Designing Superior Evolutionary Algorithms via Insights From Black-Box Complexity Theory

Jing Yang

université PARIS-SACLAY

NNT : 2018SACLX054

Thèse de doctorat

From a Complexity Theory of Evolutionary Computation to Superior Randomized Search Heuristics

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Ecole Polytechnique

Ecole doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 04 Sep 2018, par

MME JING YANG

Composition du Jury :

Mme Johanne Cohen Directrice de recherche, Université Paris-Sud (Unité de recherche)	Président
Mme Laetitia Jourdan Professeur, Université de Lille (Unité de recherche)	Rapporteur
M. Christoph Durr Directeur de recherche, Sorbonne Université (Unité de recherche)	Rapporteur
Benjamin Doerr Professeur, École Polytechnique (Unité de recherche)	Directeur de thèse
Carola Doerr Chargé de recherche, Sorbonne Université (Unité de recherche)	Co-directeur de thèse
Timo Kötzing Senior researcher, HPI Potsdam (Unité de recherche)	Examinateur
Michalis Vazirgiannis Professeur, École Polytechnique (Unité de recherche)	Examinateur

Résumé

De nombreux problèmes d'optimisation du monde réel sont trop complexes pour être résolus en temps polynomial. Un moyen de traiter de tels problèmes utilise « randomized search heuristics » (RSHs), qui produisent des solutions plus efficacement en compromettant l'optimalité, l'exhaustivité, l'exactitude ou la précision. Les RSHs initialisent d'abord un point de recherche avec une position aléatoire dans l'espace de recherche, puis résolvent les problèmes de manière itérative jusqu'à ce qu'un critère de terminaison soit rempli. À chaque itération, ils répètent les étapes suivantes : générer un ou plusieurs candidats ; évaluer la qualité des nouveaux individus ; mettre à jour les informations connues. Puisque les RSHs ne nécessitent aucune information que les valeurs objectives des points de recherche évalués, nous les appelons algorithmes d'optimisation de boîte noire.

Les RSHs sont des algorithmes paramétrés et leurs performances typiquement dépendent de manière cruciale de la valeur de ces paramètres. Un nombre de résultats montrent qu'il peut être avantageux de choisir les paramètres de manière dynamique, afin de permettre aux algorithmes de s'adapter au problème à résoudre et à l'état actuel du processus d'optimisation. Une évaluation rigoureuse des gains de performance pouvant être obtenus en modifiant les paramètres lors de l'exécution fait toutefois largement défaut.

Dans ce travail, nous contribuons à notre connaissance de la manière de contrôler les paramètres en ligne de diverses façons. Nous étudions la fonction OneMax qui est définie via $\{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} x_i$. Dans un premier temps, nous démontrons une borne inférieure pour l'efficacité de tout algorithme unaire non-biaisée de type boîte noire. Plus précisément, nous prouvons que tout algorithme de ce type nécessite $n\ln(n) - cn \pm o(n)$ itérations, en moyenne, pour trouver la solution optimale à ce problème, où c est une constante entre $0,2539$ et $0,2665$. Ce temps d'exécution peut être obtenu avec un (1+1)-type algorithme simple en utilisant un paramètre fitness-dépendant. Le meilleur algorithme non-biaisé statique a besoin de $n\ln(n) - 0.1159n$ itérations (borne optimale de RLS). Pour tout budget fixe d'évaluations de fonctions, notre algorithme génère des solutions 13% meilleures que celles proposées par le meilleur algorithme unaire non-biaisé à paramètres statiques.

Sur la base des paramètres dynamiques optimaux analysés pour OneMax, nous montrons que le paramètre fitness-dépendant peut être remplacé par un paramètre d'auto-ajustement sans perte d'efficacité. Le mécanisme d'ajustement consiste à apprendre de manière adaptative la stratégie actuellement optimale des itérations précédentes. Il utilise donc des évaluations antérieures pour estimer les progrès espérés qui peuvent être atteint par les différentes valeurs des paramètres, et choisit avidement celui qui maximise cette estimation. Seulement avec une faible probabilité $o(1)$, une valeur de paramètre uniformément aléatoire est choisie.

Nous étendons ensuite notre stratégie d'auto-ajustement aux algorithmes d'évolution (EAs) basés sur la population, qui utilisent des mécanismes inspirés par l'évolution biologique, tels que la reproduction, la mutation, la recombinaison et la sélection. Grosso modo, l'idée principale des EAs est de créer par itération λ points de recherche en utilisant deux valeurs de paramètre différentes. Chacun génère la moitié de la population. Le paramètre est ensuite mis à jour en fonction de celui utilisé dans la sous-population qui contient le meilleur point de recherche. Nous analysons comment cette $(1+\lambda)$ EA d'auto-ajustement optimise la fonction de test pour OneMax. Nous

prouvons qu'il trouve l'optimum dans un nombre espéré de $O(n\lambda/\log\lambda + n\log n)$ itérations. Ce temps est asymptotiquement inférieur au temps d'optimisation du classique $(1+\lambda)$ EA avec des paramètres statiques. Des travaux antérieurs montrent que la performance de notre algorithme est la meilleure possible parmi tous les λ-parallèle algorithmes non-biaisés mutation-basés.

Nous proposons et analysons également une version auto-adaptative de $(1,\lambda)$ EA dans laquelle le taux de mutation actuel fait partie de l'individu et est donc également soumis à la mutation. Une analyse rigoureuse du temps d'exécution de la fonction de benchmark OneMax révèle qu'un schéma de mutation locale simple pour le taux conduit à un temps d'optimisation espéré du meilleur possible $O(n\lambda/\log\lambda + n\log n)$. Notre résultat montre que l'auto-adaptation dans le calcul évolutif peut trouver des paramètres optimaux complexes à la volée. Dans le même temps, il prouve qu'un schéma d'auto-ajustement relativement compliqué peut être remplacé par le schéma endogène simple.

Abstract

Many real-world optimization problems are too complex to be solved in polynomial time. One way to deal with such problems is using randomized search heuristics (RSHs), which produce solutions more efficiently by compromising on optimality, completeness, accuracy, or precision. RSHs first initialize a search point with a random position in the search-space and then solve problems iteratively until a termination criterion is met. In each iteration, they repeat the following steps: generate one or more candidates; evaluate the quality of the new individuals; update the known information. Since RSHs do not require any other information than the objective values of the evaluated search points, we call them black-box optimization algorithms.

RSHs are parametrized algorithms, and their performance typically depends very crucially on the value of these parameters. A number of results show that it can be advantageous to chose the parameters dynamically, to allow the algorithms to adjust to the problem at hand and to the current state of the optimization process. A rigorous evaluation of the performance gains that can be obtained by changing the parameters during the execution is, however, largely missing.

In this work, we contribute to our knowledge of how to control the parameters online in various ways. We study the OneMax function which is defined via $\{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} x_i$. In a first step, we prove a lower bound for the efficiency of any unary unbiased black-box algorithm. More precisely, we prove that any such algorithm needs $n\ln(n) - cn \pm o(n)$ iterations, on average, to find an optimal solution for this problem, where c is a constant between 0.2539 and 0.2665. This runtime can be achieved with a simple (1+1)-type algorithm using a fitness-dependent parameter setting. The best static unary unbiased algorithm needs $n\ln(n) - 0.1159n$ iterations (sharp RLS bound). For any fixed budet of function evaluations, our algorithm generates solutions that are better by 13% than those offered by the best unary unbiased algorithm with static parameters.

Based on the analyzed optimal dynamic parameters for OneMax, we show that the fitness-dependent setting can be replaced by a self-adjusting one without losing efficiency. The adjusting mechanism is to adaptively learn the currently optimal strategy from previous iterations. It thus uses past evaluations to estimate the expected progress that can be achieved by the different parameter values, and greedily chooses the one that maximizes this estimation. Only with a small $o(1)$ probably a uniformly random parameter value is chosen.

We then extend our self-adjusting strategy to population-based evolutionary algorithms (EAs) which use mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Roughly speaking, the main idea of the EAs is to create per iteration λ search points using two different parameter values. Each one generates half of the population. The parameter is then updated based on the one used in that subpopulation which contains the best search point. We analyze how this self-adjusting $(1+\lambda)$ EA optimizes the OneMax test function. We prove that it finds the optimum in an expected number of $O(n\lambda/\log\lambda + n\log n)$ iterations. This time is asymptotically smaller than the optimization time of the classic $(1+\lambda)$ EA with static parameters. Previous work shows that the performance of our algorithm is best-possible among all λ-parallel mutation-based unbiased black-box algorithms.

We also propose and analyze a self-adaptive version of the $(1,\lambda)$ EA in which the current mutation rate is part of the individual and thus also subject to mutation. A rigorous runtime analysis on the OneMax benchmark function reveals that a simple

local mutation scheme for the rate leads to an expected optimization time of the best possible $O(n\lambda/\log\lambda + n\log n)$. Our result shows that self-adaptation in evolutionary computation can find complex optimal parameter settings on the fly. At the same time, it proves that a relatively complicated self-adjusting scheme can be replaced by the simple endogenous scheme.

Contents

Résumé **iii**

Abstract **v**

1 Introduction **1**
1.1 Bit-counting Problems . . . 2
1.2 Summary of Contributions . . . 3
1.3 Related Work . . . 6

2 Preliminaries **11**
2.1 Black-box Complexity . . . 11
2.2 Evolutionary Algorithms . . . 12
2.3 Randomized Local Search . . . 13
2.4 Drift Theorems . . . 13
2.5 Chernoff Bounds . . . 17
2.6 Occupation Probabilities . . . 17
2.7 Useful Equations and Inequalities . . . 20

3 Precise Unary Unbiased Black-box Complexity **23**
3.1 Problem Setting and Useful Tools . . . 25
3.2 Maximizing Drift is Near-Optimal . . . 28
3.3 Fitness-Dependent Mutation Strength . . . 37
3.4 Runtime Analysis for the Approximate Drift-Maximizer $\tilde{A}^*_\varepsilon$ . . . 51
3.5 Fixed-Budget Analysis . . . 53
3.6 Self-adjusting Mutation Rate . . . 55
3.7 Mathematical Runtime Analysis on OneMax . . . 56
3.8 Experimental Results . . . 61

4 Self-adjusting (1+λ) EA **63**
4.1 Algorithm . . . 65
4.2 Proof Overview . . . 66
4.3 The Far Region . . . 68
4.4 The Middle Region . . . 76
4.5 The Near Region . . . 78
4.6 Putting Everything Together . . . 81
4.7 Experimental Results . . . 82

5 Self-adapting (1,λ) EA **89**
5.1 Algorithm . . . 91
5.2 The Far Region . . . 91
5.3 The Near Region . . . 104
5.4 Putting Everything Together . . . 110
5.5 Experimental Results . . . 110

6 Conclusions **115**

Bibliography **117**

Chapter 1

Introduction

Many real-world optimization problems are too complex to be solved in polynomial time. One way to deal with such problems is using *heuristic* algorithms which produce solutions more efficiently by compromising on optimality, completeness, accuracy, or precision. The objective of a heuristic is to produce a "good enough" solution in a reasonable runtime. This solution may not be an optimal one, but it is still valuable because finding it does not require a prohibitively long time.

An important class of heuristic algorithms are *randomized search heuristics* (RSHs) which try to optimize a problem in an iterative *trial and error* fashion. Since these algorithms use only the function values of the evaluated solutions, and do not require any additional information (in particular, no gradients, etc.), RSH are also referred to as *black-box methods*. RSH first initialize a *search point* with a random position in the search-space and then solve problems iteratively until a termination criterion is met (e.g. number of iterations performed, or adequate objective value (referred to as *fitness*) reached). In each iteration, RSHs repeat the following steps: generate one or more solution candidates; evaluate the quality of the new individuals; update the known information.

Among the most prominent RSHs are local search variants, simulated annealing, evolutionary and genetic algorithms, and swarm intelligence algorithms. In this work, we focus on randomized local search (RLS) and evolutionary algorithms (EAs). Local search takes a potential solution to a problem and checks its immediate neighbors (that is, solutions that are similar except for very few minor details) in the hope of finding an improved solution. Evolutionary algorithms use mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Search points (*offspring*) play the role of individuals in a population, and evolution takes place at the end of each iteration based on offsprings' quality.

RSHs are parametrized algorithms, and their performance typically depends very crucially on the value of these parameters. An important goal of research on RSHs is to determine optimal parameter choices in these algorithms—a challenging task due to the complex inter-dependencies between different parameters and the random choices of the algorithms. Theory of randomized search heuristics, and in particular the sub-discipline of runtime analysis, aims to contribute to the parameter selection question by providing mathematically founded statements that lay open the influence of the parameter values on the performance.

The majority of these works investigate *static parameter settings*, i. e., the parameters are fixed before the start of the algorithm and are not changed during its execution. More recently, a number of results were shown which prove an advantage of *dynamic parameter settings*, that is, the parameters of the algorithm are changed during its execution. Many of these rely on making the parameters functionally dependent on the current state of the search process, e.g., on the fitness of the current-best individual. While this provably can lead to better performances, it

leaves the algorithm designer with an even greater parameter setting task, namely inventing a suitable functional dependence instead of fixing numerical values for the parameters.

Finding appropriate fitness-dependent parameter values is non-feasible already for quite simple optimization problems, and seems out of range for real-life optimization tasks. Luckily, it turns out that this step is not necessarily needed, as good parameter values can sometimes be found by automated parameter selection mechanisms, which take into account the recent performance. A number of results show that such *on the fly* or *self-adjusting* parameter settings can give an equally good performance as the optimal fitness-dependent parameter setting, however, with much less input from the algorithm designer. Inspiration comes from the continuous optimization, e.g., the 1/5-th rule independently discovered in [Rec73; Dev72; SS68]. In a nutshell, this rule suggests to increase the search radius when more than 1/5-th of all offspring are better than the current-best solution, and to decrease the search radius otherwise. The 1/5-th rule originates from theoretical studies of the optimization of the sphere function $\sum x_i^2$ with a so-called $(1+1)$ Evolution Strategy. This rule has inspired success-based parameter control mechanisms, which update the parameter values depending on whether or not an iteration has been successful in finding a better than the previous-best solution. This rule has found applications in continuous and in discrete optimization.

In this work, we focus on parameter control for discrete search spaces, especially for the OneMax problem, which is one of most frequently analyzed problems in the theory of evolutionary algorithms (cf. Section 1.1 for a definition). To deeply understand the influence of the parameters, we first compute a preise lower bound for the efficiency of any so-called *unary unbiased black-box algorithm* on this problem. Such unary unbiased algorithms are of $(1+1)$ type (that is, they produce and evaluate in each iteration only one new solution candidate by modifying at most one of the previously evaluated ones) and treat both the bit-positions and the bit-values in a symmetric fashion. We prove that any such algorithm needs $n\ln(n) - cn \pm o(n)$ iterations, on average, to find an optimal solution for this problem, where c is a constant between 0.2539 and 0.2665. This runtime can be achieved with a simple fitness-dependent parameter setting. In contrast, the best static unary unbiased algorithm, Randomized Local Search, needs $n\ln(n) - 0.1159n$ iterations [DD16] to optimize OneMax. This smallish looking improvement of roughly $0.14n$ in the $\Theta(n\log n)$ runtime translates into a 13% better fixed-budget performance, in the sense that after a fixed number (budget) of iterations the expected distance of the solution found by our algorithm with dynamic parameter choices to the optimal solution is roughly 13% smaller than the distance produced by RLS, provided that the budget allows for at least $0.2675n$ iterations. After we know what the suitable fitness-dependent mutation strengths look like, we design for three different algorithms strategies to control the parameters and prove, by mathematical means, that they achieve close-to-optimal performance on the OneMax problem.

1.1 Bit-counting Problems

Here we introduce three functions mapping bit strings to integers. These functions were artificially designed to highlight characteristics of the studied evolutionary algorithms (EAs) when tackling optimization problems and to build up runtime analysis.

OneMax The OneMax problem is a classic bit-counting problem. It maps each bit string to the number of ones it contains, i.e., each bitstring $x \in \{0,1\}^n$ of length n is mapped to

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

It has maximal value $\textsc{Om}(1 \dots 1) = n$ and minimal value $\textsc{Om}(0 \dots 0) = 0$.

LeadingOnes The LeadingOnes problem counts the number of one-bits from left to right. It is another classic bit-counting problem. For bitstring $x \in \{0,1\}^n$ of length n, its LeadingOnes value is defined as

$$\textsc{Lo}(x) \coloneqq \max\{i \in [0, \dots, n] \mid \forall j \leq i : x_j = 1\}.$$

It also has maximal value $\textsc{Lo}(1 \dots 1) = n$ and minimal value $\textsc{Lo}(0 \dots 0) = 0$.

Needle The Needle function returns the function value 1 for the all-ones bit string $(1, \dots, 1)$ and function value 0 anywhere else (*needle-in-a-haystack*); i.e.,

$$\textsc{Needle}(x) \coloneqq \mathbb{1}_{x=1\dots1}.$$

This function thus consist of a huge plateau of constant fitness and only one optimal point.

We notice that all three functions have unique optima. Such functions without local optima are called *unimodal*. The OneMax function is smooth and can be easily solved using hill climbing algorithms; the LeadingOnes function is one the simplest *non-separable* functions in discrete optimization, which means that the influence of individual bit values on the overall function value can depend on the other bit values; the Needle function is a typical plateau function where algorithms can basically just do a random walk. These problems are popular in the theoretical works on the analysis of the EAs.

Among these functions, and, more generally, among all benchmark functions analyzed in the theory of randomized search heuristics, the OneMax function is the most popular one since it allows to analyze how the heuristics perform on smooth parts of an optimization problem, where better function values guide the search towards the optimum. It is widely believed that heuristics which solve (much more complex) real-world problems should have no problem to optimize the OneMax problem. In the theory of randomized search heuristics, a good performance on OneMax is therefore often considered to be a minimum requirement that an algorithm needs to pass before it is investigated further. In this work, we focus on the theoretical analysis on the OneMax problem. We also do experimental runs on the LeadingOnes problem in Chapter 3. Both LeadingOnes function and Needle function will be used in the description of related works.

1.2 Summary of Contributions

1.2.1 Drift Maximizer Achieves the Precise Unary Unbiased Black-Box Complexity (Apart From an $o(n)$ Additive Term)

The inherent difficulty of proving runtime guarantees for evolutionary algorithms and other RSHs for a long time prohibited runtime results that are more precise than

giving the asymptotic order of magnitude. It is known that the unary unbiased black-box complexity of OneMax is of order $n \log n$ [LW12]. We note here that black-box complexity is the minimum expected runtime that a black-box optimization algorithm needs to solve a problem; the unary unbiased back-box model in addition requires that all search points are sampled from unary distributions which are symmetric with respect to bit values and bit positions, cf. Section 2.1 for a detailed description. The simple randomized local search heuristic (RLS), which in each iteration applies a random single bit flip on a best-so-far search point, is easily seen to have a runtime of $(1 + o(1))n \ln(n)$ by a reduction to the coupon collector problem. Its precise optimization time is $n \ln(n) + (\gamma - \ln(2)) + o(1) \approx n \ln n - 0.1159n$, where $\gamma = 0.5772\ldots$ is the Euler-Mascheroni constant [DD16]. The previously best known unary unbiased algorithm has an expected runtime that is smaller than that of RLS by an additive $\Theta(\sqrt{n \log n})$ term [LDD15].

In Chapter 3, we bring precise runtime analysis and black-box complexity together. By determining the black-box complexity more precisely than up to the asymptotic order, we shall understand more precisely the problem difficulty, but also learn improved ways to solve the problem. Taking the unary unbiased black-box complexity as first object of investigation, we derive a simple (1+1)-type algorithm that, by using a fitness-dependent mutation strength, has a time complexity larger than the theoretical optimum by only an additive εn term (ε a small constant). We show a precise bound for the unary unbiased black-box complexity of OneMax. More precisely, we show that it is $n \ln(n) - cn \pm o(n)$ for a constant c for which we show $0.2539 < c < 0.2665$. We also show how to numerically compute this constant with arbitrary precision. Equally important, our analysis reveals (and needs) a number of interesting structural results. In particular, in the language of fixed-budget computation as introduced by Jansen and Zarges [JZ14], the drift-maximizing algorithm with a budget of at least $0.2675n$ iterations computes a solution with expected fitness distance to the optimum roughly 13% smaller than the output of the previously best known algorithm. The result have been published at GECCO'16 [DDY16b].

1.2.2 Self-adjusting RLS also Achieves the Same Complexity

Based on the analysis of the near-optimal mutation strength, we have proposed at PPSN'16 [DDY16a] a variant of RLS with self-adjusting mutation strengths which can both avoid the performance loss of standard bit mutation and avoid the risk of getting stuck in local optima. The idea is to let the algorithm learn the optimal mutation rate autonomously by analyzing the past performance of the different mutation strengths. Such an on-the-fly learning are subject to an *exploration-exploitation trade-off* in that they want to exploit those parameter values that are assumed to be optimal at different stages of the optimization process, but also need to allocate some positive probability to select sub-optimal parameter values, in order to be able to detect if some of these have become superior in the meantime. Our algorithm therefore chooses in each iteration with some small probability δ a random parameter and greedily selects the parameter value from which it expects the best progress in fitness otherwise, i.e., with probability $1 - \delta$.

We experimentally analyze our new algorithm on the LeadingOnes and the minimal spinning tree (MST) problem. We observe that, for suitable parameter settings, it clearly outperforms the (1+1) EA. Interestingly, it even beats the randomized local search (RLS) heuristic (flipping always one bit) for the LeadingOnes problem and the variant of RLS flipping one or two bits for the MST problem. This shows that for these problems a better performance can be obtained from a mutation strength

that changes over time, and that our algorithm is able to find such superior fitness-dependent mutation strengths on the fly. We also make this effect mathematically precise for OneMax. We show that our algorithm essentially is able to find the close-to-optimal mutation schedule identified in our above-described work [DDY16b]. More precisely, with high probability our algorithm always (apart from a lower-order fraction of the iterations) uses a mutation strength which gives an expected progress equal to the best possible progress (again, apart from lower order terms). Consequently, it has the same optimization time (apart from an $o(n)$ additive lower order term) and the same asymptotic 13% superiority in the fixed-budget perspective as the drift maximizer proposed in [DDY16b].

1.2.3 Self-adjusting (1+λ) EA Achieves the λ-parallel Mutation-Based Unbiased Black-box Complexity

Inspired by the self-adjusting algorithm proposed in [DDY16a], we notice that some of the difficulties of the learning mechanism, e.g., the whole book-keeping being part of it and also the setting of the parameters regulating how to discount information over time, can be overcome by using a (1+λ) EA, which samples λ independent offspring per iteration. In a sense, the use of larger populations enables us to adjust the mutation rate solely on information learned in the current iteration. However, we do also use the idea of [DDY16a] to intentionally use parameter settings which appear to be slightly off the current optimum to gain additional insight and to be able to detect and to react to a change in the optimal parameter values.

In Chapter 4, we propose a new way to self-adjust the mutation rate in population-based evolutionary algorithms in discrete search spaces. It aims at overcoming some of the difficulties of the learning mechanism just described. It consists of creating half the offspring with a mutation rate that is twice the current mutation rate and the other half with half the current rate. The mutation rate is then updated according to which subpopulation contains the best offspring. Instead of always modifying the mutation rate to the rate of the best offspring, we shall take this winner's rate only with probability a half and else modify the mutation rate to a random one of the two possible values (twice and half the current rate). Our motivation for this modification is that we feel that the additional random noise will not prevent the algorithm from adjusting the mutation rate into a direction that is more profitable. However, the increased amount of randomness may allow the algorithm to leave a possible basin of attraction of a locally optimal mutation rate.

We analyze how the $(1 + \lambda)$ evolutionary algorithm with this self-adjusting mutation rate optimizes the OneMax test function. We prove that this dynamic version of the $(1 + \lambda)$ EA finds the optimum in an expected optimization time (number of fitness evaluations) of $O(n\lambda/\log\lambda + n\log n)$. This time is asymptotically smaller than the optimization time of the classic $(1 + \lambda)$ EA. Previous work in [BLS14] shows that this performance is best-possible among all λ-parallel mutation-based unbiased black-box algorithms.

As an interesting side remark, our proofs reveal that a quite non-standard but fixed mutation rate of $r = \ln(\lambda)/2$ also achieves the $\Theta(\log\log\lambda)$ improvement as it implies the bound of $\Theta(n/\log\lambda)$ generations if λ is not too small. Hence, the constant choice $r = O(1)$ as studied in [GW17] does not yield the asymptotically optimal number of generations unless λ is so small that the $n\log n$-term dominates. These results have been presented at GECCO'17 [Doe+17]. [1]

1. This is a joint work with Christian Gießen and Carsten Witt from Technical University of Denmark. Every author contributes equally to the collaboration.

1.2.4 Extend the Self-adjusting Strategy to Non-elitist Algorithm Self-adaptive (1,λ) EA

In Chapter 5, we propose and analyze a self-adaptive version of the $(1, \lambda)$ evolutionary algorithm in which the current mutation rate is part of the individual and thus also subject to mutation. A rigorous runtime analysis on the OneMax benchmark function reveals that a simple local mutation scheme for the rate leads to an expected optimization time (number of fitness evaluations of $O(n\lambda/\log\lambda + n\log n)$. This time is asymptotically smaller than the optimization time of the classic $(1, \lambda)$ EA and as mentioned above $(1+\lambda)$ EA for all static mutation rates and best possible among all λ-parallel mutation-based unbiased black-box algorithms.

Our result shows that self-adaptation in evolutionary computation can find complex optimal parameter settings on the fly. At the same time, it proves that a relatively complicated self-adjusting scheme for the mutation rate proposed in Chapter 4 can be replaced by our simple endogenous scheme. These results will be presented at GECCO'18 [DWY18]. [2].

1.3 Related Work

Since the parameter setting has a decisive influence on the performance of RSHs, it is not surprising that it is one of the most intensively studied research questions in the domain of heuristic optimization. Most works, however, study the influence of the parameters on performance by empirical means, whereas here in this work we are rather concerned with rigorous mathematical runtime results. We summarize in this section some of the few known results that are precise enough to determine optimal parameter settings. Not surprisingly, the OneMax benchmark problem also considered in this thesis plays a central role in the works.

1.3.1 Static Parameter Choices

A very early work on precise runtime analysis, apparently overlooked by many subsequent works, is the very detailed analysis on how the (1+1) EA with general mutation rate c/n, c a constant, optimizes the Needle and OneMax functions [GKS99]. Disregarding here the results on the runtime distributions, this work shows that the (1+1) EA with mutation rate c/n finds the optimum of the Needle function in $(1+o(1))\frac{1}{1-e^{-c}}2^n$ time.

For OneMax, the runtime estimate in [GKS99] is $(1+o(1))\frac{e^c}{c}n\ln(n)$, more precisely, $\frac{e^c}{c}n\ln(n) \pm O(n)$. The proof of the latter result uses several deep tools from probability theory, among them Laplace transforms and renewal theory. For a simple proof of the upper bound $(1+o(1))en\ln(n)$ for mutation rate $1/n$, see, e.g., [DJW02]. The first proofs of a lower bound of order $(1-o(1))en\ln(n)$ using more elementary methods where given, independently, in [DFW10; Sud13]. An improvement to $en\ln(n) - O(n)$ was presented in [DFW11]. Very recently, again for the case $c = 1$, a very precise analysis of the expected runtime, specifying all lower order terms larger than $\Theta(\log(n)/n)$ with precise leading constant, was given in [Hwa+18].

That the runtime bound of $(1+o(1))\frac{e^c}{c}n\ln(n)$ holds not only for OneMax, but any linear pseudo-Boolean function, was shown in [Wit13]. Since the mutation rate determines the leading constant of the runtime, a rate of $1/n$ gives the asymptotically best runtime. Using single bit flips is also a suitable choice for the classic randomized

2. This is a joint work with Carsten Witt from Technical University of Denmark. Every author contributes equally to the collaboration.

local search with static mutation strengths. By a reduction to the coupon collector problem, we obtain a runtime of $(1+o(1))n\ln(n)$ which matches the current known unary unbiased black-box complexity of order $n\log n$.

An extension of the OneMax result to $(1+\lambda)$ EAs was obtained in [GW15]. The bound of $(1+o(1))(\frac{e^c}{c}n\ln(n)+n\lambda\ln\ln(\lambda)/2\ln(\lambda))$ fitness evaluations contains the surprising result that the mutation rate is important for small offspring population sizes, but has only a lower-order influence once λ is sufficiently large (at least when restricted to mutation rates in the range $\Theta(1/n)$; note that Lemma 4.2 indicates that mutation rates of a larger order of magnitude can give asymptotically smaller runtimes for larger values of λ).

In parallel independent work, the precise expected runtime of the (1+1) EA on the LeadingOnes benchmark function was determined in [BDN10; Sud13] (note that [Sud13] is the journal version of a work that appeared at the same conference as [BDN10]). The work [Sud13] is more general in that it also regards the (1+1) EA with Best-of-μ initialization (instead of just initializing with a random individual), the approach of [BDN10] has the advantage that it also allows to determines the distribution of the runtime (this was first noted in [Doe+13a] and was formally proven in [Doe18a]). The work [BDN10] also shows that the often recommended mutation rate of $p=1/n$ is not optimal. A runtime smaller by 16% can be obtained from taking $p=1.59/n$ and another 12% can be gained by using a fitness-dependent mutation rate.

1.3.2 Dynamic Parameter Choices

While it is clear that EAs with parameters changing during the run of the algorithm (dynamic parameter settings) can be more powerful than those only using static parameter settings, only recently considerable advantages of dynamic choices could be demonstrated by mathematical means (for discrete optimization problems; in continuous optimization, step size adaptation is obviously necessary to approach arbitrarily closely a target point). To describe the different ways to dynamically control parameters and to summarize the most relevant works, we use in the following the taxonomy proposed in [DD18b], which distinguishes between state-dependent, success-based, learning-inspired, and endogenous, parameter choices.

State-Dependent Parameter Control

In the classification of [DD18b], *state-dependent parameter control* subsumes those mechanisms that depend only on the current state of the search process, e.g., the current population, its fitness values, its diversity, but also a time or iteration counter. Hence this subsumes the classic *deterministic* category in [EHM99] and all other parameter setting mechanisms which determine the current parameter values via a pre-defined function that maps algorithm states to parameter values, possibly in a randomized manner. All these mechanisms require the user to precisely specify how the parameter value depends on the current state and as such need a substantial understanding of the problem to be solved.

The first to rigorously analyze this scheme in the context of evolutionary computation are Jansen and Wegener [JW06]. They regard the performance of the (1+1) EA which uses in iteration t the mutation rate k/n, where $k\in\{1,2,\dots,2^{\lceil\log_2 n\rceil-2}\}$ is chosen such that $\log_2(k)\equiv t-1 \pmod{\lceil\log_2 n\rceil-1}$. In other words, they cyclically use the mutation rates $1/n, 2/n, 4/n, \dots, K/n$, where K is the largest power of two that is less than $n/2$. Jansen and Wegener demonstrate that there exists an example

function where this dynamic EA significantly outperforms the (1+1) EA with any static mutation rate. However, they also observe that for many classic problems, this EA is slower by a factor of $\Theta(\log n)$.

After the first the time-dependent parameter scheme, Böttcher, Doerr, and Neumann conducted a runtime analysis for a fitness-dependent parameter in [BDN10]. They proposed to use the mutation rate of $1/(\textsc{Lo}(x)+1)$ for the optimization of the LeadingOnes test function. They proved that with this choice, the runtime of the (1+1) EA improves to roughly $0.68n^2$ compared to a time of $0.86n^2$ stemming from the classic mutation rate $1/n$ or a runtime of $0.77n^2$ stemming from the asymptotically optimal static rate of approximately $1.59/n$.

For the $(1+(\lambda,\lambda))$ genetic algorithm (GA), a fitness-dependent offspring population size of order $\lambda = \Theta(\sqrt{n/d(x)})$ was suggested in [DDE15a], where $d(x)$ is the fitness-distance of the parent individual to the optimum. This choice improves the optimization time (number of fitness evaluations until the optimum is found) on OneMax from $\Theta(n\sqrt{\log(n)\log\log\log(n)/\log\log(n)})$ stemming from the optimal static parameter choice to $O(n)$, as was proven in [DD18a]. Since in this adaptive algorithm the mutation rate p is functionally dependent on the offspring population size, namely via $p=\lambda/n$, the dynamic choice of λ is equivalent to a fitness-dependent mutation rate of $1/\sqrt{nd(x)}$.

In the aforementioned work by Badkobeh et al. [BLS14], a fitness-dependent mutation rate of $\max\left\{\frac{\ln\lambda}{n\ln(en/d(x))}, \frac{1}{n}\right\}$ was shown to improve the classic runtime of $O\left(\frac{n\log\log\lambda}{\log\lambda}+\frac{n\log n}{\lambda}\right)$ to $O\left(\frac{n}{\log\lambda}+\frac{n\log n}{\lambda}\right)$ for OneMax problem. They also proved that it is the asymptotically best-possible runtime all λ-parallel mutation-based unbiased black-box algorithms. We notice here that this complexity is exactly what we aim at in Chapter 4 and Chapter 5 but using self-adjusting and self-adaptative parameter control schemes. We will discuss such schemes below.

Success-Based Parameter Control

As one important type of *self-adjusting* parameter control mechanisms, success-based parameter settings are classified as all those mechanisms which change the parameters from one iteration to the next, based on the outcome of the iteration. This includes in particular multiplicative update rules which change parameters by constant factors depending on whether the iteration was considered a success or not.

The last years have produced a profound understanding of success-based parameter choices. The first to perform a mathematical analysis were [LS11], who considered the (1+λ) EA and a simple parallel island model together with two self-adjusting mechanisms for population size or island number, including halving or doubling it depending on whether the current iteration led to an improvement or not. These mechanisms were proven to give significant improvements of the "parallel" runtime (number of generations) on various test functions without increasing significantly the "sequential" runtime (number of fitness evaluations).

In [DD15] it was shown by empirical means that the fitness-dependent choice of λ for the $(1+(\lambda,\lambda))$ GA can also be found in a self-adjusting way. To this aim, another success-based mechanism was proposed, which imitates the 1/5-th rule from evolution strategies. It was later proven to yield indeed an optimal linear runtime on OneMax [DD18a]. With some modifications, the self-adjusting mechanism also works on random satisfiability problems [BD17]. For the problem of optimizing an r-valued OneMax function, a self-adjustment of the step size inspired by the 1/5-th rule was found to find the asymptotically best possible runtime in [DDK16]. Empirical

results show that similar mechanisms work very well also to control the mutation rate of the (1+1) EA optimizing OneMax and LeadingOnes [DW18].

Our (1+λ) EA in Chapter 4 uses a success-based parameter control mechanism, of a different and novel type. The idea is that the mutation rate is updated according to the rate which generates the best offspring. We add one modification to this basic idea, by taking the winner's rate only with probability a half and else modifying the mutation rate randomly to a rate used in that iteration.

Learning-Inspired Parameter Control

In contrast to the success-based parameter control mechanisms, *learning-inspired* parameter control mechanisms are classified as all those schemes which aim at exploiting more than one iteration.

To allow such learning mechanisms to also adapt quickly to changing environments, older information is taken into account to a lesser extend than more recent ones. This can be achieved by only regarding information from (static or sliding) time windows or by discounting the importance of older information via weights that decrease (usually exponentially) with the anciency of the data.

Most experimental results within the evolutionary computation context borrow tools from machine learning. The upper confidence bound (UCB)-algorithm originally proposed in [PA02] is an example. It plays an important role in machine learning, as it is one of the few strategies that can be proven to behave optimally in a classical operator selection problem.

Our learning-based RLS in Chapter 3 is the only theoretical result so far for a learning-inspired parameter control mechanism in the context of RSHs.

Endogenous Parameter Control (Self-adaptation)

Endogenous parameter control corresponds to the *self-adaptive* parameter control mechanisms in the taxonomy of [EHM99]. Such mechanisms are classified as all those schemes where the parameter is encoded in the genome and thus subject to variation and selection. The understanding of self-adaptation is still very limited. The only theoretical work on this topic [DL16], however, is promising and shows examples where self-adaptation can lead to significant speed-ups for non-elitist evolutionary algorithms.

Our (1,λ) EA in Chapter 5 is the first non-artificial example for the use of self-adaptation in the theory literature. The mutation strength and bit string is combined together in an individual. Therefore, parameter is adapted automatically when selecting the best offspring.

Chapter 2

Preliminaries

In this chapter we formally define the black-box setting and the algorithm analyzed in this work. We present some fundamental probabilities tools for the analysis including drift theorems, Chernoff bounds, and occupation probability.

2.1 Black-box Complexity

The black-box complexity of a problem is the minimum average number of fitness evaluations that a black-box optimization algorithm (that is, an algorithm that has only access to the objective function, but not to an explicit problem description) needs in order to find an optimal solution. This notion was introduced in [DJW06] and has attracted significant attention in the past years. In particular, black-box complexity insight has been used to design better algorithms [DDE15b], and specific black-box models (allowing only a restricted class of black-box algorithms) have been proposed to understand particular properties of evolutionary algorithms and to better reflect how typical evolutionary algorithms look like.

In this work, we build on the unary unbiased black-box complexity [LW12], which is a reasonable model for mutation-based search heuristics. In simple words, a unary unbiased black-box algorithm is allowed (i) to sample random search points and (ii) to generate new search points from applying unbiased mutation operators to previously found search points. We first limit the number of search points that are used to generate the offspring. By looking at unary variation operators, we consider using only one search point to produce new search point. Moreover, unbiased means that the operator is invariant under automorphisms of the hypercube, so in particular, it is not allowed to prefer certain bit-positions or bit-values. Furthermore, all selection operations have to be independent of the bit-string representation of the individual. The algorithms follows the blueprint described in Algorithm 1.

The unary unbiased black-box complexity of OneMax is the smallest expected number of function evaluations that any algorithm following the structure of Algorithm 1 exhibits on this problem. In line 9 of Algorithm 1 a unary unbiased variation operator is asked for. In the context of optimizing pseudo-Boolean functions, a unary operator is an algorithm that is build upon a family $(p(\cdot \mid x))_{x \in \{0,1\}^n}$ of probability distributions over $\{0,1\}^n$. Given some input x it outputs a new string that it samples from the distribution $p(\cdot \mid x)$. A unary operator is unbiased if all members of its underlying family of probability distributions are symmetric with respect to the bit positions $[n] = [1..n]$ where notation $[l..r]$ is defined as the set of integers from l to r inclusively and the bit values 0 and 1 (cf. [LW12] for a discussion).

Algorithm 1 Blueprint of a unary unbiased black-box algorithm

$t \leftarrow 0$.
Choose $x(t)$ uniformly at random from $S = \{0,1\}^n$.
for $t \leftarrow 1, 2, \ldots$ **do**
  Compute $f(x(t-1))$.
  Choose probability distribution p_s on $[0..t-1]$.
  Randomly choose an index i according to p_s.
  Choose a unary unbiased operator $(p(\cdot|x))_{x\in\{0,1\}^n}$.
  Generate $x(t)$ according to $p(\cdot|x(i))$.

2.2 Evolutionary Algorithms

All mutation-based $(\mu+\lambda)$ or (μ,λ) EAs with standard bit mutation or 1-bit flips belong to the class of unbiased black-box algorithms. In this work, we analyze the performance of (1,λ) EA and (1+λ) EA on OneMax problem. In each generation, λ parallel offspring are generated using the same single parent. Below are the structures of these two algorithms.

Algorithm 2 (1+λ) EA with static mutation strength r for minimizing $f : \{0,1\}^n \to \mathbb{R}$

Select x uniformly at random from $\{0,1\}^n$
for $t \leftarrow 1, 2, \ldots$ **do**
  for $i \leftarrow 1, \ldots, \lambda$ **do**
    Create x_i by flipping each bit in a copy of x independently with probability r/n.
  $x^* \leftarrow \arg\min_{x_i} f(x_i)$ (breaking ties randomly).
  if $f(x^*) \leq f(x)$ **then**
    $x \leftarrow x^*$.

Algorithm 3 (1,λ) EA with static mutation strength r for minimizing $f : \{0,1\}^n \to \mathbb{R}$

Select x uniformly at random from $\{0,1\}^n$
for $t \leftarrow 1, 2, \ldots$ **do**
  for $i \leftarrow 1, \ldots, \lambda$ **do**
    Create x_i by flipping each bit in a copy of x independently with probability r/n.
  $x \leftarrow \arg\min_{x_i} f(x_i)$ (breaking ties randomly).

The only difference between the (1+λ) EA and the (1,λ) EA is whether the parent is updated to the best offspring when all offspring are worse than the parent. The (1+λ) EA takes the best search point among λ offspring and the parent itself, whereas the (1,λ) EA simply replaced the parent by the winner of λ offspring. We notice that the (1+λ) EA is an elitist algorithm since the fitness distance is non-increasing. However, the (1,λ) EA is non-elitist and may result in exponential runtime due to insufficient population size. We shall analyze the performance of the above two algorithms on the OneMax in Chapter 4 and Chapter 5 respectively.

2.3 Randomized Local Search

Randomized Local Search is very similar to the simplest evolutionary algorithm $(1+1)$ EA. They both generate one search point in each iteration and do selection between the parent and its single offspring. The only difference is that, the $(1+1)$ EA generates offspring using standard bit mutation whereas RLS generates offspring from the neighborhood. Assuming using mutation strength of r for both, the $(1+1)$ EA flips every bit independently with probability r/n, whereas the RLS chooses r bits uniformly at random from all n bits and flip exactly these bits. Below is the structure of RLS.

Algorithm 4 RLS with static mutation strength r for minimizing $f : \{0,1\}^n \to \mathbb{R}$

Select x uniformly at random from $\{0,1\}^n$
for $t \leftarrow 1, 2, \ldots$ **do**
  Choose a r-subset $S \subset [1..n]$ uniformly at random.
  Create x^* by flipping bits in S in a copy of x.
  if $f(x^*) \leq f(x)$ **then**
    $x \leftarrow x^*$.

Classic RLS applies static mutation rate. By taking $r = 1$, solving the OneMax problem is reduced to a coupon collect problem which has a known complexity $\Theta(n \ln n)$. We prove in Chapter 3 that RLS with fitness dependent mutation rate is the nearly optimal unary unbiased black-box algorithm apart from an additional $o(n)$ term. To analyze the runtime impact caused by different mutation strengths, we introduce drift theorems which relate expected progress to expected runtime.

2.4 Drift Theorems

The main tool in our work is drift analysis, a well-established method in the theory of randomized search heuristics. Drift analysis tries to translate information about the expected progress an algorithm does into information about the expected runtime. Drift analysis was introduced to the field in the seminal work of He and Yao [HY04]. They proved the following additive drift theorem, which assumes a uniform bound on the expected progress.

Theorem 2.1 (Additive drift theorem [HY04])**.** *Let $(X_t)_{t\geq 0}$ be a sequence of non-negative random variables over a finite state space in $\mathbb{R}$. Let T be the random variable that denotes the earliest point in time $t \geq 0$ such that $X_t = 0$. If there exist $c, d > 0$ such that*

$$c \leq \mathrm{E}\left(X_t - X_{t+1} \mid T > t\right) \leq d,$$

then

$$\frac{X_0}{d} \leq \mathrm{E}\left(T \mid X_0\right) \leq \frac{X_0}{c}.$$

A uniform bound on the drift is often not convenient, since the progress of many algorithms slows down the closer one is to the optimum. For this reason, multiplicative drift assuming a progress proportional to the distance to the optimum was proposed in [DJW12]. While it is often a good tool for the analysis of randomized search heuristic on the OneMax test function and related problems, for the very precise bound we aim at in Chapter 3 it does not suffice to approximate the true drift behavior via a linear progress estimate. For this reason, we resort to the most general technique known in the area of drift analysis, which is called variable drift, and which

assumes no particular behavior of the progress. Variable drift was independently developed in [MRC09] and [Joh10]. The latter result reads as follows.

Theorem 2.2 (Johannsen's Theorem [Joh10])**.** *Let $(X_t)_{t\geq 0}$ be a sequence of non-negative random variables over a finite state space in $\mathcal{S} \subset \mathbb{R}$ and let $x_{\min} = \min\{x \in \mathcal{S} : x > 0\}$. Furthermore, let T be the random variable that denotes the earliest point in time $t \geq 0$ such that $X_t = 0$. Suppose that there exists a continuous and monotonically increasing function $h : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ such that for all $t < T$ it holds that*

$$\mathrm{E}\left(X_t - X_{t+1} \mid X_t\right) \geq h(X_t).$$

Then

$$\mathrm{E}\left(T \mid X_0\right) \leq \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X_0} \frac{1}{h(x)} dx.$$

The variable drift theorem is often applied in the special case of *additive drift* in discrete spaces: assuming $E(X_t - X_{t+1} \mid X_t = x; X_t > 0) \geq \varepsilon$ for some constant ε, one obtains $E(T \mid X_0 = x') \leq x'/\varepsilon$.

