_{\ell+1}, x^t) \geq \varepsilon(1+\delta/4)$.

Before we come to the proof, let us briefly explain why the lemma is useful. It is tailored to support an inductive proof for Theorem 6 for HotTopic. In this induction, we will show that $d([n], x^t) \geq \varepsilon(1+\delta)$ for exponential time. In fact, when the algorithm enters a new level then the density is at least $\varepsilon(1+2\delta)$. Moreover, one can show that whp the new hot topic did not influence the algorithm up to this point, so it behaves just as a random subset of positions of size αn. In particular, whp its density is at least $\varepsilon(1+\delta)$, so the assumptions of the lemma are satisfied. As long as the level does not change, the HotTopic function is identical to f_ℓ, so we may apply Lemma 16. The first item implies what we actually want to prove, at least as long as we stay on the same level. For the second item, by the construction of the HotTopic function the level increases when $d(A_{\ell+1}, x^t) \approx \varepsilon < \varepsilon(1+\delta/4)$. So the second item implies that at this point in time we have $d([n], x^t) \geq \varepsilon(1+2\delta)$, which is the requirement for the next step in the induction. Note that we can't just merge the items into one. For example, if we would weaken the second item to assert $d([n], x^t) \geq \varepsilon(1+\delta)$ at the beginning of a level, then we could not conclude that the next offspring satisfies the same bound with exponentially small error probability.

Proof of Lemma 16. Let i_0 be the largest rank in the initial population, i.e., the largest number of one-bits in $A_{\ell+1}$ in the initial population. We fix an offset $a \in \{0, \dots, K-1\}$ and consider the sequence of random variables $Y_{i,a} := Z_{i_0+a+iK}/\log \mu$, where i is a non-negative integer. In the initial population, each individual has at most $n\,(1-\varepsilon(1+2\delta))$ one-bits by assumption. Hence, we also have $Z_{i_0+a} \leq n(1-\varepsilon(1+3\delta/2))$ with probability $1-\exp\{-\Omega(\varepsilon n)\}$ for all offsets $a \in \{0, \dots, K-1\}$, since otherwise at least one of the K mutations would need to flip $\Omega(\varepsilon n)$ bits, which happens only with probability $\exp\{-\Omega(\varepsilon n)\}$ by the Chernoff bound. Thus for the first statement it suffices to show that $Y_{i,a} \leq Y_{0,a} + \varepsilon\delta n/(2\log \mu)$ for all $i \geq 0$. Since that is equivalent to $Z_{i_0+a+iK} \leq Z_{i_0+a} + \varepsilon\delta n/2$ for all $i \geq 0$, and we

already have $Z_{i_0+a} \leq n(1-\varepsilon(1+3\delta/2))$ for all a with high probability, altogether it implies $Z_{i'} \leq n(1-\varepsilon(1+\delta))$ for all $i' \geq i_0$. As $Z_{i'}$ denotes the maximum number of one-bits in rank i', we conclude that $d([n], x^t) \geq \varepsilon(1+\delta)$ holds for any individual x^t of rank $i' \geq i_0$. For the second statement, we distinguish between two cases. Note that the index i counts, up to the factor K, the increase in one-bits in $A_{\ell+1}$. If $i \leq \alpha n\varepsilon\delta/(4K) - 1$, then for any x^t of rank $i_0 + a + iK$, $d(A_{\ell+1}, x^t) \geq (\alpha n\varepsilon(1+\delta/2) - a - iK)/(\alpha n) > \varepsilon(1+\delta/2) - (i+1)K/(\alpha n) \geq \varepsilon(1+\delta/4)$. For $i > \alpha n\varepsilon\delta/(4K) - 1$, we aim to show that $Y_{i,a} \leq Y_{0,a}$.

We would like to apply the negative drift theorem to $Y_{i,a}$ for the range $[(1-\varepsilon(1+3\delta/2))n/\log\mu, (1-\varepsilon(1+\delta))n/\log\mu]$. First note that we study a linear function, and that the bits in A_ℓ have larger weights than the remaining bits. Thus, it can be shown by a coupling argument (Lemma 4.2 in [13]) that if $d(A_{\ell+1}, x) \leq d([n], x) + \delta\varepsilon$ holds initially, then the slightly weaker condition $d(A_{\ell+1}, x) \leq d([n], x) + 2\delta\varepsilon$ remains true for all individuals in the population for the next L rounds, with probability at least $1 - Le^{-\Omega(\varepsilon n)}$. By choosing the constant parameter ρ in the definition of $L = \exp\{\rho\varepsilon n/\log^2\mu\}$ small enough, the factor L can be swallowed by the term $e^{-\Omega(\varepsilon n)}$. Thus we may assume that whenever $Y_{i,a}$ is in the range $[(1-\varepsilon(1+3\delta/2))n/\log\mu, (1-\varepsilon(1+\delta))n/\log\mu]$ then $d(A_{\ell+1}, x) \leq d([n], x) + 2\delta\varepsilon \leq \varepsilon(1+3\delta/2+2\delta) \leq 2\varepsilon$ as $\delta < 2/7$. In addition, we have $d(A_{\ell+1}, x) \geq \varepsilon/2$ before the level changes, since otherwise with probability $1 - e^{-\Omega(\varepsilon n)}$ it holds that $d(B_{\ell+1}, x) < \varepsilon$, which implies an increase of level. Thus the conditions in (2.3) are satisfied, and thus Lemma 6 is applicable.

So let us study the drift of $Y_{i,a}$ in the range $[(1-\varepsilon(1+3\delta/2))n/\log\mu, (1-\varepsilon(1+\delta))n/\log\mu]$. First note that the probability to jump over more than half of this interval is $\exp\{-\Omega(\varepsilon n/\log^2\mu)\}$: for $Y_{i,a} \geq (1-4\varepsilon)n/\log\mu$ this follows from the first statement in Lemma 12, for $Y_{i,a} < (1-4\varepsilon)n/\log\mu$ it follows from the second statement in Lemma 12. So we may assume that $Y_{i,a}$ is contained in the first half of the interval for some $i = i^*$. To ease notation, we will assume $i^* = 0$. Inside of the interval, by Theorem 15, it holds that

$$\mathbb{E}[Y_{i+1,a} - Y_{i,a}] = \mathbb{E}[Z_{i_0+a+(i+1)k} - Z_{i_0+a+ik}]/\log\mu \leq -1/\log\mu.$$

Moreover, by Lemma 12 the sequence of random variables $(Y_{i,a})_{i\geq 0}$ has an upper exponential tail bound, i.e., $\Pr[Y_{i+1,a} - Y_{i,a} \geq K \cdot \beta C_2] \leq K \cdot 2^{-\beta}$ for all $1 \leq \beta \leq \varepsilon n/\log^2\mu$. (In particular, the probability that there is ever a jump larger than $KC_2\varepsilon n/\log^2\mu$ within L steps is at most $O(L \cdot 2^{-\varepsilon n/\log^2\mu}) = o(1)$,

so we may assume that such jumps never occur.) To show sub-Gaussianity, we should extend the inequality also for $\beta < 1$. Since any probability is bounded by 1, the bound $\Pr[Y_{i+1,a} - Y_{i,a} + 1/\log\mu \geq K\beta C_2 + 1/\log\mu] \leq 2K \cdot 2^{-\beta}$ is not just true for $\beta \geq 1$, but also trivially satisfied for any $\beta \in [0,1]$. Therefore, for any $y \geq 0$, it holds that

$$\begin{aligned}\Pr[Y_{i+1,a} - Y_{i,a} + 1/\log\mu \geq y] &\leq 2^{1+1/(KC_2\log\mu)} K (2^{1/(KC_2)})^{-y} \\ &\leq 2^{1+1/(KC_2\log 2)} K (2^{1/(KC_2)})^{-y}.\end{aligned}$$

However, we need exponential tail bounds in both directions, so we need to truncate the downwards steps of $Y_{i,a}$ as follows. We set $\tilde{Y}_{0,a} := Y_{0,a}$, and we define $\tilde{Y}_{i,a}$ recursively by $\tilde{Y}_{i,a} - \tilde{Y}_{i+1,a} := \min\{Y_{i,a} - Y_{i+1,a}, 1/\log\mu\}$. Then clearly we have $\tilde{Y}_{i,a} \geq Y_{i,a}$ for all $i \geq 0$, and $\tilde{Y}_{i,a}$ satisfies the tail bound condition that

$$\Pr[\tilde{Y}_{i,a} - \tilde{Y}_{i+1,a} - 1/\log\mu > 0] = 0.$$

Therefore, by Theorem 4, $(\tilde{Y}_{i,a} + i/\log\mu)_{i\geq 0}$ is $(128c'\delta'^{-3}, \delta'/4)$-sub-Gaussian, where $c' = 2^{1+1/(KC_2\log 2)}K$ and $\delta' = 2^{1/(KC_2)} - 1$. And by Theorem 5,

$$\Pr\Big[\max_{0\leq j\leq i}(Y_{j,a} - Y_{0,a}) \geq -i/\log\mu + y\Big] \leq \exp\Big(-\frac{\delta' y}{8}\min\Big(1, \frac{\delta'^2}{32c'}\cdot\frac{y}{i}\Big)\Big).$$

Now for any $i \geq 0$ and $y = i/\log\mu + \varepsilon\delta n/(4\log\mu)$, with probability $1 - \exp\{-\Omega(\varepsilon n/\log^2\mu)\}$ we have $Y_{i,a} \leq Y_{0,a} + \varepsilon\delta n/(4\log\mu)$. Note that $\varepsilon\delta n/(4\log\mu)$ is half of the length of the interval of interest, which implies that $Y_{i,a}$ does not go beyond the interval with high probability. Similarly, for every fixed $i \geq \alpha n\varepsilon\delta/(4K)$ and $y = i/\log\mu$ we have $Y_{i,a} \leq Y_{0,a}$ with probability $1 - \exp\{-\Omega(\varepsilon n/\log^2\mu)\}$. The proof is concluded by a union bound over all possible i. Since there are at most n possible values, this increases the error probability by a factor of n, which we can swallow in the expression $\exp\{-\Omega(\varepsilon n/\log^2\mu)\}$. □

Finally, we have collected all ingredients to prove our main result.

Proof of Theorem 6. Let $L := \exp\{\rho\varepsilon n/\log^2\mu\}$ be the number of levels. For the proof, we will consider an auxiliary run of the $(\mu+1)$-EA with a dynamic fitness function $\tilde{f}$ in which we only allow the levels to increase by one. In particular, the function $\tilde{f}$ does not only depend on the current state of the algorithm, but also on the algorithm's history. More precisely, we define an auxiliary level $\tilde{\ell}(x,t)$ of a search point x, which we only allow to increase by at most one per round. Recall that $\ell(x)$ was defined in (2.1) as

$\ell(x) = \max\{\ell' \in [L] : d(B_{\ell'}, x) \leq \varepsilon\}$. For $\tilde{\ell}(t)$, we use the same definition except that we let the maximum go over only $\ell' \leq \min\{\tilde{\ell}(t-1)+1, L\}$. I.e., we set $\tilde{\ell}(0) := 0$, and if an offspring y^t of x^t enters the population in round t, then we set $\tilde{\ell}(y^t, t) := \max\{\ell' \in [\min\{\tilde{\ell}(x^t, t-1)+1, L\}] : d(B_{\ell'}, y^t) \leq \varepsilon\}$. (If the population stays the same in round t, then we leave $\tilde{\ell}$ unchanged.) Then we define the auxiliary fitness of y^t as

$$\tilde{f}(y^t) := \tilde{\ell}(y^t, t) \cdot n^2 + \sum_{i \in A_{\tilde{\ell}(y^t,t)+1}} y_i^t \cdot n + \sum_{i \in R_{\tilde{\ell}(y^t,t)+1}} y_i^t,$$

i.e., we use the same definition as for the HotTopic function except that we replace $\ell(y^t)$ by $\tilde{\ell}(t)$. Then we proceed as the $(\mu+1)$-EA, i.e., in each round we compute and store the auxiliary fitness of the new offspring (which may depend on the whole history of the algorithm), and we remove the search point for which we have stored the lowest auxiliary fitness. This definition does not make much sense from an algorithmic perspective, but we will see in hindsight that the auxiliary process behaves identical to the actual $(\mu+1)$-EA. We will next argue why this is the case.

For the auxiliary process, it is obvious that we only need to uncover the set A_{i+1} and B_{i+1} when we reach level $\tilde{\ell}(t) = i$. As we will show later for the auxiliary process, with high probability the density $d([n], x^t)$ stays strictly above $\varepsilon \cdot (1+\delta)$ for a suitable constant $\delta > 0$. Now fix any round t with auxiliary level $\tilde{\ell}(t)$. Since we do need to uncover $B_{\tilde{\ell}(t)+2}$ at some point after time t, its choice does not influence the behavior of the auxiliary process until time t. Hence, we can first let the auxiliary process run until time t, and afterwards uncover the set $B_{\tilde{\ell}(t)+2}$. Since $B_{\tilde{\ell}(t)+2} \subset [n]$ is a uniformly random subset of size βn, it contains at least $\beta\varepsilon(1+\delta)n$ zero-bits in expectation, and the probability that $B_{\tilde{\ell}(t)+2}$ contains at most $\beta\varepsilon n$ zero-bits is $\exp\{-\Omega(\beta\varepsilon n)\}$. The same argument also holds for $B_{\tilde{\ell}(t)+3}, \dots, B_L$. Since $L = \exp\{\rho\varepsilon n / \log^2 \mu\}$ with desirably small $\rho > 0$, we can afford a union bound over all such sets and all times $t \leq L$, which is a union bound over less than $L^2 = \exp\{2\rho\varepsilon n / \log^2 \mu\}$ terms. Hence, with high probability we have $d(B_i, x^t) > \varepsilon$ for all $1 \leq t \leq L$ and all $\tilde{\ell}(t)+2 \leq i \leq L$. A straightforward induction shows that this implies $\ell(t) = \tilde{\ell}(t)$ for all $t \leq L$, and thus the $(\mu+1)$-EA behaves identical to the auxiliary process. Note that this already implies that the $(\mu+1)$-EA visits each of the L levels, which implies the desired runtime bound. It only remains to show that there is a constant $\delta > 0$ such that the auxiliary process satisfies $d([n], x^t) > \varepsilon \cdot (1+\delta)$ for all $t \leq L$.

The advantage of the auxiliary process is that we may postpone drawing $A_{\ell+1}$ until we reach level $\tilde{\ell} = \ell$. In particular, since $A_{\ell+1} \subseteq [n]$ is a uniformly random subset, we may use the same argument as before and conclude that $|d(A_{\ell+1}, x) - d([n], x)| < \delta\varepsilon$ holds with probability $1 - \exp\{-\Omega(\varepsilon n)\}$ for any constant $\delta > 0$ that we desire, and for all members x of the population when we reach level ℓ. In fact, we have exponentially small error probability, so we may afford a union bound and conclude that with high probability the same holds for all ℓ. We want to show that the auxiliary process, if running on level ℓ and starting with a population that initially satisfies $|d(A_{\ell+1}, x) - d([n], x)| < \delta\varepsilon$ for $\delta < 2/7$, maintains $d([n], x^t) \geq \varepsilon(1+\delta)$ for all new search points x^t until $t > L$.

By the first conclusion from Lemma 16, $d([n], x^t) \geq \varepsilon(1+\delta)$ holds as long as the level remains to be ℓ and $t \leq L$. When a point x reaches level $\ell+1$, by definition we have $d(B_{\ell+1}, x) < \varepsilon$. Since $B_{\ell+1}$ is a uniformly random subset of $A_{\ell+1}$, by the Chernoff bound $d(A_{\ell+1}, x) < \varepsilon(1+\delta/4)$ holds with probability $1 - \exp\{-\Omega(\varepsilon n)\}$. So we apply the second conclusion of Lemma 16 to x and conclude that $d([n], x) \geq \varepsilon(1+2\delta)$. With high probability, it holds that $d(A_{\ell+2}, x) \geq \varepsilon(1+2\delta) - \varepsilon\delta$ and the conditions in Lemma 16 are satisfied again for level $\ell+1$. By induction we obtain $d([n], x^t) \geq \varepsilon(1+\delta)$ for all $t \leq L$. As the choice of ℓ is arbitrary, we start with $\ell = 0$ and $d([n], x^t) \geq \varepsilon(1+\delta)$ holds for all $t \leq L$. This concludes the proof. □

2.7 SIMULATIONS

In this section we will illustrate the detrimental effect of large populations on $(\mu+1)$-EAs by numerical simulations. Unless otherwise stated, the parameters that we used to generate the HOTTOPIC functions are $n = 10000$, $L = 100$, $\alpha = 0.25$, $\beta = 0.05$ and $\varepsilon = 0.05$. Each data point is obtained by 10 independent runs on the same HOTTOPIC function with different random seeds. Our implementation is available at https://github.com/zuxu/MuOneEA-HotTopic.

2.7.1 *Population size*

First of all, we plot typical behaviours of evolutionary algorithms with small, medium and large population sizes. Figure 2.2 shows the distance between the optimum and the fittest point p^* in the population with respect to time. We have two metrics for the distance: the density of 0-bits in p^*

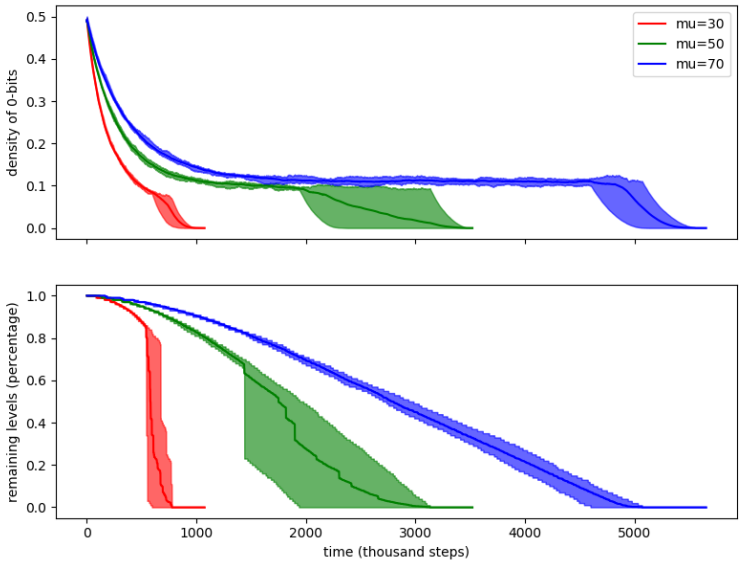

FIGURE 2.2: Distance to the optimum as $(\mu + 1)$-EAs with $c = 1.0$ proceed. The solid lines are the mean of 10 simulations, while the shaded areas are bounded by corresponding minimum and maxmium values at each time step.

and the remaining levels of p^* divided by L. As indicated by the sudden drops in level, for a small population size ($\mu = 30$), the algorithm skips many levels and reaches the optimum quickly. In contrast, an algorithm with large $\mu = 70$ visits the levels one by one, without improvement on the fitness. This happens where the density of 1-bits is relatively high, such that even though it gradually improves on the current hot topic, it often accepts offspring that flip 1-bits to 0-bits outside of the hot topic. With such offspring accumulating in a large population, the average density of 0-bits remains significantly above ε before reaching the last level. Therefore, with high probability the algorithm does not skip any level. Once the highest level is reached, the remaining bits can be optimized easily as in the coupon collector. For $\mu = 50$, the density gets close, but slightly above ε, so that it depends on chance whether levels are skipped or not. This leads to a high variance in the running time.

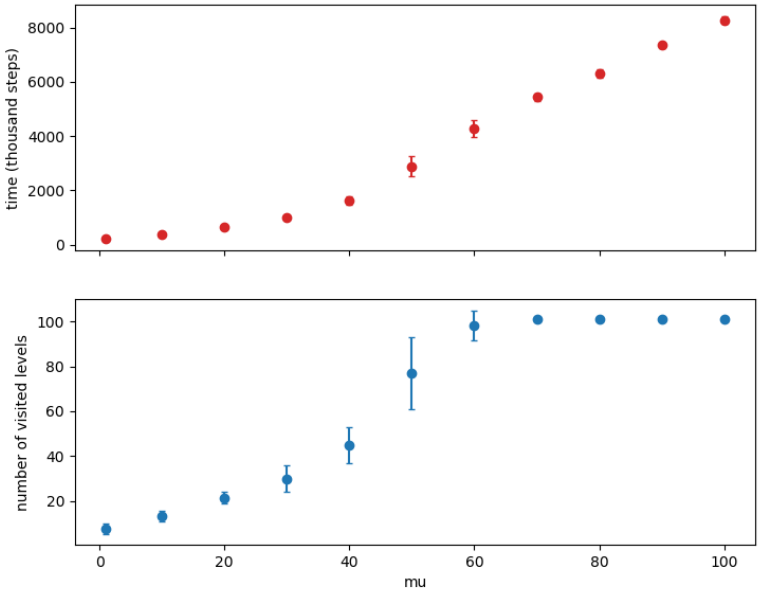

FIGURE 2.3: Running time and number of visited levels for $(\mu+1)$-EAs with different values of μ and $c = 1.0$. Solid dots indicate the means and error bars show the standard deviations.

In Figure 2.3, we show the running time and the number of visited levels for a wide range of μ. The running time is highly concentrated when μ is very small or very large. The reason is that the algorithm keeps skipping levels with small μ and visits all levels with large μ. For a medium sized μ like 50, level skipping only happens a few times. Since each time when the algorithm skips a level, it lands at some higher level uniformly at random due to the definition of the HOTTOPIC function, which results in a larger variance in the running time.