Since we will make frequent use of it in the following sections as well, we will also give the version of the *Multiplicative Drift Theorem* for upper bounds, due to [DJW12]. Again, this is implied by the previous variable drift theorem.

Theorem 2.3 (Multiplicative drift [DJW12])**.** *Let $(X_t)_{t\geq 0}$ be random variables describing a Markov process over a finite state space $S \subseteq \mathbb{R}_0^+$ and let $x_{\min} := \min\{x \in S \mid x > 0\}$. Let T be the random variable that denotes the earliest point in time $t \geq 0$ such that $X_t = 0$. If there exist $\delta > 0$ such that for all $x \in S$ with $\Pr(X_t = x) > 0$ we have*

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

then for all $x' \in S$ with $\Pr(X_0 = x') > 0$,

$$E(T \mid X_0 = x') \leq \frac{1 + \ln\left(\frac{x'}{x_{\min}}\right)}{\delta} \ .$$

We recall that our search space is discrete. Therefore by redoing the proof of Theorem in [Joh10], we derive a discrete drift theorem, which uses the elementary idea to approximate a step function by continuous functions.

Theorem 2.4 (Discrete variable drift, upper bound)**.** *Let $(X_t)_{t\geq 0}$ be a sequence of random variables in $[0..n]$ and let T be the random variable that denotes the earliest point in time $t \geq 0$ such that $X_t = 0$. Suppose that there exists a monotonically increasing function $h : \{1, \ldots, n\} \to \mathbb{R}_0^+$ such that*

$$\mathrm{E}\left(X_t - X_{t+1} \mid X_t\right) \geq h(X_t)$$

holds for all $t < T$. Then

$$\mathrm{E}\left(T \mid X_0\right) \leq \sum_{i=1}^{X_0} \frac{1}{h(i)}.$$

Proof. For all $0 < \varepsilon < 1$, define a function $h_\varepsilon : [1, n] \to \mathbb{R}^+$ by

$$h_\varepsilon(x) := \begin{cases} h(\lfloor x \rfloor) + \frac{x - \lfloor x \rfloor}{\varepsilon}\Big(h(\lceil x \rceil) - h(\lfloor x \rfloor)\Big) & \text{for } 0 < x - \lfloor x \rfloor \leq \varepsilon, \\ h(\lceil x \rceil) & \text{for } \varepsilon < x - \lfloor x \rfloor, \end{cases} \tag{2.1}$$

The continuous monotone function h_ε satisfies $\mathrm{E}(X_t - X_{t+1} \mid X_t) \geq h_\varepsilon(X_t)$ for all $t < T$. We compute

$$\begin{aligned}\int_1^{X_0} \frac{1}{h_\varepsilon(x)} dx &\leq \sum_{i=2}^{X_0} \frac{\varepsilon}{h(i-1)} + \frac{1-\varepsilon}{h(i)} \\ &= \frac{\varepsilon}{h(1)} - \frac{\varepsilon}{h(X_0)} + \sum_{i=2}^{X_0} \frac{1}{h(i)}.\end{aligned}$$

Hence by Theorem 2.2, we have $\mathrm{E}(T \mid X_0) \leq \frac{\varepsilon}{h(1)} - \frac{\varepsilon}{h(X_0)} + \sum_{i=1}^{X_0} \frac{1}{h(i)}$ for all $\varepsilon > 0$, which proves the claim. □

Inspired by Theorem 2.1, it is intuitive to think that there is a lower bound variable drift theorem which has similar structure as the upper bound variable drift theorem. Theorem 2.5 in [DFW11] shows how lower bounds on the expected first hitting time can be derived from upper bounds on the drift. As a main difference to the upper bound case, it is now required that the process with probability one in each round does not move too far towards the target (first condition of the theorem below).

Theorem 2.5 (Variable drift, lower bound [DFW11])**.** *Let $(X_t)_{t\geq 0}$ be a decreasing sequence of non-negative random variables over a finite state space in $\mathcal{S} \subset \mathbb{R}$ and let $x_{\min} = \min\{x \in \mathcal{S} : x > 0\}$. Furthermore, let T be the random variable that denotes the earliest point in time $t \geq 0$ such that $X_t = 0$. Suppose that there exists two continuous and monotonically increasing function $c, h : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ such that*

1. *$X_{t+1} \geq c(X_t)$,*
2. *$\mathrm{E}(X_t - X_{t+1} \mid X_t) \leq h(c(X_t))$.*

Then

$$\mathrm{E}(T \mid X_0) \geq \frac{x_{\min}}{h(x_{\min})} + \int_{x_{\min}}^{X_0} \frac{1}{h(x)} dx.$$

The first condition of the theorem above, that with probability one no progress beyond a given limit is made, is a substantial restriction to the applicability of the theorem, in particular, when working with evolutionary algorithms, where often any parent can give birth to any offspring (though usually with very small probability). In [DFW11], this problem was overcome with the simple argument that a progress of more than $\sqrt{x}$ from a search point with fitness distance x occurs with such a small probability that it does with high probability not occur in a run of typical length.

For the precise bounds that we aim at, we show a different version of the lower bound variable drift theorem, which allows larger jumps, if only they occur sufficiently rarely. The way to such a theorem is, in fact, quite easy. Instead of using a blunt union bound over all bad events of too large jumps (as sketched above), we take these large jumps into account when computing the (additive) drift in a proof analogous to the one of [DFW11]. This idea was already used in [GW16], where an essentially similar, but slightly more technical drift theorem for random processes in arbitrary domains is derived. Note here that in contrast to [GW16] we restrict ourselves to processes over the non-negative integers.

Theorem 2.6 (Discrete variable drift, lower bound)**.** *Let $(X_t)_{t\geq 0}$ be a sequence of decreasing random variables in $[0..n]$ and let T be the random variable that denotes the earliest point in time $t \geq 0$ such that $X_t = 0$. Suppose that there exists two monotonically increasing functions $c : [n] \to [0..n]$ and $h : [0..n] \to \mathbb{R}_0^+$, and a constant $0 \leq p < 1$ such that*

1. $X_{t+1} \geq c(X_t)$ *with probability at least* $1-p$ *for all* $t < T$,
2. $\mathrm{E}(X_t - X_{t+1} \mid X_t) \leq h(c(X_t))$ *holds for all* $t < T$.

Let $g : [0..n] \to \mathbb{R}_0^+$ *be the function defined by* $g(x) = \sum_{i=0}^{x-1} \frac{1}{h(i)}$. *Then*

$$\mathrm{E}(T \mid X_0) \geq g(X_0) - \frac{g^2(X_0)p}{1+g(X_0)p}.$$

Proof. The function g is strictly monotonically increasing. We have $g(X_t) = 0$ if and only if $X_t = 0$. Using condition 1 and the monotonicity of h, in the case $X_{t+1} \geq c(X_t)$ we have

$$g(X_t) - g(X_{t+1}) = \sum_{i=X_{t+1}}^{X_t-1} \frac{1}{h(i)} \leq \frac{X_t - X_{t+1}}{h(X_{t+1})} \leq \frac{X_t - X_{t+1}}{h(c(X_t))},$$

and otherwise

$$g(X_t) - g(X_{t+1}) \leq g(X_t) \leq g(X_0).$$

Using inequality $\mathrm{E}(X_t - X_{t+1} \mid X_t) \leq \mathrm{E}(X_t - X_{t+1} \mid X_{t+1} \geq c(X_t))$ and condition 2, we have

$$\begin{aligned} \mathrm{E}\left(g(X_t) - g(X_{t+1}) \mid g(X_t)\right) &\leq \mathrm{E}\left(\frac{X_t - X_{t+1}}{h(c(X_t))} \mid X_{t+1} \geq c(X_t)\right)(1-p) + g(X_0)p \\ &\leq 1 + g(X_0)p. \end{aligned}$$

Applying the additive drift theorem 2.1 to $g(X_t)_{t\geq 0}$, we obtain

$$\mathrm{E}\left(T \mid X_0\right) = \mathrm{E}\left(T \mid g(X_0)\right) \geq \frac{g(X_0)}{1+g(X_0)p} = g(X_0) - \frac{g^2(X_0)p}{1+g(X_0)p}.$$

□

To apply the drift theorem above (or Theorem 2.5) one needs to guess a suitable function h such that $h \circ c$ is an upper bound for the drift. The following simple reformulation overcomes this difficulty by making $h(x)$ simply an upper bound for the drift from a state x. This also makes the result easier to interpret. The influence of large jumps, as quantified by c, now is that the runtime bound is not anymore the sum of all $h(x)^{-1}$ as in the upper bound theorem, but of all $h(\mu(x))^{-1}$, where $\mu(x)$ is the largest point y such that $c(y) \leq x$. So in simple words, we have to replace the drift at x pessimistically by the largest drift among the points from which we can go to x (or further) in one round with probability more than p.

Theorem 2.7 (Discrete variable drift, lower bound)**.** *Let* $(X_t)_{t\geq 0}$ *be a sequence of decreasing random variables in* $[0..n]$ *and let* T *be the random variable that denotes the earliest point in time* $t \geq 0$ *such that* $X_t = 0$. *Suppose that there exists two functions* $c : \{1, \ldots, n\} \to [0..n]$ *and monotonically increasing* $h : [0..n] \to \mathbb{R}_0^+$, *and a constant* $0 \leq p < 1$ *such that*

1. $X_{t+1} \geq c(X_t)$ *with probability at least* $1-p$ *for all* $t < T$,
2. $\mathrm{E}(X_t - X_{t+1} \mid X_t) \leq h(X_t)$ *holds for all* $t < T$.

Let $\mu : [0..n] \to [0..n]$ *be the function defined by* $\mu(x) = \max\{i \mid c(i) \leq x\}$ *and* $g : [0..n] \to \mathbb{R}_0^+$ *be the function defined by* $g(x) = \sum_{i=0}^{x-1} \frac{1}{h(\mu(i))}$. *Then*

$$\mathrm{E}(T \mid X_0) \geq g(X_0) - \frac{g^2(X_0)p}{1+g(X_0)p}.$$

Proof. Let $\hat{c} : [0..n] \to [0..n], r \mapsto \hat{c}(r) := \min\{i|\mu(i) \geq r\}$. By definition, $\hat{c}$ is monotonically increasing. Let $r \in [n]$. Since $r \in \{i|c(i) \leq c(r)\}$, we have $\mu(c(r)) \geq r$, which implies $c(r) \in \{i|\mu(i) \geq r\}$. Hence $\hat{c}(r) \leq c(r)$. This shows that we have $X_{t+1} \geq \hat{c}(X_t)$ for all $t < T$.

The definition of $\hat{c}$ implies that $\mu(\hat{c}(r)) \geq r$. Together with the monotonicity of h, we obtain $\mathrm{E}(X_t - X_{t+1} \mid X_t) \leq h(X_t) \leq h(\mu(\hat{c}(X_t)))$. Let $\hat{h} : [0..n] \to \mathbb{R}_0^+, r \mapsto \hat{h}(r) := h(\mu(r))$. It is easy to see from the definition that μ is monotonically increasing, therefore, $\hat{h}$ is also monotonically increasing and it satisfies $\mathrm{E}(X_t - X_{t+1} \mid X_t) \leq \hat{h}(\hat{c}(X_t))$. Applying Theorem 2.6 to $\hat{h}$ and $\hat{c}$ shows the claim. □

2.5 Chernoff Bounds

In probability theory, the Chernoff bound gives exponentially decreasing bounds on tail distributions of sums of independent random variables. It is a sharper bound than the known first or second moment based tail bounds such as Markov's inequality or Chebyshev inequality, which only yield power-law bounds on the tail decay. We notice that the Chernoff bound requires that the variates be independent. However, the Chernoff bound also holds when the variates are dependent, but the sums follows a hypergeometric distribution.

For reasons of self-containedness, we state the well-known multiplicative Chernoff bounds, additive Chernoff bounds, and a lesser known additive Chernoff bound that is also known in the literature as Bernstein's inequality. These bounds can be found, e.g., in [Doe11].

Theorem 2.8 (Bernstein's inequality, Chernoff bounds)**.** *Let $X_1, \ldots, X_n$ be independent random variables and $X = \sum_{i=1}^n X_i$.*

1. *Let b be such that $E(X_i) - b \leq X_i \leq E(X_i) + b$ for all $i = 1, \ldots, n$. Let $\sigma^2 = \sum_{i=1}^n \mathrm{Var}(X_i) = \mathrm{Var}[X]$. Then for all $\Delta \geq 0$,*

$$\Pr(X \geq E(X) + \Delta) \leq \exp\left(-\frac{\Delta^2}{2(\sigma^2 + \frac{1}{3}b\Delta)}\right).$$

2. *Assume that for all $i = 1, \ldots, n$, the random variable X_i takes values in $[0, 1]$ only. Then*
   (a) $\Pr(X \leq (1-\delta)E(X)) \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{E(X)} \leq \exp\left(\frac{\delta^2 E(X)}{-2}\right)$ *for all* $0 \leq \delta \leq 1$,
   (b) $\Pr(X \geq (1+\delta)E(X)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{E(X)} \leq \exp\left(\frac{-\delta^2 E(X)}{2+\delta}\right)$ *for all* $\delta \geq 0$,
   (c) $\Pr(X \leq E(X) - \lambda) \leq \exp\left(-2\lambda^2/n\right)$ *for all* $\lambda > 0$,
   (d) $\Pr(X \geq E(X) + \lambda) \leq \exp\left(-2\lambda^2/n\right)$ *for all* $\lambda > 0$.
3. *All bounds in item 2 are valid also if X is a random variable with hypergeometric distribution.*

2.6 Occupation Probabilities

In Chapter 4 and Chapter 5, we design a scheme for evolutionary algorithms to adjust the mutation strength automatically based on the performance of all λ offspring. Thus, we will be analyzing two depending stochastic processes: the random decrease of fitness distance and the random change of the mutation rate. Often, we will prove by drift analysis that the rate is drifting towards values that yield an

almost-optimal distance decrease. However, once the rate has drifted towards such values, we would also like the rates to stay in the vicinity of these values in subsequent steps. To this end, we state two occupation theorems here. One is Theorem 7 from [KLW15], which assumes that the process in each step moves at most some constant distance. The other is Theorem 2.3 in [Haj82] which is a stronger because it considers the strong drift toward the target.

Theorem 2.9 (Theorem 7 in [KLW15])**.** *Let a Markov process $(X_t)_{t\geq 0}$ on $\mathbb{R}_0^+$, where $|X_t - X_{t+1}| \leq c$, with additive drift of at least d towards 0 be given (i. e., $E(X_t - X_{t+1} \mid X_t; X_t > 0) \geq d$), starting at 0 (i.e. $X_0 = 0$). Then we have, for all $t \in \mathbb{N}$ and $b \in \mathbb{R}_0^+$,*

$$\Pr(X_t \geq b) \leq 2e^{\frac{2d}{3c}(1-b/c)}.$$

Theorem 2.10 (Theorem 2.3 in [Haj82])**.** *Suppose that $(\mathcal{F}_k)_{k\geq 0}$ is an increasing family of sub-σ-fields of $\mathcal{F}$ and $(Y_k)_{k\geq 0}$ is adapted to $(\mathcal{F}_k)$. If*

$$E\left((e^{\eta(Y_{k+1}-Y_k)}; Y_k > a \mid \mathcal{F}_k\right) \leq \rho \text{ and } E\left(e^{\eta(Y_{k+1}-1)}; Y_k \leq a \mid \mathcal{F}_k\right) \leq D,$$

then

$$\Pr\left(Y_k \geq b \mid \mathcal{F}_0\right) \leq \rho^k e^{\eta(Y_0-b)} + \frac{1-\rho^k}{1-\rho} D e^{\eta(a-b)}.$$

We rewrite the above two theorems into the following lemmas respectively such that they can be directly applied.

Lemma 2.11. *If there is a point $a \geq 4$ such that $\Pr(r_{t+1} < r_t \mid r_t > a) \geq 1/2 + \varepsilon$ for some constant $\varepsilon > 0$, then for all $t' \geq \min\{t \mid r_t \leq a\}$ and all $b \geq 2$ it holds $\Pr(r_{t'} \geq a \cdot 2^{b+1}) \leq 2e^{-2b\varepsilon/3}$.*

Proof. Apply Lemma 2.9 to the process $X_t := \max\{0, \lfloor \log_2(r_t/a) \rfloor\}$. Note that this process is on $\mathbb{N}_0$, moves by an absolute value of at most 1 and has drift $E(X_t - X_{t+1} \mid X_t; X_t > 0) \geq 2\varepsilon$. We use $c := 1$ and $d := 2\varepsilon$ in the theorem and estimate $1 - b \leq -b/2$. □

Lemma 2.12. *Consider a stochastic process X_t, $t \geq 0$, on $\mathbb{R}$ such that for some $p \leq 1/25$ the transition probabilities for all $t \geq 0$ satisfy $\Pr(X_{t+1} \geq X_t + a \mid X_t > 1) \leq p^{a+1}$ for all $a \geq -1/2$ as well as $\Pr(X_{t+1} \geq a + 1 \mid X_t \leq 1) \leq p^{a+1}$ for all $a \geq 0$. If $X_0 \leq 1$ then for all $t \geq 1$ and $k > 1$ it holds that*

$$\Pr(X_t \geq 1 + k) \leq 11\,(ep)^k.$$

Proof. We aim at applying Theorem 2.10. There are two cases depending on X_t: for $X_t \leq 1$, using the monotonicity of $e^{\lambda(X_{t+1}-1)}$ with respect to $X_{t+1} - 1$, we obtain

$$\begin{aligned} D(p,\lambda) &:= \mathrm{E}(e^{\lambda(X_{t+1}-1)} \mid X_t \leq 1) \leq \mathrm{E}(e^{\lambda \max\{\lceil X_{t+1}-1\rceil, 0\}} \mid X_t \leq 1) \\ &= e^0 \Pr(X_{t+1} \leq 1 \mid X_t \leq 1) + \sum_{a=1}^{\infty} e^{\lambda a} \Pr(a < X_{t+1} \leq a+1 \mid X_t \leq 1) \\ &\leq e^0 + \sum_{a=1}^{\infty} e^{\lambda a} \Pr(X_t > a \mid X_t \leq 1), \end{aligned}$$

using the assumption that $\Pr(X_{t+1} \geq a + 1 \mid X_t \leq 1) \leq p^{a+1}$ for all $a \geq 0$ then

$$D(p,\lambda) \leq 1 + \sum_{a=1}^{\infty} e^{\lambda a} p^a = 1 + \frac{e^\lambda p}{1 - e^\lambda p};$$

and for $X_t > 1$, using the monotonicity of $e^{\lambda(X_{t+1}-X_t)}$ respect to $X_{t+1} - X_t$, we have

$$\rho(p,\lambda) \coloneqq \mathrm{E}(e^{\lambda(X_{t+1}-X_t)} \mid X_t > 1) \le \mathrm{E}(e^{\lambda \max\{\lceil 2(X_{t+1}-X_t)\rceil/2,-1/2\}} \mid X_t > 1)$$
$$= e^{-\lambda/2} \Pr\left(X_{t+1} - X_t \le -\frac{1}{2} \mid X_t > 1\right) + \sum_{a=0}^{\infty} e^{\lambda a/2} \Pr\left(\frac{a-1}{2} < X_{t+1} - X_t \le \frac{a}{2} \mid X_t > 1\right)$$
$$\le e^{-\lambda/2} + \sum_{a=0}^{\infty} e^{\lambda a/2} \Pr\left(X_{t+1} - X_t > \frac{a-1}{2} \mid X_t > 1\right),$$

using the assumption that $\Pr(X_{t+1} \ge X_t + a \mid X_t > 1) \le p^{a+1}$ for all $a \ge -1/2$ then

$$\rho(p,\lambda) \coloneqq e^{-\lambda/2} + \sum_{a=0}^{\infty} e^{\lambda a/2} p^{(a+1)/2} = \frac{p^{1/2}}{(e^\lambda p)^{1/2}} + \frac{p^{1/2}}{1-(e^\lambda p)^{1/2}}.$$

Using $\lambda \coloneqq \ln(1/(ep))$ such that $e^\lambda p = 1/e$, we have

$$\rho \coloneqq \rho(p,\lambda) \le e^{1/2} p^{1/2} + \frac{p^{1/2}}{1-e^{-1/2}} \le \frac{e^{1/2}}{5} + \frac{1/5}{1-e^{-1/2}} < 0.84,$$

$$D \coloneqq D(p,\lambda) \le 1 + (1/e)/(1-1/e) < 1.6.$$

Theorem 2.3, inequality (2.8) in [Haj82] yields with $a \coloneqq 1$ and $b \coloneqq 1 + k$ that

$$\Pr(X_t \ge 1 + k \mid X_0) \le \rho^t e^{-\lambda(1+k-X_0)} + \frac{1}{1-\rho} D e^{-\lambda k}$$
$$\le (ep)^k + \frac{1.6}{1-0.84}(ep)^k = 11(ep)^k.$$

□

For the simpler case of a random process that runs on the positive integers and that has a strong drift to the left, we have the following estimate for the occupation probabilities.

Lemma 2.13. *Consider a random process defined on the positive integers $1, 2, \ldots$. Assume that from each state i different from 1, only the two neighboring states $i-1$ and $i+1$ can be reached (and there is no self-loop on state i). From state 1, only state 2 can be reached and the process can stay on state 1. Let p_i be an upper bound for the transition probability from state i to state $i+1$ (valid in each iteration regardless of the past). Assume that*

$$p_{i-1} \ge \frac{p_i}{1-p_i}$$

holds for all $i \ge 2$. Assume that the process starts in state 1. Then at all times, the probability to be in state i is at most

$$q_i \coloneqq \prod_{j=1}^{i-1} \frac{p_j}{1-p_j},$$

where as usual we read the empty product as $q_1 = 1$.

Proof. The claimed bound on the occupation probabilities is clearly true at the start of the process. Assume that it is true at some time. By this assumption and the assumptions on the process, the probability to be in state $i \ge 2$ after one step is at

most

$$\begin{aligned} q_{i-1}p_i + q_{i+1} &= q_{i-1}\left(p_i + \frac{p_{i-1}}{1-p_{i-1}}\frac{p_i}{1-p_i}\right) \\ &\le q_{i-1}\left(\frac{p_i}{1-p_i} + \frac{p_{i-1}}{1-p_{i-1}}\frac{p_i}{1-p_i}\right) \\ &= q_{i-1}\left(\frac{p_i(1-p_{i-1})}{(1-p_{i-1})(1-p_i)} + \frac{p_{i-1}}{1-p_{i-1}}\frac{p_i}{1-p_i}\right) \\ &\le q_{i-1}\frac{p_i}{(1-p_{i-1})(1-p_i)} \le q_{i-1}\frac{p_{i-1}}{1-p_{i-1}} = q_i. \end{aligned}$$

Trivially, the probability to be in state 1 after one step is at most $q_1 = 1$. Hence, by induction over time, we see that q_i is an upper bound for the probability to be in state i at all times. □

2.7 Useful Equations and Inequalities

In this section, we list frequently used equations and estimations. Lemma 2.14 provides some useful inequalities involving logarithmic function and Lemma 2.15 is about binomial coefficient.

Lemma 2.14. *1. For all $x > 0$, we have $\ln x \le x/e$.*

2. For all $x \in \mathbb{R}$, we have $1 + x \le e^x$.

3. For all $0 \le x \le 2/3$, we have $1 - x \ge e^{-x-x^2}$.

4. For all $0 \le x \le 1/2$, we have $1 - x \ge e^{-3x/2}$.

5. Weierstrass product inequality: For all $p_1, \ldots, p_n \in [0, 1]$,

$$1 - \sum_{i=1}^{n} p_i \le \prod_{i=1}^{n}(1 - p_i).$$

Proof. Since $(\ln x - x/e)' = 1/x - 1/e, x > 0$, then $(e, 0)$ is the maximum point of $\ln x - x/e, x > 0$. Thus part 1 holds.

We notice that $(1 + x - e^x)' = 1 - e^x$, then $(0, 0)$ is the maximal point of $1 + x - e^x$. Therefore part 2 holds.

For the proof of part 3, let $f(x) = 1 - x - e^{-x-x^2}$ for $0 \le x \le 2/3$. Then $f'(x) = (1 + 2x)e^{-x-x^2} - 1$ and $f''(x) = (1 - 4x - 4x^2)e^{-x-x^2}$. We notice that $f''(x)$ has only one zero point between 0 and $2/3$. Let x_0 denote the zero point. Thus $f''(x)$ is positive before x_0, and then becomes negative after x_0. Using the fact $f'(0) = 0$ and $f'(2/3) < 0$, we can easily see that $f'(x)$ first increases from 0 to $f'(x_0) > 0$, and then decreases below 0. This means the minimum of $f(x)$ is attained at 0 or $2/3$. Due to the fact that $f(0) = 0$ and $f(2/3) > 0$, the first inequality in part 3 holds.

For the proof of part 4, We notice that $(1 - x - e^{-3x/2})' = -1 + (3/2)e^{-3x/2}$. It is positive for $0 < x < (2/3)\ln(3/2)$ and negative for $(2/3)\ln(3/2) < x \le 1/2$. Since $1 - 0 = e^0$ and $1 - 1/2 > e^{-3/4}$, the statement holds.

Using the fact that $p_n \ge (\prod_{i=1}^{n-1}(1 - p_i))p_n$, part 5 is easily derived by doing induction over n. □

Lemma 2.15. *Consider $m, n \in \mathbb{N}_0$ and $m \le n$. We abbreviate $(n)_m := n(n - 1)\ldots(n - m + 1) = \prod_{i=0}^{m-1} n - i$.*

1. $\binom{n}{m} = \frac{n!}{m!(n-m)!} = \frac{(n)_m}{m!} \ge \left(\frac{n}{m}\right)^m$ and $\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$.

2. $(n-m+1)^m \leq (n)_m \leq n^m$.
3. $\binom{n}{m} = \binom{n}{n-m}$, $\binom{n}{m}m = n\binom{n-1}{m-1}$, *and* $\binom{n}{m}(n-i) = n\binom{n-1}{m}$.

In our analysis, we deal with binomial distributions frequently. Here we list some results from [Doe18b, Lemma 10.20] and [DD18a, Lemma 1].

Lemma 2.16. *Let* $n \in \mathbb{N}, p \in [0,1]$, *and* $\mu = \lceil np \rceil$. *Let* $X \sim \mathrm{Bin}(n,p)$.

1. *if* $1/n < p < 1$, *then* $\Pr(X > E(X)+1) > 1/2 - \sqrt{\frac{n}{2\pi\mu(n-\mu)}}$.
2. *if* $1/n < p < 1 - 1/n$, *then* $\Pr(X > E(X)+1) \geq 0.037$.
3. *if* $0.29/n < p$, *then* $\Pr(X > E(X)) \geq 1/4$.
4. *if* $k \in [0..n]$, *then* $E(X \mid X \geq k) \leq E(X) + k$.

Chapter 3

Precise Unary Unbiased Black-box Complexity

With the goal of starting a complexity theory targeting precise results, we analyze the unary unbiased black-box complexity of the OneMax benchmark function. It is known that the unary unbiased black-box complexity of OneMax is of order $\Theta(n \log n)$ [LW12]. The simple randomized local search heuristic, a hill-climber flipping single random bits, is easily seen to have a runtime of at most $n \ln(n) + \gamma n + \frac{1}{2} \approx n \ln(n) + 0.5772n$ where $\gamma = 0.5772\ldots$ is the Euler-Mascheroni constant.

In this chapter, we show that the unary unbiased black-box complexity of OneMax is

$$n \ln(n) - cn \pm o(n)$$

for a constant c for which we show $0.2539 < c < 0.2665$. We propose a fitness-dependent randomized local search which achieves this complexity. We also show that such state-dependent parameter control can be replaced by a learning-inspired parameter control.

The proof of our results is in the following structures.

Maximizing Drift is Near-Optimal

Evolutionary algorithms build on the idea that iteratively maximizing the fitness is a good approach. This suggests to try to generate the offspring in a way that the expected fitness gain over the best-so-far solution is maximized. Clearly, this is not a successful idea for each and every problem, as easily demonstrated by examples like the DISTANCE and the TRAP functions [DJW02], where the fitness leads the algorithm into a local optimum, or the difficult-to-optimize monotonic functions constructed in [Doe+13c; Len18], where the fitness leads to the optimum, but via a prohibitively long trajectory. Still, one might hope that for problems with a good fitness-distance correlation (and Om has the perfect fitness-distance correlation), maximizing the expected fitness gain is a good approach, and this is roughly what we are able to show.

More precisely, we cannot show that maximizing the expected fitness gain leads to the optimal unary unbiased black-box algorithm. It turns out this is also not true, even if we restrict ourselves to elitist algorithms, which cannot use tricks like minimizing the fitness and inverting the search point once the all-zero string was found. In fact, the elitist algorithm flipping in each iteration the number of bits that minimizes the expected remaining runtime is different from the drift-maximizing one. For all realistic problem sizes, however, the differences between the expected runtime of our drift maximizer and that of the optimal elitist algorithm are negligibly small [BD18].

What we can prove, however, is that the algorithm which in each iteration takes the best-so-far solution and applies to it the unary unbiased mutation operator maximizing the expected fitness gain has an expected optimization time which exceeds the unary unbiased black-box complexity by at most a term of order $O(n^{2/3}\log^9 n)$.

We note that this result, while natural, is quite difficult to obtain and relies on a number of properties particular to this process, in particular, the fact that we have a good structural understanding of the maximal drift.

Maximizing the Drift via the Right Fitness-Dependent Mutation Strength

Once we decided how to choose the parent individual, in principle it is easy to mutate it in such a way that the drift is maximized. Since any unary unbiased mutation operator can be seen as a convex combination of r-bit flip operators, $r \in [0..n]$, and since the drift stemming from such an operator is the corresponding convex combination of the drifts stemming from the r-bit flip operators, we only need to determine, depending on the fitness of the parent, a value for r such that flipping r random bits maximizes the fitness gain over the parent. For a concrete value of n and a concrete fitness of the parent, one can compute the best value of r in time $O(n^2)$.

We need some more mathematical arguments to (i) obtain a structural understanding of this r-value and (ii) to obtain a runtime estimate valid for all values of n. To this aim, we shall first argue that when the parent fitness distance d is at most $(\frac{1}{2}-\varepsilon)n$ for an arbitrarily small constant ε, the optimal drift is obtained from flipping a constant number r of bits (Lemma 3.14). For any constant r, the drift obtained from flipping r bits can be well approximated by a degree r polynomial in the relative fitness distance d/n (Theorem 3.16). By this, we overcome the dependence on n, that is, apart from this small approximation error we can, by regarding these polynomials, determine a function $\tilde{R}_{\text{opt}} : [0, \frac{1}{2}-\varepsilon] \to \mathbb{N}$ such that for all $n \in \mathbb{N}$ and all $d \in [0..(\frac{1}{2}-\varepsilon)n]$ the near-optimal number of bits to flip (that is, optimal apart from the approximation error) is $\tilde{R}_{\text{opt}}(d/n)$. This $\tilde{R}_{\text{opt}}$ is decreasing, that is, the closer we are to the optimum, the smaller is the optimal number of bits to flip. Interestingly, $\tilde{R}_{\text{opt}}$ is never even, so the optimal number of bits to flip is always odd. These (and some more) properties allow to compute numerically the interval in which flipping r bits is optimal (for all odd r). From these we estimate the runtime by numerically approximating the integral describing the runtime via the resulting drifts. This gives approximations for both the runtime of our drift-maximizer and the unary unbiased black-box complexity, which are precise apart from an arbitrary small $O(n)$ term.

We note that previous works have studied drift maximizing variants of RLS and the (1+1) EA by empirical means. For $n = 100$ Bäck [Bäc92] computed the drift-maximizing mutation rates for different $(1+\lambda)$ and $(1,\lambda)$-EAs. Fialho and co-authors considered $(1+\lambda)$-type RLS-variants which choose between either flippling exactly 1, 3, or 5 bits or applying standard bit mutation with mutation rate $p = 1/n$. A simple empirical Monte Carlo evaluation is conducted to estimate the average progress obtained by any of these four operators. [Fia+08] studies the $n = 1,000$-dimensional OneMax function, and [Fia+09] the case $n = 10,000$. None of the three mentioned works, however, further investigates the difference between the drift maximizing algorithm and the optimal (i.e., time-minimizing) one.

Fixed-Budget Result

Computing the runtime of our drift-maximizing algorithm, we observe that the fitness-dependent mutation strength gives a smallish-looking improvement of roughly

$0.14n$ in the $\Theta(n \log n)$ runtime. However, if we view our result in the fixed-budget perspective [JZ14], then (after using the Azuma inequality in the martingale version to show sufficient concentration) we see that if we take the expected solution quality after a fixed number (budget) of iterations as performance measure, then our algorithm gives a roughly 13% smaller fitness distance to the optimum compared to the previous-best algorithm (provided that the budget is at least $0.2675n$).

RLS with learning-inspired parameter control

The example of solving the OneMax problem shows that using a problem-specific optimal mutation strength can lead to a fair speed-up over standard bit mutation, however, with the risk of making the algorithm fail badly when choosing a wrong mutation strength. For this reason, we design a simple hill climber that autonomously tries to choose the optimal mutation rate by analyzing the past performance of the different mutation strengths. This aims both at exploiting that different problems ask for different mutation strengths and at exploiting that for a fixed problem the optimal mutation strength may change during the optimization process; a problem even less understood than what is the right problem-specific static mutation strength.

The heart of this work is making this effect mathematically precise for the OneMax function. For our new algorithm with self-adjusting mutation strength, we show that it essentially is able to find this optimal mutation schedule on the fly. More precisely, with high probability our algorithm always (apart from a lower-order fraction of the iterations) uses a mutation strength which gives an expected progress equal to the best possible progress (again, apart from lower order terms). Consequently, our algorithm has the same optimization time (apart from an $o(n)$ additive lower order term) and the same asymptotic 13% superiority in the fixed budget perspective as the fitness- dependent algorithm.

We also experimentally analyze our new algorithm on the LeadingOnes and the MST problem. We observe that for suitable parameter settings, it clearly outperforms the (1+1) EA. Interestingly, it even beats the randomized local search (RLS) heuristic (flipping always one bit) for the LeadingOnes problem and the variant of RLS flipping one or two bits for the MST problem. This shows that for these problems a better performance can be obtained from a mutation strength that changes over time, and that our algorithm is able to find such superior fitness-dependent mutation strengths.

3.1 Problem Setting and Useful Tools

In this section we briefly describe the black-box setting regarded in this work, the unary unbiased model proposed by Lehre and Witt [LW12]. The variation operators that are admissible in this model are characterized in Lemma 3.1.

The main goal of our work is to determine a precise bound for the unary unbiased black-box complexity of OneMax, the problem of maximizing the function $\textsc{Om} : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} x_i$, which assigns to each bit string the number of ones in it. That is, we aim at identifying a best-possible mutation-based algorithm for this problem.

The unary unbiased black-box complexity of OneMax is the smallest expected number of function evaluations that any algorithm following the structure of Algorithm 1 (cf. Section 2.1) exhibits on this problem. In the optimization process of Algorithm 1 a unary unbiased variation operator is asked for. In the context of optimizing pseudo-Boolean functions, a *unary* operator is an algorithm that is build upon

a family $(p(\cdot \mid x))_{x\in\{0,1\}^n}$ of probability distributions over $\{0,1\}^n$. Given some input x it outputs a new string that it samples from the distribution $p(\cdot \mid x)$. A unary operator is *unbiased* if all members of its underlying family of probability distributions are symmetric with respect to the bit positions $[n] := \{1,\dots,n\}$ and the bit values 0 and 1 (cf. [LW12] for a discussion). Unary unbiased variation operators are also referred to as *mutation operators*.

The following characterization of unary unbiased variation operators states that each such operator is uniquely defined via a probability distribution r_p over the set $[0..n] := \{0\} \cup [n]$ describing how many bits (chosen uniformly at random without replacement) are flipped in the argument x to create a new search point y. Finding a best possible mutation-based algorithm is thus identical to identifying an optimal strategy to select the distribution r_p. This characterization can also be derived from [Doe+13b, Proposition 19], although the original formulation of Proposition 19 in [Doe+13b] requires as search space $[n]^{n-1}$.

Lemma 3.1. *For every unary unbiased variation operator $(p(\cdot|x))_{x\in\{0,1\}^n}$ there exists a probability distribution r_p on $[0..n]$ such that for all $x, y \in \{0,1\}^n$ the probability $p(y|x)$ that $(p(\cdot|x))_{x\in\{0,1\}^n}$ samples y from x equals the probability of sampling a random number r from r_p and then flipping r bits in x to create y. On the other hand, each distribution r_p on $[0..n]$ induces a unary unbiased variation operator.*

To prove Lemma 3.1, we introduce the following notation.

Definition 3.2. *Let $r \in [0..n]$. For every $x \in \{0,1\}^n$ the operator flip_r creates an offspring y from x by selecting r positions $i_1,\dots,i_r$ in $[n]$ uniformly at random (without replacement), setting $y_i := 1 - x_i$ for $i \in \{i_1,\dots,i_r\}$, and copying $y_i := x_i$ for all other bit positions $i \in [n] \setminus \{i_1,\dots,i_r\}$.*

With this notation, Lemma 3.1 states that every unary unbiased variation operator $(p(\cdot|x))_{x\in\{0,1\}^n}$ can be described by Algorithm 5, for a suitably chosen probability distribution r_p.

Since it will be needed several times in the remainder of this work, we briefly recall the following simple fact about the expected OM-value of an offspring generated by flip_r.

Remark 3.3. *Let $x \in \{0,1\}^n$ and $r \in [0..n]$. The number of 1-bits that are flipped by the variation operator flip_r follows a hypergeometric distribution with expectation equal to $r\mathrm{OM}(x)/n$. The expected number of 0-bits that are flipped by the operator flip_r equals $r(n - \mathrm{OM}(x))/n$. The expected OM-value of an offspring generated from x by applying flip_r is thus equal to $\mathrm{OM}(x) - r\mathrm{OM}(x)/n + r(n - \mathrm{OM}(x))/n = (1 - 2r/n)\mathrm{OM}(x) + r$.*

Algorithm 5 The unary unbiased operator r_p samples for a given x an offspring by sampling from r_p the number r of bits to flip and then applying flip_r to x.

Choose an integer $r \in [0..n]$ according to r_p;
Sample $y \leftarrow \mathrm{flip}_r(x)$;

To prove Lemma 3.1 we show the following.

Lemma 3.4. *For every unary unbiased variation operator $(p(\cdot|x))_{x\in\{0,1\}^n}$ there exists a exists a probability distribution r_p on $[0..n]$ such that for all $x, y \in \{0,1\}^n$*

$$p(y|x) = r_p(H(x,y))/\binom{n}{H(x,y)}, \tag{3.1}$$

where here and henceforth $H(x, y)$ denotes the Hamming distance of x and y.

Proof. Let p be a unary unbiased variation operator. We first show that for all $x, y_1, y_2 \in \{0,1\}^n$ with $H(x, y_1) = H(x, y_2)$, the equality $p(y_1|x) = p(y_2|x)$ holds. This shows that for any fixed Hamming distance $d \in [n]$ and every string x, the probability distribution on the d-neighborhood $\mathcal{N}_d(x) := \left\{y \in \{0,1\}^n, H(x, y) = d\right\}$ of x is uniform.

Using the fact that the bit-wise XOR operator $\oplus$ preserves the Hamming distance, we obtain that for any $y_1, y_2 \in \mathcal{N}_d(x)$ it holds that

$$H(x \oplus x, y_1 \oplus x) = H(x \oplus x, y_2 \oplus x) = d.$$

Since $x \oplus x = (0, \ldots, 0)$, we thus observe that

$$\sum_{i=1}^{n} (y_1 \oplus x)_i = \sum_{i=1}^{n} (y_2 \oplus x)_i = d.$$

This implies that there exists a permutation $\sigma \in S_n$ such that

$$\sigma(y_1 \oplus x) = y_2 \oplus x.$$

According to the definition of unary unbiased variation operators, p is invariant under "$\oplus$" and "σ", yielding

$$\begin{aligned} p(y_1|x) &= p(y_1 \oplus x | x \oplus x) \\ &= p(\sigma(y_1 \oplus x) | \sigma(x \oplus x)) \\ &= p(y_2 \oplus x | x \oplus x) = p(y_2|x). \end{aligned}$$

This shows that $p(\cdot|x)$ is uniformly distributed on $\mathcal{N}_d(x)$ for any $d \in [n]$ and any $x \in \{0,1\}^n$.

For every $x \in \{0,1\}^n$ and every $d \in [n]$ let $p_{(d,x)}$ denote the probability of sampling a specific point at distance d from x. That is, for $y_1 \in \mathcal{N}_d(x)$ we have $p(y_1|x) = p_{(d,x)}$. For $x' \neq x$ let y' denote $y_1 \oplus (x \oplus x')$. Then by the unbiasedness of p we obtain that

$$\begin{aligned} &H(y', x') = H(y_1, x) = d, \text{ and} \\ &p_{(d,x')} = p(y'|x') = p(y' \oplus x \oplus x' \mid x' \oplus x \oplus x') = p(y_1|x) = p_{(d,x)}. \end{aligned}$$

Thus, for all $x, x' \in \{0,1\}^n$ it holds that $p_{(d,x)} = p_{(d,x')} =: p_{(d)}$.

For the unary unbiased variation operator p we can therefore define a distribution r_p on $[0..n]$ by setting

$$r_p(d) = \binom{n}{d} p_{(d)}.$$

For all $x, y \in \{0,1\}^n$ we obtain

$$p(y|x) = p_{(d,x)} = p_{(d)} = r_p(d) / \binom{n}{d},$$

where we abbreviate $d := H(x, y)$. This shows the desired equation (3.1). □

3.2 Maximizing Drift is Near-Optimal

The goal of this section is to show that the algorithm which maximizes the expected progress over the best-so-far search point is optimal, apart from lower-order terms $o(n)$, among all unary unbiased black-box algorithms. Consequently, its expected optimization time is essentially the unary unbiased black-box complexity.

3.2.1 The Drift Maximizing Algorithm

We regard as drift maximizing algorithm the algorithm summarized in Algorithm 6. We denote this algorithm by A^*. A^* starts by querying a uniform solution $x \in \{0,1\}$. In each iteration of the main loop the algorithm generates a new solution y from x by flipping exactly $R(\textsc{Om}(x))$ bits in x, where $R : [0..n] \to [0..n]$ is a function that assigns to each fitness value the number of bits that should be flipped in a search point of this quality. We choose R such that the expected progress (*drift*) is maximized. When there is more than one value maximizing the drift, R chooses the smallest among these. That is,

$$R(\textsc{Om}(x)) := \min\left\{\arg\max\left(\max\{\textsc{Om}(\text{flip}_r(x)) - \textsc{Om}(x), 0\}\right) \mid r \in [0..n]\right\}. \quad (3.2)$$

Algorithm 6 RLS for minimizing $f : \{0,1\}^n \to R$

Choose x uniformly at random from $S = \{0,1\}^n$.
for $t \leftarrow 1, 2, \ldots$ **do**
  $y \leftarrow \text{flip}_{R(\textsc{Om}(x))}(x)$
  if $\textsc{Om}(y) \geq \textsc{Om}(x)$ **then**
    $x \leftarrow y$.

In this definition, we make use of the fact that the expected progress $\mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(x)) - \textsc{Om}(x), 0\}\right)$ depends only on the fitness of x but not on its structure. This is due to the symmetry of the function Om. The offspring y replaces its parent x if and only if $\textsc{Om}(y) \geq \textsc{Om}(x)$.

Note here that the function R is deterministic, i.e., we only make use of a unary unbiased mutation operator which deterministically depends on the fitness of the current-best solution. Note further that the search point kept in the memory of algorithm A^* is always a best-so-far solution. A^* can be seen as an RLS-variant with fitness-dependent mutation strength.

3.2.2 Main Result and Proof Strategy

The main result of this entire section is the following statement, which says that the expected runtime of A^* cannot be much worse than the unary unbiased black-box complexity of OneMax.

Theorem 3.5. *Let A be a unary unbiased black-box algorithm. Denote by $T(A)$ its runtime on* OneMax *and by $T(A^*)$ the runtime of A^*. Then $\mathrm{E}(T(A)) \geq \mathrm{E}(T(A^*)) - O(n^{2/3}\ln^9(n))$.*

For the proof of Theorem 3.5 we derive in Section 3.2.3 a lower bound for the expected runtime of any unary unbiased algorithm, cf. Theorem 3.8. We then prove in Section 3.2.4 an upper bound for the expected runtime of A^*, cf. Theorem 3.9. For both statements, we use the variable drift theorems presented in Section 2.4.

We therefore need to define suitable drift functions which bound from below the expected progress that can be made by algorithm A^* and from above the maximal expected progress that any unary unbiased algorithm can make at every step of the optimization process. This is the purpose of the remainder of this subsection.

The Distance Function

Before we define the drift functions, we first note that in order to maximize the function OM, an optimal algorithm may choose to first minimize the function, and to then flip all bits at once to obtain the optimal OM-solution. Instead of regarding the *maximization of the function* OM*,* we therefore regard in the following the problem of *minimizing the distance function* d, which assigns to each string x the value $d(x) := \min\{n - \textsc{Om}(x), \textsc{Om}(x)\}$. The black-box complexities of both problems are almost identical, as the following lemma shows.

Lemma 3.6. *The unary unbiased black-box complexities of maximizing* ONEMAX *is at least as large as that of minimizing* d *and it is larger by at most one.*

Proof. For the first statement, it suffices to observe that we can simulate the optimization of d when ONEMAX-values are available. The second statement follows from the already mentioned fact that once we have found a string x of distance value $d(x) = 0$, then either x or its bit-wise complement $\bar{x}$ has maximal ONEMAX-value. □

Drift Expressions

For the definition of the drift functions used in the proofs of Theorems 3.8 and 3.9, we use the following notation. For a unary unbiased algorithm A we denote by $(x(0), x(1), \dots, x(t))$ the sequence of the first $t+1$ search points evaluated by A. For every such sequence of search points, we abbreviate by

$$X_t := \min\{d(x(i)) | i \in [0..t]\}$$

the quality of a best-so-far solution with respect to the distance function d. Note that for all $t \geq 0$ it holds that $X_t \geq X_{t+1}$, i.e., the sequence $(X_t)_{t\geq 0}$ is monotonically decreasing in t. For each $k \in [0..n]$ let $\mathcal{H}(t,k)$ be the collection of all sequences $(x(0), x(1), \dots, x(t))$ of search points for which $X_t(x(0), x(1), \dots, x(t)) = k$. Abusing notation, we write $x \in ((x(0), x(1), \dots, x(t)))$ when $x = x(i)$ for some $i \in [0..t]$.

Denoting by $\mathcal{U}$ the set of all unary unbiased operators acting on $\{0,1\}^n$, the maximal possible drift that can be achieved by a unary unbiased variation operator when the best-so-far distance is equal to $k \in [0..n]$ is equal to

$$\hat{h}(k) := \max\left\{\max\left\{k - \mathrm{E}(\min\{k, d(\tau(x))\}) \mid \tau \in \mathcal{U}, x \in H\right\} \mid t \in \mathbb{N}, H \in \mathcal{H}(t,k)\right\}.$$

Using Lemma 3.1 it is not difficult to show that

$$\hat{h}(k) = \max\left\{k - \mathrm{E}(\min\{k, d(\mathrm{flip}_r(x))\}) \mid r \in [0..n], x \in \{0,1\}^n \text{ with } d(x) \geq k\right\}. \quad (3.3)$$

To obtain a monotonically increasing function (as required by Theorem 2.7), we set

$$h(k) := \max\{\hat{h}(i) \mid i \in [0..k]\}. \quad (3.4)$$

Note that $h(k) \geq \hat{h}(k)$ for all $k \in [0..n]$.

Finally, we set

$$\tilde{h}(k) := \max\{\mathrm{E}(\max\{\textsc{Om}(\mathrm{flip}_r(x)) - \textsc{Om}(x), 0\}) \mid x \in \{0,1\}^n \text{ with } \textsc{Om}(x) = n-k, r \in [0..n]\}, \tag{3.5}$$

the maximal Om-drift that can be obtained from a search point whose Om-value is exactly equal to $n-k$. We certainly have $\tilde{h}(k) \leq h(k)$ for all $k \leq n/2$. However, we will show in Section 3.2.4 that for all values $k \leq n/2 - n^{0.6}$ the difference between the functions h and $\tilde{h}$ is small, showing that we can approximate the maximal drift h by mutating a best-so-far solution. This will be the key step in proving the upper bound for the expected runtime of algorithm A^*. We also notice that $\tilde{h}(k) \geq \max\{\textsc{Om}(\bar{x}) - \textsc{Om}(x), 0\} \geq 2k - n$. Therefore for any $x \in \{0,1\}^n$ with $\textsc{Om}(x) = n-k$ we have $\textsc{Om}(x) + \tilde{h}(k) \geq \max\{\textsc{Om}(x), \textsc{Om}(\bar{x})\} \geq n/2$, so that A^* very quickly has a search point of function value $\textsc{Om}(x) \geq n/2$. Informally, the interesting part of the runtime analysis for A^* is therefore the fitness increase from a value around $n/2$ to n.

3.2.3 A Lower Bound for all Unary Unbiased Algorithms

Before proving the lower bound, we first introduce the following lemma arguing that the probability to make a large fitness gain is bounded by a small probability, as required by the condition to apply Theorem 2.7.

Lemma 3.7. *There exists an $n_0 \in \mathbb{N}$ such that, for all $n \geq n_0$, for all $r \in [0..n]$, and for all $x \in \{0,1\}^n$, it holds that*

$$\Pr\left(d\left(\mathrm{flip}_r(x)\right) \geq \tilde{c}\left(d(x)\right)\right) \geq 1 - n^{-4/3}\ln^7(n), \tag{3.6}$$

where

$$\tilde{c} : [n] \to [0..n], i \mapsto \tilde{c}(i) := \begin{cases} i - \sqrt{n}\ln n & \text{for } i \geq n/6, \\ i - \ln^2(n) & \text{for } n^{1/3} \leq i < n/6, \\ i - 1 & \text{for } i < n^{1/3}. \end{cases}$$

Proof. Set $p := n^{-4/3}\ln^7(n)$. To show (3.6), we first note that we can assume without loss of generality that $\textsc{Om}(x) \geq n/2$. This is due to the symmetry of the distance function. In addition, we can assume that $0 < r \leq n/2$, because $d(\mathrm{flip}_r(x))$ and $d(\mathrm{flip}_{n-r}(x))$ are identically distributed.

We make a case distinction according to the size of $d := d(x)$. For all different cases we note that the event $d(\mathrm{flip}_r(x)) < d$ happens in two cases, namely if $\textsc{Om}(\mathrm{flip}_r(x)) > \textsc{Om}(x)$ or if $\textsc{Om}(\mathrm{flip}_r(x)) < n - \textsc{Om}(x)$. We denote by Z the number of good flips in an application of flip_r to x; i.e., the number of bits flipping from 0 to 1. Z follows a hypergeometric distribution and $\mathrm{E}(Z) = rd(x)/n$, as discussed in Remark 3.3.

Case 1: $d(x) \geq n/6$. We first regard the case that $d := d(x) \geq n/6$. The Chernoff bound (e.g., the variants presented in Theorems 1.11 and 1.17 in [Doe11]) applied to Z show that, for all $\lambda > 0$, $\Pr(Z > \mathrm{E}(Z) + \lambda) \leq \exp(-2\lambda^2/n)$ and $\Pr(Z < \mathrm{E}(Z) - \lambda) \leq \exp(-2\lambda^2/n)$. Using that $\textsc{Om}(\mathrm{flip}_r(x)) = \textsc{Om}(x) + 2Z - r$ and that $n - \textsc{Om}(x) \leq \mathrm{E}(\textsc{Om}(\mathrm{flip}_r(x))) \leq \textsc{Om}(x)$, we obtain that

$$\begin{aligned} &\Pr\left(\textsc{Om}(\mathrm{flip}_r(x)) < n - \textsc{Om}(x) - 2\lambda\right) \\ &\leq \Pr\left(\textsc{Om}(\mathrm{flip}_r(x)) < \mathrm{E}(\textsc{Om}(\mathrm{flip}_r(x))) - 2\lambda\right) \\ &= \Pr\left(\textsc{Om}(x) + 2Z - r < \textsc{Om}(x) + E(2Z) - r - 2\lambda\right) \\ &= \Pr\left(2Z < E(2Z) - 2\lambda\right) \leq \exp(-2\lambda^2/n), \end{aligned}$$

and that

$$\begin{aligned}&\Pr\left(\textsc{Om}(\text{flip}_r(x)) > \textsc{Om}(x) + 2\lambda\right)\\ &\leq \Pr\left(\textsc{Om}(\text{flip}_r(x)) > \mathrm{E}(\textsc{Om}(\text{flip}_r(x))) + 2\lambda\right)\\ &= \Pr\left(2Z > E(2Z) + 2\lambda\right) \leq \exp(-2\lambda^2/n).\end{aligned}$$

From these two inequalities we conclude that

$$\Pr(d(\text{flip}_r(x)) < \tilde{c}(d(x)) = \Pr(d(\text{flip}_r(x)) < d - \sqrt{n}\ln n) \leq 2\exp(-\ln^2(n)/2) < p$$

for all $d \geq n/6$.

Case 2: $n^{1/3} \leq d(x) < n/6$. By our assumptions $r \leq n/2$ and $\textsc{Om}(x) \geq n/2$ we obtain from Remark 3.3 that $\mathrm{E}(\textsc{Om}(\text{flip}_r(x))) = (1 - \frac{2r}{n})\textsc{Om}(x) + r \geq (1 - \frac{2r}{n})\frac{n}{2} + r = n/2$. Using Chernoff bounds (e.g., Theorems 1.11 and 1.17 in [Doe11]) we thus get $\Pr(\textsc{Om}(\text{flip}_r(x)) < n/6) \leq \exp(-2(n/2 - n/6)^2/n) = o(p)$.

Consider the event $\textsc{Om}(\text{flip}_r(x)) \geq \textsc{Om}(x) + \ln^2(n)$. It intrinsically requires that $r \geq \ln^2(n)$. We apply the Chernoff bound from Corollary 1.10 (b) in [Doe11] to Z and obtain that

$$\begin{aligned}\Pr\left(\textsc{Om}(\text{flip}_r(x)) > \textsc{Om}(x)\right) = \Pr\left(Z > \frac{r}{2}\right) = \Pr\left(Z > \frac{n}{2d}\mathrm{E}(Z)\right) \leq \left(\frac{2de}{n}\right)^{r/2}\\ \leq \left(\frac{e}{3}\right)^{r/2} = O(n^{-\Omega(\ln(n))}) = o(p).\end{aligned} \tag{3.7}$$

Therefore

$$\begin{aligned}&\Pr(d(\text{flip}_r(x)) < \tilde{c}(d(x)) = \Pr(d(\text{flip}_r(x)) < d - \ln^2 n)\\ &\leq \Pr(\textsc{Om}(\text{flip}_r(x)) < n/6) + \Pr\left(\textsc{Om}(\text{flip}_r(x)) > \textsc{Om}(x)\right) = o(p)\end{aligned}$$

for all $n^{1/3} \leq d < n/6$.

Case 3: $d(x) < n^{1/3}$. We finally consider the case $d < n^{1/3}$. Applying this condition to the first line of Equation (3.7), we obtain $\Pr(\textsc{Om}(\text{flip}_r(x)) > \textsc{Om}(x)) \leq \Theta(n^{-4/3}) = o(p)$ for all $r \geq 4$. For $r < 4$ we consider the operators separately and observe that

$$\Pr\left(\textsc{Om}(\text{flip}_3(x)) > \textsc{Om}(x)\right) = \frac{\binom{d}{2}\binom{n-d}{1} + \binom{d}{3}}{\binom{n}{3}} = O(n^{-4/3}) = o(p)$$

and

$$\Pr\left(\textsc{Om}(\text{flip}_2(x)) > \textsc{Om}(x)\right) = \frac{\binom{d}{2}}{\binom{n}{2}} = O(n^{-4/3}) = o(p).$$

Thus, altogether, we obtain that

$$\begin{aligned}&\Pr(d(\text{flip}_r(x)) < \tilde{c}(d(x)) = \Pr(d(\text{flip}_r(x)) < d - 1)\\ &\leq \Pr(\textsc{Om}(\text{flip}_r(x)) < n/6) + \Pr\left(\textsc{Om}(\text{flip}_r(x)) > \textsc{Om}(x)\right) = o(p)\end{aligned}$$

for all $r \geq 2$ and $d < n^{1/3}$. Needless to say that $\Pr(d(\text{flip}_1(x)) < \tilde{c}(d(x)) = 0$. This proves Equation (3.6). □

Using Lemma 3.7 and Theorem 2.7, we are now ready to show the following lower bound.

Theorem 3.8. *Let $s := n/2 - n^{0.6}$. The expected runtime of any unary unbiased algorithm A on the* OneMax *problem satisfies*

$$E(T_A) \geq \sum_{x=1}^{s} \frac{1}{h(x)} - \Theta(n^{2/3}\ln^9(n)).$$

Proof. For convenience, we assume that n is sufficiently large. Let A be a unary unbiased algorithm. We recall that by $(x(0), x(1), \dots)$ we denote the sequence of search points evaluated by A and by $X_t = \min\{d(x(i)) \mid i \in [0..t]\}$ the best-so-far distance after first t iterations of the main loop. Let $T := \min\{t \in \mathbb{N} \mid X_t = 0\}$. Then $T \leq T_A := \min\{t \in \mathbb{N} \mid \textsc{Om}(x(t)) = n\}$. We prove a lower bound on T.

Every unary unbiased black-box algorithm has to create its first search point uniformly at random. This initial search point x_0 has expected fitness $\mathrm{E}(\textsc{Om}(x_0)) = n/2$. By Chernoff's bound (we can use, for example, the variant presented in Theorem 1.11 in [Doe11]) it furthermore holds that $\Pr(X_0 < s) = \Pr(|\textsc{Om}(x_0) - n/2| > n^{0.6}) \leq 2\exp(-2n^{0.2})$. This probability is small enough such that even optimistically assuming $T_A = 0$ whenever $X_0 < s$, the contribution of such events affect the lower bound by a term of $O(n^{2/3}\ln^9(n))$. We can therefore safely assume that $X_0 \geq s$.

For all $i \in [n]$ let $c(i) := \min\{\tilde{c}(j) \mid j \geq i\}$, where $\tilde{c}$ is the function defined in Lemma 3.7. For all $r \in [0..n]$, $x \in \{0,1\}^n$, and all distance levels $d' \leq d(x)$ it holds that

$$c(d') \leq \tilde{c}(d(x)) \text{ and } \Pr\left(d(\mathrm{flip}_r(x)) > c(d')\right) \geq \Pr\left(d(\mathrm{flip}_r(x)) > \tilde{c}(d(x))\right) \geq 1 - p.$$

By Lemma 3.1, this statement can be extended to arbitrary unary unbiased variation operators. Therefore,

$$\Pr\left(X_{t+1} > c(X_t)\right) \geq 1 - p \text{ for all } t \in \mathbb{N}.$$

We apply Theorem 2.7 to c and h. We first compute μ as in Theorem 2.7. By definition, $\mu(i) = \max\{x \mid c(x) \leq i\} = \max\{x \mid \min\{\tilde{c}(y) \mid y \geq x\} \leq i\} = \max\{x \mid \tilde{c}(x) \leq i\}$, giving

$$\mu(i) := \begin{cases} i+1 & \text{for } i < n^{1/3} - \ln^2(n), \\ i + \ln^2(n) & \text{for } n^{1/3} - \ln^2(n) \leq i < n/6 - \sqrt{n}\ln n, \\ i + \sqrt{n}\ln n & \text{for } n/6 - \sqrt{n}\ln n \leq i < n/2 - \sqrt{n}\ln n, \\ \lfloor n/2 \rfloor & \text{for } n/2 - \sqrt{n}\ln n \leq i \leq n/2. \end{cases}$$

According to Theorem 2.7, we can thus bound $\mathrm{E}(T \mid X_0)$ by

$$E(T \mid X_0) \geq g(X_0) - \frac{g^2(X_0)p}{1 + g(X_0)p} \text{ with } g(x) = \sum_{i=1}^{x-1} \frac{1}{h(\mu(x))}.$$

Since $h(\mu(x)) \geq \tilde{h}(\mu(x)) \geq \mathrm{E}(\max\{\textsc{Om}(\mathrm{flip}_1(x)) - \textsc{Om}(x), 0\} \mid x \in \{0,1\}^n \text{ with } \textsc{Om}(x) = n - \mu(x)) = \mu(x)/n \geq x/n$, we obtain $g(X_0) \leq \sum_{i=1}^{X_0-1} \frac{n}{i} = O(n\ln(n))$. Therefore $\frac{g^2(X_0)p}{1+g(X_0)p} = O(n^{2/3}\ln^9(n))$ for $p = n^{-4/3}\ln^7(n)$. Using the

monotonicity of h and the fact that all summands are positive, we estimate $g(X_0)$ by

$$\begin{aligned}\sum_{x=0}^{X_0-1} \frac{1}{h(\mu(x))} &\geq \sum_{x=1}^{n^{1/3}-\ln^2(n)} \frac{1}{h(x)} + \sum_{x=n^{1/3}}^{n/6-\sqrt{n}\ln n+\ln^2(n)} \frac{1}{h(x)} + \sum_{x=n/6}^{X_0} \frac{1}{h(x)} \\ &\geq \sum_{x=1}^{X_0} \frac{1}{h(x)} - \frac{\ln^2(n)}{h(n^{1/3}-\ln^2(n))} - \frac{\sqrt{n}\ln n - \ln^2(n)}{h(n/6-\sqrt{n}\ln n+\ln^2(n))} \\ &\geq \sum_{x=1}^{X_0} \frac{1}{h(x)} - \Theta(n^{2/3}\ln^2(n)).\end{aligned}$$

Therefore we obtain $E(T_A) \geq \mathrm{E}(T \mid X_0 \geq s) - O(n^{2/3}\ln^9(n)) \geq \sum_{x=1}^{s} \frac{1}{h(x)} - \Theta(n^{2/3}\ln^9(n))$.

□

3.2.4 Upper Bound for the Drift Maximizer

The lower bound in Theorem 3.8 also holds for drift-maximizer A^* described in the beginning of this section. We next show that A^* achieves this runtime bound apart from the lower order term $\Theta(n^{2/3}\ln^9(n))$.

Theorem 3.9. *Let $s := n/2 - n^{0.6}$. The expected runtime of algorithm A^* on* OneMax *satisfies*

$$E(T_{A^*}) \leq \sum_{x=1}^{s} \frac{1}{h(x)} + \Theta(n^{0.6}).$$

From the variable drift theorem, Theorem 2.4, we easily get $\sum_{x=1}^{s} \frac{1}{\tilde{h}(x)}$ as upper bound for the expected runtime of A^*. We therefore need to show that the difference between $h(X_t)$ and $\tilde{h}(X_t)$ is small. This is the purpose of the next subsection.

Maximizing Drift by Mutating a Best-So-Far Solution

As mentioned above, we show that the expected drift of any unary unbiased algorithm cannot be significantly better than that of A^*. The main result of this subsection is the following lemma.

Lemma 3.10. *For sufficiently sufficiently large n and $k \leq n/2 - n^{0.6}$ it holds that $0 \leq h(k) - \tilde{h}(k) \leq n\exp(-\Omega(n^{0.2}))$.*

We start our proof of Lemma 3.10 by observing that the expected Om-value of the search point obtained from mutating and selecting the best of parent and offspring is strictly increasing with the quality of the parent. The proof is by induction. The base case is covered by the following lemma.

Lemma 3.11. *Let $x, y \in \{0,1\}^n$ with $\mathrm{Om}(y) = \mathrm{Om}(x) + 1 \geq n/2$ and let $r \in [0..n/2]$. For all $t \geq 1$ it holds that*

$$\Pr(\mathrm{Om}(\mathrm{flip}_r(x)) = \mathrm{Om}(y) + t) \leq \Pr(\mathrm{Om}(\mathrm{flip}_{r-1}(y)) = \mathrm{Om}(y) + t). \tag{3.8}$$

Proof. We first notice that for $t > r - 1$ both two probabilities are zero. We can therefore assume that $1 \leq t \leq r - 1$. Let $d := n - \mathrm{Om}(y)$, and let i be the number of zeros in y that flip_{r-1} flips from zero to one. Then there are $r - 1 - i$ ones that flip to zero. We thus have $\mathrm{Om}(\mathrm{flip}_{r-1}(y)) = \mathrm{Om}(y) + t$ if and only if $i - (r - 1 - i) = t$; i.e.,

if and only if $i = (t + r - 1)/2$. By the same reasoning $\textsc{Om}(\text{flip}_r(x)) = \textsc{Om}(y) + t$ if and only if $i' - (r - i') = t + 1$ for i' being the number of zeros flipped by flip_r. This implies $i' = (r - 1 + t)/2 = i + 1$. We thus obtain

$$\Pr\left(\textsc{Om}(\text{flip}_{r-1}(y)) = \textsc{Om}(y) + t\right) = \frac{\binom{d}{i}\binom{n-d}{r-1-i}}{\binom{n}{r-1}}$$

$$\Pr\left(\textsc{Om}(\text{flip}_{r}(x)) = \textsc{Om}(y) + t\right) = \frac{\binom{d+1}{i+1}\binom{n-d-1}{r-(i+1)}}{\binom{n}{r}}$$

To show equation (3.8), we abbreviate $j := r - 1 - i$ and use the facts that $\binom{d+1}{i+1} = \binom{d}{i}\frac{d+1}{i+1}$, $\binom{n-d}{j} = \binom{n-d-1}{j}\frac{n-d}{n-d-j}$, and $\binom{n}{i+j+1} = \binom{n}{i+j}\frac{n-i-j}{i+j+1}$ to obtain that

$$\frac{\frac{\binom{d+1}{i+1}\binom{n-d-1}{r-(i+1)}}{\binom{n}{r}}}{\frac{\binom{d}{i}\binom{n-d}{r-1-i}}{\binom{n}{r-1}}} = \frac{(d+1)(n-d-j)(1+i+j)}{(i+1)(n-d)(n-i-j)}.$$

We aim to show the above ratio less or equal to 1. To this end, we compute the difference between the numerator and the denominator, and obtain $(d+1)(n-d-j)(1+i+j) - (i+1)(n-d)(n-i-j) = (1+i+j+d-n)(n+in-(1+i+j)d) - j) = (r+d-n)(n+in-rd-j)$. The first factor in this expression is negative, since both $r \leq n/2$ and $d \leq n/2$. The second factor is positive, because $n > j$, $i > r/2$ and $d \leq n/2$ implying that $in - rd > (r/2)n - r(n/2) \geq 0$. □

We now regard the case that the same number r of bits are flipped in the two strings x and y.

Lemma 3.12. *Let $x, y \in \{0,1\}^n$ with $\textsc{Om}(y) = \textsc{Om}(x) + 1$ and let $r \in [0..n]$. It holds that*

$$\mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(x)) - \textsc{Om}(x), 0\}\right) \geq \mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(y)) - \textsc{Om}(y), 0\}\right). \quad (3.9)$$

Proof. Since permutation on bit-positions does not affect the analysis, we assume that x and y are of the following form.

$$x = \underbrace{11\cdots11}_{\frac{n}{2}+\delta}\underbrace{00\cdots00}_{\frac{n}{2}-\delta-1}0 \text{ and } y = \underbrace{11\cdots11}_{\frac{n}{2}+\delta}\underbrace{00\cdots00}_{\frac{n}{2}-\delta-1}1. \quad (3.10)$$

For any r-sized subset S of $[n]$ let x_S (y_S) denote the offspring of x (y) in which the r positions in S are flipped. Then x_S and y_S differ only in the last bit and we have $\textsc{Om}(x_S) - \textsc{Om}(y_S) \in \{-1, 1\}$ for all S. Therefore

$$\max\{\textsc{Om}(\text{flip}_r(x)), \textsc{Om}(x)\} - \max\{\textsc{Om}(\text{flip}_r(y)), \textsc{Om}(y)\} \geq -1 \text{ for all } S$$

Using that $\textsc{Om}(y) - \textsc{Om}(x) = 1$ we obtain

$$\begin{aligned} & \mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(x)) - \textsc{Om}(x), 0\}\right) - \mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(y)) - \textsc{Om}(y), 0\}\right) \\ = \; & \mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(x)), \textsc{Om}(x)\} - \textsc{Om}(x)\right) - \mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(y)), \textsc{Om}(y)\} - \textsc{Om}(y)\right) \\ = \; & \mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(x)), \textsc{Om}(x)\} - \max\{\textsc{Om}(\text{flip}_r(y)), \textsc{Om}(y)\}\right) + 1 \geq 0. \end{aligned}$$

□

We now extend the last two lemmas to the case $\textsc{Om}(y) - \textsc{Om}(x) > 1$.

Corollary 3.13. *Let $x, y \in \{0,1\}^n$ with $\textsc{Om}(y) > \textsc{Om}(x)$.*

1. *if $\textsc{Om}(x) \geq n/2$, $r \in [0..n/2]$, and $r' = \max\{r - (\textsc{Om}(y) - \textsc{Om}(x)), 0\}$. Then for all $t \in \mathbb{N}_{\geq 1}$ we have*

$$\Pr\left(\textsc{Om}(\text{flip}_r(x)) = \textsc{Om}(y) + t\right) \leq \Pr\left(\textsc{Om}(\text{flip}_{r'}(y)) = \textsc{Om}(y) + t\right). \quad (3.11)$$

2. *It also holds that*

$$\tilde{h}(\textsc{Om}(x)) \geq \tilde{h}(\textsc{Om}(y)).$$

Proof. To see the first statement, we first regard the case that $r' = 0$. In this case, we have $r \leq \textsc{Om}(y) - \textsc{Om}(x)$, so that the probability that $\textsc{Om}(\text{flip}_r(x)) = \textsc{Om}(y) + t$ is zero for $t > 0$. For $t = 0$ the statement trivially holds, since the right-hand side of (3.11) is equal to one. For $r' > 0$ the first statement follows from Lemma 3.11 and an induction over $\textsc{Om}(y) - \textsc{Om}(x)$.

To prove the second statement we first assume that $\textsc{Om}(y) - \textsc{Om}(x) = 1$. Let r be the value that maximizes $\mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(y)) - \textsc{Om}(y), 0\}\right)$. Using Lemma 3.12 we obtain

$$\begin{aligned}\tilde{h}(\textsc{Om}(y)) &= \mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(y)) - \textsc{Om}(y), 0\}\right) \\ &\leq \mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(x)) - \textsc{Om}(x), 0\}\right) \leq \tilde{h}(\textsc{Om}(x)).\end{aligned}$$

The general statement now follows by induction over $\textsc{Om}(y) - \textsc{Om}(x)$.

□

We are now ready to prove the main result of this subsection, Lemma 3.10.

of Lemma 3.10. Let $k \leq n/2 - n^{0.6}$, x a search point with $d(x) \geq k$ and let $r \in [0..n]$. Since all random variables $d(\text{flip}_r(x))$, $d(\text{flip}_{n-r}(x))$, $d(\text{flip}_r(\bar{x}))$, and $d(\text{flip}_{n-r}(\bar{x}))$ are identically distributed, we can assume without loss of generality that $\textsc{Om}(x) \geq n/2$ and that $r \leq n/2$.

Using this and the observations made in Remark 3.3, we easily see that

$$\begin{aligned}\mathrm{E}(\textsc{Om}(\text{flip}_r(x))) &= \textsc{Om}(x) - r\tfrac{\textsc{Om}(x)}{n} + r\tfrac{n - \textsc{Om}(x)}{n} \\ &= \textsc{Om}(x)(1 - r/n) + (n - \textsc{Om}(x))r/n \geq n/2.\end{aligned}$$

This shows that $\mathrm{E}(\textsc{Om}(\text{flip}_r(x))) - n^{0.6} \geq n/2 - n^{0.6} \geq k$. Together with a Chernoff bound applied to $\textsc{Om}(\text{flip}_r(x))$ we thus obtain

$$\begin{aligned}&\Pr\left(\textsc{Om}(\text{flip}_r(x)) < k\right) \\ &\leq \Pr\left(\textsc{Om}(\text{flip}_r(x)) < \mathrm{E}(\textsc{Om}(\text{flip}_r(x))) - n^{0.6}\right) = \exp(-\Omega(n^{0.2})). \quad (3.12)\end{aligned}$$

We first aim at bounding $\hat{h}(k)$. To this end, we use the estimate 3.12 to obtain

$$\begin{aligned}&\mathrm{E}\left(k - \min\{k, d(\text{flip}_r(x))\}\right) \\ &= \mathrm{E}\left(\textsc{Om}(\text{flip}_r(x)) - (n-k) \mid \textsc{Om}(\text{flip}_r(x)) > n - k\right) \Pr\left(\textsc{Om}(\text{flip}_r(x)) > n - k\right) \\ &\quad + \mathrm{E}\left(k - \textsc{Om}(\text{flip}_r(x)) \mid \textsc{Om}(\text{flip}_r(x)) < k\right) \Pr\left(\textsc{Om}(\text{flip}_r(x)) < k\right) \\ &\leq \mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(x)) - (n-k), 0\}\right) + n\exp(-\Omega(n^{0.2})).\end{aligned}$$

Let x' be a search point with $\textsc{Om}(x') = n - k$, and let $r' = \max\{r - (d(x) - d(x')), 0\}$. According to Corollary 3.13 it holds for all $i \in \mathbb{N}$ that

$$\Pr(\textsc{Om}(\text{flip}_r(x)) = n - k + i) \leq \Pr(\textsc{Om}(\text{flip}_{r'}(x')) = n - k + i).$$

Using that $\mathrm{E}\left(\max\{\textsc{Om}(\mathrm{flip}_r(x)) - (n-k), 0\}\right) = \sum_{i\geq 1} i \Pr(\textsc{Om}(\mathrm{flip}_r(x)) = n-k+i)$ we obtain

$$\mathrm{E}\left(k - \min\{k, d(\mathrm{flip}_r(x))\}\right) \leq \mathrm{E}\left(\max\{\textsc{Om}(\mathrm{flip}_{r'}(x')) - (n-k), 0\}\right) + n\exp(-\Omega(n^{0.2})).$$

Referring to the definition of $\hat{h}$ and $\tilde{h}$ in equations (3.3) and (3.5), and using the symmetries mentioned in the beginning of this proof, we bound

$$\begin{aligned}
&\hat{h}(k) = \max\{k - \mathrm{E}\left(\min\{k, d(\mathrm{flip}_r(x))\}\right) \mid r \in [0..n], x \in \{0,1\}^n \text{ with } d(x) \geq k\} \\
&= \max\{k - \mathrm{E}\left(\min\{k, d(\mathrm{flip}_r(x))\}\right) \mid r \in [0..n/2], x \in \{0,1\}^n \text{ with } n/2 \leq \textsc{Om}(x) \leq n-k\} \\
&\leq \max\{\mathrm{E}\left(\max\{\textsc{Om}(\mathrm{flip}_{r'}(x')) - (n-k), 0\}\right) \mid r' \in [0..n/2], x' \in \{0,1\}^n \text{ with } \textsc{Om}(x') = n-k\} \\
&\quad + n\exp(-\Omega(n^{0.2})) \\
&\leq \tilde{h}(k) + n\exp(-\Omega(n^{0.2})).
\end{aligned}$$

According to the definition of $h(k)$ in Equation (3.4), we obtain

$$\begin{aligned}
h(k) &= \max\{\hat{h}(i) \mid i \in [0..k]\} \leq \max\{\tilde{h}(i) \mid i \in [0..k]\} + n\exp(-\Omega(n^{0.2}) \\
&= \tilde{h}(k) + n\exp(-\Omega(n^{0.2}),
\end{aligned}$$

where the last equality uses the monotonicity of $\tilde{h}(k)$ with respect to k shown in Corollary 3.13. □

As we will see in the next subsection, the $n\exp(-\Omega(n^{0.2}))$ term in this bound accounts for an additive $O(1)$ error in the runtime estimate only.

Proof of Theorem 3.9

With Lemma 3.10 at hand, we are now ready to prove Theorem 3.9.

Proof of Theorem 3.9. As mentioned above, we easily obtain from Theorem 2.4 that

$$E\left(T_{A^*} \mid x(0)\right) \leq \sum_{x=1}^{n-\textsc{Om}(x(0))} \frac{1}{\tilde{h}(x)}. \tag{3.13}$$

According to Lemma 3.10 it holds that $0 < h(x) - \tilde{h}(x) \leq n\exp(-\Omega(n^{0.2}))$ for $x \leq n/2 - n^{0.6}$. Using again that flipping a single bit on a bit string with x zeros gives an expected progress in the Om-value of x/n, we recall that $h(x) \geq \tilde{h}(x) \geq x/n$ for all $x \in [n/2]$. Therefore,

$$\frac{1}{\tilde{h}(x)} - \frac{1}{h(x)} = \frac{h(x) - \tilde{h}(x)}{\tilde{h}(x)h(x)} \leq \frac{n\exp(-\Omega(n^{0.2}))}{(x/n)(x/n)} \leq n^3\exp(-\Omega(n^{0.2})) \text{ for all } 1 \leq x \leq s.$$

Replacing $\tilde{h}(x)$ by $h(x)$ in inequality (3.13) and pessimistically assuming $\textsc{Om}(x(0)) = 0$ we thus obtain

$$\begin{aligned}
E\left(T_{A^*}\right) &\leq \sum_{x=1}^{s} \left(\frac{1}{h(x)} + n^3\exp(-\Omega(n^{0.2}))\right) + \sum_{x=s}^{n} \frac{1}{\tilde{h}(x)} \\
&= \sum_{x=1}^{s} \frac{1}{h(x)} + \sum_{x=s}^{n} \frac{1}{\tilde{h}(x)} + O(1).
\end{aligned}$$

Using that $\tilde{h}(x) \geq x/n$ for all $0 < x \leq n$ and $\tilde{h}(x) \geq 2x - n$ for all $n/2 < x \leq n$, we conclude

$$\sum_{x=s}^{n} \frac{1}{\tilde{h}(x)} \leq \sum_{x=s}^{n/2+n^{0.6}} \frac{n}{x} + \sum_{x=n/2+n^{0.6}}^{n} \frac{1}{2x-n} \leq \frac{2n^{1.6}}{s} + \frac{n/2}{2n^{0.6}} = \Theta(n^{0.6}).$$

□

Theorem 3.5 follows from Theorems 3.8 and 3.9 by observing that $\Theta(n^{0.6}) = o(n^{2/3}\ln^9(n))$.

3.3 Fitness-Dependent Mutation Strength

In the previous section we have seen that in order to compute the expected runtime of a best possible unary unbiased black-box algorithm for OneMax we can regard the algorithm A^* that maximizes at any point in time the fitness drift. By Theorems 3.8 and 3.9 this algorithm cannot be worse (in expectation) than an optimal unary unbiased one by more than an additive $\Theta(n^{2/3}\ln^9(n))$ term.

In this section we give a relatively concise description of A^*, i.e., we compute approximately the number of bits that need to be flipped in order to maximize the fitness drift. Since we are here talking about the drift in the fitness, it will be convenient to denote in this section by $d(x) = n - \textsc{Om}(x)$ the fitness distance to the target. We also denote by

$$R_{\text{opt}}(d,n) := \min\Big\{ \arg\max \mathrm{E}\left(\max\{\textsc{Om}(\text{flip}_r(x)) - \textsc{Om}(x), 0\}\right) \\ \mid r \in [0..n], \textsc{Om}(x) = n-d\Big\}, \tag{3.14}$$

the number of bits that need to be flipped in a search point $x \in \{0,1\}^n$ with $\textsc{Om}(x) = n-d$ such that the expected drift $\mathrm{E}(\max\{0, \textsc{Om}(\text{flip}_{R_{\text{opt}}(d,n)}(x)) - \textsc{Om}(x)\})$ is maximized (breaking ties by flipping fewer bits).

The exact analysis of R_{opt} is rather tedious, as we will demonstrate below. Luckily, it turns out that we can safely approximate this point-wise drift maximizing function $R_{\text{opt}}(d,n)$ by some a function $\tilde{R}_{\text{opt}} : [0,1] \to [0..n]$ which maps the relative fitness distance $d(x)/n$ to a mutation strength. Since $\tilde{R}_{\text{opt}}$ is much easier to work with, this is the focus of Section 3.3.2. For the approximation $\tilde{R}_{\text{opt}}$ we make use of the fact that for values of r that are reasonably small compared to the problem dimension n and the current fitness distance $d(x)$, the expected drift $\mathrm{E}(\max\{0, \textsc{Om}(\text{flip}_{R_{\text{opt}}(d,n)}(x)) - \textsc{Om}(x)\})$ is almost determined by the relative fitness distance $d(x)/n$. For very small $d(x) = o(n)$ the fitness drift of flipping $R_{\text{opt}}(d/n) = 1$ bit is exactly $d(x)/n$, without any estimation error. We will also see in Section 3.3.1 that it suffice to regard constant values r.

Once the approximation of the function R_{opt} by $\tilde{R}_{\text{opt}}$ is established, we demonstrate in Section 3.3.3 a few properties of these two functions that will be useful in our subsequent computations; in particular for the numerical approximation of $\tilde{R}_{\text{opt}}$, which is carried out in Section 3.4.2. Most importantly, we shall see that $\tilde{R}_{\text{opt}}$ is monotone, i.e., the number of bits to flip in order to maximize the approximated point-wise drift decreases with increasing fitness. We also show that both R_{opt} and $\tilde{R}_{\text{opt}}$ take only odd values, implying that flipping an even number of bits is suboptimal in all stages of the optimization process.

To ease the computation of $\tilde{R}_{\text{opt}}$, we analyze in detail the mutation rate for search points $x(t)$ with fitness distance $X_t \leq (1/2-\varepsilon)n$, where the constant ε satisfies $0 < \varepsilon < 1/2$. Notice that by selecting between parent and its offspring at the end of each iteration in Algorithm 6 we have $\textsc{Om}(x(t)) = n - X_t$. For the remaining fitness distances, we simply take $\tilde{R}_{\text{opt}}(1/2-\varepsilon)$ for all $(1/2-\varepsilon)n < X_t \leq n/2$ and n for all $X_t > n/2$. A detailed definition of $\tilde{R}_{\text{opt},\varepsilon}$ is provided in equation (3.19). We will prove in Theorem 3.28 that our adhoc definition of $\tilde{R}_{\text{opt},\varepsilon}(p)$ for $p \geq 1/2-\varepsilon$ only causes an error term of $O(\varepsilon n)$ in the runtime.

3.3.1 The Exact Fitness Drift $B(n,d,r)$

In this subsection, we compute the exact fitness gain obtained from flipping r bits. We shall then argue that once we have a fitness of at least $(\frac{1}{2}+\varepsilon)n$ for some constant ε, the maximal fitness drift stems from flipping some constant number of bits.

Let x be a binary string of length n with fitness distance d, that is, with OneMax-value $n-d$. By the symmetry of the OneMax function, the expected progress of flipping r bits in x does not depend on the structure of x but only on its fitness. We can therefore define the expected fitness gain from flipping r random bits in x by

$$B(n,d,r) := \mathrm{E}\left(\max\{0, d - d(\mathrm{flip}_r(x))\} \mid \textsc{Om}(x) = n-d\right).$$

To compute $B(n,d,r)$ arithmetically, let us assume that $i \in [0..r]$ is the number of bits flipped from 0 to 1. Then $r-i$ bits have flipped in the opposite direction from 1 to 0, resulting in a progress of $i-(r-i)$. This progress is positive if and only if $i > r-i$, i.e., if and only if $i > r/2$. The probability for i bits flipping in the "good" direction is $\binom{d}{i}\binom{n-d}{r-i}/\binom{n}{r}$. We therefore obtain

$$B(n,d,r) = \sum_{i=\lceil r/2 \rceil}^{r} \frac{\binom{d}{i}\binom{n-d}{r-i}\,(2i-r)}{\binom{n}{r}}.$$

We show that the maximal fitness drift is obtained from flipping a constant number of bits once we have a fitness of at least $(\frac{1}{2}+\varepsilon)n$. The main argument is that flipping a single random bit already gives a better expected fitness gain than flipping many bits, which is due to the fact that when flipping many bits, the strong concentration of the hypergeometric distributions renders it highly unlikely that a fitness gain is obtained at all.

Lemma 3.14. *Let $0 < \varepsilon < 1/2$ and $\alpha := 2\log(4/(\varepsilon^2(1/2-\varepsilon))$. Then for all $n \in \mathbb{N}$, $d \leq (1/2-\varepsilon)n$, and all $r \geq 2\alpha/\varepsilon^2$, we have $B(n,d,r) < B(n,d,1)/2$.*

Proof. Let Z denote the number of "good" flips (i.e., the number of bits flipping from 0 to 1). As discussed in Remark 3.3, the random variable Z follows a hypergeometric distribution with mean value $\mathrm{E}(Z) = dr/n = pr < r/2$, where we abbreviate $p := d/n$. Applying the Chernoff bound presented in Theorem 1.9 (b) in [Doe11] to Z, we obtain

$$\begin{aligned} \Pr\left(Z > r/2\right) = \Pr\left(Z > \mathrm{E}(Z)/(2p)\right) &\leq \left(\frac{e^{1/(2p)-1}}{(1/(2p))^{1/(2p)}}\right)^{\mathrm{E}(Z)} \\ = \left((2p)^{1/(2p)}e^{1/(2p)-1}\right)^{rp} &= \left(\frac{(2pe)^{1/(2p)}}{e}\right)^{rp} = \left(\frac{2pe}{e^{2p}}\right)^{r/2}. \end{aligned} \tag{3.15}$$

We then regard $B(n,d,r)/(B,n,d,1) \le r\Pr(Z > r/2)/(d/n) = (r/p)\left(\frac{2pe}{e^{2p}}\right)^{r/2} = r(2e)^{r/2}p^{r/2-1}e^{-pr}$. We notice that for fixed $r \ge 2\alpha/\varepsilon^2$ and $0 < p < 1/2-\varepsilon$,

$$\begin{aligned}(p^{r/2-1}e^{-pr})' &= (r/2-1)p^{r/2-2}e^{-pr} - rp^{r/2-1}e^{-pr} = p^{r/2-2}e^{-pr}(r/2-1-rp)\\ &\ge p^{r/2-2}e^{-pr}(2\alpha/\varepsilon - 1) > 0,\end{aligned}$$

thus it remains to check the statement for $d = (1/2-\varepsilon)n$. Using the Taylor expansion $e^{2\delta} = 1 + 2\delta + 2\delta^2 + (4/3)\delta^3 + O(\delta^4) < 1 + 2\delta + 3\delta^2$ we see that $\lim_{p\to 1/2-\varepsilon} 2pe/e^{2p} = \lim_{\delta\to\varepsilon} 2(1/2-\delta)e/e^{2(1/2-\delta)} = \lim_{\delta\to\varepsilon}(1-2\delta)e^{2\delta} < \lim_{\delta\to\varepsilon}(1-2\delta)(1+2\delta+3\delta^2) = 1-\varepsilon^2-6\varepsilon^3 < 1-\varepsilon^2$. Therefore we obtain $B(n,d,r) \le r\Pr(Z > r/2) \le r(2pe/e^{2p})^{r/2} < r(1-\varepsilon^2)^{r/2}$. Using the fact that $(1-\varepsilon^2)^{1/\varepsilon^2} < 1/e$, $r \ge 2\alpha/\varepsilon^2 > 2/\varepsilon^2$, and $r(1-\varepsilon^2)^{r/2}$ monotonically decreases when $r > 2/\varepsilon^2$, we obtain $B(n,d,r) < r\exp(-\alpha) \le 2\alpha\exp(-\alpha)/\varepsilon^2$. Since $\log(\alpha\exp(-\alpha)) = \log(\alpha) - \alpha < -\alpha/2$, then $B(n,d,r) < 2\alpha\exp(-\alpha)/\varepsilon^2 < 2\exp(-\alpha/2)/\varepsilon^2 = (2/\varepsilon^2)(\varepsilon^2(1/2-\varepsilon)/4) = (1/2-\varepsilon)/2 = B(n,d,1)/2$ for $d = (1/2-\varepsilon)n$. □

3.3.2 Approximating $B(n,d,r)$ via $A(r, \frac{d}{n}, 1-\frac{d}{n})$

When n and d are large compared to r, the expected progress $B(n,d,r)$ is almost determined by d/n. This inspires the following definition of $A(r,p,q)$ which will have the property that $A(r, \frac{d}{n}, 1-\frac{d}{n})$ is a good approximation of $B(n,d,r)$. The definition for general p and q instead of p and $q = 1-p$ will be useful in the following proofs.

Definition 3.15. *For all $r \in \mathbb{N}$, $p \in [0,1]$, and $q \in [0,1]$, let*

$$A(r,p,q) := \sum_{i=\lceil r/2\rceil}^{r} \binom{r}{i}(2i-r)p^i q^{r-i}. \tag{3.16}$$

The following Theorem 3.16 makes precise how well for $p = d/n$ and $q = 1-p$ the value $A(r,p,q)$ approximates the expected progress $B(n,d,r)$.