2.7.2 *Mutation rate*

The mutation rate c is the other factor that affects the magnitude of the negative drift, so we also plot the running time for various values of c, see

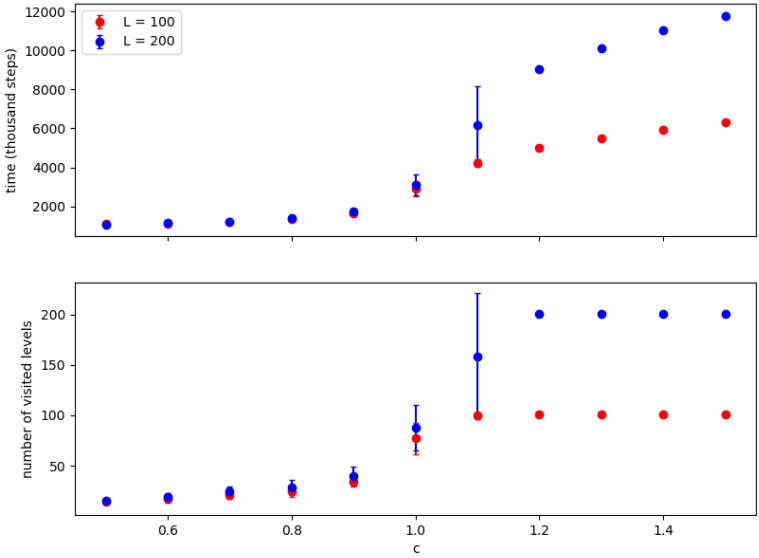

FIGURE 2.4: Running time and number of visited levels for $(\mu + 1)$-EAs with different values of c and $\mu = 50$. Solid dots indicate the means and error bars show the standard deviations.

Figure 2.4. For a small c, detrimental mutations do not occur frequently and thus the average density of 1-bits in the population keeps increasing. Conversely, with a large c, the algorithm tends to visit all the levels. To demonstrate the resulting effect on the running time, we compare the cases where $L = 100$ and $L = 200$. If c is small ($c \leq 0.9$), the algorithm skips levels quickly, and the running time is almost independent of the number of levels. On the other hand, if c is large ($c \geq 1.2$) then the algorithm visits every level. In this case, the running time is essentially proportional to the number of levels, plus some initial phase. Note that in this range the running time can get almost arbitrarily bad, since doubling the number of levels L will essentially lead to a doubling of the running time. As our theoretical analysis shows, this holds even when L becomes exponential in n, but for so many levels the running time becomes too large to run experiments. Finally, for a medium sized c like 1.1, level skipping only happens a few times. Each time when the algorithm skips a level, it lands at

some higher level uniformly at random, which results in a larger variance in the running time, similar to Figure 2.3.

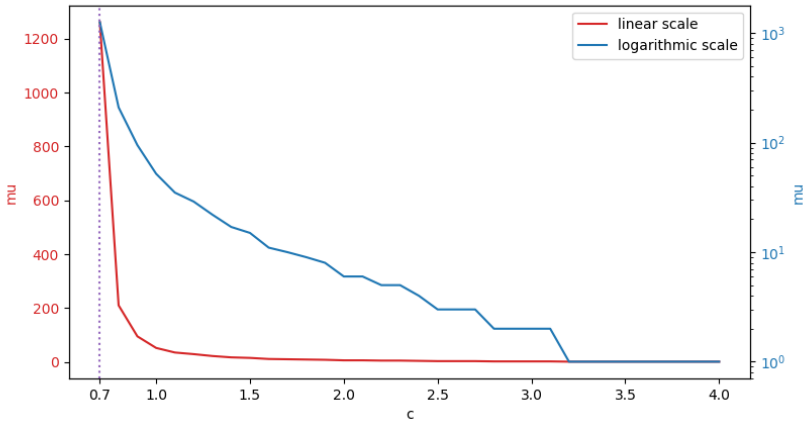

FIGURE 2.5: Minimum values of μ with respect to c such that the $(\mu+1)$-EA visits all levels in at least 5 out of 10 simulations. The choices of c ranges from 0.7 to 4.0 with step size 0.1.

Finally, we investigate how values of μ and c jointly influence the behaviour of a $(\mu+1)$-EA. For a fixed c, we search for the minimum value of μ such that the algorithm visits all levels in at least half of the 10 simulations. That is, we seek for a minimum μ that induces long running times with constant probability. As we can see from Figure 2.5, large values of c (≥ 3.2) are extremely harmful even when there is only one individual in the population. An algorithm with a large population can benefit greatly from having a small mutation rate. We did not observe a stable slowdown for $c = 0.7$ until we raise the value of μ to more than 1000.

2.8 CONCLUSION

We have shown that the $(\mu+1)$-EA with arbitrary mutation parameter $c > 0$ needs exponential time on some monotone functions if μ is too large. This is one of the very few known situations in which even a slightly larger population size μ can lead to a drastic decrease in performance. The main reason is that, if progress is steady enough that the population does not degenerate, the search points that produce offspring are typically not

the fittest ones. We believe that this is an interesting phenomenon which deserves further investigations, also in less artificial contexts.

For example, consider the $(\mu+1)$-EA on weighted linear functions with a skewed distribution (e.g., on BINVAL), and with a fixed time budget (so that the action happens away from the optimum). It is quite conceivable that the same effect hurts performance, i.e., if the algorithm flips a high-weight bit, it will allow (almost) any offspring of this individual into the population, even though this offspring has probably fewer correct bits than other search points in the population. Does that mean that the fixed-budget performance of the $(\mu+1)$-EA on BINVAL deteriorates with increasing μ? Are the resulting individuals further away from the optimum?

An even more pressing question is about crossover. We have studied the $(\mu+1)$-EA, but do the same results also apply for the $(\mu+1)$-GA? In [14] it was shown that close to the optimum (for small values of the HOTTOPIC parameter ε) crossover helps dramatically, and that a large population size can even counterbalance large mutation parameters c. So, close to the optimum, for the $(\mu+1)$-GA the effect of large population size was beneficial, while for the $(\mu+1)$-EA it was neutral and did not affect the threshold c_0. Thus if we study the $(\mu+1)$-GA on HOTTOPIC functions with large ε, then a beneficial effect of large populations is competing with a detrimental effect. Understanding this interplay would be a major step towards a better understanding of crossover in general.

Similarly, since the problems originate in non-trivial populations, what happens if we equip the $(\mu+1)$-EA with a diversity mechanism (duplication avoidance, genotypical or phenotypical niching), and study it close to the optimum? Does it fall for the same traps? This question was already asked in [14], but our results shed additional light on the question.

Finally, it is open whether the $(\mu+1)$-EA is fast on any monotone function if it starts close enough to the optimum. i.e., for every $\mu \in \mathbb{N}$, does there exist an $\varepsilon = \varepsilon(\mu)$ such that the $(\mu+1)$-EA, initialized with a random search point with εn zero-bits, has runtime $O(n \log n)$ for every monotone function? Of course, the same question also applies to other algorithms like the $(\mu+1)$-GA and the 'fast' counterparts of the $(\mu+1)$-EA and the $(\mu+1)$-GA. Interestingly, the result in [14] that the 'fast $(1+\lambda)$-EA' with good parameters is efficient for every monotone function was only proven under this assumption, that the algorithm starts close to the optimum. So this also raises the question whether there are traps for the 'fast $(1+\lambda)$-EA' that only take effect far away from the optimum.

3

ONEMAX IS NOT THE EASIEST FUNCTION FOR FITNESS IMPROVEMENTS

This chapter is based on joint work with Marc Kaufmann, Maxime Larcher, and Johannes Lengler [70], which was presented in the *European Conference on Evolutionary Computation in Combinatorial Optimisation* (*EvoCOP 2023*).

3.1 INTRODUCTION

The OneMax function assigns to a bit string x the number of one-bits in x. Despite, or rather because of its simplicity, this function remains one of the most important unimodal benchmarks for theoretical analysis of randomized optimization heuristics, and specifically of Evolutionary Algorithms (EAs). A reason for the special role of this function is the result by Doerr, Johannsen and Winzen [30] that it is the easiest function with a unique optimum for the $(1+1)$-EA in terms of expected optimization time. This result has later been extended to many other EAs [31] and to stochastic dominance instead of expectations [32]. Easiest and hardest functions have become research topics of their own [71–73].

Whether a benchmark is easy or hard is crucial for parameter control mechanisms (PCMs) [10, 74]. Such mechanisms address the classical problem of setting the parameters of algorithms. They can be regarded as meta-heuristics which automatically tune the parameters of the underlying algorithm. The hope is that (i) optimization is more robust with respect to the meta-parameters of the PCM than to the parameters of the underlying algorithm, and (ii) PCMs can deal with situations where different optimization phases require different parameter settings for optimal performance [75–79].

To this end, PCMs often rely on an (often implicit) measure of how easy the optimization process currently is. One of the most famous examples is the $(1:s+1)$-*rule* for step size adaptation in continuous optimization [20–23]. It is based on the heuristic that improving steps are easier to find if the step size is small, but that larger step sizes are better at exploiting improvements, if improvements are found at all. Thus we have conflicting goals requiring small and large step sizes, respectively, and we need a

compromise between those goals. The $(1 : s+1)$-rule resolves this conflict by defining a *target success rate*[1] of $q_s = 1/(s+1)$, and increasing the step size if the success rate (the fraction of steps which find an improvement) is above q_s, and decreasing the step size otherwise. Thus it chooses larger step sizes in environments where improvements are easy to find, and chooses smaller step sizes in more difficult environments.

More recently, the $(1 : s+1)$-rule has been extended to parameters in discrete domains, in particular to the mutation rate [24, 25] and offspring population size [26, 27] of EAs. For the self-adapting $(1, \lambda)$-EA with the $(1 : s+1)$-rule, or SA-$(1, \lambda)$-EA for short, Hevia Fajardo and Sudholt showed in [26, 27] an interesting collection of results on OneMax. They showed that optimization is highly efficient if the success ratio s is less than one. In this case, the algorithm achieves optimal population sizes λ throughout the course of optimization, ranging from constant population sizes at early stages to almost linear (in the problem dimension n) values of λ for the last steps. On the other hand, the mechanism provably fails completely if $s \geq 18$.[2] Then the algorithm does not even manage to obtain an 85% approximation of the optimum in polynomial time.

Even more interesting than the results themselves are the reasons for failure. The problem is that for large values of s, the algorithm implicitly targets a population size λ^* with a rather small success rate.[3] However, the $(1, \lambda)$-EA is a non-elitist algorithm, i.e., the fitness of its population can decrease over time. This is particularly likely if λ is small. So for large values of s, the PCM chooses population sizes that revolve around a rather small target value λ^*. It is still guaranteed that the algorithm makes progress in successful steps, which comprise a $\approx 1/(s+1)$ fraction of all steps. But due to the small population size, it loses performance in some of the remaining $\approx s/(s+1)$ fraction of steps, and this loss cannot be compensated by the gain of successful steps.

A highly surprising aspect of this bad trade-off is that it only happens when success is too easy. If success is hard, then the target population size λ^* is also large. In this case, the losses in unsuccessful steps are limited: most of the time, the offspring population contains a duplicate of the parent, in which case the loss is zero. So counter-intuitively, *easy fitness landscapes lead to a high runtime*. For OneMax, this means that the problems do not occur close to the optimum, but only at a linear distance from the optimum. This result is implicitly contained in [26, 27] and explicitly in [28]: for every

1 Traditionally the most popular value is $s = 4$, leading to the famous *one-fifth rule* [80].
2 Empirically they found the threshold between the two regimes to be around $s \approx 3.4$.
3 See Section 2.1 in [28] for a detailed discussion of the target population size λ^*.

$s > 0$ there is $\varepsilon > 0$ such that if the SA-$(1, \lambda)$-EA with success ratio s starts within distance at most εn from the optimum, then it is efficient with high probability, i.e., with probability $1 - o(1)$.

The results by Hevia Fajardo and Sudholt in [26, 27] were for OneMax, but in [28, 29] we could show that this result holds for all monotone, and even for all *dynamic monotone functions*[4]. Only the threshold for s changes, but it is a universal threshold: there exist $s_1 > s_0 > 0$ such that for every $s < s_0$, the SA-$(1, \lambda)$-EA is efficient on *every* (static or dynamic) monotone function, while for $s > s_1$ the SA-$(1, \lambda)$-EA fails for *every* (static or dynamic) monotone function to find the optimum in polynomial time. Moreover, for all $s > 0$ there is $\varepsilon > 0$ such that with high probability the SA-$(1, \lambda)$-EA with parameter s finds the optimum of every (static or dynamic) monotone function efficiently if it starts at distance εn from the optimum. Hence, all positive and negative results from [26] are not specific to OneMax, but generalize to every single function in the class of dynamic monotone functions. We also note that this class falls into more general frameworks of partially ordered functions that are easy to optimize under certain generic assumptions [56, 81].

To summarize, small success rates (large values of s) are problematic, but only if the fitness landscape is too easy. Based on this insight, and on the aforementioned fact that OneMax is the easiest function for the $(1+1)$-EA, Hevia Fajardo and Sudholt conjectured that OneMax is the most problematic situation for the SA-$(1, \lambda)$-EA: "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, we conjecture that any s that is efficient on OneMax would also be a good choice for any other problem." In the terminology above, the conjecture says that the threshold s_0 below which the SA-$(1, \lambda)$-EA is efficient for all dynamic monotone functions, is the same as the threshold s_0' below which the SA-$(1, \lambda)$-EA is efficient on OneMax. Note that the exact value of s_0' is not known theoretically except for the bounds $1 \leq s_0' \leq 18$, but that empirically $s_0' \approx 3.4$ [26]. If the conjecture was true, then experiments on OneMax could provide parameter control settings for the SA-$(1, \lambda)$-EA that work in much more general settings.

However, in this chapter we disprove the conjecture. Moreover, our result makes it more transparent in which sense OneMax is the easiest benchmark

4 Recall that a function $f : \{0,1\}^n \to \mathbb{R}$ is monotone if flipping a zero-bit into a one-bit always improves the fitness. In the dynamic monotone setting, selection may be based on a different function in each generation, but it must always be a monotone function. The formal definition is not relevant for this chapter, but can be found in [28].

for the $(1+1)$-EA or the $(1,\lambda)$-EA, and in which sense it is not. It is the easiest benchmark in the sense that for no other function with a unique global optimum, the distance to the optimum decreases faster than for OneMax [30, 32]. However, it is *not* the easiest function in the sense that it is the easiest to make a fitness improvement, i.e., to find a successful step. Rephrased, other functions make it easier to find a fitness improvement than OneMax. For the problems of the $(1:s+1)$-rule described above, the latter variant is the important one, since the $(1:s+1)$-rule adjusts its population size based on the success probability of finding a fitness improvement.

3.1.1 *Our result*

We are far from being able to determine the precise efficiency threshold s_0' even in the simple setting of OneMax, and the upper and lower bound $1 \leq s_0' \leq 18$ are far apart from each other. Therefore, it is no option to just compute and compare the thresholds for different functions. Instead, we will identify a setting in which we can indirectly compare the efficiency thresholds for OneMax and for some other function, without being able to compute either of the thresholds explicitly. For this reason, we only study the following, rather specific setting that makes the proof feasible.

We show that there are $\varepsilon > 0$ and $s > 0$ such that with high probability the SA-$(1,\lambda)$-EA with parameter s (and suitably chosen other parameters), started at any search point at distance exactly εn from the optimum

- finds the optimum of OneMax in $O(n)$ generations;
- does not find the optimum of Dynamic BinVal in polynomial time.

The definition of the Dynamic BinVal function can be found in Section 3.2.2. The key ingredient to the proof is showing that at distance εn from the optimum, Dynamic BinVal makes it easier to find a fitness improvement than OneMax (Lemma 23). Since *easy* fitness landscapes translate into poor choices of the population size of the SA-$(1,\lambda)$-EA, and thus to *large* runtimes, we are able to find a value of s that separates the two functions: for this s and for distance εn from the optimum, the algorithm will have drift *away* from the optimum for Dynamic BinVal (leading to exponential runtime), but drift *towards* the optimum for OneMax. Since the fitness landscape for OneMax only gets harder closer to the optimum, we can then show that the drift remains positive all the way to the optimum for OneMax. A high-level sketch with more detail can be found in Section 3.3.1.

A limitation of our approach is that we start with a search point at distance εn from the optimum, instead of a uniformly random search point in $\{0,1\}^n$. This simplifies the calculations substantially, and it disproves the strong "local" interpretation of the conjecture in [26] that an s that works for OneMax in some specific part of the search space also works in the same part for all other dynamic monotone functions. Our choice leaves open whether some weaker version of the conjecture in [26] might still be true. But since our argument refutes the intuitive foundation of the conjecture, we do not think that this limitation is severe.

Another limitation is that we use a dynamic monotone function instead of a static one. So we show that OneMax is not the easiest function in the class of dynamic monotone functions, but it could still be easiest in the smaller class of static monotone functions. Again, we have decided for this option for technical simplicity. We believe that our results for Dynamic BinVal could also be obtained with very similar arguments for a static HotTopic function as introduced in [15]. However, Dynamic BinVal is simpler than HotTopic functions, and the dynamic setting allows us to avoid some technical difficulties. We thus restrict ourselves to experiments in Section 3.4 for this hypothesis, and find that OneMax indeed has a harder fitness landscape (in terms of improvement probability) than other static monotone or even linear functions, and consistently (but counter-intuitive) the SA-$(1,\lambda)$-EA chooses a higher population size for OneMax. For some values of s, this leads to positive drift and efficient runtime on OneMax, while the same algorithm has negative drift and fails on other functions.

Finally, apart from the success ratio s, the SA-$(1,\lambda)$-EA also comes with other parameters. For the *mutation rate* we use the standard choice $1/n$, and any c/n for a constant $0 < c < 1$ would also work. The *update strength* $F > 1$ is the factor by which λ is reduced in case of success, see Section 3.2.1 for details. A slight mismatch with [26] is that we choose $F = 1 + o(1)$, while [26] focused on constant F. Again, this simplifies the analysis, but the restriction does not seem crucial for the conceptual understanding that we gain in this work.

3.2 PRELIMINARIES AND DEFINITIONS

Our search space is always $\{0,1\}^n$. Throughout the chapter we will assume that $s > 0$ is independent of n while $n \to \infty$, but $F = 1 + o(1)$ will depend on n. We say that an event $\mathcal{E} = \mathcal{E}(n)$ holds *with high probability* or *whp* if $\Pr[\mathcal{E}] \to 1$ for $n \to \infty$. We will write $x = a \pm b$ as shortcut for

$x \in [a-b, a+b]$. Throughout the chapter we will measure drift *towards* the optimum, so a positive drift always points towards the optimum, and a negative drift points away from the optimum.

3.2.1 *The algorithm: SA-$(1,\lambda)$-EA*

The $(1,\lambda)$-EA is the algorithm that generates λ offspring in each generation, and picks the fittest one as the unique parent for the next generation. All offspring are generated by standard bit mutation, where each of the n bits of the parent is flipped independently with probability $1/n$. The performance of the $(1,\lambda)$-EA for static population size λ is well-understood [82, 83].

We will consider the self-adjusting $(1,\lambda)$-EA with $(1:s+1)$-success rule to control the population size λ, with success rate s and update strength F, and we denote this algorithm by SA-$(1,\lambda)$-EA. It is given by the following pseudocode. The key difference from the standard $(1,\lambda)$-EA is that the population size λ is updated at each step: whenever a fitness improvement is found, the population is reduced to λ/F and otherwise the population is increased to $\lambda F^{1/s}$. Note that the parameter λ may take non-integral values during the execution of the algorithm, and the number of children is the integer $\lfloor \lambda \rceil$ closest to λ.

One way to think about the SA-$(1,\lambda)$-EA is that for each search point x it implicitly has a *target population size* $\lambda^* = \lambda^*(x)$ such that, up to rounding, the probability to have success (the fittest of λ^* offspring is strictly fitter than the parent) equals the *target success rate* $s^* = 1/(s+1)$. The $(1:s+1)$-rule ensures that there is a drift towards λ^*: whenever $\lfloor \lambda \rceil > \lambda^*$, then λ decreases in expectation, and it increases for $\lfloor \lambda \rceil < \lambda^*$, both on a logarithmic scale. We refer the reader to Section 2.1 in [28] for a more detailed discussion.

For the results of this chapter, we will specify $s > 0$ as a suitable constant, the initial population size is $\lambda^{\text{init}} = 1$, the initial search point has exactly εn zero-bits for a given ε, and the update strength is $F = 1+\eta$ for some $\eta \in \omega(\log n/n) \cap o(1/\log n)$. We will often omit the index t if it is clear from the context.

3.2.2 *The benchmarks:* OneMax *and* Dynamic BinVal

The first benchmark, the OneMax function, counts the number of one-bits

$$\textsc{Om}(x) = \textsc{OneMax}(x) = \textstyle\sum_{i=1}^{n} x_i.$$

Algorithm 2: SA-$(1,\lambda)$-EA with success rate s, update strength F, mutation rate c/n, initial start point $x^{\text{init}} \in \{0,1\}^n$ and initial population size $\lambda^{\text{init}} = 1$ for maximizing a fitness function $f : \{0,1\}^n \to \mathbb{R}$.