Theorem 3.16. *Let $0 < \varepsilon < 1/2$ and $\alpha = 2\log(4/(\varepsilon^2(1/2-\varepsilon)))$ (as in Lemma 3.14). Then for all $n \in \mathbb{N}$ large enough, all $r < 2\alpha/\varepsilon^2$, and all $2r \le d \le (1/2-\varepsilon)n$, we have*

$$\left| A\left(r, \frac{d}{n}, \frac{n-d}{n}\right) - B(n,d,r)\right| < \frac{3r^3}{d}. \tag{3.17}$$

The first step in the proof of Theorem 3.16 is the following statement, which compares suitable A-values with B-values. Note that here we profit from the general definition of $A(r,p,q)$ instead of the special case $A(r,p,1-p)$.

Lemma 3.17. *Consider $n \in \mathbb{N}$ large enough and $1 \le r \le d \le (1/2-\varepsilon)n$ with $0 < \varepsilon < 1/2$. It holds that*

$$A\left(r, \frac{d}{n}, \frac{n-d}{n-r}\right) \ge B(n,d,r) \ge A\left(r, \frac{d-r}{n}, \frac{n-d-r}{n}\right). \tag{3.18}$$

Proof. For any two positive integers r and $i \le r$ we abbreviate

$$(r)_i := r(r-1)\dots(r-i+1) = \prod_{j=0}^{i-1}(r-j).$$

With this notation, we can express $B(n,m,r)$ as

$$B(n,d,r) = \sum_{i=\lceil r/2\rceil}^{r} \frac{\binom{d}{i}\binom{n-d}{r-i}(2i-r)}{\binom{n}{r}} = \sum_{i=\lceil r/2\rceil}^{r} \frac{(d)_i(n-d)_{r-i}}{(n)_r}\binom{r}{i}(2i-r).$$

From the elementary fact that for all $\lceil r/2\rceil \le i \le r$, we have $(n-d)_{r-i} \le (n-d)^{r-i}$ and $(n)_r \ge (n)_i(n-r)^{r-i}$, we obtain

$$\frac{(d)_i(n-d)_{r-i}}{(n)_r} \le \frac{(d)_i(n-d)_{r-i}}{(n)_i(n-r)^{r-i}} \le \left(\frac{d}{n}\right)^i \left(\frac{n-d}{n-r}\right)^{r-i}.$$

This shows $B(n,d,r) \le A\left(r, \frac{d}{n}, \frac{n-d}{n-r}\right)$.

To show the second inequality, we use the fact that for all $i \le r \le n$, we have $(n)_r \le n^r$, $(d)_i \ge (d-r)^i$, and $(n-d)_{r-i} \ge (n-d-r)^{r-i}$. Consequently,

$$\frac{(d)_i(n-d)_{r-i}}{(n)_r} \ge \frac{(d-r)^i(n-d-r)^{r-i}}{n^r} = \left(\frac{d-r}{n}\right)^i \left(\frac{n-d-r}{n}\right)^{r-i},$$

yielding $B(n,d,r) \ge A\left(r, \frac{d-r}{n}, \frac{n-d-r}{n}\right)$. □

With Lemma 3.17 at hand, we now prove Theorem 3.16.

Proof of Theorem 3.16. According to the definition of $A(r,p,q)$ in (3.16) we have

$$\begin{aligned}\frac{\partial A(r,p,q)}{\partial q} &= \sum_{i=\lceil r/2\rceil}^{r} (r-i)\binom{r}{i}(2i-r)p^i q^{r-i-1} \\ &< \frac{r}{2}\sum_{i=\lceil r/2\rceil}^{r}\binom{r}{i}(2i-r)p^i q^{r-i-1} = \frac{rA(r,p,q)}{2q} < \frac{r^2}{2q},\end{aligned}$$

where we have used in the last step that $A(r,p,q) \le r$.

Using that $\frac{n-d}{n} > 1/2$ and $0 < \frac{n-d}{n-r} - \frac{n-d}{n} = \frac{n-d}{n-r}\cdot\frac{r}{n} < (1+o(1))\frac{r}{n}$, we bound

$$A\left(r,\frac{d}{n},\frac{n-d}{n-r}\right) - A\left(r,\frac{d}{n},\frac{n-d}{n}\right) \le \frac{r^2}{2(1/2)}\left(\frac{n-d}{n-r} - \frac{n-d}{n}\right) \le \frac{(1+o(1))r^3}{n}.$$

Similarly we have

$$\begin{aligned}\frac{\partial A(r,p,q)}{\partial p} &= \sum_{i=\lceil r/2\rceil}^{r} i\binom{r}{i}(2i-r)p^{i-1}q^{r-i} \\ &< r\sum_{i=\lceil r/2\rceil}^{r}\binom{r}{i}(2i-r)p^{i-1}q^{r-i} = \frac{rA(r,p,q)}{p} < \frac{r^2}{p}.\end{aligned}$$

Using $d \ge 2r$ we obtain

$$A\left(r,\frac{d}{n},\frac{n-d-r}{n}\right) - A\left(r,\frac{d-r}{n},\frac{n-d-r}{n}\right) \le \frac{r^2}{(d-r)/n}\cdot\frac{r}{n} = \frac{r^3}{d-r} > \frac{2r^3}{d}.$$

Therefore

$$A\left(r,\frac{d}{n},\frac{n-d}{n-r}\right) - A\left(r,\frac{d-r}{n},\frac{n-d-r}{n}\right) = \frac{(1+o(1))r^3}{n} + \frac{2r^3}{d} < \frac{3r^3}{d}.$$

By Lemma 3.17, it suffices to estimate $|A(r, \frac{d}{n}, \frac{n-d}{n}) - B(n,d,r)| < A(r, \frac{d}{n}, \frac{n-d}{n-r}) - A(r, \frac{d-r}{n}, \frac{n-d-r}{n}) < \frac{3r^3}{d}$. □

3.3.3 Approximate Optimal Number of Bits to Flip

The goal of this section is to approximate the function R_{opt} which tells us how many bits one should flip in order to maximize the point-wise drift. Given Theorem 3.16 above, it is tempting to assume that the map $p \mapsto \arg\max_{r \in \mathbb{N}} A(r,p,1-p)$ should do. Analogous to Lemma 3.14 we show in the following lemma that it suffices to regard constant r for the approximated drift.

Lemma 3.18. *Let $0 < \varepsilon < 1/2$ and $\alpha := 2\log(4/(\varepsilon^2(1/2-\varepsilon))$. For all $n \in \mathbb{N}$ with $d \leq (1/2-\varepsilon)n$ and all $r \geq 2\alpha/\varepsilon^2$, the expected approximated drift $A(r, \frac{d}{n}, \frac{n-d}{n}) < A(1, \frac{d}{n}, \frac{n-d}{n})/2$.*

Proof. Consider the binomial random variable $Z \sim \text{Bin}(r, d/n)$. Let $p = d/n$ then $\text{E}(Z) = pr$. Applying the Chernoff bound presented in Theorem 1.9 (b) in [Doe11] to Z, we obtain

$$\Pr\left(Z > r/2\right) = \Pr\left(Z > \text{E}(Z)/(2p)\right) \leq \left(\frac{e^{1/(2p)-1}}{(1/(2p))^{1/(2p)}}\right)^{\text{E}(Z)} = \left(\frac{2pe}{e^{2p}}\right)^{r/2},$$

which is the same inequality as (3.15) in Lemma 3.14. Since $A(r,p,1-p) := \sum_{i=\lceil r/2 \rceil}^{r} \binom{r}{i}(2i-r)p^i(1-p)^{r-i} \leq r\sum_{i=0}^{r} \mathbb{1}_{2i>r}\binom{r}{i}p^i(1-p)^{r-i} = r\Pr(Z > r/2)$ and $A(1,p,1-p) = p$, apply the samle analysis as in Lemma 3.14, the statement holds. □

In the remainder of this section we show that flipping a number r of bits that maximizes $A(r, d/n, 1-d/n)$ yields indeed a good approximation of the best possible expected progress. Since in principle there could be more than one r maximizing $A(r,p,1-p)$ for a given relative distance $p \in (0, 1/2)$, we break ties by preferring smaller values of r. For $p \geq 1/2 - \varepsilon$, where our reasoning above was not applicable, we do not try to find an optimal number of bits to flip, but rather one that does the job of giving a near-optimal runtime. Since a random initial search point has a fitness close to $n/2$, not too much time is spent in this regime anyway. Consequently, we define, for all $\varepsilon > 0$,

$$\tilde{R}_{\text{opt},\varepsilon}(p) := \begin{cases} \min\left\{\arg\max_{r \in \mathbb{N}} A(r,p,1-p)\right\} \text{ for } 0 < p \leq 1/2-\varepsilon, \\ \tilde{R}_{\text{opt}}\left(1/2-\varepsilon\right) \text{ for } 1/2-\varepsilon < p \leq 1/2, \\ n \text{ for } p > 1/2, \end{cases} \tag{3.19}$$

According to Lemma 3.18 the function $\tilde{R}_{\text{opt},\varepsilon}$ is well defined (for all $\varepsilon > 0$).

We prove two important properties of the functions $\tilde{R}_{\text{opt},\varepsilon}$, which are summarized in the following theorem.

Theorem 3.19. *For all $\varepsilon > 0$ the function $\tilde{R}_{\text{opt},\varepsilon}$ is monotonically increasing with respect to p. For all $d \leq n/2$, $\tilde{R}_{\text{opt},\varepsilon}(d/n)$ and $R_{\text{opt}}(d,n)$ are odd values.*

The proof of the second claim in Theorem 3.19 will be carried out in Section 3.3.3. It is purely combinatorial. The proof of the monotonicity of $\tilde{R}_{\text{opt},\varepsilon}$, in contrast, is surprisingly technical. It will be carried out in Section 3.3.3.

R_{opt} and $\tilde{R}_{\text{opt}}$ Attain Only Odd Values

One possibly surprising property of the functions R_{opt} and $\tilde{R}_{\text{opt},\varepsilon}$ is that they take only odd values. That is, regardless of how far we are from the optimum, the maximal drift is obtained for an odd number of bit flips. The following two lemmas show this statement for the approximate and the exact drift, respectively.

Lemma 3.20 (flipping even numbers of bits is sub-optimal, statement for the approximated drift A)**.** *For all $k \in \mathbb{N}$ and $p \in (0,1)$ it holds that $\frac{A(2k,p,1-p)}{2k} = \frac{A(2k+1,p,1-p)}{2k+1}$. Consequently $\tilde{R}_{\text{opt},\varepsilon}(d/n)$ takes odd values for all $d \leq n/2$ and all $\varepsilon > 0$.*

Proof. By definition of the function A in (3.16) and using the facts that for all $r \in \mathbb{N}$ and all $i \leq r$ we have

$$\binom{r}{i} = \binom{r}{r-i}, \ \binom{r}{i} i = r\binom{r-1}{i-1}, \text{ and } \binom{r}{i}(r-i) = r\binom{r-1}{i},$$

we easily see that

$$\begin{aligned} A(r,p,q) &= \sum_{i=\lceil r/2 \rceil}^{r} \binom{r}{i}(2i-r)p^i q^{r-i} \\ &= \sum_{i=\lceil r/2 \rceil}^{r} \left(\binom{r}{i} i - \binom{r}{i}(r-i)\right) p^i q^{r-i} \\ &= r \sum_{i=\lceil r/2 \rceil}^{r} \left(\binom{r-1}{i-1} - \binom{r-1}{i}\right) p^i q^{r-i}. \end{aligned} \tag{3.20}$$

This shows that, for all $k \in \mathbb{N}$,

$$\begin{aligned} \frac{A(2k+1,p,q)}{2k+1} &= \sum_{i=k+1}^{2k+1} \left(\binom{2k}{i-1} - \binom{2k}{i}\right) p^i q^{2k+1-i} \\ &= \sum_{i=k+1}^{2k+1} \left(\binom{2k-1}{i-1} - \binom{2k-1}{i} + \binom{2k-1}{i-2} - \binom{2k-1}{i-1}\right) p^i q^{2k+1-i} \\ &= \sum_{i=k+1}^{2k} \left(\binom{2k-1}{i-1} - \binom{2k-1}{i}\right) p^i q^{2k+1-i} + \sum_{i=k+1}^{2k+1} \left(\binom{2k-1}{i-2} - \binom{2k-1}{i-1}\right) p^i q^{2k+1-i} \\ &= \sum_{i=k+1}^{2k} \left(\binom{2k-1}{i-1} - \binom{2k-1}{i}\right) p^i q^{2k+1-i} + \sum_{i=k}^{2k} \left(\binom{2k-1}{i-1} - \binom{2k-1}{i}\right) p^{i+1} q^{2k-i} \\ &= \sum_{i=k}^{2k} \left(\binom{2k-1}{i-1} - \binom{2k-1}{i}\right) \left(p^i q^{2k+1-i} + p^{i+1} q^{2k-i}\right) - \left(\binom{2k-1}{k-1} - \binom{2k-1}{k}\right) p^k q^{k+1} \\ &= \sum_{i=k}^{2k} \left(\binom{2k-1}{i-1} - \binom{2k-1}{i}\right) p^i q^{2k-i} = \frac{A(2k,p,q)}{2k}, \end{aligned}$$

where we have used in the last step that $p+q=1$ and $\binom{2k-1}{k-1} = \binom{2k-1}{k}$. □

Lemma 3.20 is not an artifact of the approximation of the drift by function A but also holds for the exact drift-maximizing function B. This lemma will not be needed in the following, but we believe it to be interesting in its own right. The reader only interested in the proof of the main results of this work can skip this proof.

Lemma 3.21 (flipping even numbers of bits is sub-optimal, statement for exact drift B)**.** *For all $n, d, k \in \mathbb{N}$ satisfying $0 < d \leq \frac{n}{2}$ and $0 < 2k+1 \leq n$, it holds that $B(n,d,2k) < B(n,d,2k+1)$. Moreover, $\frac{B(n,d,2k)}{2k} = \frac{B(n,d,2k+1)}{2k+1}$ holds.*

Proof. Using again the shorthand $(r)_i := r(r-1)\cdots(r-i+1)$ for all positive integers r and $i \leq r$, we get

$$\begin{aligned} B(n,d,2k+1) &= \frac{1}{\binom{n}{2k+1}} \sum_{i=k+1}^{2k+1} \binom{d}{i}\binom{n-d}{2k-i+1}(2i-2k-1) \\ &= \frac{1}{\binom{n}{2k+1}} \sum_{i=0}^{k} \binom{d}{i+k+1}\binom{n-d}{k-i}(2i+1) \\ &= \frac{(n-2k-1)!}{n!} \sum_{i=0}^{k} \binom{2k+1}{k-i}(d)_{i+k+1}(n-d)_{k-i}(2i+1). \end{aligned}$$

Similarly, we obtain

$$\begin{aligned} B(n,d,2k) &= \frac{1}{\binom{n}{2k}} \sum_{i=k+1}^{2k} \binom{d}{i}\binom{n-d}{2k-i}(2i-2k) \\ &= \frac{1}{\binom{n}{2k}} \sum_{i=0}^{k-1} \binom{d}{i+k+1}\binom{n-d}{k-i-1}(2i+2) \\ &= \frac{(n-2k)!}{n!} \sum_{i=0}^{k-1} \binom{2k}{k-i-1}(d)_{i+k+1}(n-d)_{k-i-1}(2i+2). \end{aligned}$$

Therefore, the ratio of $B(n,d,2k)$ and $B(n,d,2k+1)$ is

$$\begin{aligned} \frac{B(n,d,2k)}{B(n,d,2k+1)} &= \frac{(n-2k)\sum_{i=0}^{k-1}\binom{2k}{k-i-1}(d)_{i+k+1}(n-d)_{k-i-1}(2i+2)}{\sum_{i=0}^{k}\binom{2k+1}{k-i}(d)_{i+k+1}(n-d)_{k-i}(2i+1)} \\ &= \frac{(n-2k)\sum_{i=0}^{k-1}\binom{2k}{k-i-1}(d-k-1)_i(n-d)_{k-i-1}(2i+2)}{\sum_{i=0}^{k}\binom{2k+1}{k-i}(d-k-1)_i(n-d)_{k-i}(2i+1)}. \end{aligned}$$

Replacing $(d-k)$ with u and $(n-d)$ with v gives

$$\frac{B(n,d,2k)}{B(n,d,2k+1)} = \frac{(u+v-k)\sum_{i=0}^{k-1}\binom{2k}{k-i-1}(u-1)_i(v)_{k-i-1}(2i+2)}{\sum_{i=0}^{k}\binom{2k+1}{k-i}(u-1)_i(v)_{k-i}(2i+1)}.$$

Using the shorthand λ_i^j for the coefficient of r^j in the polynomial $(r)_i$, we now take a close look at the denominator, which is a polynomial in u and v. It is not difficult to see that for all $a, b \in \mathbb{N} \cup \{0\}$, the coefficient of the term $u^a v^b$ in the denominator equals

$$\psi(a,b) = \sum_{i=0}^{k} \binom{2k+1}{k-i} \lambda_{i+1}^{a+1} \lambda_{k-i}^{b} (2i+1),$$

while the coefficient of term $u^a v^b$ in numerator equals

$$\phi(a,b) = \sum_{i=0}^{k-1} \binom{2k}{k-i-1} \left(\lambda_{i+1}^{a}\lambda_{k-i-1}^{b} + \lambda_{i+1}^{a+1}\lambda_{k-i-1}^{b-1} - k\lambda_{i+1}^{a+1}\lambda_{k-i-1}^{b}\right)(2i+2).$$

Since $(r)_{i+1} = (r-i)(r)_i$, it is easily verified that

$$\lambda_i^j - i\lambda_i^{j+1} = \lambda_{i+1}^{j+1}. \tag{3.21}$$

We use (3.21) to simplify $\phi(a,b)$ in the following way.

$$\begin{aligned}
\phi(a,b) &= \sum_{i=0}^{k-1} \binom{2k}{k-i-1} \left(\lambda_{i+2}^{a+1}\lambda_{k-i-1}^{b} + \lambda_{i+1}^{a+1}\lambda_{k-i}^{b}\right)(2i+2) \\
&= \sum_{i=0}^{k} \left(\binom{2k}{k-i-1}(2i+2) + \binom{2k}{k-i}(2i)\right)\lambda_{i+1}^{a+1}\lambda_{k-i}^{b} \\
&= \sum_{i=0}^{k} \left((2i+2) + \frac{k+i+1}{k-i}(2i)\right)\binom{2k}{k-i-1}\lambda_{i+1}^{a+1}\lambda_{k-i}^{b} \\
&= \sum_{i=0}^{k} \frac{2k(2i+1)}{k-i}\binom{2k}{k-i-1}\lambda_{i+1}^{a+1}\lambda_{k-i}^{b} \\
&= \frac{2k}{2k+1}\sum_{i=0}^{k}\binom{2k+1}{k-i}\lambda_{i+1}^{a+1}\lambda_{k-i}^{b}(2i+1) \\
&= \frac{2k}{2k+1}\psi(a,b).
\end{aligned}$$

The above holds for all $0 \le a, b \le k$, showing that indeed

$$\frac{B(n,d,2k)}{B(n,d,2k+1)} = \frac{2k}{2k+1}.$$

□

Monotonicity of $\tilde{R}_{\mathrm{opt},\varepsilon}$

We now argue that, for all $\varepsilon > 0$, the function $\tilde{R}_{\mathrm{opt},\varepsilon}$ is monotone. It seems quite intuitive that the optimal number of bit flips should decrease with decreasing distance to the optimum, and this has been previously observed empirically, e.g., in [Bäc92; Fia+08; Fia+09]. However, formally proving the desired monotonic relationship requires substantial technical work. We note that, as a side result, Lemma 3.26 shows that for search points having a distance of less than $n/3$ to the optimum (or its complement), the maximal approximated fitness gain is obtained by 1-bit flips.

Lemma 3.22 (and definition of cut-off points)**.** *For any two integers $0 \le k_1 < k_2$, the functions $p \mapsto A(2k_1+1, p, 1-p)$ and $p \mapsto A(2k_2+1, p, 1-p)$ intersect exactly once in the interval $(0, 1/2]$. Denoting this intersection p_0 and letting $A_0 := A(2k_1+1, p_0, 1-p_0)$, we call (p_0, A_0) the* **cut-off point** *of $A(2k_1+1, p, 1-p)$ and $A(2k_2+1, p, 1-p)$.*

We have $A(2k_1+1, p, 1-p) > A(2k_2+1, p, 1-p)$ if and only if $0 < p < p_0$.

The graph in Figure 3.1 illustrates the functions $p \mapsto A(k, p, 1-p)$ for $k = 1, 3, 5, 7$. The precise cut-off points will be computed numerically in Section 3.4.2.

In order to prove Lemma 3.22, we first show the following combinatorial lemma.

Lemma 3.23. *For all $k, r \in \mathbb{N} \cup \{0\}$ and $0 \le q \le 1$ it holds that*

$$\sum_{i=0}^{k}\binom{k+r+1}{i}(1-q)^{k-i}q^i = \sum_{i=0}^{k}\binom{i+r}{r}q^i. \tag{3.22}$$

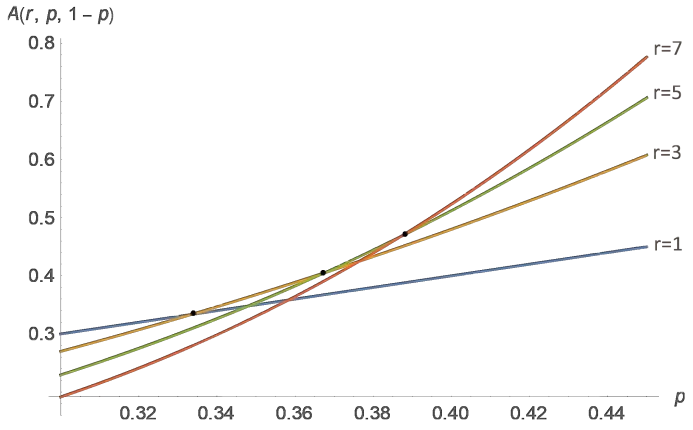

FIGURE 3.1: The approximated drift $A(k,p,1-p)$ for $k=1,3,5,7$. By Lemma 3.26 the function $r \mapsto A(r,p,1-p)$ is maximized for $r=1$ whenever $p < 1/3$.

Proof. We prove the equation by induction. It is obvious that equation (3.22) holds for all $r \geq 0$ and $k = 0$. Assume that it holds for some pair of integers $(k, r+1)$, then the following computation shows that it also holds for $(k+1, r)$.

$$\begin{aligned}
&\sum_{i=0}^{k+1} \binom{k+1+r+1}{i} (1-q)^{k-i+1} q^i \\
= &\binom{k+r+2}{k+1} q^{k+1} + (1-q) \sum_{i=0}^{k} \binom{k+(r+1)+1}{i} (1-q)^{k-i} q^i \\
= &\binom{k+r+2}{r+1} q^{k+1} + (1-q) \sum_{i=0}^{k} \binom{i+r+1}{r+1} q^i \\
= &\sum_{i=0}^{k+1} \binom{i+r}{r} q^i.
\end{aligned}$$

For arbitrary combinations of k and r, we thus get the desired correctness of (3.22) for the pair (k,r) inductively from that of the pair $(0, r+k)$. □

We use Lemma 3.23 to compute the second derivative of $A(r,p,q)$ for q in Lemma 3.24 and then obtain the second derivative of $A(r,p,q)$ for p in Lemma 3.25. We first notice that $A(1,p,1-p) = p$ and $\mathrm{d}A(1,p,1-p)/\mathrm{d}p = 1$. Thus we only look at the second derivative for $r > 1$.

Lemma 3.24. *For all $k \in \mathbb{N}$ and all $0 < p \leq 1/2$, it holds that*

$$\frac{\mathrm{d}^2 A(2k+1,p,1-p)}{(\mathrm{d}(1-p))^2} = c_k p^{k-1} (1-p)^{k-1}, \tag{3.23}$$

where c_k is a constant related to k via

$$c_k := 2(2k-1)(2k+1)\binom{2k-2}{k-1} = \frac{4k+2}{\beta(k,k)},$$

and $\beta(x,y) := \int_0^1 t^{x-1}(1-t)^{y-1} dt = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$ is the well-known beta function.

Proof. Set $q := 1-p$. We expand $A(2k+1,p,q)$ according to Equation (3.20) and use Lemma 3.23 to obtain the following

$$\begin{aligned}
A(2k+1,p,q) &= (2k+1)\sum_{i=k}^{2k}\left(\binom{2k}{i}-\binom{2k}{i+1}\right)p^{i+1}q^{2k-i}\\
&= (2k+1)\sum_{i=0}^{k}\left(\binom{2k}{k+i}-\binom{2k}{k+i+1}\right)p^{k+i+1}q^{k-i}\\
&= (2k+1)p^{k+1}\cdot\sum_{i=0}^{k}\left(\binom{2k}{i}-\binom{2k}{i-1}\right)p^{k-i}q^{i}\\
&= (2k+1)p^{k+1}\left[\sum_{i=0}^{k}\binom{k+(k-1)+1}{i}p^{k-i}q^{i}+q\sum_{i=0}^{k-1}\binom{(k-1)+k+1}{i}p^{k-1-i}q^{i}\right]\\
&= (2k+1)p^{k+1}\left[\sum_{i=0}^{k}\binom{i+(k-1)}{i}q^{i}+q\sum_{i=0}^{k-1}\binom{i+k}{i}q^{i}\right]\\
&= (2k+1)p^{k+1}\cdot\sum_{i=0}^{k}\left(\binom{k+i-1}{i}-\binom{k+i-1}{i-1}\right)q^{i}.
\end{aligned}$$

We extract the term p^{k+1} in the above equation and define the polynomial $f_k(q) := \sum_{i=0}^{\infty} a_k^i q^i$ with coefficient $a_k^i = 0$ when $i > k$ and

$$a_k^i := \binom{k+i-1}{i}-\binom{k+i-1}{i-1} = \binom{k+i-1}{i}\frac{k-i}{k} \text{ when } i \le k. \quad (3.24)$$

Then $A(2k+1,p,q) = (2k+1)p^{k+1}f_k(q)$. We use the general Leibniz rule for the second derivative (informally, this rule states that $(fg)'' = f''g + 2f'g' + fg''$) and obtain

$$\frac{\mathrm{d}^2A(2k+1,p,q)}{\mathrm{d}^2q} = (2k+1)p^{k-1}\cdot\left(p^2f_k''(q)-2(k+1)pf_k'(q)+(k+1)kf_k(q)\right).$$

It remains to prove that

$$p^2f_k''(q)-2(k+1)pf_k'(q)+(k+1)kf_k(q) = 2(2k-1)\binom{2k-2}{k-1}q^{k-1}. \quad (3.25)$$

We look at the coefficient of q^i in the left part of equation (3.25) and we denote it by c_k^i. By expanding $f_k(q)$ into a polynomial and replacing p by $1-q$, we see that c_k^i equals the coefficient of q^i in the following expression

$$\begin{aligned}
&(1-q)^2(a_k^iq^i+a_k^{i+1}q^{i+1}+a_k^{i+2}q^{i+2})''\\
&-2(k+1)(1-q)(a_k^iq^i+a_k^{i+1}q^{i+1})'\\
&+(k+1)k(a_k^iq^i).
\end{aligned}$$

This shows that c_k^i is equal to

$$\begin{aligned}
& a_k^i\left((k+1)k+2(k+1)i+i(i-1)\right)\\
+\;& a_k^{i+1}\left(-2(k+1)(i+1)-2(i+1)i\right)\\
+\;& a_k^{i+2}\left((i+2)(i+1)\right).
\end{aligned}$$

According to (3.24) the coefficient a_k^i satisfies

$$a_k^{i+1} = a_k^i + a_{k-1}^{i+1}, \text{ and} \tag{3.26}$$

$$a_{k-1}^{i+1} = a_k^i \cdot \frac{k}{i+1} \cdot \frac{k-i-2}{k-i}, \text{ for } k > i. \tag{3.27}$$

We use Equation (3.26) to rewrite the expression of c_k^i for $i \leq k-2$, and then use Equation (3.27) to simplify the equation in the following way

$$\begin{aligned} c_k^i &= a_k^i\,(k(k-1)) + a_{k-1}^{i+1}\,(-2(k-1)(i+1)) + a_{k-2}^{i+2}\,((i+2)(i+1)) \\ &= a_k^i k(k-1) - 2a_k^i k(k-1)\frac{k-i-2}{k-i} + a_k^i k(k-1)\frac{k-i-4}{k-i} \\ &= 0. \end{aligned}$$

Noticing that $a_k^k = 0$ shows that $f_k(q)$ has a degree of $k-1$. This implies that the term q^{k-1} has the highest degree in (3.25). Its coefficient is

$$\begin{aligned} c_k^{k-1} &= a_k^{k-1}\,((k+1)k + 2(k+1)(k-1) + (k-1)(k-2)) \\ &= 2(2k-1)\binom{2k-2}{k-1}. \end{aligned}$$

This proves the claimed equality in (3.23). □

Lemma 3.25. *For all $k \in \mathbb{N}$ and all $0 < p \leq 1/2$, it holds that*

$$\begin{aligned} \frac{\mathrm{d}^2 A(2k+1,p,1-p)}{\mathrm{d}p^2} &= c_k p^{k-1}(1-p)^{k-1}, \\ \frac{\mathrm{d}A(2k+1,p,1-p)}{\mathrm{d}p} &= c_k \int_0^p x^{k-1}(1-x)^{k-1}\mathrm{d}x \text{ and} \\ A(2k+1,p,1-p) &= c_k \int_0^p \int_0^y x^{k-1}(1-x)^{k-1}\mathrm{d}x\mathrm{d}y. \end{aligned}$$

Furthermore for all $k \in \mathbb{N}_0$ we can write

$$A(2k+1,p,1-p) = \int_0^p \frac{\mathrm{d}A(2k+1,x,1-x)}{\mathrm{d}p}\mathrm{d}x.$$

Proof. The first equality can be easily obtained from the equality in (3.23). Consequently,

$$\begin{aligned} \frac{\mathrm{d}A(2k+1,p,q)}{\mathrm{d}p} &= c_k \int_0^p x^{k-1}(1-x)^{k-1}\mathrm{d}x + C_1 \text{ with } C_1 \in \mathbb{R}, \\ A(2k+1,p,q) &= c_k \int_0^p \int_0^y x^{k-1}(1-x)^{k-1}\mathrm{d}x\mathrm{d}y + C_1 p + C_2 \text{ with } C_2 \in \mathbb{R}. \end{aligned}$$

Using the fact that $\lim_{p\to 0} A(2k+1,p,q) = o(p)$ for $k \geq 1$, we obtain $C_1 = C_2 = 0$ as claimed.

For the last statement, we only need to consider the case $k = 0$. Recalling that $A(1,p,1-p) = p$ and $\mathrm{d}A(1,p,1-p)/\mathrm{d}p = 1$ shows that the equality also applies to this case. □

We next prove Lemma 3.22.

Proof of Lemma 3.22. Using the notation from Lemma 3.24, we first notice that for all $k > 0$ we have

$$\frac{c_{k+1}}{c_k} = \frac{2(2k+1)(2k+3)\binom{2k}{k}}{2(2k-1)(2k+1)\binom{2k-2}{k-1}} = \frac{4k+6}{k} > 4. \tag{3.28}$$

Let $0 < k_1 < k_2$. By the above, we have $4 < (4 + 6/k_2)^{k_2-k_1} < c_{k_2}/c_{k_1} < (4 + 6/k_1)^{k_2-k_1}$. Notice that $c_{k_1}(pq)^{k_1-1} - c_{k_2}(pq)^{k_2-1} = (pq)^{k_1-1}(c_{k_1} - c_{k_2}(pq)^{k_2-k_1})$. We now use the fact that $\lim_{p\to 0} pq = 0$ and $\lim_{p\to 1/2} pq = 1/4$ to obtain that for all $k_2 > k_1 > 0$,

$$\lim_{p\to 0}(c_{k_1} - c_{k_2}(pq)^{k_2-k_1}) > 0 \text{ while } \lim_{p\to 1/2}\left(c_{k_1} - c_{k_2}(pq)^{k_2-k_1}\right) < 0.$$

This shows that the function $p \mapsto c_{k_1}(pq)^{k_1-1}$ intersects with $p \mapsto c_{k_2}(pq)^{k_2-1}$ in at most one point $p_I \in (0, 1/2)$. Moreover, we have that $c_{k_1}(pq)^{k_1-1} \geq c_{k_2}(pq)^{k_2-1}$ if and only if $p \in [0, p_I]$. Therefore, when $p > 0$, the function $p \mapsto \int_0^p c_{k_1}(x - x^2)^{k_1-1}\mathrm{d}x$ intersects with the function $p \mapsto \int_0^p c_{k_2}(x - x^2)^{k_2-1}\mathrm{d}x$ at most once for $p \in (0, 1/2)$. We now prove that the intersection exists.

Notice that for all $k > 1$ we have

$$c_k \int_0^{0.5} (x - x^2)^{k-1}\mathrm{d}x = \frac{c_k}{2}\int_0^1 (x - x^2)^{k-1}\mathrm{d}x = \frac{c_k}{2}\beta(k, k) = 2k + 1,$$

and thus

$$\lim_{p\to 1/2}\left(\int_0^p c_{k_1}(x - x^2)^{k_1-1}\mathrm{d}x - \int_0^p c_{k_2}(x - x^2)^{k_2-1}\mathrm{d}x\right) = 2(k_1 - k_2) < 0,$$

while

$$\int_0^p c_{k_1}(x - x^2)^{k_1-1}\mathrm{d}x > \int_0^p c_{k_2}(x - x^2)^{k_2-1}\mathrm{d}x \text{ for all } p \in (0, p_I).$$

There exists a intersection point $p_{II} \in (0, 1/2)$ such that

$$\int_0^p c_{k_1}(x - x^2)^{k_1-1}\mathrm{d}x \geq \int_0^p c_{k_2}(x - x^2)^{k_2-1}\mathrm{d}x \text{ if and only if } p \in [0, p_{II}].$$

This shows that for all $k_2 > k_1 > 0$ there exists a point $p_{II} \in (0, 1/2)$ such that

$$\frac{\mathrm{d}A(2k_1 + 1, p, q)}{\mathrm{d}p} \geq \frac{\mathrm{d}A(2k_2 + 1, p, q)}{\mathrm{d}p} \text{ if and only if } p \in [0, p_{II}]. \tag{3.29}$$

To extend the conclusion to $k_1 = 0$, let $k_2 > k_1 = 0$. We have $\lim_{p\to 1/2}\int_0^p c_{k_2}(x - x^2)^{k_2-1}dx = 2k_2 + 1$ and $\lim_{p\to 0}\int_0^p c_{k_2}(x - x^2)^{k_2-1}dx = 0$, while $\mathrm{d}A(1, p, q)/\mathrm{d}p = 1$. Therefore the intersection point p_{II} still exists and there is a unique such point.

As a result we see that the function $\int_0^p \mathrm{d}A(2k_1 + 1, x, 1 - x)$ intersects with the function $\int_0^p \mathrm{d}A(2k_2 + 1, x, 1 - x)$ at most once for $p \in (0, 1/2)$ and

$$\int_0^p \mathrm{d}A(2k_1 + 1, x, q) > \int_0^p \mathrm{d}A(2k_2 + 1, x, q) \text{ for all } p \in (0, p_{II}) \text{ while}$$
$$\lim_{p\to 1/2}(A(2k_1 + 1, p, q) - A(2k_2 + 1, p, q)) < 0.$$

This shows that $A(2k_1 + 1, p, q)$ intersects with $A(2k_2 + 1, p, q)$ exactly once at some value $p_{III} < 1/2$.

□

We are now ready to prove the monotonicity of $\tilde{R}_{\text{opt},\varepsilon}$.

Proof of the first part of Theorem 3.19. Let $\varepsilon > 0$, let $p_0 \in (0,1)$, and set $q_0 := 1 - p_0$. By Lemma 3.20 it holds that $A(2k, p_0, q_0) < A(2k+1, p_0, q_0)$. This shows that $\tilde{R}_{\text{opt},\varepsilon}(p_0)$ is odd. Let $k \in \mathbb{N} \cup \{0\}$ such that $\tilde{R}_{\text{opt},\varepsilon}(p_0) = 2k+1$. By definition of $\tilde{R}_{\text{opt},\varepsilon}$ (cf. equation (3.19)), k is the smallest integer obtaining a drift of $A(2k+1, p_0, q_0)$. For all integers $k' < k$ we thus obtain

$$A(2k+1, p_0, q_0) > A(2k'+1, p_0, q_0). \tag{3.30}$$

By Lemma 3.22 we also get that for all $p > p_0$ it holds that

$$A(2k+1, p, q) > A(2k'+1, p, q). \tag{3.31}$$

Therefore $\tilde{R}_{\text{opt},\varepsilon}(p) \geq 2k+1$ for all $p > p_0$. Since the statement holds for all $p_0 \in (0, 1/2 - \varepsilon]$ we obtain the monotonicity of $\tilde{R}_{\text{opt},\varepsilon}$. □

$\tilde{R}_{\text{opt},\varepsilon}(p) = 1$ when $0 < p < 1/3$ and $R_{\text{opt}}(d,n) = 1$ when $0 < d = o(n)$

We first show that flipping one bit is optimal for the approximated drift when the distance to the optimal solution is less than $n/3$.

Lemma 3.26. *For all $\varepsilon > 0$ and all $0 < p < 1/3$ it holds that $\tilde{R}_{\text{opt},\varepsilon}(p) = 1$.*

Proof. Let $\varepsilon > 0$. Due to the monotonicity of $\tilde{R}_{\text{opt},\varepsilon}$, it suffices to show that $\tilde{R}_{\text{opt},\varepsilon}(1/3) = 1$. By Lemma 3.20 we only need to consider odd values of r. According to Lemma 3.25 if the second derivative $c_k x^{k-1}(1-x)^{k-1} > c_{k+1} x^k (1-x)^k$ for all $x \in (0, 1/3)$ then $A(2k-1, 1/3, 2/3) > A(2k+1, 1/3, 2/3)$ and thus $\tilde{R}_{\text{opt},\varepsilon} \neq 2k+1$. We notice that

$$\frac{c_{k+1}\, x^k (1-x)^k}{c_k\, x^{k-1}(1-x)^{k-1}} = \frac{4k+6}{k} x(1-x) \leq \frac{4k+6}{k} \cdot \frac{2}{9} \text{ for all } 0 < x < \frac{1}{3}.$$

For all $k > 12$ it holds that $(4k+6)/k < 9/2$, which implies that $c_k x^{k-1}(1-x)^{k-1} > c_{k+1} x^k (1-x)^k$. We therefore obtain that $\tilde{R}_{\text{opt},\varepsilon}(1/3) \leq 25$. For the remaining values, i.e., for $r = 1, 3, \ldots, 25$, we can compute $A(r, 1/3, 2/3)$ numerically. This numerical evaluation shows that the maximum value $1/3$ is obtained (only) by $r = 1$ and $r = 3$. This proves $\tilde{R}_{\text{opt},\varepsilon}(1/3) = 1$. □

We show in Lemma 3.27 that flipping one bit is also optimal for the exact fitness drift when the distance is a lower-order term of n.

Lemma 3.27. *For all $0 < d = o(n)$ it holds that $R_{\text{opt}}(d,n) = 1$.*

Proof. Since $d < n/4$, Lemma 3.14 yields with $\varepsilon := 1/4$ that $R_{\text{opt}}(d,n) < 4^4 \log(4)$. Referring to Lemma 3.17, we obtain for all $3 \leq r \leq 4^4 \log(4)$ that $B(n,d,r) < A(r, d/n, 1) = \Theta((d/n)^{(r+1)/2}) = o(d/n)$. Since R_{opt} attains only odd values, we obtain $R_{\text{opt}}(d,n) = 1$ for all $0 < d = o(n)$. □

3.3.4 Runtime Loss From Using the Approximated Drift

We show in this section that the expected runtimes of the exact and the approximate drift maximizer do not differ substantially. More precisely, we show that also

the approximate drift maximizer also obtains an expected runtime on OneMax that is very close to that of an optimal unary unbiased black-box algorithm, cf. Corollary 3.29. To make things precise, we denote for every $\varepsilon > 0$ by $\tilde{A}^*_\varepsilon$ the algorithm which we obtain from Algorithm 6 by replacing the mutation rate $R(\textsc{Om}(x))$ by $\tilde{R}_{\text{opt},\varepsilon}(1 - \textsc{Om}(x)/n)$.

To state the main result, for all $\varepsilon > 0$, for all $n \in \mathbb{N}$, and all $0 < p \leq 1/2 - \varepsilon$ we abbreviate

$$A_{\max,\varepsilon}(p) := A(\tilde{R}_{\text{opt},\varepsilon}(p), p, 1-p) \text{ and } B_{\max}(p,n) := B(n, \lfloor pn \rfloor, R_{\text{opt}}(\lfloor pn \rfloor, n)). \tag{3.32}$$

We notice from Lemma 3.14 and Lemma 3.18 that $\tilde{R}_{\text{opt},\varepsilon}(1/2-\varepsilon) = \Theta(1)$ and $R_{\text{opt}}(\lfloor (1/2-\varepsilon)n \rfloor, n) = \Theta(1)$. Considering the drift of single bit flip, we see that $A_{\max,\varepsilon}(1/2-\varepsilon) \geq 1/2 - \varepsilon$. According to the definition of $\tilde{h}$ in equation (3.5), we have $B_{\max}(d/n, n) = \tilde{h}(d) \geq d/n$ for all $0 < d \leq (1/2-\varepsilon)n$.

Theorem 3.28. *For all constant $0 < \varepsilon < 1/2$ the expected runtime of algorithm $\tilde{A}^*_\varepsilon$ on* OneMax *satisfies*

$$E\left(T_{\tilde{A}^*_\varepsilon}\right) \leq \sum_{x=1}^{(1/2-\varepsilon)n} \frac{1}{h(x)} + \frac{\varepsilon n}{A_{\max,\varepsilon}(1/2-\varepsilon)} + o(n).$$

Moreover,

$$E\left(T_{\tilde{A}^*_\varepsilon}\right) \leq \sum_{x=1}^{(1/2-\varepsilon)n} \frac{1}{A_{\max,\varepsilon}(x/n)} + \frac{\varepsilon n}{A_{\max,\varepsilon}(1/2-\varepsilon)} + o(n).$$

Proof. Let constant $0 < \varepsilon < 1/2$. It is easily seen from Theorem 2.4 and from the definition of $\tilde{R}_{\text{opt},\varepsilon}$ in (3.19) that

$$E\left(T_{\tilde{A}^*_\varepsilon} \mid x(0)\right) \leq \sum_{x=1}^{n-\textsc{Om}(x(0))} \frac{1}{B(n,x,\tilde{R}_{\text{opt},\varepsilon}(x/n))} \leq \sum_{x=1}^{n/2} \frac{1}{B(n,x,\tilde{R}_{\text{opt},\varepsilon}(x/n))} + 1.$$

To ease representation, let $r_A(x) := \tilde{R}_{\text{opt},\varepsilon}(x/n)$ and $r_B(x) := R_{\text{opt}}(x,n)$ for all $0 < x \leq (1/2-\varepsilon)n$. According to Theorem 3.16 we have

$$\left| A\left(r_A(x), \frac{x}{n}, 1-\frac{x}{n}\right) - B\left(n,x,r_A(x)\right)\right| = O(1/x) \text{ and}$$
$$\left| A\left(r_B(x), \frac{x}{n}, 1-\frac{x}{n}\right) - B(n,x,r_B(x))\right| = O(1/x).$$

Since $A(r_B(x), x/n, 1-x/n) \leq A(r_A(x), x/n, (n-x)/n) = A_{\max,\varepsilon}(x/n)$ and $B(n,x,r_A(x)) \leq B(n,x,r_B(x)) = B_{\max}(x/n,n) = \tilde{h}(x)$, we obtain from Theorem 3.16 that, for all $0 < x < (1/2-\varepsilon)n$,

$$\left|B(n,x,r_A(x)) - \tilde{h}(x)\right| = O(1/x) \text{ and}$$
$$|A_{\max,\varepsilon}(x/n) - B(n,x,r_A(x))| = O(1/x),$$

where we use the fact that $r_A(x) = \Theta(1)$ according to Lemma 3.18. Referring to Lemma 3.26 and Lemma 3.27, we have $r_B(x) = r_A(x) = 1$ when $0 < x = o(n)$. Therefore,

$$A_{\max,\varepsilon}(x/n) = B(n,x,r_A(x)) = \tilde{h}(x) \text{ for all } 0 < x = o(n).$$

Notice that $h(x) \geq B(n,x,1) = x/n$ for all $x > 0$. Using the fact that $0 < h(x) - \tilde{h}(x) \leq n\exp(-\Omega(n^{0.2}))$ for $0 < x < n/2 - n^{0.6}$ in Lemma 3.10, we obtain

$$\left|\frac{1}{B(n,x,r_A(x))} - \frac{1}{h(x)}\right| = \frac{|h(x) - B(n,x,r_A(x))|}{B(n,x,r_A(x))h(x)} \leq \begin{cases} o(1/n) \text{ for } 0 < x \leq n^{0.99}, \\ O((1/x)/(x/n)^2) \text{ for } n^{0.99} < x \leq (1/2-\varepsilon)n. \end{cases}.$$

Referring to Lemma 3.12 and the definition of $B(n,x,r)$ we see that, for all fixed n and r, the fitness drift $B(n,x,r)$ monotonically increases with respect to x. Therefore, for all $x > (1/2-\varepsilon)n$, we have $r_A(x) = r_A((1/2-\varepsilon)n)$ and $B(n,x,r_A(x)) \geq B(n,(1/2-\varepsilon)n, r_A((1/2-\varepsilon)n)) \geq A_{\max,\varepsilon}(1/2-\varepsilon) - O(1/n) \geq 1/2 - \varepsilon - O(1/n) = \Omega(1)$, thus

$$\mathrm{E}(T_{\tilde{A}^*_\varepsilon}) \leq \sum_{x=1}^{(1/2-\varepsilon)n} \frac{1}{h(x)} + O(1) + O(n^{0.03}) + \frac{\varepsilon n}{A_{\max,\varepsilon}(1/2-\varepsilon)} + o(n).$$

This proves the first statement.