Initialization:
  $x^0 \leftarrow x^{\text{init}}$; $\lambda^0 \leftarrow 1$;
Optimization:
  for $t = 0, 1, \ldots$ **do**
    Mutation:
    for $j \in \{1, \ldots, \lfloor \lambda^t \rceil\}$ **do**
      $y^{t,j} \leftarrow$ mutate(x^t) by flipping each bit of x^t independently with probability $1/n$;
    Selection:
    Choose $y^t = \arg\max\{f(y^{t,1}), \ldots, f(y^{t,\lfloor \lambda^t \rceil})\}$, breaking ties randomly;
    Update:
    if $f(y^t) > f(x^t)$ **then**
      $\lambda^{t+1} \leftarrow \max\{1, \lambda^t / F\}$;
    else
      $\lambda^{t+1} \leftarrow F^{1/s} \lambda^t$
    $x^{t+1} \leftarrow y^t$;

of $x \in \{0,1\}^n$. We also define the ZeroMax function $Z(x) := n - \textsc{Om}(x)$ as the number of zero-bits in x. Throughout this chapter, we will denote $Z^t := Z(x^t)$, and we will frequently use the scaling $\varepsilon = Z/n$.

Our other benchmark is a *dynamic* function [63]. That means that in each generation t, we choose a different function f^t and use f^t in the selection update step of Algorithm 2. We choose Dynamic BinVal or DBv [84, 85], which is the binary value function BinVal, applied to a randomly selected permutation of the positions of the input string. This function has been used to model dynamic environments [84, 85] and uncertain objectives [86]. In detail, BinVal is the function that interprets a bit string as an integer representation and returns its value, so $\textsc{BinVal}(x) = \sum_{i=1}^{n} 2^{i-1} \cdot x_i$. For the dynamic version, for each generation t we draw uniformly at random a permutation π^t of the set $\{1, \dots, n\}$. The DBv function for generation t is then defined as

$$\text{DBv}^t(x) = \sum_{i=1}^{n} 2^{i-1} \cdot x_{\pi^t(i)}.$$

Note that in the update step, a re-evaluation of the parent x^t with respect to π^t is needed when comparing the fitness of x^t and y^t.

3.2.3 *Tools*

We will use drift analysis [87] to analyze two random quantities: The distance $Z^t = Z(x^t)$ of the current search point from the optimum, and the population size λ^t (or rather, $\log \lambda^t$). We use the following drift theorems to transfer results on the drift into expected hitting times.

Theorem 17 (Tail Bound for Additive Drift [66]). *Let $(X^t)_{t \geq 0}$ be a sequence of random variables over $\mathbb{R}$, each with finite expectation and let $n > 0$. With $T = \min\{t \geq 0 : X_t \geq n \mid X_0 \geq 0\}$, we denote the random variable describing the earliest point at which the random process exceeds n, given a starting value of at least 0. Suppose there are $\varepsilon, c > 0$ such that, for all t,*

1. $\mathbf{E}[X^{t+1} - X^t \mid X^0, ..., X^t, T > t] \geq \varepsilon$, *and*
2. $|X^t - X^{t+1}| < c$.

Then, for all $s \geq \frac{2n}{\varepsilon}$,

$$\Pr(T \geq s) \leq \exp\left(-\frac{s\varepsilon^2}{8c^2}\right).$$

Theorem 18 (Negative Drift Theorem [66], [88]). *Let $(X^t)_{t\geq 0}$ be a sequence of random variables over $\mathbb{R}$, each with finite expectation and let $n > 0$. With $T = \min\{t \geq 0 : X^t \geq n \mid X^0 \leq 0\}$, we denote the random variable describing the earliest point at which the random process exceeds n, given a starting value of at most 0. Suppose there are $0 < c < n$ and $\varepsilon > 0$ such that, for all t,*

1. $\mathbf{E}[X^{t+1} - X^t \mid X^0, \ldots, X^t, T > t] \leq -\varepsilon$, *and*
2. $|X^t - X^{t+1}| < c$.

Then, for all $s \geq 0$,

$$\Pr(T \leq s) \leq s \exp\left(-\frac{n\varepsilon}{2c^2}\right).$$

To switch between differences and exponentials, we will frequently make use of the following estimates, taken from Lemma 1.4.2 – Lemma 1.4.8 in [64].

Lemma 19. 1. *For all $r \geq 1$ and $0 \leq s \leq r$,*

$$(1-1/r)^r \leq 1/e \leq (1-1/r)^{r-1} \text{ and } (1-s/r)^r \leq e^{-s} \leq (1-s/r)^{r-s}.$$

2. *For all $0 \leq x \leq 1$,*

$$1 - e^{-x} \geq x/2.$$

3. *For all $0 \leq x \leq 1$ and all $y \geq 1$,*

$$\tfrac{xy}{1+xy} \leq 1 - (1-x)^y \leq xy.$$

3.3 MAIN PROOF

We start this section by defining some helpful notation. Afterwards, we give an informal sketch of the main ideas, before we give the full proof.

Definition 20. *Consider the SA-$(1,\lambda)$-EA optimizing a dynamic function $f = f^t$, and let $Z^t = Z(x^t)$. For all times t and all $i \in \mathbb{Z}$, we define*

$$p_i^{f,t} := \Pr[Z^t - Z^{t+1} = i \mid x^t, \lambda^t] \qquad \text{and} \qquad \Delta_i^{f,t} := i \cdot p_i^{f,t}.$$

We will often drop the superscripts f and t when the function and the time are clear from context. We also define $p_{\geq i} := \sum_{j=i}^{\infty} p_j$ and $\Delta_{\geq i} := \sum_{j=i}^{\infty} \Delta_j$; both $p_{\leq i}$ and $\Delta_{\leq i}$ are defined analogously. Finally, we write

$$\Delta^{f,t} := \mathbf{E}[Z^t - Z^{t+1} \mid x^t, \lambda^t] = \sum_{i=-\infty}^{\infty} \Delta_i^{f,t}.$$

Note that $i > 0$ and $\Delta > 0$ corresponds to steps/drift *towards* the optimum and $i < 0$ and $\Delta < 0$ *away from* the optimum.

Definition 21 (Improvement Probability, Equilibrium Population Size)**.** *Let $x \in \{0,1\}^n$ and f be a strictly monotone function. Let y be obtained from x by flipping every bit independently with probability $1/n$. We define*

$$p^f_{\text{imp}}(x) := \Pr[f(y) > f(x)] \qquad \text{and} \qquad q^f_{\text{imp}}(x,\lambda) := 1 - (1 - p^f_{\text{imp}}(x))^\lambda,$$

as the probability that respectively a single offspring or any offspring improves the fitness of x. We also define the equilibrium population size *as*

$$\lambda^{*,f}(x,s) := \log_{(1-p^f_{\text{imp}}(x))}\left(\tfrac{s}{1+s}\right). \tag{3.1}$$

As usual, we drop the superscript when f is clear from context. As the two functions we consider are symmetric (i.e. all bits play the same role) p_{imp} only depends on $Z(x) = \varepsilon n$ so in a slight abuse of notation we sometimes write $p_{\text{imp}}(\varepsilon n)$ instead of $p_{\text{imp}}(x)$, and sometimes we also drop the parameters by writing just p_{imp} when they are clear from the context. Similarly we sometimes write q_{imp} and λ^*.

Remark 22. *For all x, s and f, $\lambda^{*,f}(x,s)$ is chosen to satisfy*

$$q^f_{\text{imp}}(x, \lambda^{*,f}(x,s)) = \tfrac{1}{s+1}.$$

Note that the equilibrium population size λ^* need not be an integer. Moreover, rounding λ to the next integer can change the success probability by a constant factor. Thus we must take the effect of rounding into account. Fortunately, as we will show, the effect of changing the function f from OneMax *to* DBv *is much larger than such rounding effects.*

3.3.1 *Sketch of proof*

We have three quantities that depend on each other: the target population size λ^*, the target success rate $1/(s+1)$ and the distance $\varepsilon := Z/n$ of the starting point from the optimum. Essentially, choosing any two of them determines the third one. In the proof we will choose λ^* and s to be large, and ε to be small. As ε is small, it is very unlikely to flip more than one zero-bit and the positive contribution to the drift is dominated by the term Δ_1 (Lemma 24 (ii)). For OneMax we are also able to give a tight estimation of $\Delta_{\leq -1}$: for λ large enough we can guarantee that $|\Delta_{\leq -1}| \approx (1 - e^{-1})^\lambda$ (Lemma 24 (iii)).

The key to the proof is that under the above assumptions, the improvement probability p_{imp} for DBV is by a constant factor larger than for ONEMAX (Lemma 23). This is not hard to understand. Since it is unlikely to flip more than one zero-bit, the main way to improve the fitness for ONEMAX is by flipping a single zero-bit and no one-bits. Likewise, DBV also improves the fitness in this situation. However, DBV may also improve the fitness if it flips, for example, exactly one zero-bit and one one-bit. This improves the fitness if the zero-bit has higher weight, which happens with probability $1/2$. This already makes $p_{\text{imp}}^{\text{DBV}}$ by a constant factor larger than $p_{\text{imp}}^{\text{OM}}$. (There are actually even more ways to improve the fitness for DBV.) As a consequence, for the same values of s and ε, the target population size λ^* for DBV is by a constant factor smaller than for ONEMAX (Lemma 23 (iii)).

This enables us to (mentally) fix some large λ, choose ε such that the drift for ONEMAX at $Z = \varepsilon n$ is slightly positive (towards the optimum) and choose the s that satisfies $\lambda^{*,\text{OM}}(\varepsilon n, s) = \lambda$. Here, 'slightly positive' means that $\Delta_1 \approx 4|\Delta_{\leq -1}|$. This may seem like a big difference, but in terms of λ it is not. Changing λ only affects Δ_1 mildly. But adding just a single child (increasing λ by one) reduces $\Delta_{\leq -1}$ by a factor of $\approx 1 - 1/e$, which is the probability that the additional child is not a copy of the parent. So our choice of λ^*, ε and s ensures positive drift for ONEMAX as long as $\lambda^t \geq \lambda^* - 1$, but not for a much wider range. However, as we show, λ^t stays concentrated in this small range due to our choice of $F = 1 + o(1)$. This already would yield progress for ONEMAX in a small range around $Z = \varepsilon n$. To extend this to all values $Z \leq \varepsilon n$, we consider the potential function $G^t = Z^t - K \log_F(\lambda^t)$, and show that this potential function has drift towards zero (Corollary 28) whenever $Z > 0$, similar to [26, 28]. For DBV, we show that λ^t stays in a range below $\lambda^{*,\text{DBV}}(\varepsilon n, s) + 1$, which is much smaller than $\lambda^{*,\text{OM}}(\varepsilon n, s)$, and that such small values of λ^t give a negative drift on Z^t (away from the optimum, Lemma 29). Hence, the algorithm is not able to cross the point at $Z = \varepsilon n$ for DBV.

3.3.2 *Full proof*

In the remainder of this section we give the full proof, which follows the intuitive arguments presented above. In particular, we derive some relations between λ, s, ε to find a suitable such triple. Those relations and constructions only hold if the number of bits n is large enough. For instance, we wish to start at a distance εn from the optimum, meaning we need εn to

be an integer. In all following statements, we implicitly assume that εn is a positive integer. In particular, this implies $\varepsilon \geq 1/n$.[5]

We start with the following lemma whose purpose is twofold. On the one hand it gives useful bounds and estimations of the probabilities of improvement; on the other hand it compares those probabilities of improvement for OneMax and Dynamic BinVal. In particular, the success probability for OneMax is substantially smaller than for Dynamic BinVal, meaning that Dynamic BinVal is easier than OneMax with respect to fitness improvements.

Lemma 23. *Let f be any dynamic monotone function and $1/n \leq \varepsilon \leq 1$. Then $p^f_{\text{imp}}(\varepsilon n) \leq \varepsilon$. More specifically for* Om *and* DBv*, for all $n \geq 10$ we have*

$$p^{\text{Om}}_{\text{imp}}(\varepsilon n) = e^{-1}\varepsilon \pm 2\varepsilon^2 \qquad \text{and} \qquad p^{\text{DBv}}_{\text{imp}}(\varepsilon n) = (1-e^{-1})\varepsilon \pm 11\varepsilon^2.$$

In particular, there exists $c > 0$ such that the following holds.

(i) For every $\delta \leq 1$, $\lambda \in \mathbb{N}$, $\lambda \leq c\delta/\varepsilon$, and every dynamic monotone function f we have

$$q^f_{\text{imp}}(\varepsilon n, \lambda) = (1 \pm \delta)\lambda p^f_{\text{imp}}(\varepsilon n).$$

(ii) For every $s \geq 1$, every constant $0 < \varepsilon \leq c$, and every dynamic monotone function f there exists a constant $\varepsilon' > 0$ such that

$$\lambda^{*,f}((\varepsilon - \varepsilon')n, s) - \lambda^{*,f}((\varepsilon + \varepsilon')n, s) \leq 1/4.$$

(iii) For every $1/n \leq \varepsilon \leq c$ and $s > 0$ we have

$$0.5\lambda^{*,\text{Om}}(\varepsilon n, s) \leq \lambda^{*,\text{DBv}}(\varepsilon n, s) \leq 0.6\lambda^{*,\text{Om}}(\varepsilon n, s).$$

Proof. For general f, due to monotonicity the fitness can only improve if at least one zero-bit is flipped. Since the i-th zero-bit flips with probability $1/n$, by a union bound over the εn zero-bits we have $p^f_{\text{imp}}(\varepsilon n) \leq \varepsilon$.

5 In Lemma 23 (ii) and Lemma 29 we consider a constant $\varepsilon > 0$, introduce an $\varepsilon' = \varepsilon'(\varepsilon)$ and look at all states in the range $(\varepsilon \pm \varepsilon')n$. Again, we implicitly assume that $(\varepsilon - \varepsilon')n$ and $(\varepsilon + \varepsilon')n$ are integers, since we use those in the calculations.

For the next line, note that the statement is trivial for $1/2 \leq \varepsilon \leq 1$, so we may assume $\varepsilon \leq 1/2$. Let A_1 be the event of flipping exactly one zero-bit and an arbitrary number of one-bits, $A_{0,1}$ the sub-event of flipping exactly one zero-bit and no one-bits, and $A_{\geq 2}$ the event of flipping at least two zero-bits. Let $q_1 := \mathbb{P}(A_1)$, $q_{0,1} := \mathbb{P}(A_{0,1})$ and $q_{\geq 2} := \mathbb{P}(A_{\geq 2})$. We have $q_1 = \varepsilon(1-1/n)^{\varepsilon n-1}$. By Lemma 19, we may bound $(1-1/n)^{\varepsilon n-1} \geq 1-(\varepsilon n-1)/n \geq 1-\varepsilon$, which gives $\varepsilon - \varepsilon^2 \leq q_1 \leq \varepsilon$. For $q_{0,1}$, we have $q_{0,1} = \varepsilon(1-1/n)^{n-1}$. Using Lemma 19, $e^{-1} \leq (1-1/n)^{n-1}$ and $(1-1/n)^{n-1} \leq e^{-1}e^{1/n} \leq e^{-1}(1+2/n) \leq e^{-1}+\varepsilon$ for all $n \geq 1$, which gives $e^{-1}\varepsilon \leq q_{0,1} \leq e^{-1}\varepsilon + \varepsilon^2$. Finally, for $q_{\geq 2}$, any fixed pair of two zero-bits has probability $1/n^2$ to be flipped, so by a union bound over all $\binom{\varepsilon n}{2}$ such pairs we have $q_{\geq 2} \leq \varepsilon^2/2$. Summarizing, we have shown

$$q_1 = \varepsilon \pm \varepsilon^2 \qquad \text{and} \qquad q_{0,1} = e^{-1}\varepsilon \pm \varepsilon^2 \qquad \text{and} \qquad q_{\geq 2} \leq \varepsilon^2/2.$$

For OneMax, the fitness always improves in the case $A_{0,1}$, may or may not improve in the case $A_{\geq 2}$, and does not improve in any other case. Hence, $q_{0,1} \leq p^{\text{OM}}_{\text{imp}} \leq q_{0,1} + q_{\geq 2}$, which implies the claim for OneMax.

For Dynamic BinVal, we have to work a bit harder to compute the improvement probability in the case of A_1. Consider this case, i.e., assume that exactly one zero-bit and an undetermined number of one-bits is flipped. Now we sort this zero-bit together with the $(1-\varepsilon)n$ one-bits decreasingly by weight, thus obtaining a list of $(1-\varepsilon)n+1$ bits. The zero-bit is equally likely to take any position $i \in [(1-\varepsilon)n+1]$ in this list. Then the offspring is fitter than the parent if and only if none of the $i-1$ one-bits to the left of i are flipped, which has probability $(1-1/n)^{i-1}$. Hence, conditional on A_1, the probability $q^{\text{DBv}}_{|1}$ of improvement is

$$\begin{aligned} q^{\text{DBv}}_{|1} &= \frac{1}{(1-\varepsilon)n+1} \sum_{i=1}^{(1-\varepsilon)n+1} (1-1/n)^{i-1} \\ &= \frac{1}{(1-\varepsilon)n+1} \cdot \frac{1-(1-1/n)^{(1-\varepsilon)n+1}}{1/n} \\ &= \frac{1}{(1-\varepsilon)+1/n} \left(1-(1-1/n)^{(1-\varepsilon)n+1}\right), \end{aligned} \tag{3.2}$$

where in the second step we used the formula for the geometric sum, $\sum_{i=1}^{k} x^{i-1} = (1-x^k)/(1-x)$. We separately estimate upper and lower bounds for this expression. For the first factor, we use $\frac{1}{(1-\varepsilon)+1/n} \geq 1$ since

$\varepsilon \geq 1/n$, and $\frac{1}{(1-\varepsilon)+1/n} \leq \frac{1}{1-\varepsilon} \leq 1+2\varepsilon$, which holds for $\varepsilon \leq 1/2$. For the second factor, we use Lemma 19 to obtain

$$(1-1/n)^{(1-\varepsilon)n+1} \leq e^{-(1-\varepsilon)-1/n} \leq e^{-1}e^{\varepsilon} \leq e^{-1}(1+2\varepsilon) \leq e^{-1}+\varepsilon$$

and

$$(1-1/n)^{(1-\varepsilon)n+1} \geq e^{-1}(1-1/n)^{\varepsilon n+2} \geq e^{-1}(1-\varepsilon-2/n) \geq e^{-1}-2\varepsilon,$$

where the last step holds since $\varepsilon \geq 1/n$. Plugging these bound into (3.2) (mind the "1−" in the bracket of (3.2)) yields

$$q_{|1}^{\mathrm{DBv}} \geq 1-e^{-1}-\varepsilon \quad \text{and} \quad q_{|1}^{\mathrm{DBv}} \leq (1+2\varepsilon)(1-e^{-1}+2\varepsilon) \leq 1-e^{-1}+6\varepsilon.$$

Now we may use $q_1 \cdot q_{|1}^{\mathrm{DBv}} \leq p_{\mathrm{imp}}^{\mathrm{DBv}} \leq q_1 \cdot q_{|1}^{\mathrm{DBv}} + q_{\geq 2}$, where

$$q_1 \cdot q_{|1}^{\mathrm{DBv}} = (\varepsilon \pm \varepsilon^2)(1-e^{-1} \pm 6\varepsilon) = (1-e^{-1})\varepsilon \pm 10\varepsilon^2.$$

Together with $q_{\geq 2} \leq \varepsilon^2/2$, this proves the claim for $p_{\mathrm{imp}}^{\mathrm{DBv}}$.

We prove the remaining items in order, now focusing on (i). Recall that $q_{\mathrm{imp}} = 1-(1-p_{\mathrm{imp}})^{\lambda}$, by Lemma 19 this can be bounded by

$$\frac{\lambda p_{\mathrm{imp}}}{1+\lambda p_{\mathrm{imp}}} \leq q_{\mathrm{imp}} \leq \lambda p_{\mathrm{imp}}.$$

Since $p_{\mathrm{imp}}^{f} \leq \varepsilon$, if we choose $\varepsilon \leq c\delta/\lambda$ for some absolute constant c small enough we indeed get $q_{\mathrm{imp}}^{f} = (1 \pm \delta)\lambda p_{\mathrm{imp}}^{f}$ for every dynamic monotone function f.

The second result (ii) is a consequence of the definition:

$$\lambda^{*,f}(\varepsilon n, s) = \log_{1-p_{\mathrm{imp}}^{f}}(s/(s+1)) = \frac{\log(s/(s+1))}{\log(1-p_{\mathrm{imp}}^{f})}.$$

Again, $p_{\mathrm{imp}}^{f} \leq \varepsilon$, so choosing c very small and $\varepsilon \leq c$ gives

$$\lambda^{*,f}(\varepsilon n, s) = \frac{\log(s/(s+1))}{-(1 \pm C\varepsilon)p_{\mathrm{imp}}^{f}},$$

for some (possibly large, but absolute) constant C. Since $s \geq 1$, changing ε by $\varepsilon' = \varepsilon^2$ may change λ^* by $C'\varepsilon \cdot \log 2$, where C' is again a large but absolute

constant. Up to possibly choosing c smaller, this quantity is bounded by $1/4$.

Lastly, we prove (iii). Again from the definition of λ^* we have

$$\frac{\lambda^{*,\mathrm{DBv}}(\varepsilon n, s)}{\lambda^{*,\mathrm{OM}}(\varepsilon n, s)} = \frac{\log_{(1-p^{\mathrm{DBv}}_{\mathrm{imp}}(\varepsilon n))}\left(\frac{s}{1+s}\right)}{\log_{(1-p^{\mathrm{OM}}_{\mathrm{imp}}(\varepsilon n))}\left(\frac{s}{1+s}\right)} = \frac{\log(1-p^{\mathrm{OM}}_{\mathrm{imp}}(\varepsilon n))}{\log(1-p^{\mathrm{DBv}}_{\mathrm{imp}}(\varepsilon n))}$$
$$= \frac{\log(1-e^{-1}\varepsilon \pm 2\varepsilon^2)}{\log(1-(1-e^{-1})\varepsilon \pm 11\varepsilon^2)}.$$

Since $\log(1-x) \approx -x$ for $x \to 0$, the above ration tends to $e^{-1}/(1-e^{-1}) = 1/(e-1) = 0.58...$ as ε tends to zero. In particular, since $\varepsilon < c$ is assumed small enough, the ratio is at least 0.5 and at most 0.6. □

We follow with a lemma that gives estimates of the drift of Z. In particular, the first statement is one way of stating that OneMax is the easiest function with respect to minimizing the distance from the optimum. We will only apply it with $f = \mathrm{DBv}$, but we believe that the result is interesting enough to be mentioned.

Lemma 24. *There exists a constant $c > 0$ such that the following holds for every dynamic monotone function f. Let $\delta > 0$, there exists λ_0 such that the following holds.*

(i) For all $\lambda \geq 1$, all $x \in \{0,1\}^n$ and all $i \in \mathbb{Z}^+$ we have

$$\Delta^f_{\geq i}(x,\lambda) \leq \Delta^{\mathrm{OM}}_{\geq i}(x,\lambda) \qquad \text{and} \qquad |\Delta^f_{\leq -i}(x,\lambda)| \geq |\Delta^{\mathrm{OM}}_{\leq -i}(x,\lambda)|.$$

(ii) For every integer $\lambda \geq \lambda_0$ and all $1/n \leq \varepsilon \leq c\delta/\lambda$ we have

$$\Delta^f_{\geq 2}(\varepsilon n,\lambda) \leq \delta \Delta^{\mathrm{OM}}_1(\varepsilon n,\lambda).$$

(iii) For every integer $\lambda \geq \lambda_0$ and all $1/n \leq \varepsilon \leq c\delta/\lambda$ we have

$$|\Delta^{\mathrm{OM}}_{\leq -1}(\varepsilon n,\lambda)| = (1 \pm \delta)(1-e^{-1})^\lambda.$$

Note that in the second point the right hand side is the drift with respect to OneMax*, not to f.*

Proof. We prove items in order and start with (i). For any $j \in \mathbb{Z}$, the event $Z^{t+1} \geq Z^t - j$ can only happen for OneMax if $Z(y) \geq Z(x) - j$ holds for all offspring y of x, since OneMax picks the offspring which minimizes $Z(y)$. In this case, regardless of the selection of f, the selected offspring also satisfies $Z(y) \geq Z(x) - j$. Hence, $p^{\text{OM}}_{\leq j} \leq p^f_{\leq j}$ and $p^{\text{OM}}_{\geq j} \geq p^f_{\geq j}$ for all $j \in \mathbb{Z}$. In particular, for all $i \geq 0$

$$\Delta^{\text{OM}}_{\leq -i} = -i \cdot p^{\text{OM}}_{\leq -i} - \sum_{j=-\infty}^{-i-1} p^{\text{OM}}_{\leq j} \geq -i \cdot p^f_{\leq -i} - \sum_{j=-\infty}^{-i-1} p^f_{\leq j} = \Delta^f_{\leq -i},$$

and

$$\Delta^{\text{OM}}_{\geq i} = i \cdot p^{\text{OM}}_{\geq i} + \sum_{j=i+1}^{\infty} p^{\text{OM}}_{\geq j} \geq i \cdot p^f_{\geq i} + \sum_{j=i+1}^{\infty} p^f_{\geq j} = \Delta^f_{\geq i}.$$

We now turn to the proof of (ii), starting with $f = \text{OM}$. Consider a given offspring. For any i zero-bits, there is probability n^{-i} that they all flip to one-bits. By union bound over $\binom{\varepsilon n}{i} \leq (\varepsilon n)^i / i!$ such choices, the probability that at least i zero-bits are flipped is at most $\varepsilon^i / i!$. Now union bounding over all children we find

$$p^{\text{OM}}_{\geq i} = \Pr[Z^t - Z^{t+1} \geq i] \leq \lambda \varepsilon^i / i!.$$

In turn this implies

$$\Delta_{\geq 2} \;\leq\; \sum_{i=2}^{\infty} i p_{\geq i} \;\leq\; 2\lambda\varepsilon^2.$$

As a single child flipping exactly a zero-bit and no one-bit implies that Z^t decreases, we must also have

$$\Delta_1 = p_1 \geq \varepsilon(1-1/n)^{n-1} \geq e^{-1}\varepsilon,$$

by Lemma 19. Choosing $c = 1/(2e)$ ensures that $\Delta_{\geq 2} \leq \delta\Delta_1$, and the corresponding bound for $\Delta^{\text{DBv}}_{\geq 2}$ follows from (i).

We finish with the proof of (iii) and start with the lower bound. Clearly, if every child flips at least a one-bit and no zero-bit, then Z^t must increase. Hence we have

$$\begin{aligned} |\Delta_{\leq -1}| &\geq \left((1-(1-1/n)^{(1-\varepsilon)n}) \cdot (1-1/n)^{\varepsilon n}\right)^{\lambda} \\ &= \left((1-1/n)^{\varepsilon n} - (1-1/n)^n\right)^{\lambda}. \end{aligned}$$

Using Lemma 19, the first term in parentheses may be bounded from below by $(1-1/n)^{\varepsilon n} \geq (1-1/n)^{\varepsilon} e^{-\varepsilon}$; since $\varepsilon n \geq 1$, we have $(1-1/n)^{\varepsilon} \geq$

$(1-\varepsilon)^{\varepsilon} \geq 1-\varepsilon$ and we also have $e^{-\varepsilon} \geq 1-\varepsilon$ by Lemma 19. The second term of the display above may simply be handled by Lemma 19: $(1-1/n)^n \leq e^{-1}$. Combining everything we find

$$\begin{aligned}|\Delta_{\leq -1}| &\geq \left((1-\varepsilon)^2 - e^{-1}\right)^{\lambda} \geq (1-e^{-1})^{\lambda} \cdot \left(1 - \tfrac{2}{1-e^{-1}}\varepsilon\right)^{\lambda} \\ &\geq (1-e^{-1})^{\lambda} \cdot \left(1 - \tfrac{2\lambda}{1-e^{-1}}\varepsilon\right),\end{aligned}$$

where the last step follows from an application of the last inequality of Lemma 19. Since c is assumed to be sufficiently small and $\varepsilon < c\delta/\lambda$, the above is at least $|\Delta_{\leq -1}| \geq (1-\delta)(1-e^{-1})^{\lambda}$.

The lower bound is proved and we now focus on the upper bound. Since we may express $|\Delta_{\leq -1}| = \sum_{i \geq 1} p_{\leq -i}$, it suffices to bound each term appearing in the sum. As we are considering the fitness function OneMax, the number of zero-bits Z^t increases by at least i only if every child flips at least i one-bits. Hence by Lemma 19

$$p_{\leq -1} \leq \left(1 - (1-1/n)^{(1-\varepsilon)n}\right)^{\lambda} \leq (1-e^{-1})^{\lambda},$$

and

$$p_{\leq -i} \leq \left(\binom{(1-\varepsilon)n}{i} n^{-i}\right)^{\lambda} \leq (i!)^{-\lambda} \leq 2^{(1-i)\lambda}.$$

This immediately implies

$$\begin{aligned}|\Delta_{\leq -1}| &= p_{\leq -1} + \sum_{i=2}^{\infty} p_{\leq -2} \\ &\leq (1-e^{-1})^{\lambda} + 2^{1-\lambda}.\end{aligned}$$

The second term decays faster than the first since $(1-e^{-1}) \geq 2^{-1}$. Therefore, for a sufficiently large λ_0, the second term of the RHS is at most $\delta \cdot (1-e^{-1})^{\lambda}$ and this finishes the proof of (iii). □

We now come to the heart of our proof, which is finding a suitable triple $\lambda^*, \varepsilon, s$.

Lemma 25. *For every $\delta > 0$ there exists $\lambda_0 \geq 1$ such that the following holds. For every integer $\lambda \geq \lambda_0$, there exist constants $\tilde{\varepsilon}, \tilde{s}$ depending only on λ such that $\lambda = \lambda^{*,\mathrm{OM}}(\tilde{\varepsilon}n, \tilde{s})$ and*

$$\Delta^{\mathrm{OM}}_{\geq 1}(\tilde{\varepsilon}n, \lambda) = (4 \pm \delta)|\Delta^{\mathrm{OM}}_{\leq -1}(\tilde{\varepsilon}n, \lambda)| = (1 \pm \delta)/(\tilde{s}+1).$$

Additionally $\tilde{\varepsilon}(\lambda) = o_\lambda(1/\lambda)$ *and* $\tilde{s}(\lambda) = \omega_\lambda(1)$.[6] *In particular, for every* $\delta > 0$, *a sufficiently large* λ_0 *guarantees that one may apply Lemmas 23 and 24.*

Proof. Take some arbitrary $\delta > 0$, λ_0 very large and consider $\lambda \geq \lambda_0$. Choose also some $\varepsilon \leq c\delta/\lambda$, with c the constant appearing in Lemmas 23 and 24.

Using Lemma 24 (ii) followed by Lemma 23 (i) for this pair (λ, ε) we find

$$\Delta_{\geq 1} = (1 \pm \delta) p_{\geq 1} = (1 \pm \delta) q_{\text{imp}} = (1 \pm \delta)^2 p_{\text{imp}} \lambda.$$

Since $p_{\text{imp}} = e^{-1}\varepsilon \pm 4\varepsilon^2$ by Lemma 23 and we chose $\varepsilon \leq c\delta/\lambda$ with $\lambda \geq \lambda_0$ large enough, we may further write $\Delta_{\geq 1} \geq (1 \pm \delta)^3 e^{-1}\varepsilon\lambda$. With Lemma 24 (iii) we may now estimate the negative contribution as

$$\Delta_{\leq -1} = -(1 \pm \delta)(1 - e^{-1})^\lambda.$$

If we choose $\varepsilon := \tilde{\varepsilon} = 4e(1 - e^{-1})^\lambda/\lambda$, then we get $\Delta_{\geq 1} = (1 \pm \delta)^4 \cdot 4|\Delta_{\leq -1}|$, which is what we want (up to originally choosing a smaller δ). Note that this argument works because we assumed λ_0 large enough, which implies $\tilde{\varepsilon} \leq c\delta/\lambda$. Also, it is clear from the formula that $\tilde{\varepsilon} = o_\lambda(1/\lambda)$.

Define $\tilde{s}$ so that $1/(\tilde{s}+1) = q_{\text{imp}}(\tilde{\varepsilon}n, \lambda)$. Since $\Delta_{\geq 1} = (1 \pm \delta) q_{\text{imp}}$ the equality is proved (again, up to originally choosing a smaller δ). All that remains is to check that $\tilde{s}$ has the correct order: this comes from $\tilde{s} = 1/q_{\text{imp}} - 1$ is of order $1/(\tilde{\varepsilon}\lambda) = \omega_\lambda(1)$ by Lemma 23.

□

Naturally, the lemma above implies that for parameters $\lambda, \tilde{s}$ and at distance $\tilde{\varepsilon}n$ from the optimum, the drift of Z for OneMax is roughly $\Delta^{\text{OM}} = \Delta^{\text{OM}}_{\geq 1} + \Delta^{\text{OM}}_{\leq -1} \approx \frac{3}{4}\Delta^{\text{OM}}_{\geq 1} > 0$. Moreover, we want to show that the SA-$(1, \lambda)$-EA can not only pass this point, but continues all the way to the optimum. To this end, we define a more general potential function already used in [26] and [28].

Definition 26. *We define*

$$h(\lambda) := -K \log_F \lambda,$$

with $K = 1/2$. *We also define*

$$g(x, \lambda) := Z(x) + h(\lambda).$$

6 The subscript indicates dependency on λ, i.e., for all $c, C > 0$ there exists λ_0 such that for all $\lambda \geq \lambda_0$ we have $\tilde{\varepsilon}(\lambda) \leq c/\lambda$ and $\tilde{s}(\lambda) \geq C$.

For convenience we will write $Z^t = Z(x^t)$, $H^t = h(\lambda^t)$ and $G^t = Z^t + H^t = g(x^t, \lambda^t)$.

Lemma 27. *Let $f = \textsc{Om}$. At all times t such that $\lambda^t \geq F$ we have*

$$\mathbf{E}\left[H^t - H^{t+1} \mid x^t, \lambda^t\right] = \tfrac{K}{s}(1 - (s+1)q_{\mathrm{imp}}(x^t, \lceil \lambda^t \rceil)).$$

Proof. We have $H^{t+1} = -K \log_F(\lambda^t / F) = H^t + K$ in case of an improvement, and $H^{t+1} = H^t - K/s$ otherwise. Hence, the drift of H^t is

$$\mathbf{E}[H^t - H^{t+1} \mid x^t, \lambda^t] = -Kq_{\mathrm{imp}} + (1 - q_{\mathrm{imp}})\tfrac{K}{s} = \tfrac{K}{s}(1 - (s+1)q_{\mathrm{imp}}).$$

□

With these choices, the drift of G^t is positive for *all* $\varepsilon \leq \tilde{\varepsilon}$ and $\lambda^t \geq \lambda - 1$.

Corollary 28. *Let $f = \textsc{Om}$. There exists $\lambda_0 \geq 1$ such that the following holds for all $\lambda \geq \lambda_0$. Let $s = \tilde{s} = \tilde{s}(\lambda)$, $\varepsilon = \tilde{\varepsilon} = \tilde{\varepsilon}(\lambda)$ be as in Lemma 25. There exist $\rho(\lambda), \varepsilon'(\lambda)$ such that if $1 \leq Z^t \leq (\varepsilon + \varepsilon')n$ and $\lambda^t \geq \lambda - 1$, then*

$$\mathbf{E}[G^t - G^{t+1} \mid Z^t, \lambda^t] \geq \rho.$$

Proof. Let $\delta > 0$ be a sufficiently small constant and let λ_0 be sufficiently large so that Lemmas 23, 24 and 25 hold. By Lemma 23 (ii), there exists a small $\varepsilon' > 0$ such that $\lambda^{*,\textsc{Om}}((\varepsilon + \varepsilon')n, s) \geq \lambda - 1/2$. We seek to estimate the drift of G when $Z^t \leq (\varepsilon + \varepsilon')n$ and $\lambda^t \geq \lambda - 1$. Using Lemma 27, we see that this drift is

$$\begin{aligned}\mathbf{E}[G^t - G^{t+1} \mid Z^t, \lambda^t] = \Delta_{\geq 1}(Z^t, \lambda^t) + \Delta_{\leq -1}(Z^t, \lambda^t) \\ - Kq_{\mathrm{imp}}(Z^t, \lambda^t) + K(1 - q_{\mathrm{imp}}(Z^t, \lambda^t))/s.\end{aligned}$$

From Lemma 24 (iii) applied for both Z^t and $\tilde{\varepsilon}n$ and from Lemma 25 we know that

$$\begin{aligned}|\Delta_{\leq -1}(Z^t, \lambda)| &= (1 \pm \delta)(1 - e^{-1})^{\lambda} = (1 \pm \delta)^2 |\Delta_{\leq -1}(\varepsilon n, \lambda)| \\ &= (1 \pm \delta)^3 / (4(s+1)).\end{aligned}$$

Applying Lemma 24 (iii), now for $\lambda - 1$, we have $|\Delta_{\leq -1}(Z^t, \lambda - 1)| = (1 \pm \delta)(1 - e^{-1})^{\lambda - 1}$, which by the above must be at most

$$|\Delta_{\leq -1}(Z^t, \lambda - 1)| \leq (1 \pm \delta)^4 \frac{1}{4(1 - e^{-1})(s+1)} \leq \frac{1}{2(s+1)}.$$

As $\Delta_{\leq -1}(Z^t, \cdot)$ is decreasing, this same bound must hold whenever $\lambda^t \geq \lambda - 1$. We also observe that since we are considering OM, we have $\Delta_{\geq 1} \geq q_{\text{imp}}$.

Now replacing in the expression of the drift with $K = 1/2$, the drift is at least

$$\begin{aligned}\mathbf{E}[G^t - G^{t+1} \mid Z^t, \lambda^t] &\geq q_{\text{imp}} - \frac{1}{2(s+1)} - \frac{q_{\text{imp}}}{2} + \frac{1}{2s} - \frac{q_{\text{imp}}}{2s} \\ &= \frac{1}{2s(s+1)} + \frac{q_{\text{imp}}}{2}(1 - 1/s) \\ &\geq \frac{1}{2s(s+1)},\end{aligned}$$

since for a choice of λ_0 large enough we may assume that $s > 1$ by Lemma 25. Setting $\rho(\lambda) = 1/(2s(s+1))$ concludes the proof. □

We have just shown that when within distance at most $\tilde{\varepsilon}n$ from the optimum, ONEMAX has drift *towards* the optimum. We now turn to DYNAMIC BINVAL: the following lemma states that at distance $\tilde{\varepsilon}n$ from the optimum, DYNAMIC BINVAL has drift *away* from the optimum.

Lemma 29. *Let $f = \text{DBV}$. There exists $\lambda_0 \geq 1$ such that for all $\lambda \geq \lambda_0$ there are $\nu, \varepsilon' > 0$ such that the following holds. Let $\tilde{s} = \tilde{s}(\lambda), \tilde{\varepsilon} = \tilde{\varepsilon}(\lambda)$ as in Lemma 25. If $Z^t = (\tilde{\varepsilon} \pm \varepsilon')n$ and $\lambda^t \leq \lambda^{*,\text{DBV}}(\tilde{\varepsilon}n, \tilde{s}) + 1$ we have*

$$\mathbf{E}[Z^t - Z^{t+1} \mid x^t, \lambda^t] \leq -\nu.$$

Proof. Let δ be a small constant and choose a large λ_0. Consider some $\lambda \geq \lambda_0$ and let $\tilde{\varepsilon} = \tilde{\varepsilon}(\lambda), \tilde{s} = \tilde{s}(\lambda)$. Since $\tilde{\varepsilon} = o(1/\lambda)$ by Lemma 25, we may assume $\tilde{\varepsilon}$ is small enough to apply first Lemma 23 (ii) and then Lemma 23 (iii): there exists an ε' such that

$$\lambda^{*,\text{DBV}}((\tilde{\varepsilon} + \varepsilon')n, \tilde{s}) \geq \lambda^{*,\text{DBV}}(\tilde{\varepsilon}n, \tilde{s}) - 1 \geq 0.4\lambda.$$

Up to taking a larger λ_0 (giving a smaller $\tilde{\varepsilon}$) and smaller ε', we may assume that $\tilde{\varepsilon} + \varepsilon' < \frac{0.4c\delta}{\lambda}$, with c the constant appearing in Lemmas 23 and 24. In particular, we may apply those lemmas with $\tilde{\varepsilon} + \varepsilon'$ and $\lambda^{*,f}((\tilde{\varepsilon} + \varepsilon')n, \tilde{s})$ for both $f = \text{OM}$ and $f = \text{DBV}$.

The drift may be expressed as

$$\Delta^{\text{DBV}}_{\geq 1}(Z^t, \lambda^t) + \Delta^{\text{DBV}}_{\leq -1}(Z^t, \lambda^t) \leq \Delta^{\text{OM}}_{\geq 1}(Z^t, \lambda^t) + \Delta^{\text{OM}}_{\leq -1}(Z^t, \lambda^t),$$

using Lemma 24 (i), and we wish to show that this is negative. Lemma 24 (ii) guarantees that the positive contribution of this drift is at most

$$\begin{aligned}(1+\delta)\Delta_1^{\text{OM}}(Z^t,\lambda^t) &\le (1+\delta)q_{\text{imp}}^{\text{OM}}(Z^t,\lambda^t)\\ &\le (1+\delta)q_{\text{imp}}^{\text{OM}}\left((\tilde{\varepsilon}+\varepsilon')n,\lambda^{*,\text{DBV}}((\tilde{\varepsilon}+\varepsilon')n,\tilde{s})+2\right),\end{aligned}$$

where the last step follows from the monotonicity of q_{imp} in both Z and λ. We know from Lemma 23 (iii) that $\lambda^{*,\text{DBV}}((\tilde{\varepsilon}+\varepsilon')n,\tilde{s})+2 \le 0.6\lambda^{*,\text{OM}}((\tilde{\varepsilon}+\varepsilon')n,\tilde{s})+2 \le \lambda^{*,\text{OM}}((\tilde{\varepsilon}+\varepsilon')n,\tilde{s})$ as we assume λ_0 large enough. In particular, this implies that the positive contribution to the drift is at most

$$(1+\delta)q_{\text{imp}}^{\text{OM}}\left((\tilde{\varepsilon}+\varepsilon')n,\lambda^{*,\text{OM}}((\tilde{\varepsilon}+\varepsilon')n,\tilde{s})\right) = (1+\delta)/(\tilde{s}+1).$$

Let us now show that the negative contribution is larger. We first use Lemma 23 (iii) to obtain $\lambda^t \le \lambda^{*,\text{DBV}}(\tilde{\varepsilon}n,\tilde{s})+1 \le 0.7\lambda$. An application of Lemma 24 (iii) then gives

$$|\Delta_{\le -1}^{\text{OM}}(Z^t,\lambda^t)| \ge (1\pm\delta)(1-e^{-1})^{0.7\lambda}.$$

By Lemma 25, we have $|\Delta_{\le -1}^{\text{OM}}(\tilde{\varepsilon}n,\lambda)| = (1\pm\delta)\frac{1}{4(\tilde{s}+1)}$, and a new application of Lemma 24 (iii) gives that $(1\pm\delta)/(4(\tilde{s}+1)) = (1\pm\delta)(1-e^{-1})^{\lambda}$. Replacing in the display above gives

$$|\Delta_{\le -1}^{\text{OM}}(Z^t,\lambda^t)| \ge (1\pm\delta)^2(1-e^{-1})^{-0.3\lambda}\frac{1}{4(\tilde{s}+1)} \ge \frac{2}{\tilde{s}+1},$$

as λ may be chosen large enough compared to δ.