The second statement can be shown by using similar methods to bound the absolute difference $|1/B(n,x,r_A(x)) - 1/A_{\max,\varepsilon}(x/n)|$ for $0 < x \leq n/2$. □

Corollary 3.29. *For all constant $0 < \varepsilon < 1/2$ the difference between the expected runtime of $\tilde{A}^*_\varepsilon$ on* OneMax *and that of an optimal unary unbiased black-box algorithm is $O(\varepsilon n)$. Furthermore, the absolute difference between the expected runtimes of A^* and $\tilde{A}^*_\varepsilon$ is also $O(\varepsilon n)$.*

Proof. Let constant $0 < \varepsilon < 1/2$ and let A be an arbitrary unary unbiased black-box algorithm. By Theorems 3.8 and 3.28 it holds that

$$\begin{aligned} E(T_A) &\geq \sum_{x=1}^{n/2-n^{0.6}} \frac{1}{h(x)} - \Theta(n^{2/3}\ln^9(n)) \\ &\geq \sum_{x=1}^{(1/2-\varepsilon)n} \frac{1}{h(x)} - o(n) \\ &\geq E\left(T_{\tilde{A}^*_\varepsilon}\right) - O(\varepsilon n). \end{aligned}$$

The second statement is a direct consequence of the first and Theorem 3.5. □

3.4 Runtime Analysis for the Approximate Drift-Maximizer $\tilde{A}^*_\varepsilon$

We compute in this section the expected time needed by Algorithm $\tilde{A}^*_\varepsilon$ to optimize OneMax. We fix $0 < \varepsilon < 1/2$. As proven in Lemma 3.26 algorithm $\tilde{A}^*_\varepsilon$ flips $\tilde{R}_{\mathrm{opt},\varepsilon}(p) = 1$ bit whenever $0 < p < 1/3$. In this regime $\tilde{A}^*_\varepsilon$ is thus equal to RLS. It is well known (and easy to prove by a simple fitness-level argument) that the expected time needed by RLS starting in a search point of OneMax value $2n/3$ to reach the all-ones string equals $n\sum_{i=1}^{n/3} 1/i = nH_{n/3} = n(\ln(n/3)+\gamma) + 3/2 + O(1/n)$, where $\gamma \approx 0.57721\ldots$ denotes again the Euler–Mascheroni constant. It therefore remains to compute the time needed by $\tilde{A}^*_\varepsilon$ to reach for the first time a search point having fitness at least $2n/3$.

Formally, we also need to show that the first search point having fitness at least $2n/3$ does not have a fitness value that is much larger than this. Since we flip a constant number of bits only, we get this statement for free. Note also that it is shown below that in the interval before reaching this fitness level the algorithm flips only 3 bits. Apart from this situation around fitness layer $2n/3$ we do not have to take care of jumping several fitness layers by hand, but this is taken into account already in the drift theorems from which we derive our runtime estimates.

3.4.1 Drift Analysis

As we did in Section 3.2, we employ the variable drift theorems, Theorems 2.4 and 2.7, to compute upper and lower bounds for the expected runtime of algorithm $\tilde{A}^*_\varepsilon$. We will provide a numerical evaluation of these expressions in Section 3.4.2.

Lower bound. We first compute a lower bound for the expected runtime $\mathrm{E}(T_A)$ of any unary unbiased black-box algorithm A on OneMax. According to Theorem 3.8 and using a similar method to estimate $|1/A_{\max,\varepsilon}(x/n) - 1/h(x)| = o(1)$ as in Theorem 3.28 for $0 < x \leq (1/2-\varepsilon)n$, we obtain that

$$\begin{aligned}\mathrm{E}\left(T_A\right) &\geq \sum_{x=1}^{n/2-n^{0.6}} \frac{1}{h(x)} - \Theta(n^{2/3}\ln^9(n)) \geq \sum_{x=1}^{(1/2-\varepsilon)n} \frac{1}{A_{\max}(x/n)} - o(n) \\ &= nH_{n/3} + \sum_{x=\lfloor n/3\rfloor}^{(1/2-\varepsilon)n} \frac{1}{A_{\max}(x/n)} - o(n).\end{aligned}$$

Let $k \in \mathbb{N}$ and let $1/2-\varepsilon =: p_0 > p_1 > \ldots > p_k > 1/3$. Using the fact that $A_{\max,\varepsilon}$ is increasing, we bound $\mathrm{E}(T_A)$ by

$$\mathrm{E}\left(T_A\right) \geq n\left(\ln\left(\frac{n}{3}\right) + \gamma + \sum_{i=1}^{k} \frac{p_{i-1}-p_i}{A_{\max,\varepsilon}(p_{i-1})} + \int_{1/3}^{p_k} \frac{\mathrm{d}p}{A_{\max,\varepsilon}(p)}\right) - o(n). \tag{3.33}$$

Upper bound. Using the fact that $\tilde{R}_{\mathrm{opt},\varepsilon}(x/n) = 1$ for all $0 < x \leq n/3$ and referring to Theorem 3.28, we obtain

$$E\left(T_{\tilde{A}^*_\varepsilon}\right) \leq nH_{n/3} + \sum_{x=\lfloor n/3\rfloor}^{(1/2-\varepsilon)n} \frac{1}{A_{\max,\varepsilon}(x/n)} + \frac{\varepsilon n}{A_{\max,\varepsilon}(1/2-\varepsilon)} + o(n).$$

Using the same partition points as in the lower bound statement and the monotonicity of $A_{\max,\varepsilon}$, we have

$$\mathrm{E}(T_{\tilde{A}^*_\varepsilon}) \leq n\left(\ln\left(\frac{n}{3}\right) + \gamma + \sum_{i=1}^{k} \frac{p_{i-1}-p_i}{A_{\max,\varepsilon}(p_i)} + \frac{1/2-p_0}{A_{\max,\varepsilon}(p_0)} + \int_{1/3}^{p_k} \frac{\mathrm{d}p}{A_{\max,\varepsilon}(p)}\right) + o(n). \tag{3.34}$$

3.4.2 Numerical Evaluation of the Expected Runtime

In this section we evaluate expressions (3.33) and (3.34) numerically to compute an estimate for the expected runtime of algorithm $\tilde{A}^*_\varepsilon$ on OneMax and for the unary unbiased black-box complexity.

r	L_r	R_r	$A_{\max,\varepsilon}(L_r)$	$A_{\max,\varepsilon}(R_r)$	$R_r - L_r$
3	0.333333333	0.367544468	0.333333	0.405267	0.034211135
5	0.367544468	0.386916541	0.405267	0.467174	0.019372073
7	0.386916541	0.399734261	0.467174	0.522084	0.012817721
9	0.399734261	0.409006003	0.522084	0.571870	0.009271741
11	0.409006003	0.416109983	0.571870	0.617718	0.007103980

TABLE 3.1: The optimal number of bit flips in interval $(L_r n, R_r n]$ is r.

Theorem 3.30. *For sufficiently small $\varepsilon > 0$ the expected runtime $\mathrm{E}(T_{\tilde{A}^*_\varepsilon})$ of algorithm $\tilde{A}^*_\varepsilon$ on* OneMax *is $n\ln(n) - cn \pm o(n)$ for a constant c between 0.2539 and 0.2665. This bound is also the unary unbiased black-box complexity of* OneMax.

We can rewrite the expression in Theorem 3.30 to $n\left(\ln\left(n/3\right) + \gamma + c'\right) + o(n)$ for a constant c' between 0.2549 and 0.2675 to ease a comparison with the expected runtime of the previously best known unary unbiased algorithm, which is the one presented in [LDD15]. This latter algorithm has an expected runtime equaling that of RLS up to an additive term of order $o(n)$. It is hence $n(\ln(n/2) + \gamma) \pm o(n)$. For sufficiently small $\varepsilon > 0$ Algorithm $\tilde{A}^*_\varepsilon$ is thus by an additive $(\ln(3) - \ln(2) - c')n \pm o(n)$ term faster, on average, than RLS or the algorithm presented and analyzed in [LDD15]. That is, compared to RLS, algorithm $\tilde{A}^*_\varepsilon$ saves between $0.138n \pm o(n)$ and $0.151n \pm o(n)$ iterations on average.

To compute $\mathrm{E}(T_{\tilde{A}^*_\varepsilon})$, we split the interval $(0, \frac{1}{2})$ into intervals $(L_{2i+1}, R_{2i+1}]$, $i = 0, 1, \ldots$, such that for each i and each $p \in (L_{2i+1}, R_{2i+1}]$ the number $\tilde{R}_{\mathrm{opt},\varepsilon}(p)$ of bits that need to be flipped in order to maximize the approximated expected fitness increase $A(\cdot, p, 1-p)$ is $2i+1$ (note that this is independent of ε, since ε just determines the cut-off point after which only use a bound for the drift-maximizing number of bit flips). Table 3.1 displays the first few intervals along with the corresponding drift values at the borders of the interval. We observe that the further we are away from the optimum (this corresponds to larger r by Theorem 3.19), the smaller the size of the interval.

The bound for the expected runtime of algorithm $\tilde{A}^*_\varepsilon$ reported in Theorem 3.30 is obtained by setting $p_0 = R_{4001}$, $p_k = R_9$ and using the following partition points

$$p_0 = R_{4001} > R_{3001} > R_{2001} > R_{1001} > R_{951} > R_{901} > R_{851} > \cdots > R_{200} > R_{151} > R_{101} > R_{35} > R_{34} > \cdots > R_{10} > R_9 = p_k.$$

The accuracy of our approximation can be increased by adding denser partition points, especially to the smaller side near p_k.

3.5 Fixed-Budget Analysis

In this section, we compare the algorithms developed in this work with the classic RLS heuristic (which was the essentially best previous unary unbiased algorithm for Om) in the *fixed-budget perspective*, that is, we compare the expected fitnesses obtained after a fixed budget B of iterations. This performance measure was introduced by Jansen and Zarges [JZ14] to reflect the fact that the most common use of search heuristics is not to compute an optimal solution, but only a solution of reasonable quality. We note that the time to reach a particular solution quality, called $T_{A,f}(a)$ in [Doe+13a] where this notion was first explicitly defined, would be an alternative way to phrase such results. We do not regard this performance measure here, but we

would expect that, in a similar vein in as the following analysis, also in this measure our algorithm is superior to RLS by a (small) constant percentage.

Our main result in this section is that our drift maximizer with a fixed budget compute solutions having a roughly 13% smaller fitness distance to the optimum. This result contrasts the lower-order advantage in terms of the expected runtime, i.e., the average time needed to find an optimal solution.

The main challenge is proving the innocent statement that the time taken by our algorithm to find a solution x of fitness $\textsc{Om}(x) \geq 2n/3$ is strongly concentrated. Such difficulties occur often in fixed-budget analyses, see, e.g. [Doe+13a]. We prove the desired concentration via the following well-known martingale version of Azuma's inequality [Azu67] (as opposed to the simpler method of bounded differences, which appears not to be applicable here).

Theorem 3.31 (Method of Bounded Martingale Differences)**.** *Let $X_1, X_2, \dots, X_n$ be an arbitrary sequence of random variables and let f be a function satisfying the property that for each $i \in [n]$, there is a non-negative c_i such that $|\operatorname{E}(f \mid X_0, X_1, \dots, X_{i-1}) - \operatorname{E}(f \mid X_0, X_1, \dots, X_i)| \leq c_i$. Then*

$$\Pr\left(|f - \operatorname{E}(f)| \geq \delta\right) \leq 2\exp\left(-\frac{\delta^2}{2\sum_{i=1}^n c_i^2}\right)$$

for all $\delta > 0$.

Consider a run of the algorithm $\tilde{A}^*_\varepsilon$ with small constant $0 < \varepsilon < 1/6$. Let $X_t := n - \max\{\textsc{Om}(x(i)) \mid i \in [0..t]\}$ be the current smallest fitness distance and let $r_{\max} := \tilde{R}_{\text{opt},\varepsilon}(1/2 - \varepsilon)$ be the maximal mutation strength. Let $T_{1/3}$ be the first time at which the distance to the optimum is at most $n/3$, i.e., $T_{1/3}$ is the smallest t for which $X_t \leq n/3$. Let $N := 3r_{\max}n$ and define the function f by setting $f(X_0, X_1, \dots) := \min\{N, T_{1/3}\}$.

We notice that $\Pr(T_{1/3} > f) = \Pr(T_{1/3} > N) = \Pr(X_N > n/3)$. Referring to Lemma 3.16, we obtain for all $X_t > n/3$ that $\operatorname{E}(X_t - X_{t+1} \mid X_t) = B(n, X_t, \tilde{R}_{\text{opt}}(X_t/n)) \geq A_{\max,\varepsilon}(X_t/n) - O(1/X_t) \geq 1/3 - o(1)$. Using the fact that $X_t - X_{t+1} \leq r_{\max}$, we obtain $\Pr(X_t > X_{t+1} \mid X_t > n/3) \geq 1/(3r_{\max}) - o(1)$. Define binary random variables Y_t by setting $Y_t := \mathbb{1}_{X_t > X_{t+1}}$, if $X_t > n/3$, and otherwise by having $Y_t = 1$ with probability $1/(3r_{\max}) - o(1)$ independently for all such Y_t. Note that, by definition, we have $\Pr[Y_t = 1] \geq 1/(3r_{\max}) - o(1)$ regardless of the outcomes of $Y_{t'}$, $t' < t$. Consequently, by well-known results, e.g., Lemma 3 in [Doe18a], the Y_t admit the same Chernoff bounds for the lower tail as independent binary random variables with success probability $1/(3r_{\max}) - o(1)$. We thus estimate $\Pr(X_N > n/3) \leq \Pr(Y_1 + Y_2 + \dots + Y_N < (2/3)n) = \exp(-\Omega(n))$. Consequently $E(T_{1/3} - f) < E(T)\Pr(T_{1/3} > f) = \exp(-\Omega(n))$.

Using the fact that $X_t - X_{t+1} \leq r_{\max}$, the additive drift theorem yields that the expected influence of one iteration on the remaining optimization time is at most $r_{\max}/(1/3 - o(1)) < 4r_{\max}$. Consequently, for $1 \leq i \leq N$, we have

$$|\operatorname{E}(f \mid X_0, X_1, \cdots, X_{i-1}) - \operatorname{E}(f \mid X_0, X_1, \cdots, X_i)| \leq 4r_{\max}.$$

Applying Theorem 3.31 and using the fact that $\Pr(T_{1/3} > f) = \exp(-\Omega(n))$ and $E(T_{1/3} - f) = \exp(-\Omega(n))$, we compute

$$\begin{aligned}
& \Pr\left(|T_{1/3} - \mathrm{E}(T_{1/3})| \geq n^{0.6}\right) \\
\leq\ & \Pr\left(|f - \mathrm{E}(T_{1/3})| \geq n^{0.6}\right) + \exp(-\Omega(n)) \\
\leq\ & \Pr\left(|f - \mathrm{E}(f) - \mathrm{E}(T_{1/3} - f)| \geq n^{0.6}\right) + \exp(-\Omega(n)) \\
\leq\ & \Pr\left(|f - \mathrm{E}(f)| \geq n^{0.6} - \mathrm{E}(T_{1/3} - f)\right) + \exp(-\Omega(n)) \\
\leq\ & \Pr\left(|f - \mathrm{E}(f)| \geq n^{0.6}/2\right) + \exp(-\Omega(n)) \\
\leq\ & 2\exp\left(-\frac{n^{1.2}/4}{2N(4r_{\max})^2}\right) + \exp(-\Omega(n)) = o(\exp(-n^{0.1})).
\end{aligned}$$

According to the computation in the proof of Theorem 3.30 and using the fact that $\mathrm{E}(T_{1/3}) = \mathrm{E}(T_{\tilde{A}^*_\varepsilon}) - nH_{n/3}$, we obtain with probability $1 - O(\exp(-n^{0.1}))$ that $0.2549n \leq T_{1/3} \leq 0.2675n$.

Consider a budget of $B = kn$ iterations with $k \geq 0.2675$. Let $s := \lfloor 0.2675n \rfloor$. With probability $1 - O(\exp(-n^{-0.1}))$, a run of algorithm $\tilde{A}^*_\varepsilon$ has $X_s \leq n/3$. Conditional on this, in the remainder $\tilde{A}^*_\varepsilon$ mutates exactly one bit in each iteration according to Lemma 3.26. Since $E(X_t \mid X_{t-1}) = X_{t-1}(1 - 1/n)$ in this case, we have for all $t \geq s$ that

$$\mathrm{E}(X_t \mid X_s) = \mathrm{E}(X_s)\,(1 - 1/n)^{t-s} \leq (n/3)(1 - 1/n)^{t-s}.$$

Therefore, with a budget of $B \geq 0.2675n$ algorithm $\tilde{A}^*_\varepsilon$ reaches a fitness distance X_B satisfying

$$\mathrm{E}(X_B) \leq \mathrm{E}(X_B \mid X_s \leq n/3) + n \cdot \Pr(X_s > n/3) \leq (1 + o(1))(n/3)(1 - 1/n)^{B - 0.2675n}.$$

Using the same reasoning for RLS, we compute for Y_B the fitness distance RLS reaches with the same budget of B that

$$\begin{aligned}
\mathrm{E}(Y_B) &= (n/2)(1 - 1/n)^B \\
&= (3/2)(1 - 1/n)^{0.2675n}(n/3)(1 - 1/n)^{B - 0.2675n} \\
&\geq (1 - o(1))(3/2)\exp(-0.2675)\,\mathrm{E}(X_B) \\
&= (1 - o(1))1.1479...\ \mathrm{E}(X_B).
\end{aligned}$$

In other words, $\mathrm{E}(X_B) \leq (1 + o(1))0.8711...\ \mathrm{E}(X_A)$, that is, with the same budget, Algorithm $\tilde{A}^*_\varepsilon$ is roughly 13% closer to the optimum than RLS.

3.6 Self-adjusting Mutation Rate

While it is clear that choosing the best mutation strength for each fitness level can improve the performance, it is not so clear how to find a good mutation strength function efficiently. To overcome this difficulty, we propose to choose the mutation strength in each iteration based on the experience in the optimization process so far. We enforce gaining a certain experience by designating each iteration with probability δ as *learning iteration*. In a learning iteration, we flip a random number of bits (chosen uniformly at random from a domain $[1..r_{\max}]$) and store (in an efficient manner) the progress made in these iterations. In all regular iterations, we use the experience

made in these learning iterations to determine the most promising mutation strength and create the offspring with this mutation strength.

More precisely, let us denote by x_t the search point after the t-th iteration, that is, after the mutation and selection step of iteration t. Denote by x_0 the random initial search point. If t is a learning iteration, denote by r_t the random mutation strength r used in this iteration. Otherwise set $r_t = 0$.

The main idea of our algorithm is to learn the efficiency of the mutation strengths, that is, the expected progress made when flipping r bits, for all $r \in [1..r_{\max}]$. We do so via a time-discounted average of the progresses observed in the learning iterations: We define the estimated progress, called *velocity* in the absence of a better name, after the t-th iteration by

$$v_t[r] := \frac{\sum_{s=1}^{t} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t-s}(f(x_s) - f(x_{s-1}))}{\sum_{s=1}^{t} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t-s}}. \tag{3.35}$$

In this expression, the parameter ε, called *forgetting rate*, determines the decrease of the importance of older information. Since $(1-\varepsilon)^{1/\varepsilon} \approx 1/e$, the reciprocal of the forgetting rate is (apart from constant factors) the information half-life.

We first observe that we can compute the velocities iteratively and thus, unlike equation (3.35) might suggest, do not need to store the full history of the learning iterations. To this aim, we need to store one additional value for each r, namely the sum of the $(1-\varepsilon)^{t-s}$ terms used in the weighted average, that is,

$$w_t[r] := \sum_{s=1}^{t} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t-s}.$$

Then the following recursive description of the velocities and weight sums is easily seen: If in iteration $t+1$ we have not done a learning step with mutation strength r, that is, $r_{t+1} \neq r$, then $v_{t+1}[r] = v_t[r]$ and $w_{t+1}[r] = (1-\varepsilon)w_t[r]$. If $r_{t+1} = r$, then

$$\begin{aligned} v_{t+1}[r] &= \frac{(1-\varepsilon)w_t[r]v_t[r] + f(x_{t+1}) - f(x_t)}{(1-\varepsilon)w_t[r] + 1}, \\ w_{t+1}[r] &= (1-\varepsilon)w_t[r] + 1. \end{aligned}$$

For exploiting the experience gained in the learning iterations, we adopt a greedy strategy and always choose the mutation strength with highest velocity (breaking ties randomly, but giving preference to the previous-best mutation strength). Our greedy choice of the mutation strength might be detrimental for fitness landscapes in which the optimal mutation strength changes very frequently. Therefore a velocity-weighted random choice might be more fruitful.

From this discussion, we derive the algorithm *randomized local search with self-adjusting mutation strength* (Algorithm 7).

3.7 Mathematical Runtime Analysis on OneMax

In this section, we analyze via mathematical means how self-adjusting algorithm optimizes the OneMax function. This is an asymptotic analysis in terms of the problem size n. We refer to the previous well-established runtime analysis literature for more details on the motivations of mathematical runtime analysis and on the meaning of asymptotic results.

Algorithm 7 Randomized local search with self-adjusting mutation strength. The parameters of the algorithm are the maximum mutation strength $r_{\max}$, the learning rate δ, and the forgetting rate ε. The operator $\text{flip}(x, r)$ generates from x a new search point by flipping exactly r random bit positions.

Initialization:
Choose $x \in \{0,1\}^n$ uniformly at random.
for $r \leftarrow 1, 2, \dots, r_{\max}$ **do** $(v[r], w[r]) := (0, 0)$.
Set $r^* \leftarrow 1$.
Optimization:
for $t \leftarrow 1, 2, \dots$ **do**
  $z \leftarrow \text{random}([0,1])$.
  if $z \leq \delta$ **then**% learning iteration
    $r \leftarrow \text{random}(\{1, \dots, r_{\max}\})$.
    $y \leftarrow \text{flip}(x, r)$.
    $(v[r], w[r]) \leftarrow \left(\frac{(1-\varepsilon)w[r]v[r] + \max\{0, f(y)-f(x)\}}{(1-\varepsilon)w[r]+1}, (1-\varepsilon)w[r]+1\right)$.
    for $r' \in \{1, \dots, r_{\max}\} \setminus \{r\}$ **do** $w[r'] \leftarrow (1-\varepsilon)w[r']$.
  else
    $r^+ \leftarrow \text{random}(\text{argmax}_r(v[r]))$.
    if $v[r^+] > v[r^*]$ **then** $r^* \leftarrow r^+$.
    $y \leftarrow \text{flip}(x, r^*)$.
    for $r \in \{1, \dots, r_{\max}\}$ **do** $w[r] \leftarrow (1-\varepsilon)w[r]$.
  if $f(y) \geq f(x)$ **then** $x \leftarrow y$.

The main result of this section is a proof that our algorithm with reasonable parameter settings very precisely detects the optimal mutation strength. It thus, apart from the learning iterations, has the same performance as the randomized local search algorithm with fitness dependent mutation strength. The main technical challenge in this analysis are the dependencies between the progress of the algorithm and the learning system trying to estimate the velocities. We overcome these, among others, via a domination argument developed in [Doe11] (Lemma 1.20).

Throughout this section, we assume that $r_{\max}$ is a constant independent of n. For simplicity, we only regard the parameters $\varepsilon = n^{-0.99}$ and $\delta = n^{-0.01}$, but remark that broader ranges of these parameters would work as well. In addition to the notation introduced in the previous section, we write r_t^* for the number of bits flipped in a non-learning iteration. We also define the fitness distance $d(x) = n - f(x)$ for all $x \in \{0,1\}^n$.

We start with two elementary results on the fitness progress made in one iteration. We frequently use two elementary facts about the progress from flipping a constant number of bits.

Lemma 3.32. *Consider a run of our algorithm. Let t and H be such that $d(x_t)H \geq n^{1.01}$. Then $d(x_{t+H}) \geq d(x_t)(1 - 2Hr_{\max}/n)$ with probability $1 - \exp(-\Omega(n^{0.01}))$.*

Proof. Consider an iteration $t+i$ that creates x_{t+i} from x_{t+i-1}. Independent of what happened in the previous iterations, the progress $d(x_{t+i-1}) - d(x_{t+i})$ is dominated by the number of 0-bits (of x_{t+i-1}) that are flipped in the mutation step, which again is dominated by the number of flipped positions that were 0 in x_t. Consequently, $d(x_{t+i-1}) - d(x_{t+i})$ is dominated by a hypergeometric distribution with parameters $n, d(x_t), r_{\max}$. By Lemma 1.20 of [Doe11], $d(x_t) - d(x_{t+H})$ is dominated by a sum of H independent random variables each having a hypergeometric distribution as

above. Consequently, $E((x_t) - d(x_{t+H})) \leq Hr_{\max}d(x_t)/n$ and $\Pr(d(x_t) - d(x_{t+H}) \geq 2Hr_{\max}d(x_t)/n) \leq \exp(-Hr_{\max}d(x_t)/3nr_{\max}^2)$. □

Without proof, we state the following elementary fact.

Lemma 3.33. *Let $r \in [1..r_{\max}]$. For $d \in [0..n]$, let $X_d := X_d^r := \max\{d(x) - d(\mathrm{flip}(x,r)), 0\}$, where x is any search point with $d(x) = d$. In other words, X_d describes the fitness gain in one iteration that starts with a search point with fitness $f(x_t)$ and in which the mutation consists of flipping exactly r bits. Then $E(X_d) - E(X_{d-H}) = O(H/n)$. Also, X_d stochastically dominates X_{d-H}.*

The following lemma states that, apart from an initial segment of the optimization process, the values of $w_t[r]$ can essentially assumed to be constant over time.

Lemma 3.34. *Let $r \in [1..r_{\max}]$, $t \geq H := (1/\varepsilon)\ln(n)$ and $w^* := \delta/r_{\max}\varepsilon$. Then with probability $1 - \exp(-n^{\Omega(1)})$,*

$$|w_t[r] - w^*| \leq w^* O(n^{-0.002}).$$

Proof. By definition, we have $w_t[r] = \sum_{s=1}^t \mathbf{1}_{r_s=r}(1-\varepsilon)^{t-s}$. The random variables $\mathbf{1}_{r_s=r}$ are independent Bernoulli trials with success probability $\delta/r_{\max}$. Consequently,

$$E(w_t[r]) = (\delta/r_{\max})\sum_{s=1}^{t}(1-\varepsilon)^{t-s} = (\delta/r_{\max})\frac{1-(1-\varepsilon)^{t+1}}{1-(1-\varepsilon)},$$

which is smaller than $\delta/r_{\max}\varepsilon$ and greater than $(\delta/r_{\max}\varepsilon)(1-(1-\varepsilon)^H) \geq (\delta/r_{\max}\varepsilon)(1-n^{-1})$, the latter using $t \geq H$ and the well-known estimate $(1-\varepsilon) \leq \exp(-\varepsilon)$ valid for all $\varepsilon \in \mathbb{R}$.

By the multiplicative Chernoff bound in Theorem 2.8, taking $\gamma = n^{-0.002}$, we have $\Pr[|w_t[r] - E(w_t[r])| \geq \gamma(\delta/r_{\max}\varepsilon)] \leq 2\exp(-\gamma^2(1-1/n)^2(\delta/r_{\max}\varepsilon)/3) = \exp(-\Omega(n^{0.006}))$. □

The following two lemmas show how well our learning mechanism is able to detect the currently most profitable mutation strength.

Lemma 3.35. *Let $r \in [1..r_{\max}]$, $t \geq 1$ and $H = (1/\varepsilon)2\ln(n)$. Then with probability at least $1 - \exp(-n^{\Omega(1)})$, we have $v_{t+H}[r] \leq (1 + O(n^{-0.002}))\max\{E(X^r_{f(x_{t+H})}), (\varepsilon/\delta)n^{0.01}\}$.*

Proof. Let $S := \sum_{s=1}^{t+H}\mathbf{1}_{r_s=r}(1-\varepsilon)^{t+H-s}(f(x_s) - f(x_{s-1}))$. Since $f(x_s) - f(x_{s-1}) \leq r \leq r_{\max}$, we have $S_0 := \sum_{s=1}^{t}\mathbf{1}_{r_s=r}(1-\varepsilon)^{t+H-s}(f(x_s) - f(x_{s-1})) \leq \sum_{s=1}^{t}(1-\varepsilon)^{t+H-s}r_{\max} \leq r_{\max}(1/\varepsilon)(1-\varepsilon)^H \leq r_{\max}/\varepsilon n^2$.

For the rest $S_1 = \sum_{s=t+1}^{t+H}\mathbf{1}_{r_s=r}(1-\varepsilon)^{t+H-s}(f(x_s) - f(x_{s-1}))$ of the sum S, we argue as follows. Independent of what happened before iteration s, by Lemma 3.33 the progress $f(x_s) - f(x_{s-1})$ is dominated by a random variable with distribution $X_{d(x_t)}$. By Lemma 1.20 in [Doe11], S_1 is dominated by $S' = \sum_{s=t+1}^{t+H}\mathbf{1}_{r_s=r}(1-\varepsilon)^{t+H-s}Y_s$, where the Y_s are independent copies of the random variable $X_{d(x_t)}$. We have $E(S') = \sum_{s=t+1}^{t+H}(\delta/r_{\max})(1-\varepsilon)^{t+H-s}E(X_{d(x_t)}) \leq (\delta/r_{\max}\varepsilon)E(X_{d(x_t)}) \leq (\delta/r_{\max}\varepsilon)\max\{E(X_{d(x_t)}), (\varepsilon/\delta)n^{0.01}\} \leq (1+O(H/n))(\delta/r_{\max}\varepsilon)\max\{E(X_{d(x_{t+H})}), (\varepsilon/\delta)n^{0.01}\} =: U$, where the changing x_t into x_{t+H} for an extra $(1+O(H/n))$ factor is justified by Lemma 3.33 (note that $d(x_{t+H}) \geq d(x_t) - E(X_{d(x_t)})O(H) \geq d(x_t) - O(d(x_t)/n)O(H)$ with high probability is justified by Lemma 3.32). The multiplicative Chernoff bound, recall (i) that

these bounds remain valid when the expectation is replaced by an upper bound for the expectation and (ii) that our independent summands take values in $[0, r_{\max}]$, gives that $\Pr[S' \geq (1+\gamma)U] \leq \exp(-\gamma^2 U/3r_{\max}^2) = \exp(-\Omega(\gamma^2 n^{0.01}))$, which is $\exp(-\Omega(n^{0.006}))$ for $\gamma = n^{-0.002}$.

Consequently, with probability at least $1-\exp(-\Omega(n^{0.006}))$, we have $S = S_0 + S_1 \leq r_{\max}/\varepsilon n^2 + (1+O(n^{-0.002}))(\delta/r_{\max}\varepsilon)\max\{E(X_{f(x_{t+H})}), (\varepsilon/\delta)n^{0.01}\} = (1+O(n^{-0.002}))(\delta/r_{\max}\varepsilon)\max\{E(X_{f(x_{t+H})}), (\varepsilon/\delta)n^{0.01}\}$.

By Lemma 3.34, we have $w_{t+H}[r] = (1-O(n^{-0.002}))\delta/r_{\max}\varepsilon$ with probability at least $1-\exp(-n^{\Omega(1)})$. Consequently, with probability at least $1-\exp(-n^{\Omega(1)})$, we have $v_{t+H}[r] = S/w_{t+H}[r] \leq (1+O(n^{-0.002}))\max\{E(X_{f(x_{t+H})}), (\varepsilon/\delta)n^{0.01}\}$. □

Lemma 3.36. *Let $r \in [1..r_{\max}]$, $t \geq 1$ and $H = (1/\varepsilon)\ln(n)$. Assume that $E(X^r_{d(x_{t+H})}) = \Omega(n^{0.01}\varepsilon/\delta)$. Then with probability at least $1-\exp(-n^{\Omega(1)})$, we have $v_{t+H}[r] \geq (1-O(n^{-0.002}))E(X^r_{f(x_{t+H})})$.*

Proof. Let $S := \sum_{s=1}^{t+H} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t+H-s}(f(x_s)-f(x_{s-1}))$. This dominates $S_1 = \sum_{s=t+1}^{t+H} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t+H-s}(f(x_s)-f(x_{s-1}))$. Independent of what happened in iterations before iteration s, the progress in the s-th iteration $f(x_s)-f(x_{s-1})$ dominates $X_{d(x_{t+H})}$. Let $S' := \sum_{s=t+1}^{t+H} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t+H-s}Y_s$, where the Y_s are independent random variables with distribution $X_{d(x_{t+H})}$. By Lemma 1.20 of [Doe11], S dominates S'.

By construction, S' is a sum of independent random variables each taking values in $[0, r_{\max}]$. Estimating the geometric series as in Lemma 3.34, we compute $E(S') = \sum_{s=t+1}^{t+H}(\delta/r_{\max})(1-\varepsilon)^{t+H-s}E(X_{d(x_{t+H})}) \geq (1-1/n)(\delta/r_{\max}\varepsilon)E(X_{d(x_{t+H})}) =: L$. The multiplicative Chernoff bound gives $\Pr(S' \leq (1-\gamma)L) \leq \exp(-\gamma^2 L/2r_{\max}^2)$, which is $\exp(-n^{\Omega(1)})$ for $\gamma = n^{-0.002}$ and using $E(X_{d(x_{t+H})}) = \Omega(n^{0.01}\varepsilon/\delta)$. Consequently, with probability $1-\exp(-n^{\Omega(1)})$, we have $S \geq (1-O(n^{-0.002}))(\delta/r_{\max}\varepsilon)E(X_{d(x_{t+H})})$.

Since by Lemma 3.34, we have $w_{t+H}[r] = (1+O(n^{-0.002}))\delta/r_{\max}\varepsilon$ with probability at least $1-\exp(-n^{\Omega(1)})$, we conclude that with probability at least $1-\exp(-n^{\Omega(1)})$, we have $v_{t+H}[r] = S/w_{t+H}[r] \geq (1+O(n^{-0.002}))E(X_{f(x_{t+H})})$. □

The fact that our algorithm very precisely detects the optimal mutation strength implies that its fitness progress in each iteration is very close to the maximum possible (Theorem 3.37) and that it has a performance very close to the algorithm developed in Section 3.3 (Theorem 3.38).

Theorem 3.37. *Let T be the optimization time of our algorithm with parameters $\delta = n^{-0.01}$ and $\varepsilon = n^{-0.99}$ on the* OneMax *function. Let $T' = \min\{T, 2n\ln(n)\}$. Then with probability at least $1-O(n^{0.19})$, for each non-learning iteration $t \in [2\ln(n)/\varepsilon, T']$, we have*

$$E(X^{r_t^*}_{d(x_{t-1})}) \geq (1-O(n^{-0.002}))\max\{E(X^r_{d(x_{t-1})}) \mid r \in [1..r_{\max}]\}.$$

Proof. Assume that none of the polynomially many exceptional events in Lemmas 3.34, 3.35, and 3.36 (occurring each with probability at most $\exp(-n^{\Omega(1)})$) appears in the first T' iterations. Let t be such that $d(x_{t-1}) \geq n^{0.04}$. Then $E(X^1_{d(x_{t-1})}) = d(x_{t-1})/n \geq n^{-0.96} > n^{-0.97} = n^{0.01}\varepsilon/\delta$. Consequently, by Lemma 3.35 and 3.36 and the fact that $r_t^* \in \operatorname{argmax}_r v_{t-1}[r]$, we have $E(X^{r_t^*}_{d(x_{t-1})}) \geq (1-O(n^{-0.002}))\max\{E(X^r_{d(x_{t-1})}) \mid r \in [1..r_{\max}]\}$.

We now exploit that more-bit flips have a very small chance to be accepted towards the end of the process. More precisely, for $r \geq 2$ we have $\Pr(X_d^r) = O((d/n)^2)$ with the asymptotics for $d/n \to 0$. Consequently, the probability that in some iteration with $d(x_t) \leq n^{0.3}$ a progress is made with a more-bit flip is $O(T'n^{-1.4}) = \tilde{O}(n^{-0.4})$. Let us condition on this event. Then, also by the previous paragraph, the optimization process from a fitness distance of $n^{0.3}$ to $n^{0.04}$ equal the process of the classic randomized local search heuristic (doing one-bit flips) prolonged by unsuccessful multi-bit flips at a rate of $\Theta(\delta)$. Consequently, via the standard arguments of the analysis of randomized local search on OneMax (which essentially is the coupon collector process), we see that with probability $1 - \exp(-n^{\Omega(1)})$, it takes more than $0.25n\ln(n)$ iterations to reduce the fitness distance from $n^{0.3}$ to $n^{0.04}$.

We estimate, for each $r \geq 2$, the value of $v_t[r]$ after the first iteration t with $d(x_t) = n^{0.04}$. Since the previous $t' = 0.25n\ln n$ iterations did not see any successful r-bit flip, we have $S = \sum_{s=1}^{t} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t-s}(f(x_s) - f(x_{s-1})) \leq \sum_{s=1}^{t-t'}(1-\varepsilon)^{t-s} r_{\max} \leq r_{\max}(1/\varepsilon)(1-\varepsilon)^{t'}$, $w_t[r] \leq (1 - O(n^{0.002}))\varepsilon/\delta r_{\max}$, and thus $v_t[r] = S/w_t[r] \leq (1 + O(n^{0.002}))r_{\max}^2(1/\delta)(1-\varepsilon)^{t'} =: L$. Since in the remaining run of the algorithm no positive update to $v[r]$ is made, we have $v_{t+i}[r] \leq L$ for all $i \geq 1$.

We now estimate $v_{t+i}[1]$. We have $S_{t+i} := \sum_{s=1}^{t+i} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t+i-s}(f(x_s) - f(x_{s-1})) \geq (1-\varepsilon)^i \sum_{s=1}^{t} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t-s}(f(x_s) - f(x_{s-1})) = (1-\varepsilon)^i r_t[1] \geq (1-\varepsilon)^i(1 - O(n^{-0.002}))E(X^1_{d(x_t)}) \geq (1-\varepsilon)^i(1 - O(n^{-0.002}))(d(x_t)/n) = (1-\varepsilon)^i(1 - O(n^{-0.002}))n^{-0.97}$. Consequently, again using Lemma 3.34, we have $v_{t+i}[1] = S_{t+i}/w_{t+i}[r] \geq (1-\varepsilon)^i(1 - O(n^{-0.002}))n^{-0.97}/(\delta/r_{\max}\varepsilon)$. Comparing this to L above, we see that $v_{t+i}[1]$ is significantly larger than $v_{t+i}[r]$, $r \geq 2$, as long as, e.g., $i \leq 0.24n\ln n =: t''$. Hence for the remaining t'' iterations, our algorithm performs 1-bit flips (apart from the few learning-iterations). Consequently, $E(d(x_{t+i})) \leq d(x_t)(1 - \frac{1-\delta}{n})^{t''} \leq n^{0.04}\exp(-(1-\delta)(1/n)t'') = n^{-0.2(1-\delta)}$. By Markov's inequality, the probability that the algorithm has not found the optimum until iteration $t + t''$ thus is at most $n^{-0.2(1-\delta)}$. Consequently, apart from this failure probability, our algorithm has also chosen the optimal r, in this case $r = 1$, in the time interval $[t..T]$. □

Theorem 3.38. *Let $T_{r_{\max}}$ be the minimal expected runtime on the* OneMax *problem among all randomized local search algorithms with fitness dependent mutation strength. Then the expected runtime T of our algorithm A is at most $T_{r_{\max}} + o(n)$. Consequently, by taking $r_{\max}$ large enough, our algorithm has the same expected runtime (apart from $o(n)$ terms) as the algorithm using the near-optimal fitness-dependent mutation strength of Algorithm $\tilde{A}^*_\varepsilon$ (cf. Section 3.3). Also, with any fixed-budget of $B \geq 0.2675n$ iterations, it computes a solution approximately 13% closer to the optimum than the classic RLS heuristic.*

Proof. Denote by $\mathrm{RLS}(r_{\max})$ the class of all randomized local search algorithms for functions $f : \{0,1\}^n \to [0..n]$ with fitness dependent mutation strength $r : [0..n] \to [0..r_{\max}]$. Let $A_{\max} \in \mathrm{RLS}(r_{\max})$ be the randomized local search algorithm with in each iteration t flips that number $r \in [1..r_{\max}]$ of bits that maximizes the expected progress $E(f(x_t) - f(x_{t-1}))$. Let $A^* \in \mathrm{RLS}(r_{\max})$ be the randomized local search algorithm with minimal expected optimization time (on $f =$ OneMax). In Section 3.2, it has been shown that the expected runtimes of both algorithms differ by at most $o(n)$, that is, $E(T_{A_{\max}}) \leq E(T_{A^*}) + o(n)$. We show the assertion of this theorem by showing that our algorithm A has an expected runtime satisfying $E(T_A) \leq E(T_{A_{\max}}) + o(n)$.

Denote the maximum expected progress from a search point x with fitness distance $d(x)$ by $h_{\max}(d(x)) := \max\{E(X^r_{d(x)}) \mid r \in [1..r_{\max}]\}$. Assume first that

our algorithm A for the first $T' = \min\{T_A, 2n\ln(n)\}$ iterations has $E(X^{r_t^*}_{d(x_{t-1})}) \geq (1 - O(n^{-0.002}))h_{\max}(d(x_{t-1}))$ in each non-learning iteration (this is the good case in Theorem 3.37). Then in each iteration $t \in [1..T']$, we have $h(d(x_{t-1})) := E(f(x_t) - f(x_{t-1})) \geq (1-\delta)(1 - O(n^{-0.002}))h_{\max}(d(x_{t-1}))$ by ignoring possible progress in learning iterations. By the variable drift theorem, as in Section 2.4, we have that the expected time taken from a fixed initial search point x_0 to a fitness distance of, say, $0.1n$, is $\int_{0.1n}^{f(x_0)} \frac{1}{h(t)}dt + o(n)$. Since h and $h_{\max}$ are so close, this equals $\int_{0.1n}^{f(x_0)} \frac{1}{h_{\max}(t)}dt + o(n)$, which, again as shown in Section 3.2 via the variable drift theorem for lower bounds, is a lower bound for the time $A_{\max}$ needs to optimize x_0 to a fitness distance of $0.1n$. From a fitness distance of $0.1n$ on, $A_{\max}$ does only 1-bit flips and A does only 1-bit flips apart from the learning iterations. Consequently, the classic coupon collector argument shows that $A_{\max}$ needs at least $\sum_{i=1}^{0.1n-r_{\max}}(n/i)$ iterations to reach the optimum from the first search point reached that has fitness distance at most $0.1n$, whereas $A_{\max}$ does take at most $\sum_{i=1}^{0.1n}(n/i(1-\delta))$ iterations from any search point with fitness distance $0.1n$ or less. Again, these terms are equal apart from $o(n)$ terms, showing our claim for the good case of Theorem 3.37 assuming that the initial search point has a fitness distance of at least $0.1n$.

In the exceptional case of Theorem 3.37, which occurs with probability at most $O(n^{-0.19})$, or in the regular case of the theorem but for an initial search point with fitness distance less than $0.1n$ (which occurs with probability $\exp(-\Omega(n))$), we argue as follows. By only regarding the learning iterations and only those in which $r_t = 1$ was chosen, we see that the expected time to reduce a fitness distance of d (by at least one) is at most $(n/d\delta r_{\max})$. This gives an upper bound for the expected runtime of $O(n\log(n)/\delta) = O(n^{1.01}\log n)$. Since these exceptional cases occur only with probability $O(n^{-0.19})$, they contribute at most $o(n)$ to the expected runtime of A.

Since we have essentially the same (variable) drift as $\tilde{A}^*_\varepsilon$ from Section 3.3, the proof for the second statement on the fixed-budget performance follows from the same arguments in Section 3.5. □

3.8 Experimental Results

In this section we describe some experimental results for our self-adjusting RLS. These are by no means intended to account for a thorough scientific investigation, both for reasons of space and because we feel that the mathematical investigation in the subsequent section is more insightful, also with respect to *why* the proposed ideas work well. Nevertheless, the experimental results indicate that our new algorithm gives good results also for problems other than OneMax, they give some hints on how to choose the parameters $r_{\max}$, δ and ε, and they taught us that finding suitable parameters was not very difficult—we were immediately faster than the (1+1) EA and with at most a few trials were able to beat RLS. All experiments were repeated 20 times, all numbers given below are the averages of these 20 runs.

LeadingOnes function: The LeadingOnes function is defined by $\textsc{LeadingOnes}(x) := \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\}$, that is, it counts, starting from the left end, how many consecutive ones the bit-string x contains. The expected optimization time (number of iterations until the optimum is found) for RLS is $0.5n^2 \pm O(n)$, that of the (1+1) EA with mutation rate $p = 1/n$ is $0.5n^2(1-1/n)((1-1/n)^{-n} - 1) = 0.5(e-1)n^2 \pm O(n) \approx 0.8591n^2$. When taking the asymptotically optimal mutation rate of approximately 1.59, the optimization

time drops to approximately $0.7720n^2$. When taking a (best-possible) fitness-dependent mutation rate of $p_i = 1/(i+1)$ at fitness i, then the optimization time drops to $(e/4)n^2 \pm O(n) \approx 0.6796n^2$ [BDN10].

Experimentally, for $n = 10,000$ and taking the parameters $r_{\max} = 5$, $\delta = 0.1$ and $\varepsilon = 1/(5,000,000)$, we observed an average optimization time of 45.5 million iterations, that is, $0.455n^2$, which clearly beats RLS and all (1+1) EA results described above. The relative standard deviation is low, 4.48% to be precise. [1]

Minimum spanning trees: Given a connected undirected graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_{>0}$, the *minimum spanning tree problem* asks for finding a tree in G that connects all vertices and that has minimal total weight. This problem can be solved via evolutionary methods by taking a bit-string representation (each bit describes whether some edge is part of the tree or not) and taking as fitness function (to be minimized) the sum of the weights of the edges in the string representation plus a punishment term for each connected component (except the first one). For this representation of the problem, both RLS (flipping one or two bits with equal probability) and the (1+1) EA find an optimal solution in any input in expected time $O(|E|^2 \log(|E|w_{\max})$, where $w_{\max}$ is the maximum weight of an edge (see [NW07]).

We ran the following experiments. We took as graph G the complete graph on 50 vertices (hence $|E| = 1225$) with edge weights chosen independently at random in $[0, 1]$, thus having a unique minimum spanning tree. On this instance, RLS in the variant that flips either one or two bits (random choice between these two alternatives) took $4.68 \cdot 10^6 \pm 38.43\%$ iterations. Our algorithm with $r_{\max} = 5$, $\delta = 0.1$, and $\varepsilon = 1/(20,000)$ took $2.66 \cdot 10^6 \pm 30.05\%$ iterations. Analyzing these runs in more detail, we observe that the preferred mutation strength r after a short initial phase takes the maximum value 5, then decreases to one, and finally goes back to two, which is then used for the large remainder of the optimization process. For reasons of computation time, we could not evaluate the (1+1) EA on graphs on 50 vertices. For graphs on 20 vertices, the (1+1) EA was roughly 4 times slower that RLS.

1. We report in the following the mean and relative standard deviations of our experiments by expressing, for example, the previous numbers as $45.5 \cdot 10^6 \pm 4.48\%$.

Chapter 4

Self-adjusting (1+λ) EA

In this chapter, we propose a way to adjust the mutation rate on the fly for algorithms using larger offspring populations. The simple idea is to create half the offspring with twice the current mutation rate and the other half using half the current rate. The mutation rate is then modified to the rate which was used to create the best of these offspring (choosing the winning offspring randomly among all best in case of ambiguity). We do not allow the mutation probability to leave the interval $[2/n, 1/4]$, so that the probability used in the subpopulations are always in the interval $[1/n, 1/2]$.

We add one modification to the very basic idea described in the first paragraph of this section. Instead of always modifying the mutation rate to the rate of the best offspring, we shall take this winner's rate only with probability a half and else modify the mutation rate to a random one of the two possible values (twice and half the current rate). Our motivation for this modification is that we feel that the additional random noise will not prevent the algorithm from adjusting the mutation rate into a direction that is more profitable. However, the increased amount of randomness may allow the algorithm to leave a possible basin of attraction of a locally optimal mutation rate. Observe that with probability $\Theta(1/n^2)$, a sequence of $\log_2 n$ random modifications all in the same direction appears. Hence with this inverse-polynomial rate, the algorithm can jump from any mutation rate to any other (with the restriction that only a discrete set of mutation rates can appear). We note that the existence of random modifications is also exploited in our runtime analysis, which will show that the new self-adjusting mechanism selects mutation rates good enough to lead to the asymptotically optimal runtime among all dynamic choices of the mutation rate for the (1+λ) EA.

In this first work proposing this mechanism, we shall not spend much effort fine-tuning it, but rather show in a proof-of-concept manner that it can find very good mutation rates. In a real application, it is likely that better results are obtained by working with three subpopulations, namely an additional one using (that is, exploiting) the current mutation rate. Also, it seems natural that more modest adjustments of the mutation rate, that is, multiplying and dividing the rate by a number F that is smaller than the value $F = 2$ used by our mechanism, is profitable. We conduct some elementary experiments supporting this intuition in Section 4.7.

To prove that the self-adjusting mechanism just presented can indeed find good dynamic mutation rates, we conduct a rigorous runtime analysis for our self-adjusting (1+λ) EA on the classic test function OneMax.

The runtime of the (1+λ) EA with fixed mutation rates on OneMax is well understood [DK15; GW17]. In particular, [GW17] show that the expected runtime (number of generations) is $(1 \pm o(1)) \left(\frac{1}{2} \cdot \frac{n \ln \ln \lambda}{\ln \lambda} + \frac{e^r}{r} \cdot \frac{n \ln n}{\lambda} \right)$ when a mutation rate of r/n, r a constant, is used. Thus for λ not too large, the mutation rate determines

the leading constant of the runtime, and a rate of $1/n$ gives the asymptotically best runtime.

As a consequence of their work on parallel black-box complexities, Badkobeh, Lehre, and Sudholt [BLS14] showed that the (1+λ) EA with a suitable fitness-dependent mutation rate finds the optimum of OneMax in an asymptotically better runtime of $O(\frac{n}{\log\lambda} + \frac{n\log n}{\lambda})$, where the improvement is by a factor of $\Theta(\log\log\lambda)$. [1] This runtime is best-possible among all λ-parallel unary unbiased black-box optimization algorithms. In particular, no other dynamic choice of the mutation rate in the (1+λ) EA can achieve an asymptotically better runtime. The way how the mutation rate depends on the fitness in the above result, however, is not trivial. When the parent individual has fitness distance d, then mutation rate employed is $p = \max\{\frac{\ln\lambda}{n\ln(en/d)}, \frac{1}{n}\}$.

Our main technical result is that the (1+λ) EA adjusting the mutation rate according to the mechanism described above has the same (optimal) asymptotic runtime. Consequently, the self-adjusting mechanism is able to find on the fly a mutation rate that is sufficiently close to the one proposed in [BLS14] to achieve asymptotically the same expected runtime.

Theorem 4.1. *Let $\lambda \geq 45$ and $\lambda = n^{O(1)}$. Let T denote the number of generations of the (1+λ) EA with self-adjusting mutation rate to minimize* OneMax*. Then,*

$$E(T) = \Theta\left(\frac{n}{\log\lambda} + \frac{n\log n}{\lambda}\right).$$

This corresponds to an expected number of function evaluations of $\Theta(\frac{\lambda n}{\log\lambda} + n\log n)$.

To the best of our knowledge, this is the first time that a simple mutation-based EA achieves a super-constant speed-up via a self-adjusting choice of the mutation rate.

As an interesting side remark, our proofs reveal that a quite non-standard but fixed mutation rate of $r = \ln(\lambda)/2$ also achieves the $\Theta(\log\log\lambda)$ improvement as it implies the bound of $\Theta(n/\log\lambda)$ generations if λ is not too small. Hence, the constant choice $r = O(1)$ as studied in [GW17] does not yield the asymptotically optimal number of generations unless λ is so small that the $n\log n$-term dominates.

Lemma 4.2. *Let $\lambda \geq 45$ and $\lambda = n^{O(1)}$, Let T denote the number of generations of the (1+λ) EA with fixed mutation rate $r = \ln(\lambda)/2$. Then,*

$$E(T) = O\left(\frac{n}{\log\lambda} + \frac{n\log n}{\sqrt{\lambda}}\right).$$

This corresponds to an expected number of function evaluations of

$$O\left(\frac{\lambda n}{\log\lambda} + \sqrt{\lambda} n\log n\right).$$

This chapter is structured as follows: In Section 4.1 we give the algorithm and the mutation scheme. Afterwards, Section 4.2 gives a high-level overview of our main proof strategy. The following technical sections deal with the runtime analysis of

1. As usual in the analysis of algorithms, we write log when there is no need to specify the base of the logarithm, e.g., in asymptotic terms like $\Theta(\log n)$. To avoid the possible risk of an ambiguity for small arguments, as usual, we set $\log x = \max\{1, \log x\}$. All other logarithms are used in their classic meaning, that is, for all $x > 0$, we denote by $\ln x$ the natural logarithm of x and by $\log_2 x$ its binary logarithm.

the expected time spent by the (1+λ) EA on OneMax in each of three regions of the fitness distance d. We label these regions the *far region*, *middle region* and *near region*, each of which will be dealt with in a separate section. The proof of the main theorem and of Lemma 4.2 is then given in Section 4.6. Finally, we conduct some elementary experiments supporting this intuition in Section 4.7.

4.1 Algorithm

We consider the (1+λ) EA with self-adjusting mutation rate for the minimization of pseudo-boolean functions $f : \{0,1\}^n \to \mathbb{R}$, defined as Algorithm 8.

The general idea of the mutation scheme is to adjust the mutation strength according to its success in the population. We perform mutation by applying standard bit mutation with two different mutation probabilities $r/(2n)$ and $2r/n$ and we call r the *mutation rate*. More precisely, for an even number $\lambda \geq 2$ the algorithm creates $\lambda/2$ offspring with mutation rate $r/2$ and with $2r$ each.

The mutation rate is adjusted after each selection. With probability a half, the new rate is taken as the mutation rate that the best individual (i. e. the one with the lowest fitness, ties broken uniformly at random) was created with (*success-based adjustment*). With the other 50% probability, the mutation rate is adjusted to a random value in $\{r/2, 2r\}$ (*random adjustment*). Note that the mutation rate is adjusted in each iteration, that is, also when all offspring are worse than the parent and thus the parent is kept for the next iteration.

If an adjustment of the rate results in a new rate r outside the interval $[2, n/4]$, we replace this rate with the corresponding boundary value. Note that in the case of $r < 2$, a subpopulation with rate less than 1 would be generated, which means flipping less than one bit in expectation. At a rate $r > n/4$, a subpopulation with rate larger than $n/2$ would be created, which again is not a very useful choice.

We formulate the algorithm to start with an initial mutation rate r^{init}. The only assumption on r^{init} is to be greater than or equal to 2. The (1+λ) EA with this self-adjusting choice of the mutation rate is given as pseudocode in Algorithm 8.

Algorithm 8 (1+λ) EA with two-rate standard bit mutation

Select x uniformly at random from $\{0,1\}^n$ and set $r \leftarrow r^{\text{init}}$.
for $t \leftarrow 1, 2, \ldots$ **do**
  for $i \leftarrow 1, \ldots, \lambda$ **do**
    Create x_i by flipping each bit in a copy of x independently with probability $r_t/(2n)$ if $i \leq \lambda/2$ and with probability $2r_t/n$ otherwise.
  $x^* \leftarrow \arg\min_{x_i} f(x_i)$ (breaking ties randomly).
  if $f(x^*) \leq f(x)$ **then**
    $x \leftarrow x^*$.
  Perform one of the following two actions with prob. 1/2:
— Replace r_t with the mutation rate that x^* has been created with.
— Replace r_t with either $r_t/2$ or $2r_t$, each with probability 1/2.
  Replace r_t with $\min\{\max\{2, r_t\}, n/4\}$.

Let us explain the motivation for the random adjustments of the rate. Without such random adjustments, the rate can only be changed into some direction if a winning offspring is generated with this rate. For simple functions like OneMax, this is most likely sufficient. However, when the fitness of the best of $\lambda/2$ offspring,

viewed as a function of the rate, is not unimodal, then several adjustments into a direction at first not yielding good offspring might be needed to reach good values of the rate. Here, our random adjustments enable the algorithm to cross such a valley of unfavorable rate values. We note that such ideas are not uncommon in evolutionary computation, with standard-bit mutation being the most prominent example (allowing to perform several local-search steps in one iteration to cross fitness valleys).

A different way to implement a mechanism allowing larger changes of the rate to cross unfavorable regions would have been to not only generate offspring with rates $r/2$ and $2r$, but to allow larger deviations from the current rate with some small probability. One idea could be choosing for each offspring independently the rate $r2^{-i}$ with probability $2^{-|i|-1}$ for all $i \in \mathbb{Z}$, $i \neq 0$. This should give similar results, but to us the process appears more chaotic (e.g., with not the same number of individuals produced with rates $r/2$ and $2r$).

The *runtime*, also called the *optimization time*, of the (1+λ) EA is the smallest t such that an individual of minimum f-value has been found. Note that t corresponds to a number of iterations (also called generations), where each generation creates λ offspring. Since each of these offspring has to be evaluated, the number of function evaluations, which is a classical cost measure, is by a factor of λ larger than the runtime as defined here. However, assuming a massively parallel architecture that allows for parallel evaluation of the offspring, counting the number of generations seems also a valid cost measure. In particular, a speed-up on the function $\textsc{OneMax}(x_1, \dots, x_n) := x_1 + \cdots + x_n$ by increasing λ can only be observed in terms of the number of generations. Note that for reasons of symmetry, it makes no difference whether OneMax is minimized (as in the present paper) or maximized (as in several previous research papers).

Throughout the chapter, all asymptotic notation will be with respect to the problem size n.

4.2 Proof Overview

Since the following runtime analysis of our self-adjusting (1+λ) EA on the OneMax function is slightly technical, let us outline the main proof ideas here in an informal manner. Let always x denote the parent individual of the current iterations and $k := f(x)$ its fitness distance from the optimum (recall that we are minimizing the OneMax function, hence the fitness distance equals the objective function value which in turn is the Hamming distance from the optimum $x^* = (0, \dots, 0) \in \{0,1\}^n$ and thus the number of ones in x).

The outer proof argument is variable drift, that is, for each fitness value k we estimate the expected fitness gain ("progress") in an iteration starting with a parent individual x with $f(x) = k$ and then we use the variable drift theorem (Theorem 2.4) to translate this information on the progress into an expected runtime (number of generations until the optimum is found).

To obtain sufficiently strong lower bounds on the expected progress, we need to argue that the rate self-adjustment sufficiently often sets the current rate to a sufficiently good value. This is the main technical difficulty as it needs a very precise analysis of the quality of the offspring in both subpopulations. Since this requires different arguments depending on the current fitness distance k, we partition the process into three regimes.

In the *far region* covering the fitness distance values $k \geq n/\ln(\lambda)$, we need a rate r that is at least almost logarithmic in λ to ensure that the average progress is high enough. Note that to gain the $\Theta(n)$ fitness levels from the initial value of approximately $n/2$ to $n/\ln\lambda$ in at most a time of $O(n/\log\lambda)$, we need an average progress of $\Omega(\ln\lambda)$ per iteration. We shall not be able to obtain this progress throughout this region, but we will obtain an expected progress of

$$\Omega\left(\frac{\log\lambda}{\log\frac{en}{k}}\right)$$

in an iteration with initial fitness distance k. This will be sufficient to reach a fitness distance of $n/\log\lambda$ in the desired $O(n/\log\lambda)$ iterations.

As said, the main difficulty is arguing that the self-adjusting mechanism keeps the rate sufficiently often in the range we need, which is roughly

$$\left[\frac{1}{2\ln\frac{en}{k}}\ln\lambda, \frac{4n^2}{(n-2k)^2}\ln\lambda\right]$$

for all $k \in [n/\ln\lambda, n/2-1]$. Note that these range boundaries, in particular the upper one, depend strongly on k. Hence for arguing that our rate is sufficiently often in this range, we need to consider both the rate changes from the self-adjustments and the changing k-value. In this region we profit from the fact that we work with relatively high rates, which lead to strong concentration behaviors. This allows to argue that, for example, exceeding the upper boundary already by small constant factors is highly unlikely.

The *middle region* covering the fitness distances $k \in [n/\lambda, n/\ln\lambda]$ is small enough so that we do not require the algorithm to find a near-optimal rate very often. In fact, it suffices that the rate is below $\frac{1}{2}\ln\lambda$ with constant probability. This ensures an expected progress of at least $\min\{\frac{1}{8}, \sqrt{\lambda}k/32n\}$, which is sufficient to traverse also this region in time $O(n/\log\lambda)$. Consequently, in this region we only need to argue that the rate does not become too large, whereas there is no lower bound on r which we require. Further, the upper bound of $\frac{1}{2}\ln\lambda$ is large enough so that strong concentration arguments can be exploited, which show that deviations above the upper bound are highly unlikely. Since the upper bound does not change over time, we can now conveniently use an occupation probability argument to show that the rate with constant probability is only a small constant factor over the desired range, which is enough since the random fluctuations of the rate with constant probability reduce the rate further to the desired range.

In the *near region* covering the remaining fitness distance values $k \leq n/\lambda$, the parent individual is already so close to the optimum that any rate higher than the minimal rate $r = 2$ is sub-optimal. Hence in this region, we know precisely the optimum value for the rate. Nevertheless, it is not very easy to show that the subpopulation using the smaller rate $r/2$ has a higher chance of containing the new parent individual. Due to the small rates predominant in this region and the fact that progress generally is difficult (due to the proximity to the optimum), often both subpopulations will contain best offspring (which are in fact copies of the parent). Hence the typical reason for the winning offspring stemming from the smaller-rate subpopulation is not anymore that the other subpopulation contains only worse individuals, but only that the smaller-rate subpopulation contains more best offspring.

By quantifying this effect we establish a drift of the rate down to its minimum value 2 in the near region, and another occupation probability argument is used to

show that the rate with sufficiently high probability will take this value in subsequent sets. Nevertheless, the concentration of the rate around this target value is weaker than in the other regions, and additional care has to be taken to handle iterations in the near region in which the unlikely, but still possible event occurs that the rate exceeds $\ln \lambda$. This value is a critical threshold in the near region since rates larger than $\ln \lambda$ will typically make all offspring worse than the parent. As an additional obstacle, there is a small interval of rates above the threshold in which we cannot show a drift of the rate to its minimum. To show that this small interval nevertheless is left towards its lower end with probability $\Omega(1)$, the occupation probability argument is combined with a potential function whose shape exploits the random adjustments that the (1+λ) EA performs with 50% probability.

4.3 The Far Region

In this first of three technical sections, we analyze the optimization behavior of our self-adjusting (1+λ) EA in the regime where the fitness distance k is at least $n/\ln \lambda$. Since we are relatively far from the optimum, it is relatively easy to make progress. On the other hand, this regime spans the largest number of fitness levels (namely $\Theta(n)$), so we need to exhibit a sufficient progress in each iteration. Also, this is the regime where the optimal mutation rate varies most drastically. Since it is not important for the following analysis, we remark without proof that the optimal rate is n for $k \geq n/2 + \omega(\sqrt{n \log \lambda})$, $(1 + o(1))n/2$ for $k = n/2 \pm o(\sqrt{n \log \lambda})$, and then quickly drops to $r = \Theta(\log \lambda)$ for $k \leq n/2 - \varepsilon n$. Despite these difficulties, our (1+λ) EA manages to find sufficiently good mutation rates to be able to reach a fitness distance of $k = n/\ln \lambda$ in an expected number of $O(n/\log \lambda)$ iterations.

Lemma 4.3. *Let n be sufficiently large and $0 < k < n/2$. We define $c_1(k) = (2\ln(en/k))^{-1}$ and $c_2(k) = 4n^2/(n-2k)^2$.*

1. *If $n/\ln \lambda \leq k$ and $r \leq c_1(k) \ln \lambda$, then the probability that a best offspring has been created with rate $2r$ is at least 0.64.*
2. *Let $\lambda \geq 100$. If $c_2(k) \ln \lambda \leq r \leq n/4$, then the probability that all best offspring have been created with rate $r/2$ is at least 0.51.*
3. *If $r \geq 2(1+\gamma)c_2(k) \ln \lambda$, then the probability that all best offspring are worse than the parent is at least $1 - \lambda^{-\gamma}$.*

Proof of Lemma 4.3 part 1. Let $q(k,i,r)$ and $Q(k,i,r)$ be the probabilities that standard bit mutation with mutation rate $p = r/n$ creates from a parent with fitness distance k an offspring with fitness distance exactly $k-i$ and at most $k-i$. Then

$$q(k,i,r) = \sum_{j=0}^{k-i} \binom{k}{i+j} \binom{n-k}{j} \left(\frac{r}{n}\right)^{i+2j} \left(1 - \frac{r}{n}\right)^{n-i-2j}$$

and $Q(k,i,r) = \sum_{j=i}^{k} q(k,j,r)$. We first show that the probability of not achieving $i \geq 2r$ is less than 0.2. This is because for large enough n and $r \geq 2$ we have $(1-o(1))\binom{k}{2r} \geq (1-o(1))4!(k/(2r))^{2r} > 4(k/(2r))^{2r}$ and for $r \leq c_1(k)\ln\lambda = o(\sqrt{n})$ by Lemma 2.14, part 3 we have $(1-2r/n)^n \geq e^{-2r-4r^2/n} = (1-o(1))e^{-2r}$,

thus

$$Q(k,2r,2r) \geq q(k,2r,2r) \geq \binom{k}{2r}\left(\frac{2r}{n}\right)^{2r}\left(1-\frac{2r}{n}\right)^{n}$$
$$\geq 4\left(\frac{k}{2r}\right)^{2r}\left(\frac{2r}{n}\right)^{2r}e^{-2r} \geq 4\left(\frac{k}{en}\right)^{2r} \geq 4\left(\frac{k}{en}\right)^{2c_1(k)\ln\lambda} = \frac{4}{\lambda}.$$

Considering $\lambda/2$ offspring using rate $2r$, the probability that none of them achieves a fitness improvement of at least $2r$ is less than $(1-4/\lambda)^{\lambda/2} < \exp(-4/2) < 0.2$. We next argue that an offspring having a progress of $2r$ or more with good probability comes from the $2r$-subpopulation. We notice that

$$\frac{q(k,i,2r)}{q(k,i,r/2)} \geq 4^i\left(\frac{1-2r/n}{1-r/(2n)}\right)^{n} \geq \Big(1-o(1)\Big)4^i e^{-1.5r}.$$

Thus for $r \geq 2$ and $i \geq 2r$ we have

$$\frac{q(k,i,2r)}{q(k,i,r/2)} \geq \Big(1-o(1)\Big)\left(\frac{4}{e}\right)^{2r} > 4.$$

Therefore if the best progress among all λ offspring is i and $i \geq 2r$, the conditional probability that an offspring having fitness distance $k-i$ is generated with rate $2r$ is at least $q(k,i,2r)/(q(k,i,2r)+q(k,i,r/2)) \geq 4/5$. In total, the probability that a best offspring has been created with rate $2r$ is at least

$$(1-0.2)\cdot 4/6 = 0.64.$$

□

Proof of Lemma 4.3 part 3. Let $\lambda \geq 100$ and $c_2(k)\ln\lambda \leq r \leq n/4$. Our idea is to show a probability of $o(1/\lambda)$ for the event that an offspring with rate $2r$ is not worse than the expected fitness of an offspring with rate $r/2$. Let $X(k,r)$ denote the random decrease of the fitness distance when applying standard bit mutation with probability $p = r/n$ to an individual with k ones. Then

$$E(X(k,r)) = kp-(n-k)p = \frac{(2k-n)r}{n},$$
$$\mathrm{Var}(X(k,r)) = np(1-p) = r\left(1-\frac{r}{n}\right).$$

According to Bernstein's inequality (Theorem 2.8 1), for any $\Delta > 0$ we have

$$\Pr(X \geq E(X)+\Delta) \leq \exp\left(\frac{-\Delta^2}{2\,\mathrm{Var}(X)+2\Delta/3}\right) \leq \exp\left(\frac{-\Delta^2}{2\,\mathrm{Var}(X)+2\Delta}\right)$$

We apply this bound with $X = X(k,2r)$ and $\Delta = E(X(k,r/2)) - E(X(k,2r)) = (n-2k)1.5r/n > 0$. Then,

$$\Pr(X(k,2r) \geq E(X(k,r/2))) \leq \exp\left(\frac{-\Delta^2}{2\,\mathrm{Var}(X(k,2r))+2\Delta}\right)$$
$$= \exp\left(\frac{-9(n-2k)^2 r}{4n(7n-8r-6k)}\right) \leq \exp\left(-\frac{9(n-2k)^2 c_2(k)\ln\lambda}{28n^2}\right) = \frac{1}{\lambda^{9/7}}.$$

We notice that we have $7n - 8r - 6k > 7n - 4n - 3n = 0$ in the second inequality. From a union bound we see that with probability less than $\lambda^{-9/7}(\lambda/2) < 100^{-2/7}/2 < 0.14$, the best offspring created with rate $2r$ is at least as good as the expected fitness of an individual created with rate $r/2$.

We estimate the probability that the best offspring using rate $r/2$ has a fitness distance at most $E(X(k, r/2))$. Let y be an offspring obtained from x via standard bit mutation with mutation rate $r/(2n)$. Let X^+ and X^- be the number of one-bits flipped and zero-bits flipped, respectively.

$$X^+ := |\{i \in \{1, \dots, n\} \mid x_i = 1 \wedge y_i = 0\}|,$$
$$X^- := |\{i \in \{1, \dots, n\} \mid x_i = 0 \wedge y_i = 1\}|.$$

Both X^+ and X^- are binomially distributed and $X(k, r/2) = X^+ - X^-$. We aim at a lower bound for $\Pr(X(k, r/2) \geq E(X(k, r/2)))$. We have $\Pr(X^+ \geq E(X^+) - 1) \geq 1/2$, since the median of X^+ is between $\lfloor E(X^+) \rfloor$ and $\lceil E(X^+) \rceil$ by [KB80]. It remains to bound $\Pr(X^- \leq E(X^-) - 1)$. We notice that $E(X^-) = (n-k)r/(2n) > \ln \lambda > 1$ and $E(X^-) \leq (n-k)/4$. Applying Lemma 2.16 to binomial random variable $\tilde{X}^- := (n-k) - X^-$, we obtain $\Pr(X^- \leq E(X^-) - 1) = \Pr(\tilde{X}^- \geq E(\tilde{X}^-) + 1) \geq 0.037$. Therefore

$$\begin{aligned}\Pr(X(k, r/2) \geq E(X(k, r/2))) &\geq \Pr(X^- \leq E(X^-) - 1) \Pr(X^+ \geq E(X^+) - 1) \\ &\geq 0.037 \cdot 0.5 = 0.0185.\end{aligned}$$

For $\lambda/2$ offspring using rate $r/2$, the probability that the best one has a fitness distance at most $E(X(k, r/2))$ is more than $1 - (1 - 0.0185)^{\lambda/2} \geq 1 - 0.9815^{50} \geq 0.6$. Therefore with probability at least $0.6 \cdot (1 - 0.14) > 0.51$, all best offspring are from the $r/2$-subpopulation. This proves the second statement of the lemma. □

Proof of Lemma 4.3 part 2. Let $r \geq 2(1+\gamma)c_2(k) \ln \lambda$. An offspring created with mutation rate r/n is at least as good as its parent if and only if $X(k, r) \geq 0$. By using Bernstein's inequality (Theorem 2.8 1) with $X = X(k, r)$ and $\Delta = -E(X(k, r))$, we have

$$\begin{aligned}\Pr(X(k, r) \geq 0) &\leq \exp\left(-\frac{E(X(k, r))^2}{2\operatorname{Var}(X(k, r)) + 2E(X(k, r))}\right) \\ &= \exp\left(-\frac{(n-2k)^2 r}{2n(2n-2k-r)}\right) \leq \exp\left(-\frac{(n-2k)^2 r}{4n^2}\right).\end{aligned}$$

Since $r \geq 2(1+\gamma)c_2(k) \ln \lambda$, the corresponding probabilities for rate $r/2$ and $2r$ are at most $1/\lambda^{1+\gamma}$ and $1/\lambda^{4+4\gamma}$, respectively. By a union bound, with probability at most $1/\lambda^{\gamma}$, the best offspring is at least as good as its parent. This proves part 3. □

The lemma above shows that the rate r is subject to a constant drift towards the interval $[c_1(k) \ln n, c_2(k) \ln n]$. Unfortunately, we cannot show that we obtain a sufficient fitness progress for all r-values in this range. However, we can do so for a range smaller only by constant factors. This is what we do now (for large values of k) and in Lemma 4.5 (for smaller values of k). This case distinction is motivated by the fact that $c_2(k)$ becomes very large when k approaches $n/2$. Having a good fitness drift only for such a smaller range of r-values is not a problem since the random movements of r let us enter the smaller range with constant probability. This is what we will exploit in Theorem 4.6 and its proof.

Let $2 \leq r \leq n/4$ be the current rate and let $\tilde{r} \in \{r/2, 2r\}$. Let $\tilde{\Delta}(\lambda/2, k, \tilde{r})$ denote the fitness gain of the best of $\lambda/2$ offspring generated with rate $\tilde{r}$ from a parent x with fitness distance $k := \textsc{OneMax}(x)$ and the parent itself. More precisely, let $x^{(i)}, i \in \{1, \dots, \lambda/2\}$, be independent offspring generated from x by flipping each bit independently with probability $\tilde{r}/n$. Then the random variable $\tilde{\Delta}(\lambda/2, k, \tilde{r})$ is defined by $\max\{0, k - \min\{\textsc{OneMax}(x^{(i)}) \mid i \in \{1, \dots, \lambda/2\}\}\}$. We use $\Delta := \max\{\tilde{\Delta}(\lambda/2, k, r/2), \tilde{\Delta}(\lambda/2, k, 2r)\}$ to denote the fitness gain in a iteration which uses x as parent and r as mutation rate.

We next show that a region contained in $[c_1(k) \ln \lambda, c_2(k) \ln \lambda]$ provides at least a logarithmic (in λ) drift on fitness.

Lemma 4.4. *Let n be sufficiently large, $2n/5 \leq k < n/2$ and $\lambda \geq e^{10}$. If $\ln(\lambda) \leq r \leq \min\{n^2 \ln(\lambda)/(25(n-2k)^2), n/4\}$, then $E(\Delta \mid k) \geq 10^{-6} \ln \lambda$.*

Proof. We first notice that $n^2/(25(n-2k)^2) \geq 1/(25 \cdot 0.2^2) = 1$ for all $2n/5 \leq k < n/2$. We aim to prove $\tilde{\Delta}(\lambda/2, k, \tilde{r}) \geq 10^{-5} \ln \lambda$ with $\tilde{r} = r/2$. If $\tilde{r} \leq 13$, the probability that an offspring using rate $\tilde{r}/n$ achieving a fitness distance of $k-1$ is at least

$$\binom{k}{1} \frac{\tilde{r}}{n} \left(1 - \frac{\tilde{r}}{n}\right)^{n-1} \geq (1 - o(1)) 0.4 e^{-\hat{r}} > 9 \cdot 10^{-7}.$$

For $\lambda/2$ offspring using rate $\tilde{r} \leq 13$,

$$\Pr(\Delta \geq 1 \mid k) \geq 1 - \left(1 - 9 \cdot 10^{-7}\right)^{\lambda/2} > 9 \cdot 10^{-3} \text{ for all } \lambda > e^{10}.$$

This satisfies the statement that $E(\Delta \mid k) \geq 10^{-6} \ln \lambda$ since $\ln \lambda \leq 2\tilde{r} \leq 26$. Therefore, it remains to consider $\tilde{r} > 13$.

Let X^+ and X^- be the number of one-bits flipped and zero-bits flipped, respectively, in an offspring using rate $p = \tilde{r}/n$. X^+ and X^- follow binomial distributions $\mathrm{Bin}(k, p)$ and $\mathrm{Bin}(n-k, p)$, respectively. Let $u := E(X^+) = kp$ and $\tilde{u} = \lceil u \rceil$. Then $u \leq k/8$, $u \geq (k/n)\tilde{r} \geq 5.2$, and $u \geq (k/n) \ln(\lambda)/2 \geq 0.4 \cdot 0.5 \cdot \ln \lambda = 0.2 \ln \lambda$. Furthermore, let

$$B(x) := \Pr(X^+ = x) = \binom{k}{x} p^x (1-p)^{k-x} \text{ for all } x \in \{0, 1, \dots, k\},$$

$$F(x) := \Pr(X^+ \geq x) = \sum_{i=\lceil x \rceil}^{k} B(i) \text{ for all } x \in [0, k].$$

Applying Lemma 2.16, we obtain $F(u) = \Pr(X^+ \geq E(X^+)) \geq 1/4$. Similarly $\Pr(X^- \leq E(X^-)) \geq 1/4$. We prove that for $\delta := \lceil E(X^-) - E(X^+) + 0.02 \ln \lambda \rceil = \lceil (n-2k)p + 0.02 \ln \lambda \rceil$, we have $F(u+\delta) \geq \lambda^{-0.36}/e^{13}$, and thus $\Pr(X^+ - X^- \geq 0.02 \ln \lambda) \geq \lambda^{-0.36}/(4e^{13})$. Since for any $x \in \mathbb{Z}_{\geq u}$ we have

$$\frac{B(x+1)}{B(x)} = \frac{k-x}{x+1} \cdot \frac{p}{1-p} \leq \frac{u - up}{u - up + 1 - p} < 1,$$

we obtain $B(\tilde{u}) > B(\tilde{u}+1) > \cdots > B(k)$, and thus $F(u+\delta) \geq \delta B(\tilde{u}+2\delta)$ as well as $\delta B(\tilde{u}) \geq F(u) - F(u+\delta)$. We see that

$$\begin{aligned}\frac{B(\tilde{u}+2\delta)}{B(\tilde{u})} &= \frac{(k-\tilde{u})\cdots(k-\tilde{u}-2\delta+1)}{(\tilde{u}+1)\cdots(\tilde{u}+2\delta)} \cdot \frac{p^{2\delta}}{(1-p)^{2\delta}} \\ &\geq \left(\frac{k-u-2\delta}{k(1-p)}\right)^{2\delta} \frac{u^{2\delta}}{(\tilde{u}+1)\cdots(\tilde{u}+2\delta)} \geq \left(1-\frac{2\delta}{7u}\right)^{2\delta} \frac{u^{2\delta}}{(\tilde{u}+1)\cdots(\tilde{u}+2\delta)},\end{aligned}$$

where we used $k(1-p) = k-u \geq 8u-u = 7u$. Using a sharp version of Stirling's approximation due to Robbins [Rob55], we compute

$$\begin{aligned}(\tilde{u}+2\delta)! &\leq \sqrt{2\pi(\tilde{u}+2\delta)}\left(\frac{(\tilde{u}+2\delta)}{e}\right)^{\tilde{u}+2\delta} \exp\left(\frac{1}{12(\tilde{u}+2\delta)}\right), \\ \tilde{u}! &\geq \sqrt{2\pi\tilde{u}}\left(\frac{\tilde{u}}{e}\right)^{\tilde{u}} \exp\left(\frac{1}{12\tilde{u}+1}\right).\end{aligned}$$

Notice that $12\tilde{u}+1 < 12(\tilde{u}+2\delta)$, we obtain

$$\frac{1}{(\tilde{u}+1)\cdots(\tilde{u}+2\delta)} = \frac{\tilde{u}!}{(\tilde{u}+2\delta)!} \geq \sqrt{\frac{\tilde{u}}{\tilde{u}+2\delta}} \frac{\tilde{u}^{\tilde{u}} e^{2\delta}}{(\tilde{u}+2\delta)^{\tilde{u}+2\delta}} \geq \sqrt{\frac{\tilde{u}}{\tilde{u}+2\delta}} \frac{u^{\tilde{u}} e^{2\delta}}{(\tilde{u}+2\delta)^{\tilde{u}+2\delta}}.$$

Therefore

$$\begin{aligned}\frac{B(\tilde{u}+2\delta)}{B(\tilde{u})} &\geq \left(1-\frac{2\delta}{7u}\right)^{2\delta} \sqrt{\frac{\tilde{u}}{\tilde{u}+2\delta}} \frac{u^{\tilde{u}+2\delta} e^{2\delta}}{(\tilde{u}+2\delta)^{\tilde{u}+2\delta}} \\ &\geq \sqrt{\frac{\tilde{u}}{\tilde{u}+2\delta}} \exp\left(2\delta \ln\left(1-\frac{2\delta}{7u}\right) + (\tilde{u}+2\delta)\ln\left(1-\frac{2\delta+1}{\tilde{u}+2\delta}\right) + 2\delta\right).\end{aligned}$$

We notice that $u \geq \max\{0.2\ln\lambda, 5.2\}$ and

$$(n-2k)p + 0.02\ln\lambda \leq 0.2np + 0.02\ln\lambda \leq 0.5u + 0.02\ln\lambda < 0.6u.$$

Thus $\delta < u$ and $\tilde{u} - \delta \geq 0.4u - 1 > 1$. Since $\tilde{u}-\delta$ is an interger, $\tilde{u}-\delta \geq 2$. Therefore $2\delta/(7u) < 1/2$ and $(2\delta+1)/(\tilde{u}+2\delta) < 2/3$. By Lemma 2.14 item 3 and 4, we have $\ln(1-x) \geq -x-x^2$ for $0 \leq x \leq 2/3$ and $\ln(1-x) \geq -3x/2$ for $0 \leq x \leq 1/2$. We compute

$$\begin{aligned}2\delta\ln\left(1-\frac{2\delta}{7u}\right) &\geq -\frac{3}{2}\cdot\frac{4\delta^2}{7u} \geq -\frac{\delta^2}{u}, \\ (\tilde{u}+2\delta)\ln\left(1-\frac{2\delta+1}{\tilde{u}+2\delta}\right) &\geq -(2\delta+1) - \frac{(2\delta+1)^2}{\tilde{u}+2\delta} \geq -2\delta - \frac{4\delta^2}{u} - 3, \\ \frac{B(\tilde{u}+2\delta)}{B(\tilde{u})} &\geq \sqrt{\frac{1}{3}}\exp\left(-\frac{5\delta^2}{u} - 3\right) > \exp(-0.36\ln\lambda - 11).\end{aligned}$$

where in the last inequality, using $\ln(\lambda)/2 \leq \tilde{r} \leq n^2 \ln(\lambda)/(50(n-2k)^2)$, we estimate

$$\begin{aligned}
\frac{5\delta^2}{u} &\leq \frac{5((n-2k)\tilde{r}/n + 0.02\ln\lambda + 1)^2}{k\tilde{r}/n} \\
&= \frac{5n}{k}\left(\frac{(n-2k)^2\tilde{r}}{n^2} + \frac{2(n-2k)0.02\ln\lambda}{n} + \frac{(0.02\ln\lambda)^2}{\tilde{r}} + \frac{2(n-2k)}{n} + \frac{2\cdot 0.02\ln\lambda}{\ln(\lambda)/2} + \frac{1}{\tilde{r}}\right) \\
&\leq \frac{5}{0.4}\left(\frac{\ln\lambda}{50} + 2\cdot 0.2\cdot 0.02\ln\lambda + \frac{0.02^2\ln\lambda}{0.5} + 0.4 + 0.08 + \frac{1}{13}\right) \leq 0.36\ln\lambda + 7.
\end{aligned}$$

Recalling $F(u+\delta) \geq \delta B(\tilde{u}+2\delta)$ and $\delta B(\tilde{u}) \geq F(u) - F(u+\delta)$, we compute

$$F(u+\delta) \geq \delta B(u+2\delta) \geq \delta e^{-0.36\ln\lambda-11} B(u+2\delta) \geq e^{-0.36\ln\lambda-11}(F(u) - F(u+\delta)).$$

Since $F(u) \geq 1/4$ and $\lambda > e^{10}$, we obtain

$$F(u+\delta) \geq \frac{e^{-0.36\ln\lambda-11}F(u)}{1+e^{-0.36\ln\lambda-11}} \geq \frac{\lambda^{-0.36}e^{-11}/4}{1+e^{-0.36\cdot 10-11}} > \frac{\lambda^{-0.36}}{e^{13}}.$$

Using $\Pr(X^+ - X^- \geq 0.02\ln\lambda) \geq \lambda^{-0.36}/(4e^{13})$, we bound

$$\begin{aligned}
\Pr(\tilde{\Delta}(\lambda/2,k,r/2) \geq 0.02\ln\lambda \mid k) &\geq 1 - (1-\lambda^{-0.36}/(4e^{13}))^{\lambda/2} \\
&\geq 1 - (1-e^{-0.36\cdot 10-13}/4)^{e^{10}/2} > 1.7\cdot 10^{-4}.
\end{aligned}$$

Finally $E(\Delta \mid k) \geq 1.7\cdot 10^{-4}\cdot 0.02\ln\lambda > 10^{-6}\ln\lambda$. □

We now extend the lemma to the whole region of $n/\ln\lambda \leq k < n/2$. If $k < 2n/5$ the situation becomes easier because $4 \leq c_2(k) < 100$ and every r in the smaller range $[c_1(k)\ln\lambda, \ln(\lambda)/2]$ provides at least an expected fitness improvement that is logarithmic in λ. Together with the previous lemma, we obtain the following statement for the drift in the whole region $n/\ln\lambda \leq k < n/2$.

Lemma 4.5. *Let n be sufficiently large, $n/\ln\lambda \leq k < n/2$ and $\lambda \geq e^{10}$. If $c_1(k)\ln\lambda \leq r \leq c_2(k)\ln(\lambda)/100$ with $c_1(k), c_2(k)$ defined as in Lemma 4.3, then*

$$E(\Delta \mid k) \geq 10^{-6}\ln(\lambda)/\ln(en/k).$$

Proof. If $r > \ln(\lambda)$, then $c_2(2n/5) = 100$ implies $k > 2n/5$ and the claim follows from Lemma 4.4. Hence let us assume $r \leq \ln(\lambda)$ in the remainder and compute $\tilde{\Delta}(\lambda/2,k,\tilde{r})$ with $\tilde{r} = r/2$.