In the range of Z^t,λ^t from the lemma, we see that

$$\mathbf{E}[Z^t - Z^{t+1} \mid x^t,\lambda^t] \le (1+\delta)\frac{1}{\tilde{s}+1} - \frac{2}{\tilde{s}+1} \le -\frac{1}{2(\tilde{s}+1)},$$

which concludes the proof. □

Corollary 28 and Lemma 29 respectively give drift towards the optimum for OneMax and away from the optimum for Dynamic BinVal. We are almost ready to state and prove our final theorem. A last step before this is the following lemma saying $\lambda \le n^2$ at all steps and that Z never changes by more than $\log n$.

Lemma 30. *Consider the SA-$(1,\lambda)$-EA on* OneMax *or* Dynamic BinVal. *With probability $1-n^{-\omega(1)}$, either the optimum is found or $\lambda^t \le n^2$ holds for super-polynomially many steps, and in all of these steps $|Z^t - Z^{t+1}| \le \log n$.*

Proof. In order to grow above n^2, there would need to be a non-improving step when $\lambda^t \geq n^2/F^{1/s}$. Let us call this event $\mathcal{E}^t$. The probability that a fixed offspring finds an improvement is at least $1/n \cdot (1-1/n)^{n-1} \geq 1/(en)$ (using Lemma 19), even if the algorithm is just one step away from the optimum. Hence, the probability that none of λ^t offspring finds an improvement is at most

$$\Pr[\mathcal{E}^t] \leq (1-\tfrac{1}{en})^{\lambda^t} \leq e^{-(\lambda^t-1)/(en)} = e^{-\Omega(n)}.$$

using Lemma 19 for the second inequality. Even a union bound over a super-polynomial number of steps leaves the error probability $e^{-\Omega(n)}$ unchanged. This proves the first statement.

For the step sizes, condition on $\mathcal{E}^t$. We can bound the difference $|Z^t - Z^{t+1}|$ from above by the maximal number of bits that any offspring flips. The probability that a fixed offspring flips exactly $k \geq 1$ bits is at most

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

and the probability that it flips at least k bits is at most $\sum_{i\geq k} 1/i! \leq 2/k!$, since the factorials are bounded by a geometric sum as $1/(k+1)! \leq 1/(2 \cdot k!)$. For $k = \log n$ this bound is $2/k! = e^{-\Omega(k \log k)} = e^{-\omega(\log n)} = n^{-\omega(1)}$, and the same holds after a union bound over at most n^2 offspring, and after a union bound over a super-polynomial (suitably chosen) number of steps. □

Now we are ready to formulate and prove the main result of this chapter.

Theorem 31. *Let $F = 1+\eta$ for some $\eta = \omega(\log n/n) \cap o(1/\log n)$. There exist constants s, ε such that with high probability the SA-$(1,\lambda)$-EA with success ratio s, update strength F and mutation rate $1/n$ starting with $Z^0 = \varepsilon n$,*

a) finds the optimum of OneMax *in $O(n)$ generations;*

b) does not find the optimum of Dynamic BinVal *in a polynomial number of generations.*

Proof. Let δ be a sufficiently small constant and choose λ large. Let $s := \tilde{s}(\lambda)$ and $\varepsilon = \tilde{\varepsilon}(\lambda)$ as in Lemma 25. Since $\lambda^{*,\mathrm{DBv}}(\varepsilon n, s) \geq 0.5\lambda^{*,\mathrm{Om}}(\varepsilon n, s) = 0.5\lambda$, we may assume that Lemma 24 holds for $\lambda^{*,\mathrm{DBv}}(\varepsilon n, s)$.

We first prove a). Set $f :=$ OneMax, and let $\varepsilon' = \varepsilon'(\lambda)$ be small enough to apply Corollary 28. We will first argue that λ^t quickly reaches a value of

at least $\lambda - 7/8$ and stays above $\lambda - 1$ afterwards. Note that by definition of $\varepsilon = \tilde{\varepsilon}$, we have $q_{\text{imp}}(\varepsilon n, \lambda) = 1/(s+1)$, and hence

$$q_{\text{imp}}(\varepsilon n, \lambda - 1/4) < q_{\text{imp}}(\varepsilon n, \lambda) = \tfrac{1}{s+1}.$$

Up to potentially reducing ε', we may apply Lemma 23 (ii) and get $\lambda^*((\varepsilon + \varepsilon')n, s) \geq \lambda - 1/4$, or reformulated

$$q_{\text{imp}}((\varepsilon + \varepsilon')n, \lambda - 1/4) \leq \tfrac{1}{s+1}.$$

Therefore, $\lfloor \lambda - 7/8 \rceil \leq \lambda - 3/8 < \lambda^*((\varepsilon + \varepsilon')n, s)$, and by Lemma 27 we have for all $1 < Z^t \leq (\varepsilon + \varepsilon')n$ and all $\lambda - 1 \leq \lambda^t \leq \lambda - 7/8$,

$$\begin{aligned} E[H^t - H^{t+1} \mid Z^t, \lambda^t] &= \tfrac{K}{s}(1 - (s+1)q_{\text{imp}}(Z^t, \lfloor \lambda^t \rceil)) \\ &\geq \tfrac{K}{s}(1 - (s+1)q_{\text{imp}}((\varepsilon + \varepsilon')n, \lambda - 3/8)) \geq d, \end{aligned}$$

for a suitable constant $d > 0$. Therefore, H^t has positive drift as long as $1 < Z^t \leq (\varepsilon + \varepsilon')n$ and $\lambda^t \leq \lambda - 7/8$. By Lemma 30, we may assume that Z^t changes by at most $\log n$ in all of the first $\varepsilon' n/(2 \log n)$ steps. In particular, we have $Z^t \leq (\varepsilon + \varepsilon'/2)n$ during this time. Hence, H^t has constant positive drift until either $\varepsilon' n/(2 \log n)$ steps are over or until it hits $-K \log_F(\lambda - 7/8)$. Since $H^0 = -K \log_F(1) = 0$, by the additive drift theorem, the latter option happens after at most $\frac{K}{d} \log_F(\lambda - 7/8) = O(1/\log F) = O(1/\eta)$ steps in expectation. Moreover, the same holds with high probability after at most $2\frac{K}{d} \log_F(\lambda - 7/8) = O(1/\eta)$ steps by concentration of hitting times for additive drift, Theorem 17, since H changes by definition by at most one in each step. Since $1/\eta = o(n/\log n)$, with high probability, $\lambda^t \geq \lambda - 7/8$ happens before Z^t reaches $(\varepsilon + \varepsilon'/2)n$. Let us call the first such point in time t_0.

Now we show that after time t_0, the population size λ^t stays above $\lambda - 1$. We have already established that H^t has at least positive constant drift whenever $\lambda^t \leq \lambda - 7/8$. In order for λ^t to decrease from $\lambda - 7/8$ to $\lambda - 1$ the potential must increase by $K \log_F(\lambda - 7/8) - K \log_F(\lambda - 1) = \Omega(1/\log F) = \Omega(1/\eta) = \omega(\log n)$. By the negative drift theorem, Theorem 18, with high probability this does not happen within $O(n^2)$ steps.

On the other hand, since we may assume by Lemma 30 that $\lambda^t \leq n^2$ for a superpolynomial number of steps, we have $0 \leq Z^t - G^t \leq \log_F(n^2) = O(\frac{1}{\eta} \log n) = o(n)$. In particular, the event $Z^t > (\varepsilon + \varepsilon')n$ implies $G^t > (\varepsilon + \frac{3}{4}\varepsilon')n \geq Z^{t_0} + \varepsilon' n/4 \geq G^{t_0} + \varepsilon' n/4$. In other words, Z^t can only exceed $(\varepsilon + \varepsilon')n$ if the potential G^t increases by at least $\varepsilon' n/4$. We will show in the following that this is unlikely.

By Corollary 28 the potential $G^t = Z^t + H^t$ has constant drift towards zero while $1 < Z^t \leq (\varepsilon + \varepsilon')n$ and $\lambda^t \geq \lambda - 1$. Moreover, we may assume that Z^t changes by at most $\log n$ in each step by Lemma 30, and H^t changes by at most one in each step. Therefore, by the negative drift theorem with scaling, Theorem 18, with high probability G^t does not increase by $\varepsilon' n/4$ within n^2 steps.

We have established that with high probability, for n^2 steps none of the events $\lambda^t < \lambda - 1$ and $Z^t > (\varepsilon + \varepsilon')n$ happens. Hence, by Corollary 28 the potential G^t has a positive drift for n^2 steps, or until Z^t hits zero. Since we start with a potential of at most $Z^{t_0} + H^{t_0} \leq Z^{t_0} = O(n)$, with high probability the event $Z^t = 0$ happens within $O(n)$ steps by the tail bounds on hitting times for additive drift, Theorem 17. This proves a).

For b), we set $f := \text{DBv}$, and we will work directly with the potential Z^t instead of G^t. Let $\varepsilon' > 0$ be the constant from Lemma 29. We first show that as long as $Z^t \geq (\varepsilon - \varepsilon')n$, for a superpolynomial number of steps we have $\lambda^t \leq \lambda^{*,\text{DBv}}(\varepsilon n, s) + 1$ so that we may apply Lemma 29. We may assume ε' small enough such that Lemma 23 (ii) gives $\lambda^{*,\text{DBv}}((\varepsilon - \varepsilon')n, s) \leq \lambda^{*,\text{DBv}}(\varepsilon n, s) + 1/4$. As for case a) we can achieve that for all $\lambda^t \geq \lambda^{*,\text{DBv}}(\varepsilon n, s) + 7/8 \geq \lambda^{*,\text{DBv}}((\varepsilon - \varepsilon')n, s) + 5/8$ and all $Z^t = (\varepsilon \pm \varepsilon')n$,

$$q_{\text{imp}}(Z^t, \lfloor \lambda^t \rceil) \geq q_{\text{imp}}((\varepsilon - \varepsilon')n, \lambda^{*,\text{DBv}}((\varepsilon - \varepsilon')n, s) + 1/8) > \tfrac{1}{s+1},$$

which implies a constant negative drift of λ^t. As in case a), with high probability λ^t does not exceed $\lambda^{*,\text{DBv}}(\varepsilon n, s) + 1$ for a superpolynomial number of steps, as long as $Z^t \geq (\varepsilon - \varepsilon')n$ is satisfied. On the other hand, as long as $\lambda^t \leq \lambda^{*,\text{DBv}}(\varepsilon n, s) + 1$, Z^t has constant drift away from the optimum in the interval $(\varepsilon \pm \varepsilon')n$ by Lemma 29. By the negative drift theorem with scaling, Theorem 18, with high probability Z^t stays above $(\varepsilon - \varepsilon')n$ for a superpolynomial number of steps. This concludes the proof of b). □

3.4 SIMULATIONS

This section aims to provide empirical support to our theoretical results. Namely, we show that there exist parameters s and F such that OneMax is optimized efficiently by the SA-$(1, \lambda)$-EA, while Dynamic BinVal is not. Moreover, in the simulations we find that the claim also extends to non-dynamic functions, such as BinVal and Binary[7]. In all experiments, we set $n = 1000$, the update strength $F = 1.5$, and the mutation rate to

7 Defined as $\text{BinVal}(x) = \sum_{i=1}^{n} 2^{i-1} x_i$ and $\text{Binary}(x) = \sum_{i=1}^{\lfloor n/2 \rfloor} x_i n + \sum_{i=\lfloor n/2 \rfloor + 1}^{n} x_i$.

be $1/n$. Then we start the SA-$(1,\lambda)$-EA with the zero string and an initial offspring size of $\lambda^{\text{init}} = 1$. The algorithm terminates when the optimum is found or after $500n$ generations. The code for the simulations can be found at https://github.com/zuxu/OneLambdaEA.

We first show that the improvement probability p_{imp} of OneMax is the lowest among all considered monotone functions (Figure 3.1), while it is highest for Dynamic BinVal and Binary (partly covered by by the violet line). Hence the fitness landscape looks hardest for OneMax with respect to fitness improvements. Therefore, to maintain a target success probability of $1/(1+s)$, more offspring are needed for OneMax, and the SA-$(1,\lambda)$-EA chooses a slightly higher λ (first panel of Figures 3.2 and 3.3, partly covered by the green line). This contributes positively to the drift towards the optimum, so that the drift for OneMax is higher than for the other functions (second panel).

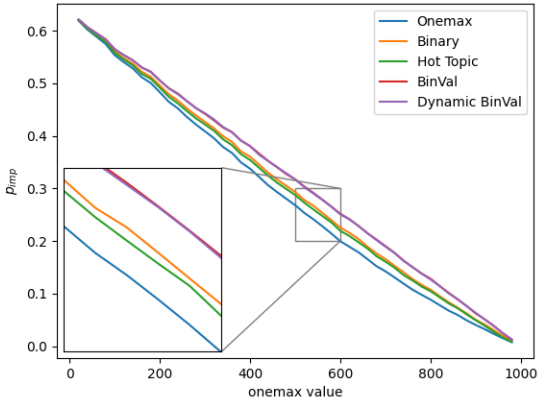

FIGURE 3.1: The probability of fitness improvement with a single offspring for search points with different OneMax values. Each data point in the figure is estimated by first sampling 1000 search points of the corresponding OneMax value, then sampling 100 offspring for each of the sampled search points, and calculating the frequency of an offspring fitter than its parent.

Figure 3.2 summarizes our main result of this section. Clearly, we observe that the SA-$(1,\lambda)$-EA gets stuck on all considered monotone functions except OneMax when the number of one-bits in the search point is between $0.55n$ and $0.65n$. Although the algorithm spends a bit more generations on

ONEMAX between $0.5n$ and $0.8n$ compared to the other parts, the optimum is found rather efficiently (not very explicit in the figure though, since the number is normalized). The reason is, all functions except ONEMAX have negative drifts somewhere within the interval $(0.58n, 0.8n)$. The drift of ONEMAX here is also small compared to the other regions, but remains positive. We also note that the optimum is also found for HOTTOPIC [15] despite its negative drift at $0.7n$, probably because the drift is so weak that it can be overcome by random fluctuations.

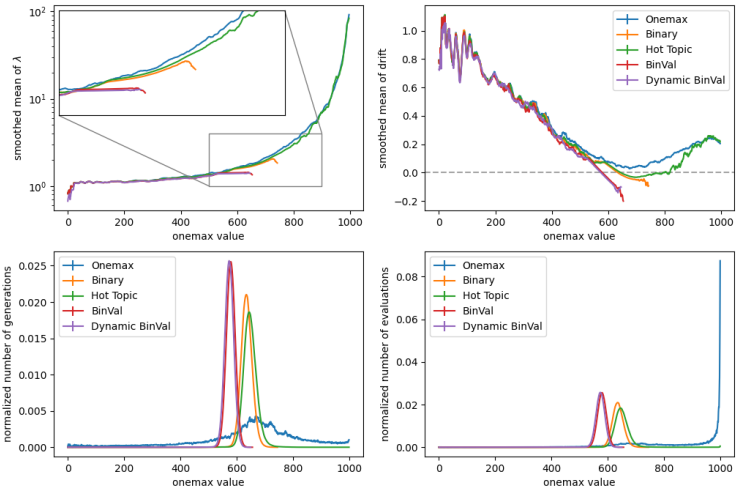

FIGURE 3.2: Smoothed average of λ, smoothed average drift, average number of generations, and average number of evaluations of the self-adjusting $(1, \lambda)$-EA with $s = 3$, $F = 1.5$, and $c = 1$ in 100 runs at each ONEMAX value when optimizing monotone functions with $n = 1000$. The parameters of HOTTOPIC (see Section 2.2.3 for its definition) are $L = 100$, $\alpha = 0.25$, $\beta = 0.05$, and $\varepsilon = 0.05$. The average of λ is shown in log scale. The average of λ and the average drift are smoothed over a window of size 15. The number of generations/evaluations is normalized such that its sum over all ONEMAX values is 1.

As a comparison, we show in Figure 3.3 the situation when s is smaller. Due to a smaller value of s, all functions have positive drifts toward the optimum most of the time during the optimization progress, and the global

optimum is found for all of them. As the 'hardest' function considered in our simulations, DYNAMIC BINVAL has a lower drift compared to the other functions after $0.6n$ and a slightly negative drift between $0.7n$ and $0.8n$, which leads to a peak in generations during this particular interval.

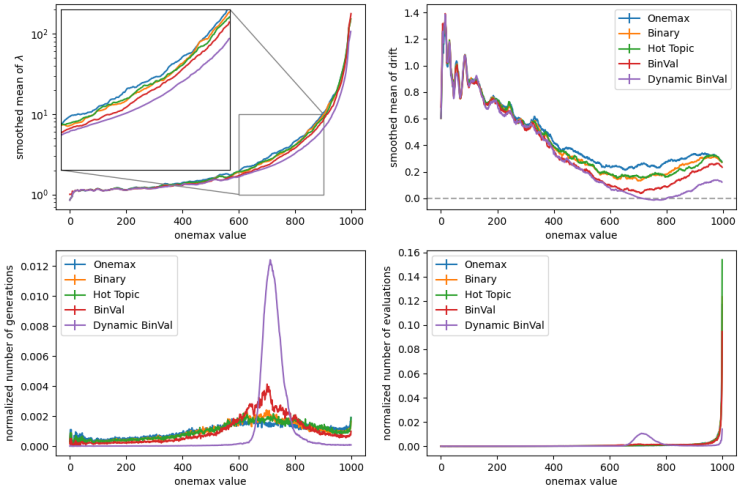

FIGURE 3.3: Similar to Figure 3.2 with the only difference being $s = 2$.

3.5 CONCLUSION

The key insight of our work is that there are two types of "easiness" of a benchmark function[8], which need to be separated carefully. The first type relates to the question of how much progress an elitist hillclimber can make on the function. In this sense, it is well-known that ONEMAX is indeed the easiest benchmark among all functions with unique global optimum. However, a second type of easiness is how likely it is that a mutation gives an improving step. Here ONEMAX is not the easiest function.

Once those concepts are mentally separated, it is indeed not hard to see that ONEMAX is not the easiest function with respect to the second type. In this chapter we have shown that DYNAMIC BINVAL is easier (Lemma 23),

8 We note that there are other types of "easiness", e. g. easiness with respect to a fixed budget.

but we conjecture that this actually holds for many other functions as well, including static ones. This is backed up by experimental data, but we are lacking a more systematic understanding of which functions are hard or easy in this aspect.

We have also shown that the second type of easiness is *relevant*. In particular, the SA-$(1, \lambda)$-EA relies on an empirical sample of the second type of easiness (aka the improvement probability) to choose the population size. Since the SA-$(1, \lambda)$-EA may make bad choices for too easy settings (of second type) if the parameter s is set too high, it is important to understand how easy a fitness landscape can get. These easiest fitness landscapes will determine the range of s that generally makes the SA-$(1, \lambda)$-EA an efficient optimizer. We have disproved the conjecture from [26, 27] that OneMax is the easiest function (of second type). As an alternative, we conjecture that the easiest function is the 'adversarial' Dynamic BinVal, defined similarly to Dynamic BinVal with the exception that the permutation is not random, but chosen so that any 0-bit is heavier than all 1-bits. With this fitness function, any mutation in which at least one 0-bit is flipped gives a fitter child, regardless of the number of 1-bit flips, so it is intuitively convincing that it should be the easiest function with respect to fitness improvement.

4

SPARSE DALE NETWORKS CAN APPROXIMATE KERNEL FUNCTIONS

This chapter is based on joint work with Angelika Steger.

4.1 INTRODUCTION

The olfactory system of Drosophila has long been studied due to its simplicity and capability of associative learning [40, 41]. One of its most distinctive characteristics is the sparse unstructured connectivity between projection neurons (PNs) and Kenyon cells (KCs), i.e. each KC integrates sensory input from 7 out of roughly 50 distinct types of PNs on average [44]. Nevertheless, Drosophilas are able to learn stereotypical responses to distinct odorants quickly [89–91].

The projection from PNs to KCs is usually modeled as a 2-layer network with PNs being the input layer and KCs being the output layer [46, 92]. In machine learning, it is known that with rectified linear units (RELU)[1] being the activation function, when the output layer contains a large number of units and their incoming weights are drawn from independent normal distributions, the network approximates, in the limit, a feature mapping that corresponds to the first order arc-cosine kernel [48]. Kernels are functions that measure the similarity between two data points in certain implicit high dimensional space [93]. Intuitively, this means that the output layer extracts random features from the input, and provides general-purpose representations for downstream tasks. However, a single PN can not excite and inhibit KCs at the same time by Dale's law. And in fact, the projection from PNs and KCs is sparse and consists of only excitatory synapses, so a natural question is whether such a construction can provide a similarly powerful feature extraction as a dense, not sign-constrained network.

In this chapter, we show that a sparse network with only excitatory connections can also approximate the first-order arc-cosine kernel, and that the key to this is an inhibitory cell: the so-called, anterior paired lateral (APL) neuron [49–51]. The APL neuron receives input from KCs and provides feedback inhibition to all KCs. We model the APL neuron in a

1 $\mathrm{RELU}(x) = \max(0, x)$.

similar way to existing computational models [46, 92], and show that this inhibition has an effect of densifying the net and shifting the mean of the weights to 0.

The quality of kernel approximation depends on the number of units in the second layer, the sparsity of the connection, as well as the distribution of the weights. We show that a discrepancy measured between the weights of the network and a rotationally invariant distribution can predict how close the approximation is. We empirically show that a reasonably large network size together with various sparsity levels and weight initialization leads to a low discrepancy, as well as good performance on downstream learning tasks. On the other hand, due to the noise in synaptic transmission, sparse connections are favored in terms of noise resistance.

4.1.1 *Related work*

In [46, 94] properties of the PN-KC network and how these depend on the sparsity are studied. In particular, [46] measures the dimension of the KC representations and shows that it correlates well with classification performance on downstream tasks. They also show that networks with experimentally observed sparsity generate representations of the largest dimension. In a more recent work, [94] shows that sparse connections emerge when training a fully connected network to perform an artificial odorant classification task. It shows that with a non-negative constraint for the network weights, gradient descent will lead the network to converge to a sparse one. Moreover, the sparse network is robust in the sense that perturbations on the weights do not result in a huge difference in the KC representation.

Similarity search, a key problem in large-scale information retrieval, also gets inspiration from the PN-KC network. The goal of similarity search is to return a set of near neighbors to the given query as fast as possible. As the database is huge, hashing is usually applied to avoid exhaustive searches. Inspired by the sparse net in Drosophila, [95] proposes fly hash, which replaces hash functions in similarity search by a sparse network. Empirical results show that sparsifying the connections as in fly hash does not affect the search precision and benefits from less computation. A follow-up paper [96] proposes dense hash, which is basically the same as fly hash except that it omits the global inhibition for KCs. Dense hash is better than fly hash in similarity search, due to the fact that global inhibition drops

more information from the input. In our work, we also observe that denser networks slightly outperform their sparser counterparts in learning tasks.

4.2 PRELIMINARIES

In this section, we introduce the necessary background regarding arc-cosine kernels and their approximations from neural networks. In terms of notation, we use bold lowercase letters to denote vectors and bold capital letters for matrices, for example, $\mathbf{x}$ and $\mathbf{W}$. When referring to elements in vectors and matrices, we use non-bold letters with subscripts such as x_i and W_{ij}. The norm of vectors and matrices considered in this work is the L^2 norm, which is simply denoted by $\|\cdot\|$.

4.2.1 *Arc-cosine kernels and neural networks*

A fully connected two-layer neural network with weights initialized by a normal distribution and a wide output layer equipped with certain non-linear activation can be regarded as feature mappings that correspond to the family of arc-cosine kernel functions [48]. Consider a neural network with d input units and m output units, let w_{ij} denote the weight from the j-th input unit to the i-th output unit. If each w_{ij} follows a standard normal distribution independently and an element-wise RELU non-linearity is applied at each output unit, the network implements a feature map from $\mathbb{R}^d$ to $\mathbb{R}^m$: $\mathbf{f}(\mathbf{x}) = \text{Relu}(\mathbf{W}\mathbf{x})$, where $\mathbf{W}$ is the weight matrix whose entry on i-th row and j-th column is w_{ij}. Let $\mathbf{w}_i$ denote the i-th row of $\mathbf{W}$, then the inner product between the output representations of two input $\mathbf{x}$ and $\mathbf{y}$ is

$$\mathbf{f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{y}) = \sum_{i=1}^{m} \text{Relu}(\mathbf{w}_i\mathbf{x}) \cdot \text{Relu}(\mathbf{w}_i\mathbf{y}).$$

The first-order arc-cosine kernel is defined as

$$k(\mathbf{x}, \mathbf{y}) := \|\mathbf{x}\|\|\mathbf{y}\| \left(\frac{1}{\pi} \sin\theta + \frac{\pi - \theta}{\pi} \cos\theta \right), \tag{4.1}$$

where $\theta = \arccos \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$. When m approaches infinity, the following equation can be established:

$$k(\mathbf{x}, \mathbf{y}) = \lim_{m\to\infty} \frac{2}{m} \mathbf{f}(\mathbf{x}) \cdot \mathbf{f}(\mathbf{y}) = 2 \int \mathbf{dw} \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{2\pi^{d/2}} \text{Relu}(\mathbf{w}\mathbf{x}) \cdot \text{Relu}(\mathbf{w}\mathbf{y}). \tag{4.2}$$

That is, the kernel evaluation can be approximated by computing the inner product between the output representation of the inputs. Details of the derivation can be found in the appendix in [48].

4.2.2 *Invariance of the weight distribution*

Actually, the distribution that $\mathbf{w}$ follows does not need to be standard normal, but it can be any rotationally invariant distribution [97]. A probability density function f is rotationally invariant if for all $\mathbf{w}$ and orthogonal matrices $\mathbf{R}$,

$$f(\mathbf{w}) = f(\mathbf{R}\mathbf{w}) = f(\|\mathbf{w}\|).$$

Therefore, as long as $\mathbf{w}$ is sampled from a rotationally invariant distribution, the distribution of $\|\mathbf{w}\|$ does not affect the kernel up to a scaling factor. However, in practice, due to the constraint of the sample size, normalization is usually applied on $\mathbf{w}$ [98] to improve sample efficiency.

4.3 RESULTS

In this section, we propose our model of the PN-KC network with global inhibition and study it as an approximation to the arc-cosine kernel. We further show that the approximation quality is correlated with a measure termed discrepancy defined on the weights on the network, and that the maximum eigenvalue of the approximated kernel could have an impact on learning tasks based on the output of the network.

4.3.1 *A network model with global inhibition*

The projections from PNs to KCs in the olfactory pathway of Drosophila can be modeled with a two-layer neural network, where we regard PNs as input units and KCs as output units. Due to biological constraints, the weight $\mathbf{W}$ should be non-negative and sparse. In this subsection, we show that a bio-plausible global inhibition mechanism can help the network to get around constraints on sparsity and non-negativity.

Instead of simply applying RELU to each hidden unit, we adopt an activation function that is similar to the one in [92]. Namely, let the summed input to the i-th KC be $\mathbf{w}_i\mathbf{x}$, the output of KC i is calculated by

$$z_i = \text{Relu}\left(\mathbf{w}_i\mathbf{x} - \frac{1}{m}\sum_{i'=1}^{m}\mathbf{w}_{i'}\mathbf{x}\right). \tag{4.3}$$

To interpret the formula, consider that all KCs project to an inhibitory APL neuron with excitatory synapses of weight 1, and the APL neuron projects back to all KCs with inhibitory synapses of weight $1/m$. Simplifying equation (4.3) by defining $w'_{ij} = w_{ij} - \frac{1}{m}\sum_{i'=1}^{m} w_{i'j}$, we obtain

$$\begin{aligned} z_i &= \text{Relu}\left(\sum_{j=1}^{d}\left(w_{ij} - \frac{1}{m}\sum_{i'=1}^{m} w_{i'j}\right)x_j\right) \\ &= \text{Relu}\left(\sum_{j=1}^{d} w'_{ij}x_j\right) = \text{Relu}(\mathbf{w}'_i\mathbf{x}). \end{aligned}$$

This implies the output of the network with weights $\mathbf{W}$ and global inhibition is equivalent to another network that is parameterized by $\mathbf{W}'$ without inhibition. In this way, even though the original weight $\mathbf{W}$ is sparse and non-negative, the equivalent weights $\mathbf{W}'$ can be dense and negative by integrating the effect of inhibition. Therefore, if the distribution of $\mathbf{W}'$ is roughly rotationally invariant, the network can be a close approximation to the arc-cosine kernel.

4.3.2 *Kernel approximation and discrepancy*

Let $\{\mathbf{x}_i\}_{i=1}^{n}$ be the set of input, $\mathbf{K}$ be the kernel matrix where $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{G}$ be the approximated kernel matrix (or Gram matrix) where $G_{ij} = c_{\mathbf{W}'}\sum_{l=1}^{m}\sigma(\mathbf{w}'_l\mathbf{x}_i)\sigma(\mathbf{w}'_l\mathbf{x}_j)$, where $c_{\mathbf{W}'} = 2d/(\sum_{i=1}^{m}\|\mathbf{w}'_i\|^2)$ is a normalization factor and σ denotes RELU. We are interested in the norm of $\mathbf{E} := \mathbf{G} - \mathbf{K}$. Similar to Appendix C in [99], we can show that the L^2 norm of $\mathbf{E}$ is

$$\|\mathbf{E}\| \leq \sqrt{\sum_{i\neq j} E_{ij}^2} + \max_i |E_{ii}|,$$

and that $\mathbb{E}[E_{ij}^2]$ and $\max_i |E_{ii}|$ can be related to the L_2-discrepancy of the weights $\mathbf{W}'$, which measures how rotationally invariant the distribution of $\mathbf{W}'$ is.

We generalize the definition of discrepancy in [99] as follows[2]. Given a probability distribution on $\mathbb{R}^d$ with density ρ, we would like to measure

2 The kernel studied there is not dependent on norms of the input. While in our context, norms are needed for evaluating the kernel function, see (4.1).

how rotationally invariant it is. To do so, we first define a rotationally invariant distribution over $\mathbb{R}^d$ whose density ϕ satisfies

$$\int_{\mathbb{S}_r^{d-1}} \rho(\mathbf{x})d\mathbf{x} = \int_{\mathbb{S}_r^{d-1}} \phi(\mathbf{x})d\mathbf{x},$$

for all $r \in (0, \infty)$, where $\mathbb{S}_r^{d-1}$ denotes the $(d-1)$-sphere with radius r. Namely, $\Pr_\rho[\|\mathbf{x}\| = r] = \Pr_\phi[\|\mathbf{x}\| = r]$. Now given a set of m points $\mathbf{W} := \{\mathbf{w}_i\}_{i=1}^m$ drawn according to ρ, ideally we want to measure how much $\mathbf{W}$ is aligned with ϕ, the rotationally invariant counterpart of ρ. Note that in this work we are always interested in the weight $\mathbf{W}'$ defined in the last subsection, here we just write $\mathbf{W}$ for simplicity. However, since an analytic formula for ϕ can be hard to obtain, we use an empirical approximation $\phi_{\mathbf{W}}$ such that

$$\Pr_{\phi_{\mathbf{W}}}[\|\mathbf{x}\| = r] = \frac{1}{m}\sum_{i=1}^m \mathbb{1}\{\|\mathbf{w}_i\| = r\}.$$

Now the discrepancy between $\mathbf{W}$ and $\phi_{\mathbf{W}}$ with respect to a non-negative function f is defined as

$$\mathrm{D}(W, f) = \left|\frac{1}{m}\sum_{i=1}^m f(\mathbf{w}_i) - \int_{\mathbb{R}^d} f(\mathbf{w})\phi_{\mathbf{W}}(\mathbf{w})d\mathbf{w}\right|. \tag{4.4}$$

The first term in the absolute value can be interpreted as a Monte Carlo estimate of the mean of f, while the second is exactly the mean with respect to $\phi_{\mathbf{W}}$.

To establish the relationship between $\mathbf{E}$ and the discrepancy, we need to work with a special family of functions. Let $\mathcal{F}$ be a family of functions on $\mathbb{R}^d$ defined as follows,

$$\mathcal{F} = \{f_{\mathbf{x},\mathbf{y}} \mid \mathbf{x}, \mathbf{y} \in \mathbb{R}^{d-1}\},$$

where $f_{\mathbf{x},\mathbf{y}}(\mathbf{w}) = \sigma(\mathbf{w}\mathbf{x})\sigma(\mathbf{w}\mathbf{y})$ and σ denotes RELU. $\mathbf{x}$ and $\mathbf{y}$ can be considered as input to the network. The L_∞ discrepancy of $\mathbf{W}$ with respect to $\mathcal{F}$ is defined as

$$L_\infty(\mathbf{W}, \mathcal{F}) := \sup_{f\in\mathcal{F}} \mathrm{D}(\mathbf{W}, f), \tag{4.5}$$

and the L_2 discrepancy

$$L_2(\mathbf{W}, \mathcal{F}) := \sqrt{\mathbb{E}_{\mathbf{x},\mathbf{y}}\,\mathrm{D}^2(\mathbf{W}, f_{\mathbf{x},\mathbf{y}})}. \tag{4.6}$$

Since $\mathcal{F}$ is clear from the context, we write $L_\infty(\mathbf{W})$ and $L_2(\mathbf{W})$ for short.

If both $\mathbf{x}$ and $\mathbf{y}$ are drawn from the input distribution of the network, we have

$$\mathbb{E}[E_{ij}^2] = c_{\mathbf{W}}^2 L_2(\mathbf{W})^2, \quad \text{and} \quad \max_i |E_{ii}| \leq c_{\mathbf{W}} L_\infty(\mathbf{W}),$$

see calculations in Section 4.6.1. Therefore, the kernel approximation is more accurate the discrepancies are small. However, since L_∞ is difficult to estimate, we will focus on L_2, which is also a lower bound for L_∞.

4.3.3 *A formula of discrepancy*

It turns out that when the input $\mathbf{x}$ and $\mathbf{y}$ are drawn from normal distribution, the L_2 discrepancy has a relatively simple expression:

$$\mathbb{E}[E_{ij}^2] = c_{\mathbf{W}}^2 L_2(\mathbf{W})^2 = \frac{d^2}{(\sum_{i=1}^m \|\mathbf{w}_i\|^2)^2} \sum_{i,j=1} k(\mathbf{w}_i, \mathbf{w}_j)^2 - \mathbb{E}_{\mathbf{x},\mathbf{y}}[k(\mathbf{x}, \mathbf{y})^2], \quad (4.7)$$

see calculations in Section 4.6.2. However, the input we are interested in does not necessarily follow a normal distribution, so we first need to argue why it is reasonable to make this assumption. For simplicity, we restrict all vectors ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{w}$) on the unit sphere, and consider how to achieve low L_∞ discrepancy defined in (4.5). When $\mathbf{x}$ and $\mathbf{y}$ that define $f_{\mathbf{x},\mathbf{y}}$ are drawn from a normal distribution, that is they are uniformly distributed on the unit sphere, the best set of $\mathbf{w}$ should be distributed on the unit sphere as uniform as possible. Another extreme case is that $\mathbf{x}$ and $\mathbf{y}$ can only be drawn from the set $\{\mathbf{o}, \mathbf{o}'\}$, whose elements are two orthogonal vectors. Now there are many good configurations of $\mathbf{w}$. Let us denote the region $\{\mathbf{w} \in \mathbb{S}_1^{d-1} \mid \mathbf{wx} > 0, \mathbf{wy} > 0\}$ by R and its complement by $\bar{R}$. Clearly, we have $f = 0$ for all $\mathbf{w} \in \bar{R}$. Let $A(\cdot)$ denote the surface area on $\mathbb{S}_1^{d-1}$. As long as the ratio between the number of points in R and $\bar{R}$ is $A(R)/A(\bar{R})$ and that the points in R are distributed uniformly, a low L_∞ can be achieved. In particular, the points in $\bar{R}$ do not need to be uniformly distributed. Therefore, the assumption of normally distributed input is a rather strong assumption, and a small value from (4.7) is likely to imply low discrepancy under other input distributions. In fact, we observe from Section 4.4.1 that the discrepancy computed with this assumption is well correlated with kernel approximation empirically.

The analytic expression (4.7) hints towards several desired properties of the weight distribution. Since $\mathbb{E}_{\mathbf{x},\mathbf{y}}[k(\mathbf{x}, \mathbf{y})^2]$ is a constant, the expected

approximation error is proportional to the sum of pairwise squared kernel value of the weight $\mathbf{W}$ up to a normalization factor. Therefore,

$$\mathbb{E}[E_{ij}^2] \propto \frac{1}{(\sum_{i=1}^m \|\mathbf{w}_i\|^2)^2} \sum_{i,j=1} \|\mathbf{w}_i\|^2 \|\mathbf{w}_j\|^2 J^2(\theta_{ij}), \tag{4.8}$$

where θ_{ij} is the angle formed by $\mathbf{w}_i$ and $\mathbf{w}_j$, and $J(\theta) = \sin\theta/\pi + (\pi - \theta)\cos\theta/\pi$ according to (4.1).

To separate the effect of norms and angles, we first make the simplification that $\|\mathbf{w}_i\| = 1$ for all i. Then it holds

$$\mathbb{E}[E_{ij}^2] \propto \sum_{i,j=1} J^2(\theta_{ij}) \geq \frac{1}{m^2}\Big(\sum_{i,j=1} J(\theta_{ij})\Big)^2, \tag{4.9}$$

where the equality can be established when $\theta_{ij} = \theta$ for all i and j. Therefore, for a low expected error, the distribution of θ_{ij} should be as uniform as possible. This is reasonable as it discourages $\mathbf{w}$ to cluster.

To investigate the impact of the norms, we assume that the distribution of θ_{ij} is independent of $\|\mathbf{w}_i\|$ and $\|\mathbf{w}_j\|$. Now the expected error is simply a weighted sum of an array of constants θ_{ij}, where the weights are $a_{ij} = \|\mathbf{w}_i\|^2\|\mathbf{w}_j\|^2/(\sum_{i=1}^m \|\mathbf{w}_i\|^2)^2$. In this case, the more uniform the distribution $\|\mathbf{w}_i\|$ is, the smaller variance of the expected error. In Section 4.4.1, we also show this lead to smaller expected errors through simulations.

To summarize, in the simplified scenarios considered above, it is beneficial to have weight distributions that lead to uniform θ_{ij} and $\|\mathbf{w}_i\|$ in terms of kernel approximation.

4.3.4 *Learning speed and discrepancy*

Next, we show that when learning a linear function via a local learning rule, better kernel approximation can imply faster learning in theory. More precisely, the maximum eigenvalue of the Gram matrix $\mathbf{G}$ controls the learning speed, and it can be bounded by the discrepancy.

Assume that there is a readout neuron that tries to predict $y_i = f(\mathbf{x}_i)$ based on the output representation $\mathbf{z}_i$, where f is a function to be approximated. Denote the adaptable weight between the readout neuron and a KC j by v_j, the output of the readout neuron is $\bar{y}_i = \sum_{j=1}^m v_j z_{ij}$. The goal is to have the mean squared error $\sum_{i=1}^n (y_i - \bar{y}_i)^2/2 = \|\mathbf{Z}\mathbf{v} - \mathbf{y}\|^2/2$ as small as possible. The reason why we pick the least square as objective function is that it can be learned via a bio-plausible local learning rule

$$v_j \leftarrow v_j + \eta z_{ij}(y_i - \bar{y}_i), \tag{4.10}$$

where η is a constant denoting the learning rate. Note that the rule is just an application of gradient descent.

By the descent lemma, if the gradient is ℓ-Lipschitz continuous, $\eta = \ell^{-1}$ is guaranteed to decrease the objective if it is not zero. Since for least squares, the minimum ℓ is the maximum eigenvalue of the Hessian $\mathbf{Z}^\top\mathbf{Z}$, we have $\eta = \lambda_{\max}^{-1}(\mathbf{Z}^\top\mathbf{Z}) = \lambda_{\max}^{-1}(\mathbf{Z}\mathbf{Z}^\top) \propto \lambda_{\max}^{-1}(\mathbf{G})$. Namely, the smaller the largest eigenvalue of $\mathbf{G}$ is, the larger learning rate we can use. Note that this is a worse case bound, in practice we can usually use much larger learning rates. When the discrepancy is low, i.e. the Gram matrix $\mathbf{G}$ is close to the kernel matrix $\mathbf{K}$, we can bound $\lambda_{\max}(\mathbf{G}) \leq \lambda_{\max}(\mathbf{K}) + \|\mathbf{G} - \mathbf{K}\|$, since $\lambda_{\max}(\cdot) = \|\cdot\|$ for L^2 norm. As $\lambda_{\max}(K)$ is fixed, being close to the kernel implies smaller $\lambda_{\max}(G)$.

4.4 SIMULATIONS

In this section, we aim to verify by simulation that sparse networks with non-negative weights and proper inhibition can be decent approximations of the first-order arc-cosine kernel. Assume that all weights are drawn from the same non-negative distribution, and let a random variable W denote the value of w_{ij}. To take sparsity into account, we further assume

$$W = \begin{cases} X & \text{with probability } p, \\ 0 & \text{otherwise,} \end{cases}$$

where $p \in (0, 1]$ controls the sparsity of the network, and X is a random variable such that $\Pr[X \leq 0] = 0$. The distributions X follows in the simulations include half normal, exponential, and log normal with default parameters. We also consider three special distributions: the first is normal distribution, which is added for comparison although it can take negative values; for the second one we have $X = 1$ with probability 1; the last one is the empirical distribution of synaptic counts between PNs and KCs, see Figure 4.1. The axon of a neuron can form multiple synapses with the dendrite of another neuron, the number of those can be taken as an approximation to their connection strength. The data is obtained through the publicly available database from [4].