We consider the probability $Q(k,i,\tilde{r})$ of creating from a parent with distance k an offspring with fitness distance at most $k-i$ via standard bit mutation with mutation rate $p = \tilde{r}/n$. Let $i := \max\{1, \lfloor c_1(k)\ln(\lambda)/2\rfloor\} \leq \max\{1, r/2\} = \tilde{r}$. If $\lfloor c_1(k)\ln(\lambda)/2\rfloor < 1$, using $(1-p)^n \geq e^{-pn-p^2n} \geq (1-o(1))e^{-\tilde{r}}$ by Lemma 2.143, we compute

$$Q(k,1,\tilde{r}) \geq kp(1-p)^n \geq \frac{k\tilde{r}}{n}(1-p)^n \geq \frac{(1-p)^n}{\ln\lambda} \geq \frac{(1-o(1))e^{-\ln(\lambda)/2}}{\ln\lambda} \geq \frac{1}{\lambda}.$$

Otherwise if $\lfloor c_1(k)\ln(\lambda)/2\rfloor > 1$, using $\binom{k}{i} \geq (k/i)^i$, we obtain

$$\begin{aligned} Q(k,i,\tilde{r}) &\geq \binom{k}{i} p^i (1-p)^n \geq \left(\frac{k}{i}\cdot\frac{\tilde{r}}{n}\right)^i e^{-\tilde{r}-p^2 n} \geq \left(\frac{k}{n}\right)^i e^{-\tilde{r}-p^2 n} \\ &\geq \left(\frac{k}{en}\right)^i e^{-\tilde{r}} \geq \left(\frac{k}{en}\right)^{\frac{\ln\lambda}{4\ln(en/k)}} e^{-\tilde{r}} = e^{-\ln(\lambda)/4-\tilde{r}} \geq e^{-\ln\lambda} = \frac{1}{\lambda}. \end{aligned}$$

Hence, $\Pr(\tilde{\Delta}(\lambda/2,k,\tilde{r}) \geq i \mid k) \geq 1-(1-1/\lambda)^{\lambda/2} \geq 1-e^{-1/2} > 0.3$. We notice that $i \geq c_1(k)\ln(\lambda)/4$, since $i = 1 \geq c_1(k)\ln(\lambda)/4$ when $c_1(k)\ln\lambda < 2$ and $i = \lfloor c_1(k)\ln(\lambda)/2\rfloor \geq c_1(k)\ln(\lambda)/4$ when $c_1(k)\ln\lambda \geq 2$. Consequently

$$E(\Delta \mid k) > \Pr(\tilde{\Delta}(\lambda/2,k,\tilde{r}) \geq i \mid k)\cdot i \geq 0.3\cdot c_1(k)\ln(\lambda)/4 \geq 10^{-6}\ln(\lambda)/\ln(en/k).$$

□

If we only consider generations that use a rate within the right region, we can bound the expected runtime to reach $k \leq n/\ln\lambda$ by $O(n/\log\lambda)$ since the drift on the fitness is of order $\log\lambda$. The following theorem shows that the additional time spent with adjusting the rate towards the right region does not change this bound on the expected runtime.

Theorem 4.6. *The (1+λ) EA with self-adjusting mutation rate reaches a* OneMax*-value of $k \leq n/\ln\lambda$ within an expected number of $O(n/\log\lambda)$ iterations. This bound is valid regardless of the initial mutation rate.*

Proof. Let us denote by $k(x)$ the fitness distance of a search point $x \in \{0,1\}^n$.

We first argue that it takes an expected number of at most $O(\sqrt{n})$ iterations to reach a fitness distance of less than $n/2$. To this end, consider a parent x with fitness distance $k(x) \geq n/2$. Let $2 \leq r \leq n/4$ be the current rate and let $\tilde{r} \in \{r/2, 2r\}$. Let y be an offspring obtained from x via standard bit mutation with mutation rate $\tilde{r}/n$. Let

$$\begin{aligned} X^+ &:= |\{i \in \{1,\dots,n\} \mid x_i = 1 \wedge y_i = 0\}|, \\ X^- &:= |\{i \in \{1,\dots,n\} \mid x_i = 0 \wedge y_i = 1\}|. \end{aligned}$$

Then the fitness improvement is $k(x)-k(y) = X^+ - X^-$ and both X^+ and X^- follow binomial distributions. From elementary properties of the binomial distribution, see, e.g., Lemma 2.16, we have $\Pr(X^+ \geq E(X^+)+1) = \Omega(1)$ and $\Pr(X^- \leq E(X^-)) = \Omega(1)$, and these are independent events. Hence with constant probability, we have

$$\begin{aligned} k(y) &= k(x) - X^+ + X^- \leq k(x) - E(X^+) - 1 + E(X^-) \\ &\leq k(x) - 1 - (2k(x)-n)\tfrac{\tilde{r}}{n} \leq k(x) - 1. \end{aligned}$$

Clearly, for the best offspring z out of the λ offspring generated in this iteration, we have $k(z) \leq k(y)$. Consequently, in each iteration starting with a parent x with $k(x) \geq n/2$, with constant probability we gain at least one fitness level. By the additive drift theorem (see Theorem 2.4 and the subsequent discussion), from the initial random search point x_0 it takes an expected $O(\max\{0, k(x_0)-n/2+1\})$ iterations to reach a parent with fitness distance less than $n/2$. Since $X := k(x_0) \sim \mathrm{Bin}(n,1/2)$,

we have

$$E(\max\{0, X - n/2\}) = \tfrac{1}{2}E(|X - E(X)|) \leq \tfrac{1}{2}\sqrt{E((X - E(X))^2)}$$
$$= \tfrac{1}{2}\sqrt{\mathrm{Var}(X)} = \tfrac{1}{4}\sqrt{n},$$

where we first exploited the symmetry of X and then the well-known estimate $E(Y)^2 \leq E(Y^2)$ valid for all random variables Y, in particular, for $Y = |X - E(X)|$. By the law of total expectation, the expected time to reach a search point with k-value below $n/2$ is $O(\sqrt{n})$.

Without loss of generality we can now assume $k < n/2$ for the initial state. Our intuition is that once we begin to use a rate $r \in [(c_1(k)/2)\ln\lambda, c_2(k)\ln\lambda]$ at some distance level k, we will have a considerable drift on the OneMax-value and the strong drift on the rate keeps r within or close to this interval. After we make progress and k decreases to a new level, the corresponding c_1 and c_2 decrease, and the algorithm may take some time to readjust r into new bounds.

We consider the stochastic process $X_t = \lfloor \log_2(r_t) \rfloor$ and the current OneMax-value Y_t. According to Lemma 4.3 item 1 and 2, there exists $\varepsilon = \Omega(1)$ such that

$$\Pr(X_{t+1} - X_t \mid X_t; X_t < \log_2(c_1(K_t)\ln\lambda) - 1) \geq \varepsilon,$$
$$\Pr(X_t - X_{t+1} \mid X_t; X_t > \log_2(c_2(K_t)\ln\lambda)) \geq \varepsilon.$$

Let $k_0 > k_1 > \cdots > k_N$ be all the different OneMax-values taken by Y_t until for the first time $Y_t \leq n/\ln\lambda$. By the additive drift theorem, it takes at most $O(\log n)$ iterations to have $r_t \in [(c_1(k_0)/2)\ln\lambda, c_2(k_0)\ln\lambda]$, regardless of how we set the initial rate. Since $c_1(Y_t)$ is non-increasing, $r_t \geq c_1(Y_t)\ln(\lambda)/2$ implies $r_t \geq c_1(Y_{t+1})\ln(\lambda)/2$. Thus, no readjustments are necessary to satisfy the lower bound $r_t \geq c_1(Y_t)\ln(\lambda)/2$ once we have $r_t \geq c_1(k_0)\ln(\lambda)/2$. We now regard the upper bound condition $r_t \leq c_2(Y_t)\ln(\lambda)$. Let (k_{s_i}) be the subsequence consisting of all k_{s_i} such that $\{t \mid r_t \leq c_2(Y_t)\ln(\lambda), Y_t = k_{s_i}\} \neq \emptyset$. If k_i is in the subsequence, once $r_t \leq c_2(Y_t)\ln(\lambda)$ is achieved, the runtime until the first time $Y_t = k_{i+1}$ can be computed using the occupation lemma and the variable drift theorem. After that, it takes at most $O(\log(c_2(k_i)/c_2(k_{i+1})))$ more iterations to readjust r_t from $c_2(k_i)\ln(\lambda)$ to $c_2(k_{i+1})\ln(\lambda)$ by the additive drift theorem. Similar arguments hold for k_{i+1} if it is also in the subsequence. Otherwise let $k_j < k_{i+1}$ denote the next OneMax-values after k_i in the subsequence. By definition, the fitness distance decreases from k_{i+1} to k_j before the rate being adjusted into the corresponding range. This means that improvements in distance are made during readjustment. Analogous to above, the expected (readjusting) time to decrease the rate from $c_2(k_{i+1})\ln(\lambda)$ to $c_2(k_j)\ln(\lambda)$ is $O(\log(c_2(k_{i+1})/c_2(k_j)))$. Therefore, the expected number of total readjusting time is at most

$$\sum_{i=1}^{s_{i+1}\leq N} O\left(\log\left(\frac{c_2(k_{s_i})}{c_2(k_{s_{i+1}})}\right)\right) = O\left(\log\left(\frac{c_2(k_0)}{c_2(k_N)}\right)\right) = O(\log n).$$

When computing the expected number of non-adjusting iterations until $Y_t = k_N$, we choose a constant $b \in \mathbb{N}_{\geq 2}$ large enough such that $2e^{-2b\varepsilon/3} \leq 1/2 - \delta/2$ holds for some positive constant $\delta > 0$. When $r_t \in [(c_1(k)/2)\ln\lambda, c_2(k)\ln\lambda]$ and $Y_t = k$ for

some t, then by Lemma 2.11, we obtain

$$\Pr(r_{t'} \geq 2^{b+1}c_2(k)\ln\lambda) \leq 2e^{-2b\varepsilon/3} \text{ and}$$
$$\Pr(r_{t'} \leq 2^{-b-2}c_1(k)\ln\lambda) \leq 2e^{-2b\varepsilon/3} \text{ for } t' \geq t.$$

Hence, we obtain that $r_{t'} \in [2^{-b-2}c_1(k)\ln\lambda, 2^{b+1}c_2(k)\ln\lambda]$ happens with probability at least δ. We see that there are at most $\lceil \log_2(100) \rceil$ steps between being in the range $[(c_1(k)/2)\ln\lambda, c_2(k)\ln\lambda]$ and being in the smaller range $[c_1(k)\ln\lambda, c_2(k)\ln(\lambda)/100]$ which is described in Lemma 4.5. If $r_t \in [2^{-b-2}c_1(k)\ln\lambda, 2^{b+1}c_2(k)\ln\lambda]$, it takes at most a constant number of iterations α in expectation to reach the smaller range $[c_1(k)\ln\lambda, c_2(k)\ln(\lambda)/100]$ because our mutation scheme employs a 50% chance to perform a random step of the mutation rate. Based on Lemma 4.5, the fitness drift at distance k of all rates within the narrow region is at least $10^{-6}\ln(\lambda)/\ln(en/k)$. This contributes to an average drift of at least

$$10^{-6} \cdot \frac{\ln(\lambda)}{\ln(en/k)} \cdot \frac{\delta}{1+\alpha} = \Omega\left(\frac{\log(\lambda)}{\log(n/k)}\right)$$

for all random rates at distance k. Using the variable drift theorem (Theorem 2.4), we estimate the runtime as

$$\begin{aligned}
&O\left(\frac{\log(\log\lambda)}{\log\lambda} + \int_{n/\log\lambda}^{n/2} \frac{\log(n/k)\mathrm{d}k}{\log\lambda}\right) = O\left(\frac{1}{\log\lambda}\int_{n/\log\lambda}^{n/2}\Big(\log(n) - \log(k)\Big)\mathrm{d}k\right)\\
=&O\left(\frac{1}{\log\lambda}\Big(\log(n)k - k\log(k) + \log(k)\Big)\Big|_{n/\log\lambda}^{n/2}\right)\\
=&O\left(\frac{1}{\log\lambda}\left(\log(n)\left(\frac{n}{2} - \frac{n}{\log\lambda}\right) - \frac{n}{2}\log\left(\frac{n}{2}\right) + \frac{n}{\log\lambda}\log\left(\frac{n}{\log\lambda}\right) + \log\left(\frac{n/2}{n/\log\lambda}\right)\right)\right)\\
=&O\left(\frac{1}{\log\lambda}\left(\frac{n\log 2}{2} - \frac{n\log(\log\lambda)}{\log\lambda} + \log(\log\lambda)\right)\right) = O\left(\frac{n}{\log\lambda}\right).
\end{aligned}$$

We notice that the expected runtime for adjusting r_t is $O(\log n) = O(n/\log\lambda)$. Therefore, the total runtime is $O(n/\log\lambda)$ in expectation. □

4.4 The Middle Region

In this section we estimate the expected number of generations until the number of one-bits has decreased from $k \leq n/\ln\lambda$ to $k \leq n/\lambda$. We first claim that the right region for r is $1 \leq r \leq \ln(\lambda)/2$. Hence, the (1+λ) EA is not very sensitive to the choice of r here. Intuitively, this is due to the fact that a total fitness improvement of only $O(n/\log\lambda)$ suffices to cross the middle region, whereas an improvement of $\Omega(n)$ is needed for the far region.

We estimate the drift of the fitness in Lemma 4.7 and apply that result afterwards to estimate the number of generations to cross the region.

Lemma 4.7. *Let $n/\lambda \leq k \leq n/\ln\lambda$, $\lambda \geq 26$ and $2 \leq r \leq \ln\lambda$. Then*

$$E(\Delta \mid k) \geq \min\left\{\frac{1}{8}, \frac{\sqrt{\lambda}k}{32n}\right\}.$$

Proof. We aim at computing $\tilde{\Delta}(\lambda/2, k, \tilde{r})$ with $\tilde{r} = r/2$. The probability that no zero-bit flips in a single offspring stemming from rate $\tilde{r}$ is $(1-\tilde{r}/n)^{n-k} \geq e^{-\tilde{r}} \geq 1/\sqrt{\lambda}$. We regard the number Z of offspring of rate $\tilde{r}$ that have no flipped zeros. The expectation $E(Z)$ is at least $(1/\sqrt{\lambda}) \cdot (\lambda/2) = \sqrt{\lambda}/2$. Applying Chernoff bounds (Theorem 2.8), we observe that Z exceeds $\lambda_0 := \sqrt{\lambda}/4$ with probability at least $1 - \exp(-\sqrt{\lambda}/16) > 1/4$ since $\lambda \geq 26$. Assuming this to happen, we look at the first λ_0 offspring without flipped zeros. For $i \in \{1, \dots, \lambda_0\}$ let X_i be the number of flipped ones in the i-th offspring. Then the X_i are i.i.d. with $X_i \sim \mathrm{Bin}(k, \tilde{r}/n)$. Let $X^* := \max\{X_i\}$. The expectation of X^* is analyzed in [GW17, Lemma 4, part 2]; more precisely the following statement is shown for constant $\tilde{r}$:

$$\text{If } \frac{\lambda_0 k \tilde{r}}{n} \geq \alpha \text{ then } E(X^*) \geq \frac{\alpha}{1+\alpha}.$$

Since $\tilde{r}$ is not necessarily assumed constant here, we generalize the result by closely following the proof in [GW17]. Note that $X^* \geq 1$ if at least one of the offspring does not flip a zero-bit. Hence,

$$\begin{aligned} E(X^*) &\geq 1 - \left(\left(1 - \frac{\tilde{r}}{n}\right)^k\right)^{\lambda_0} \geq 1 - \left(\left(1 - \frac{\tilde{r}}{n}\right)^{\alpha n/(\lambda_0 \tilde{r})}\right)^{\lambda_0} \\ &\geq 1 - \left(e^{-\alpha/\lambda_0}\right)^{\lambda_0} = 1 - e^{-\alpha} \geq 1 - \frac{1}{1+\alpha} = \frac{\alpha}{1+\alpha}, \end{aligned}$$

where the second inequality used the assumption $\lambda_0 k \tilde{r}/n \geq \alpha$, the third one $e^x \geq 1 + x$ for $x \in \mathbb{R}$. and the fifth one the equivalent to the previous inequality $e^{-x} \leq 1/(1+x)$. Hence, Lemma 4, part 2 in [GW17] holds also for arbitrary $\tilde{r}$.

Setting $\alpha := \lambda_0 k \tilde{r}/n$, we now distinguish between two cases. If $\alpha \geq 1$, we obtain $E(X^*) \geq \alpha/(1+\alpha) \geq 1/2$; otherwise we have $1 + \alpha < 2$, hence $E(X^*) \geq \alpha/2 = \lambda_0 k \tilde{r}/(2n)$. Thus,

$$E(X^*) \geq \min\{1/2, \sqrt{\lambda} k/(8n)\}.$$

Hence, using the law of total probability, we obtain the lower bound on the drift for the middle region. □

We now use our result on the drift to estimate the time spent in this region. We notice that $c_2(k) = 4 + o(1)$ when $k = o(n)$. This means we will often have $r_t \in [2, \ln(\lambda)]$ which provides the drift we need.

Theorem 4.8. *Let $\lambda \geq 26$ and $\lambda = n^{O(1)}$. Assume $k \leq n/\ln\lambda$ for the current* OneMax*-value of the self-adjusting (1+λ) EA. Then the expected number of generations until $k \leq n/\lambda$ is $O(n/\log\lambda)$.*

Proof. For $k \leq n/\ln\lambda$ the upper bound from Lemma 4.3 is $c_2(k) = 4/(1 - 2/\ln\lambda) < 11$. According to the lemma we have for $X_t := \lceil \log_2 r_t \rceil$ that $E(X_t - X_{t+1} \mid X_t; X_t > \log_2(c_2(k)\ln\lambda)) \geq 2\varepsilon = \Omega(1)$. The additive drift theorem yields that in $O(\log n)$ time we have $r_t \leq c_2(k)\ln\lambda$. We choose b large enough such that $2e^{-2b\varepsilon/3} \leq 1 - \delta$ holds for some positive constant $\delta > 0$ and note that b is constant. Applying Lemma 2.11 we obtain $\Pr(r_t \geq 2^b c_2(k)\ln\lambda) \leq 2e^{-2b\varepsilon/3}$ and $r_t \leq 2^b c_2(k)\ln\lambda$ happens with probability at least δ. Once $r_t < 2^b c_2(k)\ln\lambda < 2^b(11\ln\lambda)$ it takes at most a constant number of iterations α in expectation to draw r_t to $\ln(\lambda)$ or less. According to Lemma 4.7 this ensures a drift of at least $(1/4)\min\{1/2, \sqrt{\lambda}k/(8n)\} \geq (1/32)\min\{1, \sqrt{\lambda}k/n\}$, which implies an average drift of at least $c\min\{1, \sqrt{\lambda}k/n\}$ over all random rates at distance k, where $c > 0$ is a constant. Considering

$\min\{1, \sqrt{\lambda}k/n\}$, the minimum is taken on the first argument if $k > n/\sqrt{\lambda}$, and on the second if $k < n/\sqrt{\lambda}$.

We are interested in the expected time to reduce the OneMax-value to at most n/λ. To ease the application of drift analysis, we artificially modify the process and make it create the optimum when the state (OneMax-value) is strictly less than n/λ. Clearly, the first hitting time of state at most n/λ does not change by this modification. Applying the variable drift theorem (Theorem 2.4) with $x_{\min} = n/\lambda$, $X_0 = k \leq n/\ln\lambda$ and $h(x) = c\min\{1, \sqrt{\lambda}k/n\}$, the expected number of generations to reach state at most n/λ is bounded from above by

$$\frac{n/\lambda}{c\sqrt{\lambda}(n/\lambda)/n} + \int_{n/\lambda}^{n/\sqrt{\lambda}} \frac{n}{c\sqrt{\lambda}x}\,\mathrm{d}x + \int_{n/\sqrt{\lambda}}^{n/\ln\lambda} \frac{1}{c}\,\mathrm{d}x = O\left(\frac{n}{\sqrt{\lambda}}\right) + O\left(\frac{n\log\lambda}{\sqrt{\lambda}}\right) + O\left(\frac{n}{\log\lambda}\right),$$

which is $O(n/\log\lambda)$. The overall expected number of generations spent is $O(\log n + n/\log\lambda) = O(n/\log\lambda)$ since $\lambda = n^{O(1)}$ by assumption. □

4.5 The Near Region

In the near region, we have $k \leq n/\lambda$. Hence, the fitness is so low that we can expect only a constant number of offspring to flip at least one of the remaining one-bits. This assumes constant rate. However, higher rates are detrimental since they are more likely to destroy the zero-bits of the few individuals flipping one-bits. Hence, we expect the rate to drift towards constant values, as shown in the following lemma.

Lemma 4.9. *Let $k \leq n/\lambda$, $\lambda \geq 45$ and $4 \leq r_t \leq \ln(\lambda)/4$. Then the probability that $r_{t+1} = r_t/2$ is at least 0.5099.*

Proof. To prove the claim we exploit the fact that only few one-bits are flipped in both subpopulations. Using $r := r_t$, we shall argue as follows. With sufficiently high (constant) probability, (i) the $2r$-subpopulation contains no individual strictly better than the parent, that is, with fitness less than k, and (ii) all $2r$-offspring with fitness at least k are identical to the parent. Conditional on this, either the $r/2$-population contains individuals with fitness less than k and the winning individual surely stems from this subpopulation, or the $r/2$-population contains no better offspring. In the latter case, we argue that there are many more individuals with fitness exactly k in the $r/2$-population than in the $2r$-population, which gives a sufficiently high probability for taking the winning individual from this side (as it is chose uniformly at random from all offspring with fitness k).

Let $N_{r/2}$ and N_{2r} be the number of offspring that did not flip any zero-bits using rate $r/2$ and $2r$, respectively. Then $E(N_{r/2}) = (\lambda/2)(1 - r/(2n))^{n-k} \geq (1 - o(1))\lambda e^{-r/2}/2$, since $k \geq 1$ and

$$\left(1 - \frac{r}{2n}\right)^{n-1} \geq e^{-\frac{r}{2}}\left(1 - \frac{r}{2n}\right)^{\frac{r}{2}-1} \geq e^{-\frac{r}{2}}\left(1 - \frac{c\ln n}{8n}\right)^{\frac{c\ln n}{8}-1} = (1 - o(1))e^{-\frac{r}{2}},$$

where we used that $r \leq \ln\lambda/4$ and $\lambda = n^{O(1)}$, i. e. $\ln\lambda \leq c\ln n$ for some constant c. Using $k \leq n/\lambda$ we get $E(N_{2r}) = (\lambda/2)(1 - 2r/n)^{n-k} \leq (\lambda/2)e^{-2r(1-1/\lambda)}$. In fact, we can discriminate $N_{r/2}$ and N_{2r} by using Theorem 2.8 in the following way: we

have

$$\begin{aligned}\Pr\left(N_{r/2} \leq \frac{\lambda}{4}e^{-\frac{r}{2}}\right) &\leq \exp\left(-(1-o(1))^3\frac{\lambda}{16}e^{-\frac{r}{2}}\right)\\ &\leq \exp\left(-(1-o(1))\frac{1}{16}\lambda^{7/8}\right)\\ &< 0.175,\end{aligned}$$

for sufficiently large n, since $r \leq (\ln\lambda)/4$ and $\lambda \geq 45$. Similarly, we obtain

$$\Pr\left(N_{2r} \geq \lambda e^{-2r(1-\frac{1}{\lambda})}\right) \leq \exp\left(-\frac{1}{6}\lambda^{1-\frac{1}{2}(1-\frac{1}{\lambda})}\right) < 0.312.$$

Note that $e^{-2r(1-1/\lambda)} < e^{-r/2}/4$ holds for all $r \geq 4$ and $\lambda \geq 45$. Since the offspring are generated independently, the events $N_{r/2} > \lambda e^{-r/2}/4$ and $N_{2r} < \lambda e^{-2r(1-\frac{1}{\lambda})}$ happen together with probability at least $(1-0.175)\cdot(1-0.312) \geq 0.567 =: 1-p_{\mathrm{err}_1}$. Conditioning on this and by using a union bound the probability p_{err_2} that at least one of the N_{2r} offspring that do not flip any zero-bits flips at least one one-bit can be upper bounded by

$$p_{\mathrm{err}_2} \coloneqq N_{2r}\cdot\frac{2kr}{n} \leq 2re^{-2r\left(1-\frac{1}{\lambda}\right)} \leq 0.004$$

using $k/n \leq 1/\lambda$ in the first and $r \geq 4$ and $\lambda \geq 45$ in the last inequality. By using a union bound, we find the probability p_{err_3} that at least one $2r$-offspring flips exactly one one-bit and exactly one zero-bit to be at most

$$\begin{aligned}p_{\mathrm{err}_3} &\coloneqq \frac{\lambda}{2}\frac{2rk}{n}\left(1-\frac{2r}{n}\right)^{k-1}\frac{2r(n-k)}{n}\left(1-\frac{2r}{n}\right)^{n-k-1}\\ &\leq 2r^2\left(1-\frac{2r}{n}\right)^{n-2} \leq (1+o(1))2e^{2(\ln(r)-r)} < (2+o(1))e^{-\frac{6}{5}r} < 0.017,\end{aligned}$$

for sufficiently large n, using $k/n \leq 1/\lambda$ for the first inequality. The third inequality is due to $\ln x - x \leq -(1-e^{-1})x < -(3/5)x$ for all $x > 0$ (see Lemma 2.14.a) and the last inequality stems from $r \geq 4$. The second inequality follows from

$$\left(1-\frac{2r}{n}\right)^{n-2} \leq e^{-\frac{2r}{n}(n-2)} = e^{-2r\left(1-\frac{2}{n}\right)} = (1+o(1))e^{-2r},$$

using again $r \leq \ln\lambda \leq c\ln n$ for some constant c. Let M_r be the number of such offspring. Any other fitness-decreasing flip-combinations of zeroes and ones in the $2r$-subpopulation require an offspring to flip at least two one-bits. The probability that such an offspring is created is at most

$$p_{\mathrm{err}_4} \coloneqq \frac{\lambda}{2}\binom{k}{2}\left(\frac{2r}{n}\right)^2 \leq \frac{\ln^2\lambda}{16\lambda} < 0.021,$$

using $k \leq n/\lambda$ and $r \leq \ln\lambda/4$ and the fact that $(\ln^2 x)/x$ is decreasing for $x \geq e^2$ and $\lambda \geq 45 > e^2$.

The events $N_{r/2} > \lambda e^{-r/2}/4$, $N_{2r} < \lambda e^{-2r(1-\frac{1}{\lambda})}$, $M_r = 0$, and the event that no fitness-decreasing offspring is created in the $2r$-subpopulation are sufficient to ensure that the best individual is either surely from the $r/2$-population or chosen uniformly at random from the $N_{r/2}+N_{2r}$ offspring. Conditioning on these events, we have that

the probability that the best offspring is chosen from the $r/2$-population is at least

$$\frac{N_{r/2}}{N_{r/2}+N_{2r}} \geq \frac{1}{1+4e^{-r\left(2\left(1-\frac{1}{\lambda}\right)-\frac{1}{2}\right)}} > 0.988.$$

Hence, using a union bound for the error probabilities, the unconditional probability is at least

$$\frac{(1-p_{\mathrm{err}_1}-p_{\mathrm{err}_2}-p_{\mathrm{err}_3}-p_{\mathrm{err}_4})\cdot 0.988+\frac{1}{2}}{2} > 0.5099.$$

□

We note that the restriction $r_t \geq 4$ in the lemma above is not strictly necessary. Also for smaller r_t, the probability that the winning individual is chosen from the $r_t/2$-population is by an additive constant larger than $1/2$. Showing this, however, would need additional proof arguments as for smaller r_t, the event that both subpopulations contain individuals with fitness $k-1$ becomes more likely. We avoid this additional technicality by only arguing for $r_t \geq 4$, which is enough since any constant r_t is sufficient for the fitness drift we need (since we do not aim at making the leading constant precise).

In the following proof of the analysis of the near region, we use the above lemma (with quite some additional arguments) to argue that the r-value quickly reaches 4 or less and from then on regularly returns to this region. This allows to argue that in the near region we have a speed-up of a factor of $\Theta(\lambda)$ compared to the (1+1) EA, since every offspring only has a probability of $O(1/\lambda)$ of making progress (see also [DK15; GW17]).

Theorem 4.10. *Assume $k \leq n/\lambda$ for the current* OneMax*-value of the self-adjusting (1+λ) EA. Then the expected number of generations until the optimum is reached is $O(n\log(n)/\lambda + \log n)$.*

Proof. The aim is to estimate the OneMax-drift at the points in time (generations) t where $r_t = O(1)$. To bound the expected number of generations until the mutation rate has entered this region, we basically consider the stochastic process $Z_t := \max\{0, \lfloor \log_2(r_t/a)\rfloor\}$, where $a := 4$, which is the lower bound on r_t from Lemma 4.9. However, as we do not have proved a drift of Z_t towards smaller values in the region $L := (\ln\lambda)/4 \leq r_t \leq 16\ln\lambda =: U$ (where 16 is an upper bound on $c_2(k)$ from Lemma 4.5), we use the potential function

$$X_t(Z_t) := \begin{cases} Z_t & \text{if } a \leq r_t \leq L \\ \log_2(L/a) + \sum_{i=1}^{\lceil \log_2(r_t/L)\rceil} 4^{-i} & \text{if } L < r_t < U \\ \log_2(L/a) + \sum_{i=1}^{\log_2(U/L)+1} 4^{-i} + 4^{-\log_2(U/L)-1}\lfloor \log_2(r_t/a)\rfloor & \text{otherwise.} \end{cases}$$

assuming that L and U have been rounded down and up to the closest power of 2, respectively. We note that $X_t \geq \log_2(r_t/a) - 1 = \log_2(r_t/(2a))$ if $r_t \leq L$ and $X_t \geq (L/U)^2\log_2(r_t/(2a))$ in all three cases.

The potential function has a slope of 1 for $a \leq r_t \leq L$. Lemma 4.9 gives us the drift $E(X_t - X_{t+1} \mid X_t; a \leq r_t \leq L) \geq (0.5099 - 0.4901)/(2a) = \Omega(1)$. The function satisfies $X_t(Z_t) - X_t(Z_t - 1) \geq 4(X_t(Z_{t+1}) - X_t(Z_t))$ if $L < r_t < U$, which corresponds to the region where the probability of decreasing r_t by a factor of $1/2$ has only be bounded from below by $1/4$ due to the random steps. Still, $E(X_t - X_{t+1} \mid X_t; L < r_t < U) \geq (1/4)4^{-\log_2(U/L)-1} - (3/4)4^{-\log_2(U/L)} = \Omega(1)$ in this region due to the concavity of the potential function. Finally, $E(X_t - X_{t+1} \mid X_t; r_t \geq U) =$

$4^{-\log_2(U/L)-1}(1/(2a))\Omega(1) = \Omega(1)$ by Lemma 4.5. Hence, altogether $E(X_t - X_{t+1} \mid X_t; r_t \geq a) \geq \kappa$ for some constant $\kappa > 0$. As $X_0 = O(\log n)$, additive drift analysis yields an expected number of $O(\log n)$ generations until for the first time $X_t = 0$ holds, corresponding to $r_t \leq a$. We denote this hitting time by T.

We now consider an arbitrary point of time $t \geq T$. The aim is to show a drift on the OneMax-value, depending on the current OneMax-value Y_t, which satisfies $Y_t \leq n/\lambda$ with probability 1. To this end, we will use Lemma 2.11. We choose b large enough such that $2e^{-2b\cdot\kappa/4} \leq 1 - \delta$ holds for some positive constant $\delta > 0$ and note that b is constant. We consider two cases. If $X_t \leq b$, which happens with probability at least δ according to the lemma, then the bound $X_t \geq (L/U)^2(\log_2(r_t/(2a)))$ implies $r_t \leq 2^{(U/L)^2}2a2^b$. Hence, we have $r_t = O(1)$ in this case and obtain a probability of at least

$$1 - \left(1 - \binom{Y_t}{1}\left(\frac{2r_t}{n}\right)\left(1 - \frac{2r_t}{n}\right)^{n-1}\right)^{\lambda} \geq 1 - \left(1 - \Theta\left(\frac{Y_t}{n}\right)\right)^{\lambda} = \Omega(\lambda Y_t/n)$$

to improve the OneMax-value by 1, using that $Y_t = O(n/\lambda)$ and pessimistically assuming a rate of $2r_t$ in all offspring. If $X_t > b$, we bound the improvement from below by 0. Using the law of total probability, we obtain

$$E(Y_t - Y_{t+1} \mid Y_t; Y_t \leq n/\lambda) = \delta\Omega(\lambda Y_t/n) = \Omega(\lambda Y_t/n).$$

Now a multiplicative drift analysis with respect to the stochastic process on the Y_t, more precisely Theorem 2.3 using $\delta = \Theta(\lambda/n)$ and minimum state 1, gives an expected number of $O((n/\lambda)\log Y_0) = O(n\log(n)/\lambda)$ generations until the optimum is found. Together with the expected number $O(\log n)$ until the r-value becomes at most a, this proves the theorem. □

4.6 Putting Everything Together

In this section, we put together the analyses of the different regimes to prove our main result.

Proof of Theorem 4.1. The lower bound actually holds for all unbiased parallel black-box algorithms, as shown in [BLS14].

We add up the bounds on the expected number of generations spent in the three regimes, more precisely we add up the bounds from Theorem 4.6, Theorem 4.8 and Theorem 4.10, which gives us $O(n/\log\lambda + n\log(n)/\lambda + \log n)$ generations. Due to our assumption $\lambda = n^{O(1)}$ the bound is dominated by $O(n/\log\lambda + n\log(n)/\lambda)$ as suggested. □

Proof of Lemma 4.2. We basically revisit the regions of different OneMax-values analyzed in this chapter and bound the time spent in these regions under the assumption $r = \ln(\lambda)/2$. In the far region, Lemmas 4.4 and 4.5, applied with this value of r, imply a fitness drift of $\Omega(\log(\lambda)/\log(en/k))$ per generation, so the expected number of generations spent in the far region is $O(n/\log\lambda)$ as computed by variable drift analysis in the proof of Theorem 4.6.

The middle region is shortened at the lower end. For $k \geq n/\sqrt{\lambda}$, Lemma 4.7 gives a fitness drift of $\Omega(1)$, implying by additive drift analysis $O(n/\log\lambda)$ generations to reduce the fitness to at most $n/\sqrt{\lambda}$.

In the near region, which now starts at $n/\sqrt{\lambda}$, we have to argue slightly differently. Note that every offspring has a probability of at least $(1-r)^n \geq e^{-\ln(\lambda)/2+O(1)} =$

$\Omega(\lambda^{-1/2})$ of not flipping a zero-bit. Hence, we expect $\Omega(\sqrt{\lambda})$ such offspring. We pessimistically assume that the other individuals do not yield a fitness improvement; conceptually, this reduces the population size to $\Omega(\sqrt{\lambda})$ offspring, all of which are guaranteed not to flip a zero-bit. Adapting the arguments from the proof of Theorem 4.10, the probability that at least of one of these individuals flips at least a one-bit is at least

$$1 - \left(1 - \binom{Y_t}{1}\left(\frac{r_t}{n}\right)\right)^{\Omega(\sqrt{\lambda})} \geq 1 - \left(1 - \Theta\left(\frac{Y_t}{n}\right)\right)^{\Omega(\sqrt{\lambda})} = \Omega(\sqrt{\lambda} Y_t/n),$$

which is a lower bound on the fitness drift. Using the multiplicative drift analysis, the expected number of generations in the near region is $O(n \log(n)/\sqrt{\lambda})$. Putting the times for the regions together, we obtain the lemma. □

4.7 Experimental Results

Since our analysis is asymptotic in nature we performed some elementary experiments in order to see whether besides the asymptotic runtime improvement (showing an improvement for an unspecified large problem size n) we also see an improvement for realistic problem sizes. For this purpose we implemented the (1+λ) EA in C using the GNU Scientific Library (GSL) for the generation of pseudo-random numbers. In this section our performance measure is the runtime, represented by the number of generations until the optimum is found for the first time.

The first plot in Figure 4.1 displays the average runtime over 10000 runs of the self-adjusting (1+λ) EA on OneMax for $n = 5000$ as given in Algorithm 8 over $\lambda = 100, 200, \ldots, 1000$; the second plot shows the corresponding interquartile ranges to support that the results are statistically significant. We set the initial mutation rate to 2, i. e., the minimum mutation rate the algorithm can attain. Moreover, the plot displays the average runtimes of the classic (1+λ) EA using static mutation probabilities of $1/n$ and of $(\ln \lambda)/(2n)$, the latter of which is asymptotically optimal for large λ according to Lemma 4.2.

The average runtimes of both algorithms profit from higher offspring population sizes λ leading to lower average runtimes as λ increases. Interestingly, the two static settings of the classic (1+λ) EA outperform the self-adjusting (1+λ) EA for small values of λ up to $\lambda = 400$. For higher offspring population sizes the self-adjusting (1+λ) EA outperforms the classic ones, indicating that the theoretical performance gain of $\ln \ln \lambda$ can in fact be relevant in practice. Furthermore, we implemented the self-adjusting (1+λ) EA without the random steps, that is, when the rate is always adjusted according to how the best offspring are distributed over the two subpopulations. The experiments show that this variant of the self-adjusting (1+λ) EA performs generally slightly better on OneMax. Since the OneMax fitness landscape is structurally very simple, this result is not totally surprising. It seems very natural that the fitness of the best of $\lambda/2$ individuals, viewed as a function in the rate, is a unimodal function. In this case, the advantage of random steps to be able to leave local optima of this function is not needed. On the other hand, of course, this observation suggests to try to prove our performance bound rigorously also for the case without random rate adjustments. We currently do not see how to do this. Lastly, we implemented the (1+λ) EA using the fitness-depending mutation rate $p = \max\{\frac{\ln \lambda}{n \ln(en/d)}, \frac{1}{n}\}$ as presented in [BLS14]. The experiments suggest that this scheme outperforms all other variants considered.

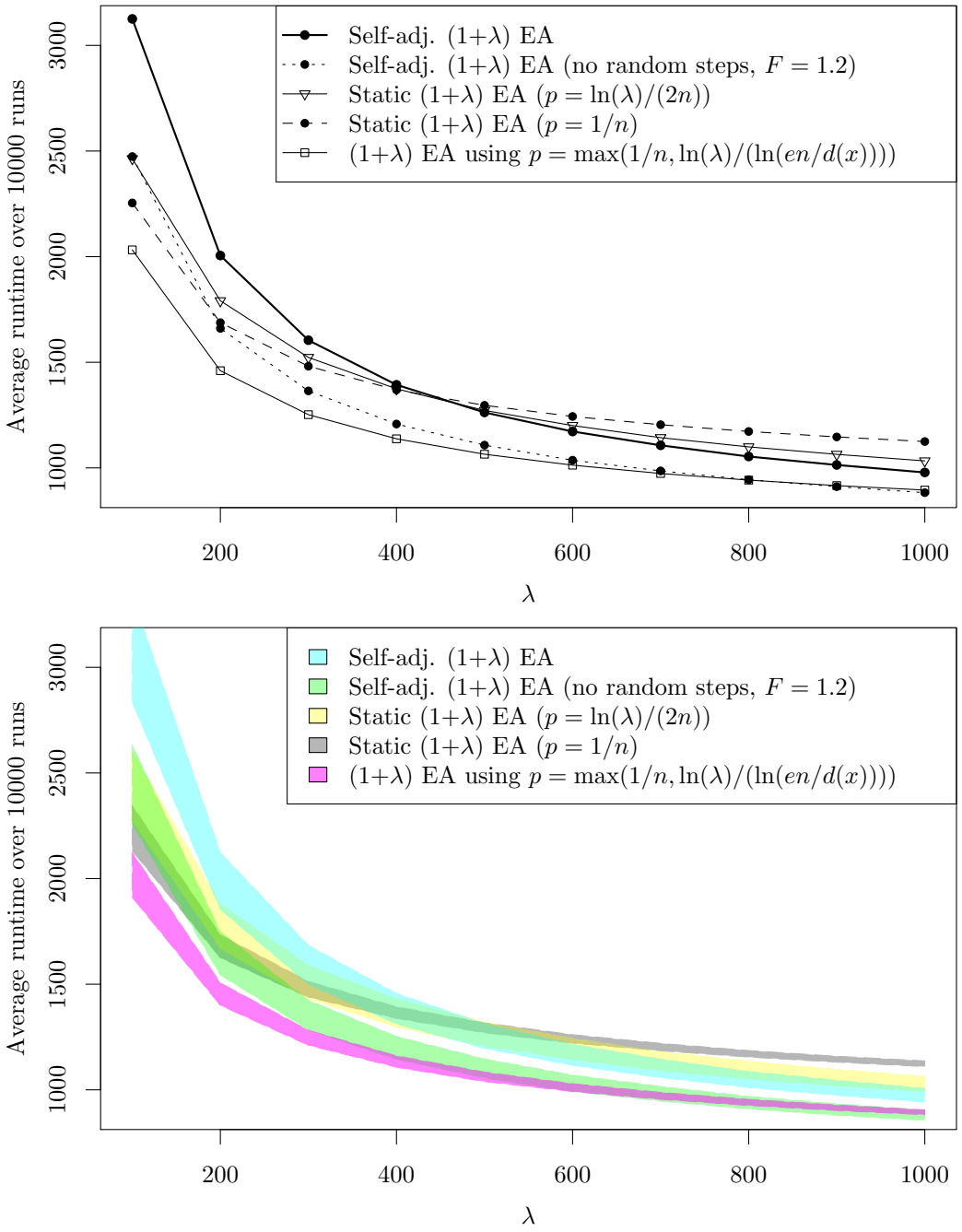

FIGURE 4.1: Static and Self-adjusting (1+λ) EA average runtime comparison ($n = 5000$) on ONEMAX and the corresponding interquartile ranges.

Additionally we implemented another variant of the self-adjusting (1+λ) EA using three equally-sized subpopulations i. e. the additional one is using (that is, exploiting) the current mutation rate. We compared this variant with the self-adjusting (1+λ) EA, both with and without using random steps. The results are shown in Figure 4.2, again with respect to average runtimes and interquartile ranges. The experiments suggest that the variant using three subpopulations outperforms the self-adjusting (1+λ) EA slightly for small population sizes. For very high population sizes, using just two subpopulations seems to be a better choice.

To gain some understanding on how the parameters influence the runtime, we implemented the self-adjusting (1+λ) EA using different mutation rate update factors, that is, we consider the self-adjusting (1+λ) EA as given in Algorithm 8 where the mutation rate r_t is increased or decreased by some factor F (instead of the choice $F = 2$ made in Algorithm 8). Note that we do not change the rule that we use the rates $r/2$ and $2r$ to create the subpopulations. Furthermore, after initialization, the algorithm starts with rate F and the rate is capped below by F and above by $1/(2F)$ during the run, accordingly.

The results are shown in Figure 4.3. The plot displays the average runtime over 10000 runs of the self-adjusting (1+λ) EA on OneMax for $n = 5000$ over $\lambda = 100, 200, \ldots, 1000$ using the update factors $F = 2.0, 1.5, 1.01$. The plot suggests that lower values of F yield a better performance. This result is not immediately obvious. Clearly, a large factor F implies that the rate changes a lot from generation to generation (namely by a factor of F). These changes prevent the algorithm from using a very good rate for several iterations in a row. On the other hand, a small value for F implies that it takes longer to adjust the rate to value that is far from the current one. We also performed experiments for $F > 2$ and observed even worse runtimes than for $F = 2.0$. The figure does not display the outcomes of these additional experiments since this would impair the readability of the diagram.

Finally, to illustrate the nontrivial development of the rate during a run of the algorithm we plotted the rate of three single runs of the self-adjusting (1+λ) EA using different factors F over the fitness in Figure 4.4. Since the algorithm initialized with the rate F, the rate increases after initialization and decreases again with decreasing fitness-distance to the optimum. The plot suggests that for higher values of F the rate is more unsteady due to the greater impact of the rate adjustments while smaller rate updates yield a more stable development of the rate. Interestingly, for all three values of F, the rates seem to correspond to the same rate after the initial increasing phase from F. Note that this illustration does not indicate the actual runtime. In fact, the specific runtimes are 19766 for $F = 1.01$, 19085 for $F = 1.05$ and 19857 for $F = 1.1$. A similar, more pronounced behaviour can be seen for $F = 2.0$; we chose these particular values of F for illustrative purposes since for $F = 2.0$ the variance in the rate can be visually confusing for the reasons given above.

While we would draw from this experiment the conclusion that a smaller choice of F is preferable in a practical application of our algorithm, the influence of the parameter on the runtime is not very large. So it might not be worth optimizing it and rather view Algorithm 8 as a parameter-less algorithm.