The default input data we use in simulations is a collection of neural recordings of 24 olfactory receptor neurons for 110 odorants [100], since we are modeling a neural network in the olfactory system. In the end, we also

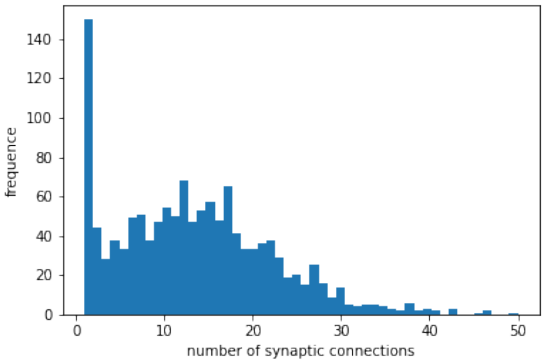

FIGURE 4.1: The empirical distribution of the number of synaptic contacts between a pair of PN and KC.

use MNIST [101] to show that the network is capable of learning a more complicated task, i. e. learning to classify handwriting digits.

To generate the random features, we feed the input through a random network parameterized by X, the distribution of the non-zero weights, p, the sparsity, and m, the number of output units. The nonlinearity in (4.3) is then applied element-wise at the output layer.

4.4.1 *Sparse nets have low discrepancy*

In Figure 4.2 and 4.3, we plot measurements of kernel approximation quality with various distributions for X and a wide range of values for p and m. In subplot (c) we have the alignment between $\mathbf{K}$ and its approximation $\mathbf{G}$, which is the inner product between $\mathbf{K}$ and $\mathbf{G}$ if we consider them as vectors. As p and m increase, all measurements indicate that the approximation gets better. The only exception is when X takes constant values, since all output units essentially extract the same feature as p approaches 1. Despite that the difference brought by different parameters is large in terms of L^2 norm, both discrepancy and alignment indicate small differences as long as the network is not too sparse and the network is not too small. This might explain why the sparse network in the brain of Drosophila works well.

In all three subplots, the ordering of the considered distributions does not vary much. Normal distribution and constant always offer the best approximation quality, which are then followed by the synaptic counts, half normal,

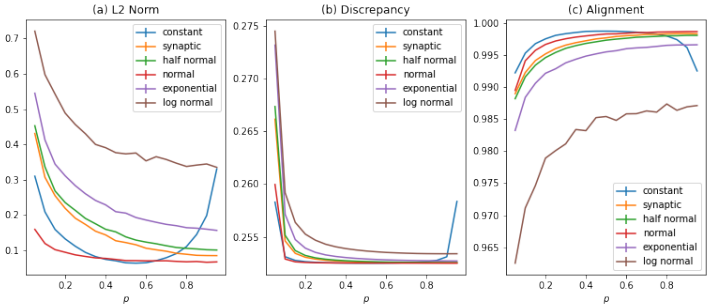

FIGURE 4.2: (a) $\|\mathbf{K} - \mathbf{G}\|/\|\mathbf{K}\|$, (b) discrepancy of $\mathbf{W}'$ with normalization, and (c) kernel alignment between $\mathbf{K}$ and $\mathbf{G}$ with respect to different values of p, where $m = 500$. Plots are obtained by averaging 1000 independent simulations.

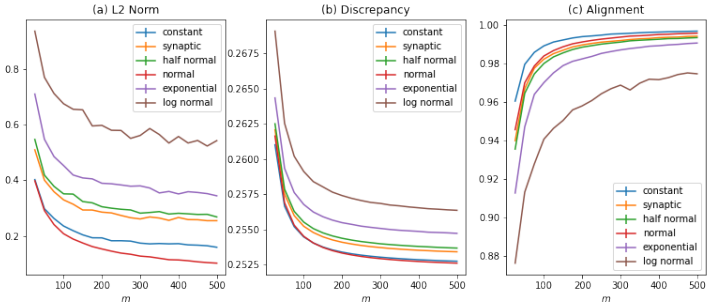

FIGURE 4.3: (a) $\|\mathbf{K} - \mathbf{G}\|/\|\mathbf{K}\|$, (b) discrepancy of $\mathbf{W}'$ with normalization, and (c) kernel alignment between $\mathbf{K}$ and $\mathbf{G}$ with respect to different values of m, where $p = 0.15$. Plots are obtained by averaging 1000 independent simulations.

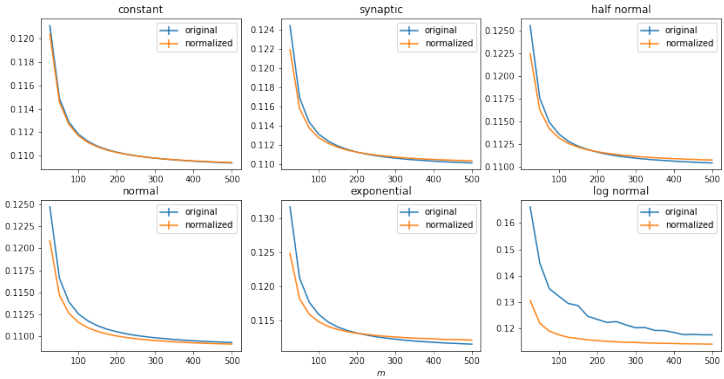

FIGURE 4.4: The discrepancy for original $\mathbf{W}'$ and normalized $\mathbf{W}'$ with respect to different values of m, where $p = 0.15$. Plots are obtained by averaging 1000 independent simulations.

exponential, and log normal. This is correlated with the boundedness of the tails of these distributions: most distributions have sub-Gaussian tails, while exponential distribution has an exponential tail and log normal distribution has a heavy tail. That is, distributions with faster-decaying tails tend to approximate the kernel better. To verify this intuition, we compare the original discrepancy and the one calculated after normalizing all $\|\mathbf{w}'_i\|$ to 1 in Figure 4.4. We find that having a constant norm can bring advantages, especially for small m and log normal. However, even after normalization, log normal is still the worst among all considered distributions. Note that in the actual neural network, it is not feasible to directly normalize $\mathbf{w}'$, as $\mathbf{w}'$ is not the actually non-negative weight $\mathbf{w}$ on the connections.

4.4.2 *The bound of learning speed*

In Figure 4.5, we plot the maximum eigenvalues of the Gram matrix $\mathbf{G}$. Similar as before, large values of p tend to result in beneficial (smaller) maximum eigenvalues, but the effect of increasing p beyond 0.3 is marginal. However, the number of output units m does not have a huge impact on the maximum eigenvalue, see subplot (b). As for the distributions, we observe a similar ordering as in the discrepancy, which is not surprising as we have the bound $\lambda_{\max}(\mathbf{G}) \leq \lambda_{\max}(\mathbf{K}) + \|\mathbf{G} - \mathbf{K}\|$ from Section 4.3.4.

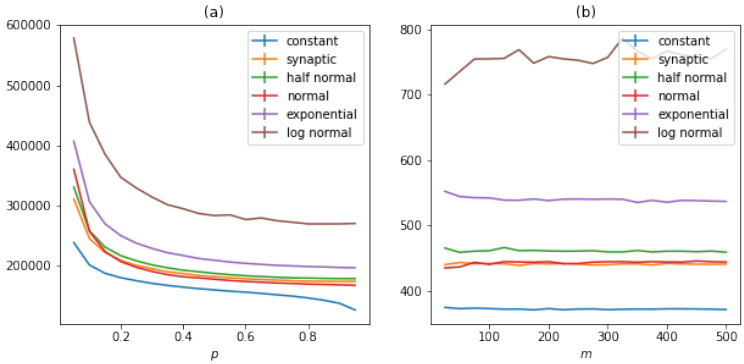

FIGURE 4.5: (a, b) maximum eigenvalues of $\mathbf{G}$ with respect to p (m=500) and m ($p = 0.15$). Note that all eigenvalues are normalized within each subplot. The plot is obtained by averaging 1000 independent simulations.

4.4.3 *Performance in linear regression*

In this subsection, we study the performance of the random networks on a regression task. To generate the target function, we use a random network with $p = 1$, $m = 250$, and weights sampled from normal distribution as the teacher network. After that, a vector $\mathbf{v}$ of 250 entries is sampled from a uniform distribution, and the inner product between the output of the teacher network and $\mathbf{v}$ gives the target linear function. This setting ensures that as long as we have a good approximation to the arc-cosine kernel, the target function can be learned with a bio-plausible local learning rule (see (4.10) in Section 4.3.4).

For any student network parameterized by p, m, and a weight distribution, the goal is to learn a m-dimensional vector $\mathbf{v}'$ such that the inner product between the output of the student network and $\mathbf{v}'$ is as close as possible to the target function value, in terms of the mean squared difference. We randomly pick 70% of the data as the training set and the remaining as the test set in each independent simulation. Training consists of 5 epochs, and in each epoch the input in the training set is used to update $\mathbf{v}'$ one by one in random order.

Figure 4.6 shows the typical situation as more and more training data is used to update $\mathbf{v}'$. The norm of the gradient drops rapidly with the first few training samples and then goes down rather slowly, and so are the training and testing losses. With the same learning rate and the same number of

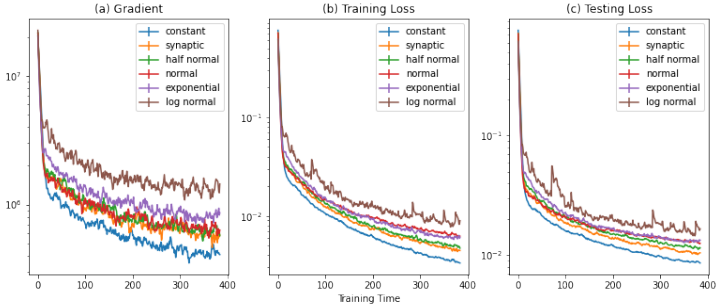

FIGURE 4.6: (a) norm of the gradient, (b) training loss, and (c) testing loss as training proceeds. The parameters for the student networks are $m = 125$ and $p = 0.15$. The learning rate is 0.0001. The plot is obtained by averaging 100 independent simulations.

training samples, distributions with lower discrepancy are able to achieve smaller gradients and losses. The effects of p and m are shown in Figure 4.7. We observe again that larger p and m leads to better performance, in this case, the training and testing losses. However, after the points $p = 0.2$ and $m = 100$ the improvements are rather marginal. The behavior of networks with weights sampled from log normal distribution is rather unstable compared to the other ones, which indicates a coarse positive correlation between discrepancy and the losses.

4.4.4 *Classification on MNIST*

The learning task in the last subsection is rather artificial in the sense that it is tailed to demonstrate the capability of sparse networks to approximate the arccos kernel. Now we aim to show that they are also suitable for learning a more complicated task. Figure 4.8 shows the training progress of a linear classifier based on the sparse random features. With all considered distributions, a testing accuracy of over 95.5% can be reached within 10 epochs, which is comparable to the state-of-the-art results for kernel methods [102]. After 10 epochs, the networks suffer from slight over-fitting, which can be avoided by reducing the learning rate. Although not shown in the plots, the accuracy is not sensitive to the choice of sparsity p.

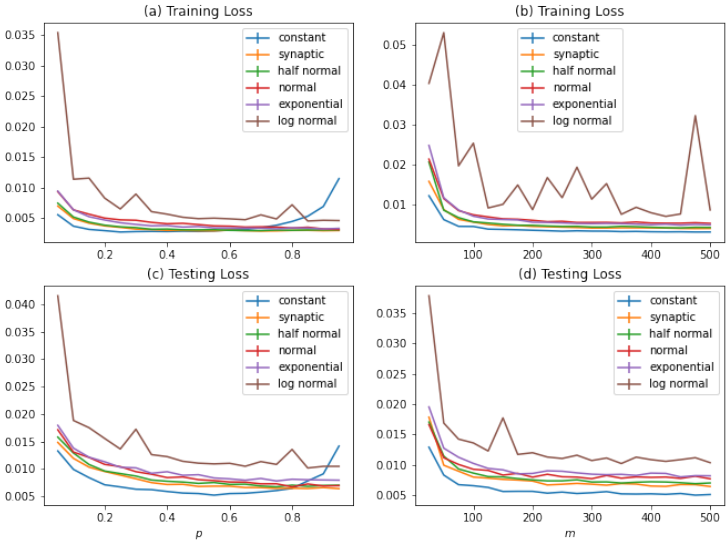

FIGURE 4.7: (a, c) training and testing loss with respect to p ($m = 250$), and (b, d) training and testing loss with respect to m ($p = 0.15$). The learning rate is 0.0001, the same one as in Figure 4.6. The plot is obtained by averaging 100 independent simulations. Standard deviations are omitted for better visualization.

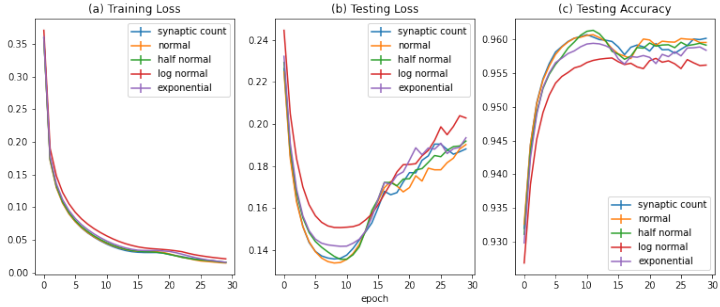

FIGURE 4.8: (a) training loss, (b) testing loss, and (c) testing accuracy on MNIST within the first 30 epochs. The images are first normalized such that each pixel is between 0 and 1. The parameters p and m are set to 0.09 and 2000. The output of the random network is then used to train a linear classifier with stochastic gradient descent and cross-entropy loss. The plot is obtained by averaging 20 independent simulations.

4.4.5 *Noise resistance*

So far, most simulations indicate that larger values of p and using constant weights are beneficial. So why is the connection between PNs and KCs sparse, and why do not two neurons always form a constant number of synapses? In terms of sparsity, one possibility is that sparse networks achieve a good balance between performance and resources. Another hypothesis is that synaptic transmission comes often with noises, and sparse connections can reduce the effect of noises [103]. To verify this hypothesis in our model, after the weights of a random network are drawn, we add independent Gaussian noise to the non-zero weights and obtain two noisy networks. The two noisy networks are then used to generate the random features of the same input, and we consider the angle between the generated features. Obviously, if the network is noise resistant, the interested angle will be small.

Figure 4.9 shows that networks with large p can not produce stable features due to the existence of noises. On the other hand, varying m has only minor effects. The ordering between the distributions is reversed. Namely, the ones with smaller discrepancy are more sensitive to noises. This might explain why KCs do not always form the same number of synapses with PNs in the Drosophila brain.

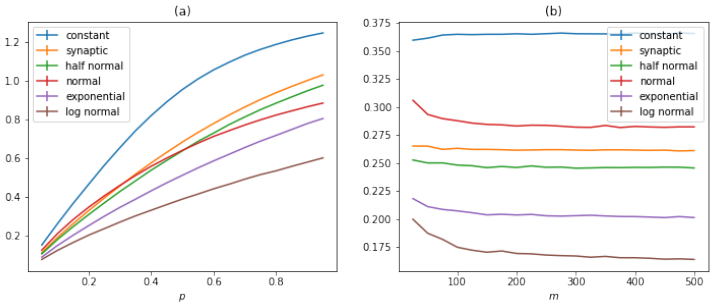

FIGURE 4.9: (a, b) Mean angle between features generated by noise-perturbed networks with respect to p ($m = 500$) and m ($p = 0.15$). The injected Gaussian noise has a standard deviation equal to $0.1 \cdot \mathbb{E}[X]$, while for the normal distribution, this value is $0.1 \cdot \mathbb{E}[|X|]$. The plot is obtained by averaging 1000 independent simulations.

4.5 CONCLUSION

We have shown that a sparse Dale network with global inhibition can approximate the first-order arc-cosine kernel well and produce representations that are suitable for associative learning under a set of mild conditions. The conditions include a reasonable number of output units and connections, as well as weight initialization with a bounded distribution. The quality of the kernel approximation is well correlated with the theoretically derived discrepancy, which measures how rotationally invariant the weight distribution is. Moreover, the discrepancy can be a good prediction of the performance of the network on regression tasks. We also identify two contradicting forces in our network model: on one hand, a low discrepancy is desired to facilitate downstream learning; on the other hand, a low discrepancy contributes negatively to noisy resistance. Therefore, the size, sparsity, and weight initialization need to achieve a good balance between these two factors.

4.6 APPENDIX

4.6.1 *Connecting kernel approximation to discrepancy*

Note that for $\mathcal{F}$, the second term in the definition of discrepancy (4.4) can be simplified as

$$\begin{aligned}\int_{\mathbb{R}^d} f_{\mathbf{x},\mathbf{y}}(\mathbf{w})\phi_{\mathbf{W}}(\mathbf{w})d\mathbf{w} &= \sum_{i=1}^{m} \Pr_{\phi_W}[\|\mathbf{w}\| = \|\mathbf{w}_i\|] \cdot \int_{\mathbb{S}^{d-1}_{\|\mathbf{w}_i\|}} \sigma(\mathbf{w}\mathbf{x})\sigma(\mathbf{w}\mathbf{y}) \frac{1}{A(\mathbb{S}^{d-1}_{\|\mathbf{w}\|})} d\mathbf{w} \\ &= \sum_{i=1}^{m} \frac{1}{m} \cdot \frac{\|\mathbf{w}_i\|^{d+1}}{A(\mathbb{S}^{d-1}_{\|\mathbf{w}_i\|})} \int_{\mathbb{S}^{d-1}_1} \sigma(\mathbf{w}\mathbf{x})\sigma(\mathbf{w}\mathbf{y}) d\mathbf{w}, \end{aligned} \tag{4.11}$$

where $A(\mathbb{S}^{d-1}_{\|\mathbf{w}_i\|})$ denotes the surface area of $\mathbb{S}^{d-1}_{\|\mathbf{w}_i\|}$, and note that in the last step, the integration is switched to the unit sphere. Let χ be the probability density of the Chi distribution with d degrees of freedom, which is the distribution of the norm of a d-dimensional vector whose entries are i.i.d. standard normal. By equation (4.1),

$$\begin{aligned} k(\mathbf{x},\mathbf{y}) &= 2\int_{\mathbb{R}^d} d\mathbf{w} \frac{e^{-\frac{\|\mathbf{w}\|^2}{2}}}{2\pi^{d/2}} \sigma(\mathbf{w}\mathbf{x}) \cdot \sigma(\mathbf{w}\mathbf{y}) \\ &= 2\int_0^\infty \chi(r) \int_{\mathbb{S}^{d-1}_r} \sigma(\mathbf{w}\mathbf{x})\sigma(\mathbf{w}\mathbf{y}) \frac{1}{A(\mathbb{S}^{d-1}_r)} d\mathbf{w} dr \\ &= 2\int_0^\infty \chi(r) \frac{r^{d+1}}{A(\mathbb{S}^{d-1}_r)} dr \int_{\mathbb{S}^{d-1}_1} \sigma(\mathbf{w}\mathbf{x})\sigma(\mathbf{w}\mathbf{y}) d\mathbf{w}. \end{aligned} \tag{4.12}$$

Combining (4.11) and (4.12), and making use of $A(\mathbb{S}^{d-1}_r) \propto r^{d-1}$ and $\int_0^\infty \chi(r) r^2 dr = d$, we obtain

$$\int_{\mathbb{R}^d} f_{\mathbf{x},\mathbf{y}}(\mathbf{w}) d\phi_{\mathbf{W}}(\mathbf{w}) = \frac{c}{m} \sum_{i=1}^{m} \frac{\|\mathbf{w}_i\|^{d+1}}{A(\mathbb{S}^{d-1}_{\|\mathbf{w}_i\|})} \cdot k(\mathbf{x},\mathbf{y}) = c_{\mathbf{W}}^{-1} \cdot k(\mathbf{x},\mathbf{y}), \tag{4.13}$$

where $c = (2\int_0^\infty \chi(r) r^{d+1} A(\mathbb{S}^{d-1}_r)^{-1} dr)^{-1}$ denotes a constant.

Now, if both $\mathbf{x}$ and $\mathbf{y}$ are drawn from the input distribution, we have

$$\mathbb{E}[E_{ij}^2] = c_{\mathbf{W}}^2 L_2(W)^2, \quad \text{and} \quad \max_i |E_{ii}| \le c_{\mathbf{W}} L_\infty(W).$$

4.6.2 *Calculating discrepancy*

Assume that $\mathbf{x}$ and $\mathbf{y}$ are drawn from a distribution with density ι, we can show that

$$L_2(\mathbf{W})^2 = \frac{1}{m^2}\sum_{i,j=1}^{m}\left(\int_{\mathbb{R}^d}\sigma(\mathbf{w}_i\mathbf{x})\sigma(\mathbf{w}_j\mathbf{x})\iota(\mathbf{x})d\mathbf{x}\right)^2 \tag{4.14}$$

$$-\frac{2}{mc_{\mathbf{W}}}\sum_{i=1}^{m}\int_{\mathbb{R}^d}\int_{\mathbb{R}^d}\sigma(\mathbf{w}_i\mathbf{x})\sigma(\mathbf{w}_i\mathbf{y})k(\mathbf{x},\mathbf{y})\iota(\mathbf{x})d\mathbf{x}\iota(\mathbf{y})d\mathbf{y} \tag{4.15}$$

$$+c_{\mathbf{W}}^{-2}\,\mathbb{E}_{\mathbf{x},\mathbf{y}}[k(\mathbf{x},\mathbf{y})^2].$$

If ι is the density of a rotationally invariant distribution, the summands above can be further simplified. To avoid complicated scaling factors, we assume each entry of $\mathbf{x}$ and $\mathbf{y}$ is drawn from the standard normal distribution independently. In particular, we have

$$(4.14) = \frac{1}{4m^2}\sum_{i,j=1}^{m}k(\mathbf{w}_i,\mathbf{w}_j)^2, \tag{4.16}$$

by (4.2) and

$$(4.15) = -\frac{2}{mc_{\mathbf{W}}}\sum_{i=1}^{m}\|\mathbf{w}_i\|^2\int_{\mathbb{R}^d}\int_{\mathbb{R}^d}\sigma\Big(\frac{\mathbf{w}_i}{\|\mathbf{w}_i\|}\mathbf{x}\Big)\sigma\Big(\frac{\mathbf{w}_i}{\|\mathbf{w}_i\|}\mathbf{y}\Big)k(\mathbf{x},\mathbf{y})\iota(\mathbf{x})d\mathbf{x}\iota(\mathbf{y})d\mathbf{y}. \tag{4.17}$$

Due to the symmetry, the value of the integral does not change if we replace $\mathbf{w}_i/\|\mathbf{w}_i\|$ with any vector on the unit sphere. Together with (4.12), the integral in (4.17) is

$$\begin{aligned}
&\int_{\mathbb{S}_1^{d-1}}\int_{\mathbb{R}^d}\int_{\mathbb{R}^d}\sigma(\mathbf{w}\mathbf{x})\sigma(\mathbf{w}\mathbf{y})k(\mathbf{x},\mathbf{y})\iota(\mathbf{x})d\mathbf{x}\iota(\mathbf{y})d\mathbf{y}\frac{1}{A(\mathbb{S}_1^{d-1})}d\mathbf{w}\\
=&\int_{\mathbb{R}^d}\int_{\mathbb{R}^d}\left(\int_{\mathbb{S}_1^{d-1}}\sigma(\mathbf{w}\mathbf{x})\sigma(\mathbf{w}\mathbf{y})\frac{1}{A(\mathbb{S}_1^{d-1})}d\mathbf{w}\right)k(\mathbf{x},\mathbf{y})\iota(\mathbf{x})d\mathbf{x}\iota(\mathbf{y})d\mathbf{y}\\
=&\frac{c}{A(\mathbb{S}_1^{d-1})}\int_{\mathbb{R}^d}\int_{\mathbb{R}^d}k(\mathbf{x},\mathbf{y})^2\iota(\mathbf{x})d\mathbf{x}\iota(\mathbf{y})d\mathbf{y}\\
=&\frac{1}{2d}\,\mathbb{E}_{\mathbf{x},\mathbf{y}}[k(\mathbf{x},\mathbf{y})^2],
\end{aligned} \tag{4.18}$$

where c is the constant in (4.13). So putting together (4.17) and (4.18), we obtain

$$(4.15) = -2c_{\mathbf{W}}^{-2}\,\mathbb{E}_{\mathbf{x},\mathbf{y}}[k(\mathbf{x},\mathbf{y})], \tag{4.19}$$

and therefore by (4.16) and (4.19),

$$L_2(\mathbf{W})^2 = \frac{1}{4m^2} \sum_{i,j=1} k(\mathbf{w}_i, \mathbf{w}_j)^2 - c_{\mathbf{W}}^{-2} \, \mathbb{E}_{\mathbf{x},\mathbf{y}}[k(\mathbf{x}, \mathbf{y})^2].$$

To summarize, we have shown that for standard normal input distribution,

$$\begin{aligned} \mathbb{E}[E_{ij}^2] = c_{\mathbf{W}}^2 L_2(\mathbf{W})^2 &= \frac{c_{\mathbf{W}}^2}{4m^2} \sum_{i,j=1} k(\mathbf{w}_i, \mathbf{w}_j)^2 - \mathbb{E}_{\mathbf{x},\mathbf{y}}[k(\mathbf{x}, \mathbf{y})^2] \\ &= \frac{d^2}{(\sum_{i=1}^m \|\mathbf{w}_i\|^2)^2} \sum_{i,j=1} k(\mathbf{w}_i, \mathbf{w}_j)^2 - \mathbb{E}_{\mathbf{x},\mathbf{y}}[k(\mathbf{x}, \mathbf{y})^2]. \end{aligned}$$

BIBLIOGRAPHY

1. Eiben, A. & Smith, J. *Introduction to Evolutionary Computing* (Springer, 2015).
2. Darwin, C. *On the origin of species, 1859* (Routledge, 2004).
3. Jansen, T. *Analyzing evolutionary algorithms: The computer science perspective* (Springer, 2013).
4. Scheffer, L. K., Xu, C. S., Januszewski, M., Lu, Z., Takemura, S.-y., Hayworth, K. J., Huang, G. B., Shinomiya, K., Maitlin-Shepard, J., Berg, S., *et al.* A connectome and analysis of the adult Drosophila central brain. *Elife* **9**, e57443 (2020).
5. Lyu, C., Abbott, L. & Maimon, G. Building an allocentric travelling direction signal via vector computation. *Nature* **601**, 92 (2022).
6. Whittington, J. C., Warren, J. & Behrens, T. E. *Relating transformers to models and neural representations of the hippocampal formation* in *International Conference on Learning Representations* (2021).
7. Doerr, B. & Neumann, F. *Theory of evolutionary computation: Recent developments in discrete optimization* (Springer Nature, 2019).
8. Zhou, Z.-H., Yu, Y. & Qian, C. *Evolutionary learning: Advances in theories and algorithms* (Springer, 2019).
9. Lobo, F., Lima, C. F. & Michalewicz, Z. *Parameter setting in evolutionary algorithms* (Springer Science & Business Media, 2007).
10. Doerr, B. & Doerr, C. Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. *Theory of evolutionary computation*, 271 (2020).
11. Doerr, B., Jansen, T., Sudholt, D., Winzen, C. & Zarges, C. *Optimizing Monotone Functions Can Be Difficult* in *International Conference on Parallel Problem Solving from Nature* (2010).
12. Doerr, B., Jansen, T., Sudholt, D., Winzen, C. & Zarges, C. Mutation rate matters even when optimizing monotonic functions. *Evolutionary Computation* **21**, 1 (2013).
13. Lengler, J. & Steger, A. Drift analysis and evolutionary algorithms revisited. *Combinatorics, Probability and Computing* **27**, 643 (2018).

14. Lengler, J. *A general dichotomy of evolutionary algorithms on monotone functions* in *International Conference on Parallel Problem Solving from Nature* (2018), 3.

15. Lengler, J. A general dichotomy of evolutionary algorithms on monotone functions. *IEEE Transactions on Evolutionary Computation* **24**, 995 (2019).

16. Witt, C. Runtime analysis of the $(\mu + 1)$ EA on simple pseudo-Boolean functions. *Evolutionary Computation* **14**, 65 (2006).

17. Antipov, D., Doerr, B., Fang, J. & Hetet, T. *A tight runtime analysis for the (μ+λ) EA* in *Genetic and Evolutionary Computation Conference* (2018), 1459.

18. Richter, J. N., Wright, A. & Paxton, J. *Ignoble trails-where crossove r is provably harmful* in *International Conference on Parallel Problem Solving from Nature* (2008), 92.

19. Witt, C. Population size versus runtime of a simple evolutionary algorithm. *Theoretical Computer Science* **403**, 104 (2008).

20. Schumer, M. & Steiglitz, K. Adaptive step size random search. *IEEE Transactions on Automatic Control* **13**, 270 (1968).

21. Devroye, L. *The compound random search* (Ph.D. dissertation, Purdue Univ., West Lafayette, IN, 1972).

22. Rechenberg, I. Evolutionsstrategien. *Simulationsmethoden in der Medizin und Biologie*, 83 (1978).

23. Kern, S., Müller, S. D., Hansen, N., Büche, D., Ocenasek, J. & Koumoutsakos, P. Learning probability distributions in continuous evolutionary algorithms–a comparative review. *Natural Computing* **3**, 77 (2004).

24. Doerr, C. & Wagner, M. *Simple on-the-fly parameter selection mechanisms for two classical discrete black-box optimization benchmark problems* in *Genetic and Evolutionary Computation Conference* (2018), 943.

25. Doerr, B., Doerr, C. & Lengler, J. Self-adjusting mutation rates with provably optimal success rules. *Algorithmica* **83**, 3108 (2021).

26. Hevia Fajardo, M. A. & Sudholt, D. Self-Adjusting Population Sizes for Non-Elitist Evolutionary Algorithms: Why Success Rates Matter. *arXiv preprint arXiv:2104.05624* (2021).

27. Hevia Fajardo, M. A. & Sudholt, D. *Self-adjusting population sizes for non-elitist evolutionary algorithms: why success rates matter* in *Genetic and Evolutionary Computation Conference* (2021), 1151.

28. Kaufmann, M., Larcher, M., Lengler, J. & Zou, X. Self-adjusting Population Sizes for the $(1, \lambda)$-EA on Monotone Functions. *arXiv preprint arXiv:2204.00531* (2022).

29. Kaufmann, M., Larcher, M., Lengler, J. & Zou, X. *Self-adjusting Population Sizes for the (1, λ)-EA on Monotone Functions* in *International Conference on Parallel Problem Solving from Nature* (2022), 569.

30. Doerr, B., Johannsen, D. & Winzen, C. Multiplicative Drift Analysis. *Algorithmica* **64**, 673 (2012).

31. Sudholt, D. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* **17**, 418 (2012).

32. Witt, C. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing* **22**, 294 (2013).

33. Ramesh, A., Dhariwal, P., Nichol, A., Chu, C. & Chen, M. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125* (2022).

34. Hafting, T., Fyhn, M., Molden, S., Moser, M.-B. & Moser, E. I. Microstructure of a spatial map in the entorhinal cortex. *Nature* **436**, 801 (2005).

35. Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M. J., Degris, T., Modayil, J., *et al.* Vector-based navigation using grid-like representations in artificial agents. *Nature* **557**, 429 (2018).

36. Cueva, C. J. & Wei, X.-X. *Emergence of grid-like representations by training recurrent neural networks to perform spatial localization* in *International Conference on Learning Representations* (2018).

37. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. Attention is all you need. *Advances in neural information processing systems* **30** (2017).

38. Hattori, D., Aso, Y., Swartz, K. J., Rubin, G. M., Abbott, L. & Axel, R. Representations of novelty and familiarity in a mushroom body compartment. *Cell* **169**, 956 (2017).

39. Dasgupta, S., Sheehan, T. C., Stevens, C. F. & Navlakha, S. A neural data structure for novelty detection. *Proceedings of the National Academy of Sciences* **115**, 13093 (2018).
40. Connolly, J. B., Roberts, I. J., Armstrong, J. D., Kaiser, K., Forte, M., Tully, T. & O'Kane, C. J. Associative learning disrupted by impaired Gs signaling in Drosophila mushroom bodies. *Science* **274**, 2104 (1996).
41. De Belle, J. S. & Heisenberg, M. Associative odor learning in Drosophila abolished by chemical ablation of mushroom bodies. *Science* **263**, 692 (1994).
42. Schiffman, S. Smell and Taste. *Encyclopedia of Gerontology (Second Edition)*, 515 (2007).
43. Fishilevich, E. & Vosshall, L. B. Genetic and functional subdivision of the Drosophila antennal lobe. *Current Biology* **15**, 1548 (2005).
44. Caron, S. J., Ruta, V., Abbott, L. & Axel, R. Random convergence of olfactory inputs in the Drosophila mushroom body. *Nature* **497**, 113 (2013).
45. Laurent, G. Olfactory network dynamics and the coding of multidimensional signals. *Nature reviews neuroscience* **3**, 884 (2002).
46. Litwin-Kumar, A., Harris, K. D., Axel, R., Sompolinsky, H. & Abbott, L. Optimal degrees of synaptic connectivity. *Neuron* **93**, 1153 (2017).
47. Rahimi, A. & Recht, B. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. *Advances in neural information processing systems* **21** (2008).
48. Cho, Y. & Saul, L. K. Large-margin classification in infinite neural networks. *Neural computation* **22**, 2678 (2010).
49. Leitch, B. & Laurent, G. GABAergic synapses in the antennal lobe and mushroom body of the locust olfactory system. *Journal of comparative Neurology* **372**, 487 (1996).
50. Liu, X. & Davis, R. L. The GABAergic anterior paired lateral neuron suppresses and is suppressed by olfactory learning. *Nature neuroscience* **12**, 53 (2009).
51. Papadopoulou, M., Cassenaer, S., Nowotny, T. & Laurent, G. Normalization for sparse encoding of odors by a wide-field interneuron. *Science* **332**, 721 (2011).
52. Claesen, M. & De Moor, B. *Hyperparameter search in machine learning* in *Metaheuristics International Conference* (2015), 1.

53. Lengler, J. & Zou, X. *Exponential slowdown for larger populations: the (μ+ 1)-EA on monotone functions* in *ACM/SIGEVO Conference on Foundations of Genetic Algorithms* (2019), 87.

54. Lengler, J. & Zou, X. Exponential slowdown for larger populations: The (μ+ 1)-EA on monotone functions. *Theoretical Computer Science* **875**, 28 (2021).

55. Antipov, D. & Doerr, B. A Tight Runtime Analysis for the $(\mu + \lambda)$ EA. *Algorithmica*, 1 (2020).

56. Jansen, T. *On the brittleness of evolutionary algorithms* in *ACM/SIGEVO Conference on Foundations of Genetic Algorithms* (2007), 54.

57. Lengler, J., Martinsson, A. & Steger, A. *When does hillclimbing fail on monotone functions: An entropy compression argument* in *Analytic Algorithmics and Combinatorics* (2019), 94.

58. Doerr, B., Le, H. P., Makhmara, R. & Nguyen, T. D. *Fast genetic algorithms* in *Genetic and Evolutionary Computation Conference* (2017).

59. Kötzing, T., Lagodzinski, J. G., Lengler, J. & Melnichenko, A. *Destructiveness of lexicographic parsimony pressure and alleviation by a concatenation crossover in genetic programming* in *International Conference on Parallel Problem Solving from Nature* (2018), 42.

60. Friedrich, T., Kötzing, T., Krejca, M. S. & Sutton, A. M. *The benefit of recombination in noisy evolutionary search* in *International Symposium on Algorithms and Computation* (2015), 140.

61. Qian, C., Yu, Y. & Zhou, Z.-H. An analysis on recombination in multi-objective evolutionary optimization. *Artificial Intelligence* **204**, 99 (2013).

62. Kötzing, T., Sudholt, D. & Theile, M. *How crossover helps in pseudo-boolean optimization* in *Genetic and Evolutionary Computation Conference* (2011), 989.

63. Lengler, J. & Schaller, U. *The* $(1+1)$*-EA on noisy linear functions with random positive weights* in *Symposium Series on Computational Intelligence* (2018), 712.

64. Doerr, B. Probabilistic tools for the analysis of randomized optimization heuristics. *Theory of evolutionary computation*, 1 (2020).

65. Witt, C. Fitness levels with tail bounds for the analysis of randomized search heuristics. *Information Processing Letters* **114**, 38 (2014).

66. Kötzing, T. Concentration of first hitting times under additive drift. *Algorithmica* **75**, 490 (2016).

67. Lehre, P. K. & Yao, X. On the impact of mutation-selection balance on the runtime of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* **16**, 225 (2012).

68. Sudholt, D. The impact of parametrization in memetic evolutionary algorithms. *Theoretical Computer Science* **410**, 2511 (2009).

69. Lässig, J. & Sudholt, D. Design and analysis of migration in parallel evolutionary algorithms. *Soft Computing* **17**, 1121 (2013).

70. Kaufmann, M., Larcher, M., Lengler, J. & Zou, X. *Onemax is not the easiest function for fitness improvements* in *European Conference on Evolutionary Computation in Combinatorial Optimization* (2023), 162.

71. He, J., Chen, T. & Yao, X. On the easiest and hardest fitness functions. *IEEE Transactions on Evolutionary Computation* **19**, 295 (2014).

72. Corus, D., He, J., Jansen, T., Oliveto, P. S., Sudholt, D. & Zarges, C. On easiest functions for mutation operators in bio-inspired optimisation. *Algorithmica* **78**, 714 (2017).

73. Droste, S. A rigorous analysis of the compact genetic algorithm for linear functions. *Natural Computing* **5**, 257 (2006).

74. Eiben, A. E., Hinterding, R. & Michalewicz, Z. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* **3**, 124 (1999).

75. Badkobeh, G., Lehre, P. K. & Sudholt, D. *Unbiased black-box complexity of parallel search* in *International Conference on Parallel Problem Solving from Nature* (2014), 892.

76. Böttcher, S., Doerr, B. & Neumann, F. *Optimal fixed and adaptive mutation rates for the LeadingOnes problem* in *International Conference on Parallel Problem Solving from Nature* (2010), 1.

77. Doerr, B., Doerr, C. & Ebel, F. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science* **567**, 87 (2015).

78. Doerr, B., Doerr, C. & Yang, J. Optimal parameter choices via precise black-box analysis. *Theoretical Computer Science* **801**, 1 (2020).

79. Doerr, B., Witt, C. & Yang, J. Runtime analysis for self-adaptive mutation rates. *Algorithmica* **83**, 1012 (2021).

80. Auger, A. *Benchmarking the (1+ 1) evolution strategy with one-fifth success rule on the BBOB-2009 function testbed* in *Genetic and Evolutionary Computation Conference* (2009), 2447.

81. Colin, S., Doerr, B. & Férey, G. *Monotonic functions in EC: anything but monotone!* in *Genetic and Evolutionary Computation Conference* (2014), 753.

82. Rowe, J. E. & Sudholt, D. The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm. *Theoretical Computer Science* **545**, 20 (2014).

83. Antipov, D., Doerr, B. & Yang, Q. *The efficiency threshold for the offspring population size of the (μ, λ) EA* in *Genetic and Evolutionary Computation Conference* (2019), 1461.

84. Lengler, J. & Meier, J. *Large population sizes and crossover help in dynamic environments* in *International Conference on Parallel Problem Solving from Nature* (2020), 610.

85. Lengler, J. & Riedi, S. *Runtime Analysis of the $(\mu + 1)$-EA on the Dynamic BinVal Function* in *European Conference on Evolutionary Computation in Combinatorial Optimization* (2021), 84.

86. Lehre, P. & Qin, X. More Precise Runtime Analyses of Non-elitist Evolutionary Algorithms in Uncertain Environments. *Algorithmica*, 1 (2022).

87. Lengler, J. Drift analysis. *Theory of Evolutionary Computation*, 89 (2020).

88. Oliveto, P. & Witt, C. On the Analysis of the Simple Genetic Algorithm. *Theoretical Computer Science* **545**, 2 (2014).

89. Murthy, M., Fiete, I. & Laurent, G. Testing odor response stereotypy in the Drosophila mushroom body. *Neuron* **59**, 1009 (2008).

90. Mittal, A. M., Gupta, D., Singh, A., Lin, A. C. & Gupta, N. Multiple network properties overcome random connectivity to enable stereotypic sensory responses. *Nature communications* **11**, 1 (2020).

91. Aso, Y., Hattori, D., Yu, Y., Johnston, R. M., Iyer, N. A., Ngo, T.-T., Dionne, H., Abbott, L., Axel, R., Tanimoto, H., *et al.* The neuronal architecture of the mushroom body provides a logic for associative learning. *Elife* **3**, e04577 (2014).

92. Endo, K., Tsuchimoto, Y. & Kazama, H. Synthesis of conserved odor object representations in a random, divergent-convergent network. *Neuron* **108**, 367 (2020).

93. Hofmann, T., Schölkopf, B. & Smola, A. J. Kernel methods in machine learning. *The annals of statistics* **36**, 1171 (2008).

94. Wang, P. Y., Sun, Y., Axel, R., Abbott, L. & Yang, G. R. Evolving the olfactory system with machine learning. *Neuron* **109**, 3879 (2021).

95. Dasgupta, S., Stevens, C. F. & Navlakha, S. A neural algorithm for a fundamental computing problem. *Science* **358**, 793 (2017).

96. Sharma, J. & Navlakha, S. Improving Similarity Search with High-dimensional Locality-sensitive Hashing. *arXiv preprint arXiv:1812.01844* (2018).

97. Tsuchida, R., Roosta, F. & Gallagher, M. *Invariance of weight distributions in rectified MLPs* in *International Conference on Machine Learning* (2018), 4995.

98. Zandieh, A., Han, I., Avron, H., Shoham, N., Kim, C. & Shin, J. Scaling neural tangent kernels via sketching and random features. *Advances in Neural Information Processing Systems* **34**, 1062 (2021).

99. Xie, B., Liang, Y. & Song, L. *Diverse neural network learns true target functions* in *Artificial Intelligence and Statistics* (2017), 1216.

100. Hallem, E. A. & Carlson, J. R. Coding of odors by a receptor repertoire. *Cell* **125**, 143 (2006).

101. LeCun, Y. The MNIST database of handwritten digits (1998).

102. Liu, F., Huang, X., Chen, Y. & Suykens, J. A. Random features for kernel approximation: A survey on algorithms, theory, and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **44**, 7128 (2021).

103. Yang, R. G., Wang, P. Y., Sun, Y., Litwin-Kumar, A., Axel, R. & Abbott, L. Evolving the olfactory system (2019).