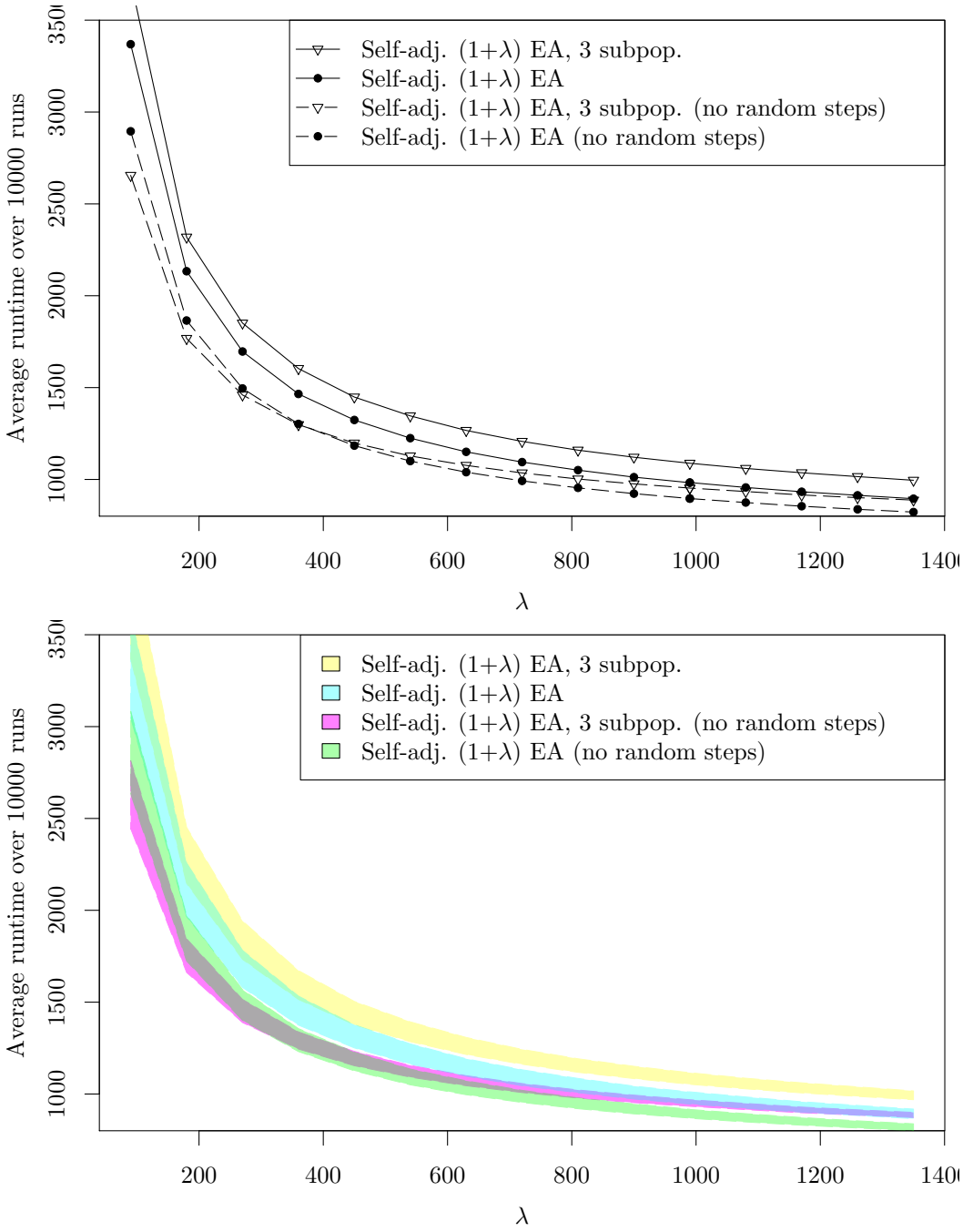

FIGURE 4.2: Average runtime of the self-adjusting (1+λ) EA with two and three subpopulations each with and without random steps on ONEMAX ($n = 5000$) with corresponding interquartile ranges.

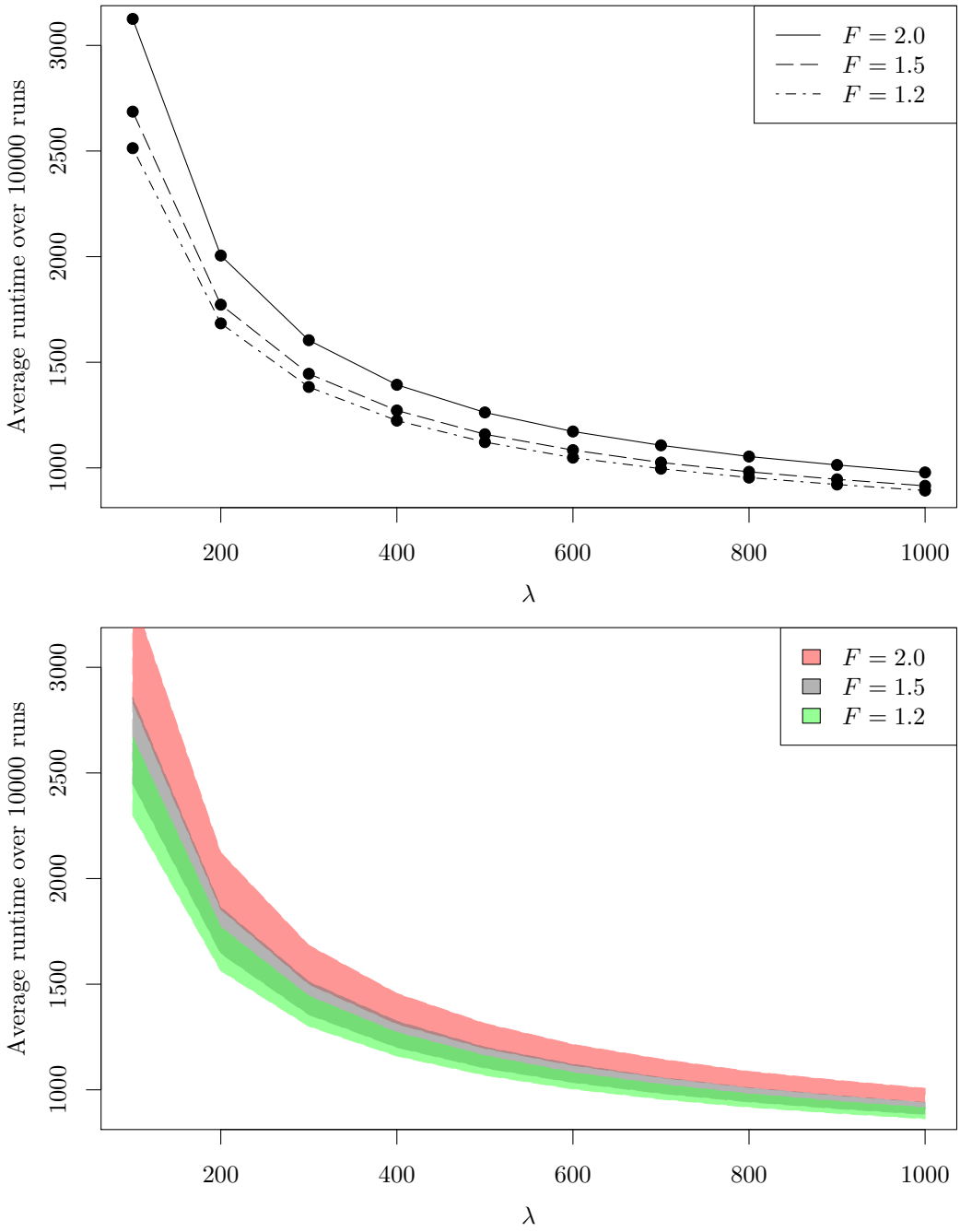

FIGURE 4.3: Average runtime of the self-adjusting (1+λ) EA with different mutation rate update factors F on ONEMAX ($n = 5000$) with corresponding interquartile ranges

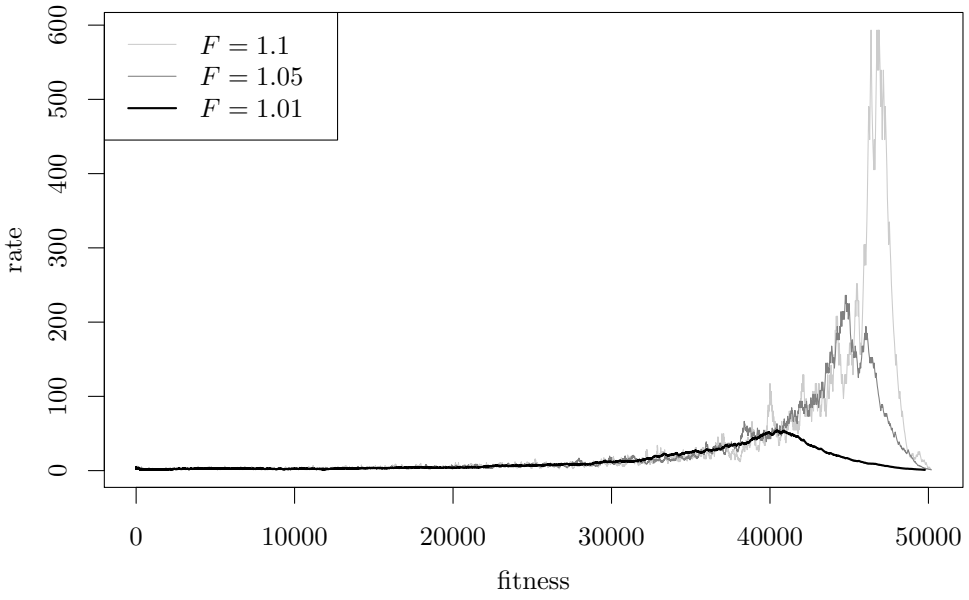

FIGURE 4.4: Development of the rate over the fitness of three example runs of the self-adjusting (1+λ) EA on ONEMAX ($n = 100000$, $\lambda = 1000$), using different factors F

Chapter 5

Self-adapting (1,λ) EA

In this chapter we propose a version of the $(1, \lambda)$ evolutionary algorithm (EA) with a natural self-adaptive choice of the mutation rate and prove that it optimizes the classic OneMax benchmark problem in a runtime that is asymptotically optimal among all λ-parallel black-box optimization algorithms and that is, for $\lambda \geq C \ln n$ with a large enough constant C and $\lambda = n^{O(1)}$, better than the runtimes of the (1,λ) EA and the $(1 + \lambda)$ EA for all static choices of the mutation rate. It obviously cannot beat the (also asymptotically optimal) runtimes of the $(1 + \lambda)$ EA with fitness-dependent mutation rate of Badkobeh, Lehre, and Sudholt [BLS14] and of the $(1 + \lambda)$ EA with self-adjusting (exogenous) mutation rate we proposed in Chapter 4. Hence the good news of our result is that this optimal runtime could be obtained in a generic manner. Note that both the fitness-dependent mutation rate of [BLS14] and the self-adjusting rate of Chapter 4 with its mix of random and greedy rate adjustments would have been hard to find without a deeper understanding of the mathematics of these algorithms.

Not surprisingly, the proof of our main result has some similarity to the analysis of the self-adjusting $(1 + \lambda)$ EA of Chapter 4. In particular, we also estimate the expected progress in one iteration and use variable drift analysis. Also, we need a careful probabilistic analysis of the progress obtained from different mutation rates to estimate which rate is encoded in the new parent individual (unfortunately, we cannot reuse the analysis of Chapter 4 since it is not always strong enough for our purposes). The reason, and this is also the main technical challenge in this work, is that the (1,λ) EA can lose fitness in one iteration. This happens almost surely when the mutation rate is too high. For this reason, we need to argue more carefully that such events do not happen regularly. To do so, among several new arguments, we also need a stronger version of the occupation probability result [KLW15, Theorem 7] since (i) we need sharper probability estimates for the case that movements away from the target are highly unlikely and (ii) for our process, the changes per time step cannot be bounded by a small constant. Note that for the $(1 + \lambda)$ EA, an excursion into unfavorable rate regions is less a problem as long as one can show that the mutation rate returns into the good region after a reasonable time. The fact that the (1,λ) EA can lose fitness also makes it more difficult to cut the analysis into regimes defined by fitness levels since it is now possible that the EA returns into a previous regime. In this work, we also gained two insights which might be useful in the design of future self-adaptive algorithms.

Need for non-elitism: Given the previous works, it would be natural to try a self-adaptive version of the $(1 + \lambda)$ EA. However, this is risky. While the self-adjusting EA of Chapter 4 can cope with the fact that the current mutation rate is far from the ideal one and in this case provably quickly changes the rate to an efficient setting, a self-adaptive algorithm cannot do so. Since the mutation rate is encoded in the individual, a change of the rate can only occur if an offspring is accepted. For an elitist algorithm like the $(1 + \lambda)$ EA, this is only possible when an offspring is generated that is good

enough to compete with the parent(s). Consequently, if the parent individual in a self-adaptive $(1+\lambda)$ EA has a high fitness, but a detrimental (that is, large) mutation rate, then the algorithm is stuck with this individual for a long time. Already for the simple OneMax function, such a situation can lead to an exponential runtime. Needless to say, when using a comma strategy we have to choose λ sufficiently large to avoid losing the current-best solution too quickly; see [RS14] for a precise analysis of this phenomenon for the (1,λ) EA using a static mutation rate of $1/n$.

Tie-breaking towards lower mutation rates: To prove our result, we need that the algorithm in case of many offspring of equal fitness prefers those with the smaller mutation rate. Given that the usual recommendation for the mutation rate is small, namely $\frac{1}{n}$, and that it is well-known that large rates can be very detrimental, it is natural to prefer smaller rates in case of ties (where, loosely speaking, the offspring population gives not hint which rate is preferable). This choice is similar to the classic tie-breaking rule of preferring offspring over parents in case of equal fitness (again, the fitness indicates no preference, but the simple fact that one is maybe working already for quite some time with this parent suggest to rather prefer the new individual). Without proof, we remark that without this tie-breaking rule we would need $\lambda = n^{\Omega(1)}$ to ensure a positive drift towards the optimum throughout the process.

The main result of this work Theorem 5.1 is a mathematical runtime analysis of the performance of the algorithm proposed above on the classic benchmark function $\textsc{OneMax} : \{0,1\}^n \to \mathbb{R}$ defined by $\textsc{OneMax}(x) = \sum_{i=1}^{n} x_i$ for all $x = (x_1, \dots, x_n) \in \{0,1\}^n$. Since such runtime analyses are by now a well-established way of understanding the performance of evolutionary algorithms, we only briefly give the most important details and refer the reader to the textbook [Jan13].

Our main result is as follows.

Theorem 5.1. *Let $\lambda \geq C \ln n$ for a sufficiently large constant C, and $\lambda = n^{O(1)}$, and $F = 32$. Then the expected number of generations of the self-adapting (1,λ) EA to minimize* OneMax *is $O(n/\log\lambda + (n\log n)/\lambda)$, corresponding to an expected number of fitness evaluations of $O(n\lambda/\log\lambda + n\log n)$.*

The aim of runtime analysis is predicting how long an evolutionary algorithm takes to find the optimum or a solution of sufficient quality. As implementation-independent performance measure usually the number of fitness evaluations performed in a run of the algorithm is taken. More precisely, the *optimization time* of an algorithm on some problem is the number of fitness evaluations performed until for the first time an optimal solution is evaluated. Obviously, for a (1,λ) EA, the optimization time is essentially λ times the number of iterations performed until an optimum is generated.

As in classic algorithms analysis, our main goal is an asymptotic understanding of how the optimization time depends on the problems size n. Hence all asymptotic notation in the chapter will be with respect to n tending to infinity.

The proof of the theorem is based on a careful, technically demanding drift analysis of both the current OneMax-value k_t (which is also the fitness distance, recall that our goal is the minimization of the objective function) and the current rate r_t of the parent. In very rough terms, a similar division of the run as in Chapter 4 into regions of large OneMax-value, the far region (Section 5.2), and of small OneMax-value, the near region (Section 5.3) is made. The middle region considered in Chapter 4 is subsumed under the far region here.

In the remaining of our analysis, we regard n sufficiently large, $\lambda \geq C \ln n$ with a large constant C, and $\lambda = n^{O(1)}$.

5.1 Algorithm

We propose a (1,λ) EA with self-adaptive mutation rate for the *minimization* of pseudo-boolean functions $f : \{0,1\}^n \to \mathbb{R}$ as defined in Algorithm 9.

To encode the mutation rate into the individual, we extend the individual representation by adding the rate parameter. Hence the extended individuals are pairs (x, r) consisting of a search point $x \in \{0,1\}^n$ and the rate parameter r, which shall indicate that r/n is the mutation rate this individual was created with.

The extended mutation operator first changes the rate to either r/F or Fr with equal probability ($F > 1$). It then performs standard bit mutation with the new rate.

In the selection step, we choose from the offspring population an individual with best fitness. If there are several such individuals, we prefer individuals having the smaller rate r/F, breaking still existing ties randomly. In this winning individual, we replace the rate by F if it was smaller and by $n/(2F)$ if it was larger.

We formulate the algorithm to start with an initial mutation rate r^{init}, which we require to be in $[F, n/(2F)]$. For the result we shall show in this work, the initial rate is not important, but without this prior knowledge we would strongly recommend to start with the smallest possible rate $r^{\text{init}} = F$. Due to the multiplicative rate adaptation, the rate can quickly grow if this is profitable. On the other hand, a too large initial rate might lead to an erratic initial behavior of the algorithm. For the adaptation parameter, we shall use $F = 32$ in our runtime analysis. Having such a large adaptation parameter eases the already technical analysis, because now the two competing rates r/F and Fr are different enough to lead to a significantly different performance. For a practical application, we suspect that a smaller value of F is preferable as it leads to a more stable optimization process. The choice of the offspring population size depends mostly on the degree of parallelism one wants to obtain. Clearly, λ should be at least logarithmic in n to prevent a too quick loss of the current-best solution. For our theoretical analysis, we require $\lambda \geq C \ln n$ for a sufficiently large constant C.

Algorithm 9 The (1,λ) EA with self-adapting mutation rate, adaptation parameter $F > 1$, and initial mutation rate r^{init}/n such that $r^{\text{init}} \in [F, n/(2F)]$ and $r^{\text{init}} = F^i$ for some $i \in \mathbb{N}$.

Select x_0 uniformly at random from $\{0,1\}^n$.
Set $r_0 \leftarrow r^{\text{init}}$.
for $t \leftarrow 1, 2, \ldots$ **do**
  for $i \leftarrow 1, \ldots, \lambda$ **do**
    Choose $r_{t,i} \in \{r_{t-1}/F, Fr_{t-1}\}$ uniformly at random.
    Create $x_{t,i}$ by flipping each bit in x independently with probability $r_{t,i}/n$.
  Choose $i \in [1..\lambda]$ such that $f(x_{t,i}) = \min_{j \in [1..\lambda]} f(x_{t,j})$; in case of a tie, prefer an i with $r_{t,i} = r_{t-1}/F$; break remaining ties randomly.
  $(x_t, r_t) \leftarrow (x_{t,i}, r_{t,i})$.
  Replace r_t with $\min\{\max\{F, r_t\}, n/(2F)\}$.

5.2 The Far Region

In this section, we analyze the optimization behavior of our self-adaptive (1,λ) EA in the regime where the fitness distance k is at least n/λ. Due to our assumption $\lambda \geq C \ln n$, it is very likely to have at least one copy of the parent among λ offspring

for $r = O(\ln \lambda)$. Thus the (1,λ) EA works almost the same as the $(1+\lambda)$ EA but can lose fitness with small probability. The following lemma is crucial in order to analyze the drift of the rate depending on k, which follows a similar scheme as with the $(1+\lambda)$ EA proposed in [Doe+17].

Roughly speaking, the rate leading to optimal fitness progress is n for $k \geq n/2 + \omega(\sqrt{n \ln(\lambda)})$, $n/2$ for $k = n/2 \pm o(\sqrt{n \log(\lambda)})$, and then the optimal rate quickly drops to $r = \Theta(\log \lambda)$ when $k \leq n/2 - \varepsilon n$.

To ease the representation, we first define two fitness dependent bounds $L(k)$ and $R(k)$.

Definition 5.2. *Let $n/\ln\lambda < k < n/2$ and $F = 32$. We define $L(k) := (F \ln(en/k))^{-1}$ and $U(k) := n(2n-k)/(22(n-2k)^2)$.*

According to the definition, both $L(k)$ and $R(k)$ monotonically increase when k increases.

Lemma 5.3. *Let $F = 32$. Consider an iteration of the self-adaptive (1,λ) EA with current fitness distance k and current rate r.*

Then:

1. *If $n/\ln\lambda < k$ and $F \leq r \leq L(k)\ln\lambda$, the probability that all best offspring have been created with rate Fr is at least $1 - O(\ln^3(\lambda)/\lambda^{1/(4\ln\ln\lambda)})$.*
2. *If $k < n/2$ and $n/(2F) \geq r \geq U(k)\ln\lambda$, then the probability that all best offspring have been created with rate r/F is at least $1 - \lambda^{1-(23/22)r/(U(k)\ln\lambda)}$.*

Proof of Lemma 5.3 part 1. Let $q(k,i,r)$ and $Q(k,i,r)$ be the probability that standard bit mutation with mutation rate $p = r/n$ creates from a parent with fitness distance k an offspring with fitness distance exactly $k-i$ and at most $k-i$, respectively. Then

$$q(k,i,r) = \sum_{j=0}^{k-i} \binom{k}{i+j} \binom{n-k}{j} p^{i+2j}(1-p)^{n-i-2j} \tag{5.1}$$

and $Q(k,i,r) = \sum_{j=i}^{k} q(k,j,r)$. We aim at finding i such that $Q(k,i,Fr) \geq \ln(\lambda)/\lambda$ while $Q(k,i,r/F) = O(\ln^3(\lambda)/\lambda^{1+1/4(\ln\ln\lambda)})$. Then we use these to bound the probability that at least one offspring using rate Fr obtains a progress of i or more while at the same time all offspring using rate r/F obtains less than i progress. Let i^* be the largest i such that $Q(k,i,Fr) \geq \ln(\lambda)/\lambda$. Using the fact that $\ln(1-x) \geq -x - x^2$ for all $0 < x < 2/3$, we notice that $(1-Fp)^{n-i} \geq (1-Fp)^n \geq e^{-Fr-(Fr)^2/n}$. By the assumption that $r \leq L(k)\ln\lambda \leq \ln\lambda$, we obtain $(Fr)^2/n = O(\ln^2\lambda/n) = o(1)$. Thus $(1-Fp)^{n-i} = (1-o(1))e^{-Fr}$. We also notice that $\binom{k}{i} = (k/i)((k-1)/(i-1))\cdots(k-i+1) > (k/i)^{i-1}(k-i) = (k/i)^i((k-i)i/k) > 2(k/i)^i$ for $2 < i < k-2$. Thus for $i > 2$ we can bound $Q(k,i,Fr)$ by

$$Q(k,i,Fr) \geq q(k,i,Fr) \geq \binom{k}{i}(Fp)^i(1-Fp)^{n-i} \geq \left(\frac{k}{i}\cdot\frac{Fr}{n}\right)^i e^{-Fr}. \tag{5.2}$$

Let $i = \max\{(F-1)r, \ln\lambda/(8\ln\ln\lambda)\}$. We prove $i^* \geq i$ by distinguishing between two cases according to which argument maximizes i.

If $i = \ln\lambda/(8\ln\ln\lambda)$, then $r \le i/(F-1)$ and $Fr \le 2i$. Referring to inequality (5.2) and using the fact that $k/n \ge 1/\ln\lambda$, $i < \ln\lambda$, and $\ln\ln(\lambda) > 1$, we obtain

$$\begin{aligned}\ln(Q(k,i,Fr)) &\ge i\ln\left(\frac{k}{in}\right) - Fr \ge -i\ln(\ln^2\lambda) - 2i \\ &= -2i\ln\ln\lambda - 2i > -4i\ln\ln\lambda = -\frac{\ln\lambda}{2} \ge \ln\left(\frac{\ln\lambda}{\lambda}\right)\end{aligned}$$

and thus $Q(k,i,Fr) \ge \ln(\lambda)/\lambda$.

If $i = (F-1)r$, then $r \ge \ln\lambda/(8(F-1)\ln\ln\lambda)$ since F is a constant. Using $r \le L(k)\ln\lambda$, we obtain $\ln\lambda \ge \ln(en/k)Fr$ which is equivalent to $(k/en)^{Fr} \ge 1/\lambda$. Furthermore, $(k/n)^i > (k/n)^{Fr}$ since $i = (F-1)r < Fr$. Thus

$$Q(k,i,Fr) \ge \left(\frac{k}{i}\cdot\frac{Fr}{n}\right)^i e^{-Fr} \ge \left(\frac{F}{F-1}\right)^{(F-1)r}\left(\frac{k}{en}\right)^{Fr} \ge 2^r\left(\frac{k}{en}\right)^{Fr} \ge \frac{\ln\lambda}{\lambda}.$$

Since $Q(k,i,r)$ is decreasing in i, we obtain $i^* \ge \max\{(F-1)r, \ln\lambda/(8\ln\ln\lambda)\}$. Using a Chernoff bound and recalling that the expected number of flipped bits is bounded by $FL(k)\ln\lambda \le \ln\lambda/\ln(2e)$, we notice that $i^* \le \ln\lambda$. This upper bound will be used to estimate $Q(k,i^*,Fr)/Q(k,i^*+1,Fr)$ in the following part of the proof.

We now prove that $Q(k,i^*,r/F) = o(1/\lambda)$. By comparing each component in $q(k,i,r/F)$ and $q(k,i,Fr)$, and applying Lemma 2.14 3 to estimate $(1-Fr/n)^n$ and $(1-r/(Fn))^n$ with $r = O(\ln\lambda) = o(n^{1/2})$ for large enough n, we obtain

$$\begin{aligned}\frac{q(k,i,Fr)}{q(k,i,r/F)} &\ge F^{2i}\frac{(1-Fr/n)^n}{(1-r/(Fn))^n} \\ &\ge \Big(1-o(1)\Big)F^{2i}e^{-(F-1/F)r} > F^{2i}e^{-Fr}.\end{aligned}$$

Therefore $Q(k,i^*,Fr)/Q(k,i^*,r/F) \ge F^{2i^*}e^{-Fr} = \exp(2i^*\ln F - Fr) \ge \exp(2i^*\ln F - Fi^*/(F-1)) > \exp(3i^*) > \lambda^{1/(4\ln\ln\lambda)}$, where in the first inequality, we use the fact that $i^* \ge (F-1)r$. To prove $Q(k,i^*,r/F) = O(\ln^3(\lambda)/\lambda^{1+1/4(\ln\ln\lambda)})$, we first show $Q(k,i^*,Fr)/Q(k,i^*+1,Fr) = O(\ln^2\lambda)$. Then we use this to bound $Q(k,i^*,Fr) = O(\ln^3(\lambda)/\lambda)$ according to the definition of i^*. Finally we obtain $Q(k,i^*,r/F) \le Q(k,i^*,Fr)/\lambda^{1/(4\ln\ln\lambda)} = O(\ln^3(\lambda)/\lambda^{1+1/4(\ln\ln\lambda)})$. It remains to bound $Q(k,i^*,Fr)/Q(k,i^*+1,Fr)$. We show that the majority of $q(k,i,r)$ are from the first $3r$ terms in the summation of equation (5.1). Let $q(k,i,r)_j$ denote the j-th item $\binom{k}{i+j}\binom{n-k}{j}p^{i+2j}(1-p)^{n-i-2j}$ in equation (5.1). Then

$$\frac{q(k,i,r)_{j+1}}{q(k,i,r)_j} = \frac{k-i-j}{i+j+1}\cdot\frac{n-k-j}{j+1}\cdot p^2\cdot(1-p)^{-2} \le \frac{r^2}{(i+j+1)(j+1)}.$$

If $j > 3r$, then $r^2/((i+j+1)(j+1)) < 1/9$, and thus

$$\begin{aligned} q(k,i,r) &\leq \left(\sum_{j=0}^{3r} q(k,i,r)_j\right) + q(k,i,r)_{3r}\left(\sum_{j=3r+1}^{k-i} (1/9)^{j-3r}\right) \\ &\leq \left(\sum_{j=0}^{3r} q(k,i,r)_j\right) + q(k,i,r)_{3r}\cdot\frac{1/9}{1-1/9} \\ &= \left(\sum_{j=0}^{3r-1} q(k,i,r)_j\right) + \frac{9}{8}\cdot q(k,i,r)_{3r} \leq \frac{9}{8}\sum_{j=0}^{3r} q(k,i,r)_j. \end{aligned}$$

We notice that

$$\frac{q(k,i+1,r)_j}{q(k,i,r)_j} = \frac{\binom{k}{i+j+1}\binom{n-k}{j}p^{i+2j+1}(1-p)^{n-i-2j-1}}{\binom{k}{i+j}\binom{n-k}{j}p^{i+2j}(1-p)^{n-i-2j}} = \frac{(k-i-j)p}{(i+j+1)(1-p)},$$

using the fact that $\sum_{j=0}^{3r} q(k,i,r)_j \leq q(k,i,r) \leq (9/8)\sum_{j=0}^{3r} q(k,i,r)_j$ for all (k,i,r), we compute

$$\frac{q(k,i^*+1,Fr)}{q(k,i^*,Fr)} \geq \frac{\sum_{j=0}^{3Fr} q(k,i^*+1,Fr)_j}{(9/8)\sum_{j=0}^{3Fr} q(k,i^*,Fr)_j} \geq \frac{8}{9}\cdot\frac{k-i^*-3Fr}{i^*+3Fr+1}\cdot\frac{p}{1-p}.$$

Since $i^* \geq (F-1)r$, $i^* \leq \ln\lambda$, and $k \geq n/\ln\lambda = \omega(\ln\lambda)$, we obtain

$$\frac{q(k,i^*+1,Fr)}{q(k,i^*,Fr)} = \Omega\left(\frac{kp}{i^*}\right) = \Omega\left(\frac{kr}{i^*n}\right) = \Omega\left(\frac{1}{\ln^2\lambda}\right).$$

Consequently we have $q(k,i^*,Fr)/Q(k,i^*+1,Fr) \leq q(k,i^*,Fr)/q(k,i^*+1,Fr) = O(\ln^2\lambda)$ and

$$\frac{Q(k,i^*,Fr)}{Q(k,i^*+1,Fr)} = 1 + \frac{q(k,i^*,Fr)}{Q(k,i^*+1,Fr)} = O(\ln^2\lambda).$$

So finally $Q(k,i^*,Fr) = O(\ln^3(\lambda)/\lambda)$ due to the definition of i^*, and

$$Q(k,i^*,r/F) \leq \frac{Q(k,i^*,Fr)}{F^{2i^*}e^{-Fr}} = O\left(\frac{\ln^3\lambda}{\lambda\cdot\lambda^{1/(4\ln\ln\lambda)}}\right).$$

A simple union bound shows that with probability $1 - O(\ln^3(\lambda)/\lambda^{1/(4\ln\ln\lambda)})$, no offspring of rate r/F manages to obtain a progress of i^* or more. However, the probability that an offspring has rate Fr and obtains at least i^* progress is $\ln(\lambda)/(2\lambda)$. Thus the probability that no offspring generated with rate Fr achieves a progress of at least i^* is at most $(1-\ln(\lambda)/(2\lambda))^\lambda \leq \lambda^{-1/2} = o(\ln^3(\lambda)/\lambda^{1/(4\ln\ln\lambda)})$. This proves the first statement of the lemma. □

Proof of Lemma 5.3 part 2. For $\tilde{r} \in \{r/F, Fr\}$ let the random variable $X(k,\tilde{r})$ denote the number of flipped bits in k ones and $Y(k,\tilde{r})$ denote the number of flipped bits in $n-k$ zeros when applying standard bit mutation with probability $p = \tilde{r}/n$. Let $Z(k,\tilde{r}) := Y(k,\tilde{r}) - X(k,\tilde{r})$ denote the improvement in fitness. Let $Z^*(k,\tilde{r})$ denote the minimal $Z(k,\tilde{r})$ among all offspring which apply rate $\tilde{r}$. $\mathrm{E}(Z(k,\tilde{r})) = (n-k)\tilde{r}/n - k\tilde{r}/n = (n-2k)\tilde{r}/n$. Our aim is to find a β such that

$\Pr(Z(k,r/F) \leq \beta) = \Theta(1)$ while $\Pr(Z(k,Fr) \leq \beta) = o(1/\lambda)$, and use this to obtain a high value for $\Pr(Z^*(k,r/F) < Z^*(k,Fr))$.

Let $\beta := \mathrm{E}(Z(k,r/F))$. We notice that $\Pr(X(k,r/F) > \mathrm{E}(X(k,r/F)) - 1) \geq 1/2$ since the median of binomial distribution $X(k,r/F)$ is $\lfloor E(X(k,r/F) \rfloor$ or $\lceil E(X(k,r/F) \rceil$. Applying Lemma 2.16 to $\Pr(Y(k,r/F) < \mathrm{E}(Y(k,r/F)) - 1)$ with $E(Y(k,r/F)) = \Omega(\ln\lambda) = \omega(1)$ by assumption $r \geq U(k)\ln\lambda$ and $E(Y(k,r/F)) < (n-k)/2$, we obtain for n sufficiently large that

$$\begin{aligned} &\Pr\Big(Y(k,r/F) < \mathrm{E}(Y(k,r/F)) - 1\Big) \\ &\geq \frac{1}{2} - \sqrt{\frac{n-k}{2\pi\lfloor(n-k)p\rfloor(n-k-\lfloor(n-k)p\rfloor)}} > \frac{2}{5}. \end{aligned} \tag{5.3}$$

Thus $\Pr(Z(k,r/F) \leq \beta) > (1/2)(2/5) = 1/5$. We use Bernstein's inequality (version Lemma 5.7) to bound $\Pr(Z(k,Fr) \leq \beta)$ and obtain

$$\Pr\Big(Z(k,Fr) \leq E(Z(k,Fr)) - \Delta\Big) \leq \exp\left(-\frac{\Delta^2}{2(\mathrm{Var}(Z(k,Fr)) + \Delta/3)}\right) \text{ for all } \Delta > 0.$$

With $\Delta = E(Z(k,Fr)) - \beta = (n-2k)(Fr/n - r/(Fn)) = (n-2k)(F^2-1)r/(Fn)$ and $\mathrm{Var}(Z(k,Fr)) = Fr(1-Fr/n) < Fr$, we compute

$$\begin{aligned} Pr\Big(Z(k,Fr) \leq \beta\Big) &\leq \exp\left(-\frac{1}{2}\cdot\frac{(F^2-1)^2(n-2k)^2r^2}{F^2n^2(Fr + (n-2k)(F^2-1)r/(3Fn))}\right) \\ &= \exp\left(-\frac{1}{2}\cdot\frac{(F^2-1)^2(n-2k)^2r}{F^3n^2 + Fn(n-2k)(F^2-1)/3}\right) \\ &\leq \exp\left(-\frac{3}{2}\cdot\frac{(F^2-1)^2(n-2k)^2r}{3F^3n^2 + F^3n(n-2k)}\right) \\ &= \exp\left(-\frac{3}{4}\cdot\frac{(F^2-1)^2(n-2k)^2r}{F^3n(2n-k)}\right). \end{aligned}$$

Given $F = 32$ and $r \geq U(k)\ln\lambda$ then

$$\Pr\Big(Z(k,Fr) \leq \beta\Big) < \exp\left(-\frac{23.9(n-2k)^2r}{n(2n-k)}\right) < \lambda^{-\frac{23.9r}{22U(k)\ln\lambda}}.$$

With a simple union bound, we obtain $\Pr(Z^*(k,Fr) \leq \beta) < \lambda\Pr(Z(k,Fr) \leq \beta) < \lambda^{1-23.9r/(22U(k)\ln\lambda)}$. The probability that an offspring has rate r/F and obtains β is at least $(1/2)(1/5) = 1/10$. Thus the probability that no offspring generated with r/F has a Z-value of at least β is at most $(1-1/10)^\lambda = \exp(-\Theta(\lambda))$. Therefore $\Pr(Z^*(k,Fr) < Z^*(k,r/F)) < \lambda^{1-23.9r/(22U(k)\ln\lambda)}(1-\exp(-\Theta(\lambda))) = o(\lambda^{1-23r/(22U(k)\ln\lambda)})$, which means with probability at least $1-\lambda^{1-(23r/(22U(k)\ln\lambda)}$ all best offspring have been created with rate r/F. □

Lemma 5.3 will be crucial in order to bound the expected progress on fitness in the far region. We notice that $\ln\lambda = o(\sqrt{n})$ in the lemma we may allow $r > \ln\lambda$ when k is large and $r = \Theta(n)$ when $k = n/2 - \Theta(\sqrt{n\ln\lambda})$. It is easy to show a positive progress on fitness for $r < \ln\lambda$ since there will be sufficiently many offspring that do not flip zeroes. When $r \geq \ln\lambda$ we expect all offspring to flip zeros, but we can still show a positive drift when $k > 7n/20$, as stated in the following lemma. The idea is that the

standard variation of the number of flipping ones is $\sqrt{kr/n(1-r/n)} = \Theta(\sqrt{r})$. This makes a deviation compensating bad flips among the remaining $n-2k$ zeros likely enough.

Lemma 5.4. *Let $7n/20 \leq k < n/2$, $F = 32$ and $\alpha = 10^{-4}$. Assume $r \leq \min\{n^2 \ln\lambda/(12(n-2k)^2), n/(2F)\}$. Assume that from a parent with fitness distance k we generate an offspring using standard bit mutation with mutation rate $p = r/n$. Then the probability that this offspring has a fitness distance of at most $k-s$ with $s := \alpha(\min\{\ln\lambda, r\} + (n-2k)r/n)$, is at least $\lambda^{-0.98}$.*

Proof. We first look at the case when $r < 1/(2\alpha)$. In this case $s \leq \alpha(r + (n-2k)r/n) \leq \alpha(2r) < 1$. Then the probability that this offspring has a fitness distance of $k-1 > k-s$ is at least

$$\binom{k}{1}\left(\frac{r}{n}\right)^1\left(1-\frac{r}{n}\right)^{n-1} = \Theta(e^{-r}) = \omega(\lambda^{-0.98}).$$

Therefore it remains to consider $r \geq 1/(2\alpha)$.

Let random variables X and Y denote the number of flips in k one-bits and $(n-k)$ zero-bits, respectively, in an offspring using rate $p = r/n$. Then $X - Y$ is the decrease of fitness distance. X and Y follow binomial distributions $\mathrm{Bin}(k,p)$ and $\mathrm{Bin}(n-k,p)$, respectively. Let

$$B(x) := \Pr(X = x) = \binom{k}{x}p^x(1-p)^{k-x} \text{ for all } x \in \{0,1,\ldots,k\},$$

$$F(x) := \Pr(X \geq x) = \sum_{i=\lceil x\rceil}^{k} B(i) \text{ for all } x \in [0,k].$$

Since $r \geq 1/(2\alpha) \geq 5000$ and $n \leq 20k/7$, then $p = r/n \geq 5000 \cdot 7/(20k) = 1750/k$. Using this and the fact that $p \leq 1/(2F)$, we apply Lemma 2.16 and obtain

$$\Pr(X > E(X)) > \frac{1}{2} - \sqrt{\frac{k}{2\pi\lfloor kp\rfloor(k-\lfloor kp\rfloor)}} > \frac{1}{2} - \sqrt{\frac{1}{2kp}} > \frac{2}{5}$$

Similarly $\Pr(Y \leq E(Y)) = 2/5$. Since $E(X-Y) = kp - (n-k)p = -(n-2k)p$, we bound

$$\begin{aligned}\Pr\Big(X - Y \geq s\Big) &\geq \Pr\Big(X \geq E(X) + (n-2k)p + s\Big)\Pr\Big(Y \leq E(Y)\Big)\\ &\geq \frac{2}{5}F\Big(kp + (n-2k)p + s\Big).\end{aligned}$$

Let $\delta := \lceil (n-2k)p + s\rceil$, $u := kp$ and $\tilde{u} := \lceil u\rceil$. We notice that $u = rk/n \geq (1/(2\alpha))(7/20) = 1750$. Furthermore, we have $\delta < \tilde{u} - 2 < u$ since

$$\begin{aligned}\delta &= \lceil (n-2k)p + s\rceil < (1+\alpha)(n-2k)p + \alpha\min\{\ln\lambda, r\} + 1\\ &\leq (1+\alpha)\frac{n-2k}{n}r + \alpha r + 1 \leq \left((1+\alpha)\frac{3}{10} + \alpha\right)r + 1\\ &= \frac{3+13\alpha}{10}\cdot\frac{n}{k}\cdot u + 1 < \frac{3+13\alpha}{10}(3u) + 1 < 0.91u + 1\\ &= u - 0.09u + 1 \leq u - (0.09\cdot 1750 - 1) = u - 156.5.\end{aligned}$$

We aim at proving $F(u+\delta) = \omega(\lambda^{-0.98})$ to obtain this lemma. If $F(u+\delta) = \Theta(1)$ then the conclusion holds. It remains to consider $F(u+\delta) = o(1)$ while $F(u) - F(u+\delta) \geq 2/5 - o(1)$ as stated in equation (5.3). For any $x \in \mathbb{Z}_{\geq u}$ we have

$$\frac{B(x+1)}{B(x)} = \frac{k-x}{x+1} \cdot \frac{p}{1-p} \leq \frac{u-up}{u-up+1-p} < 1.$$

Since $\tilde{u} = \lceil u \rceil$ then $B(\tilde{u}) > B(\tilde{u}+1) > \cdots > B(k)$, and thus $F(u+\delta) \geq \delta B(\tilde{u}+2\delta)$ as well as $F(u) - F(u+\delta) \leq \delta B(\tilde{u})$. Using the fact that $p/(1-p) = u/(k-u)$ and $\tilde{u} - 1 < u$, we see that

$$\begin{aligned}\frac{B(\tilde{u}+2\delta)}{B(\tilde{u})} &= \frac{(k-\tilde{u})\cdots(k-(\tilde{u}+2\delta)+1)}{(\tilde{u}+1)\cdots(\tilde{u}+2\delta)} \cdot \frac{p^{2\delta}}{(1-p)^{2\delta}} \\ &\geq \frac{(k-(\tilde{u}-1)-2\delta)^{2\delta}}{(\tilde{u}+1)\cdots(\tilde{u}+2\delta)} \cdot \frac{u^{2\delta}}{(k-u)^{2\delta}} \geq \left(1 - \frac{2\delta}{k-u}\right)^{2\delta} \frac{u^{2\delta}}{(\tilde{u}+1)\cdots(\tilde{u}+2\delta)}.\end{aligned}$$

We compute the following factorials using Robbins's Stirling's approximation in [Rob55]

$$\begin{aligned}(\tilde{u}+2\delta)! &\leq \sqrt{2\pi(\tilde{u}+2\delta)} \left(\frac{\tilde{u}+2\delta}{e}\right)^{\tilde{u}+2\delta} \exp\left(\frac{1}{12(\tilde{u}+2\delta)}\right), \\ \tilde{u}! &\geq \sqrt{2\pi\tilde{u}} \left(\frac{\tilde{u}}{e}\right)^{\tilde{u}} \exp\left(\frac{1}{12\tilde{u}+1}\right).\end{aligned}$$

Notice that $12\tilde{u}+1 < 12(\tilde{u}+2\delta)$, we obtain

$$\frac{1}{(\tilde{u}+1)\cdots(\tilde{u}+2\delta)} = \frac{\tilde{u}!}{(\tilde{u}+2\delta)!} \geq \sqrt{\frac{\tilde{u}}{\tilde{u}+2\delta}} \frac{\tilde{u}^{\tilde{u}} e^{2\delta}}{(\tilde{u}+2\delta)^{\tilde{u}+2\delta}} \geq \sqrt{\frac{\tilde{u}}{\tilde{u}+2\delta}} \frac{u^{\tilde{u}} e^{2\delta}}{(\tilde{u}+2\delta)^{\tilde{u}+2\delta}}.$$

Therefore

$$\begin{aligned}\frac{B(\tilde{u}+2\delta)}{B(\tilde{u})} &\geq \left(1 - \frac{2\delta}{k-u}\right)^{2\delta} \sqrt{\frac{\tilde{u}}{\tilde{u}+2\delta}} \frac{u^{\tilde{u}+2\delta} e^{2\delta}}{(\tilde{u}+2\delta)^{\tilde{u}+2\delta}} \\ &= \sqrt{\frac{\tilde{u}}{\tilde{u}+2\delta}} \exp\left(2\delta \ln\left(1 - \frac{2\delta}{k-u}\right) + (\tilde{u}+2\delta)\ln\left(\frac{u}{\tilde{u}+2\delta}\right) + 2\delta\right) \\ &\geq \sqrt{\frac{\tilde{u}}{\tilde{u}+2\delta}} \exp\left(2\delta \ln\left(1 - \frac{2\delta}{k-u}\right) + (\tilde{u}+2\delta)\ln\left(1 - \frac{2\delta+1}{\tilde{u}+2\delta}\right) + 2\delta\right).\end{aligned}$$

We notice that $2\delta/(k-u) \leq 2\delta/(2Fu-u) = 2\delta/(63u) < 2/63 < 1/2$ and $(2\delta+1)/(\tilde{u}+2\delta) < (2\delta+1)/(3\delta+2) < 2/3$. Referring to Lemma 2.14 3, 4, we compute

$$\begin{aligned}2\delta \ln\left(1 - \frac{2\delta}{k-u}\right) &\geq -\frac{3}{2} \cdot \frac{4\delta^2}{k-u} = -\frac{6\delta^2}{u/p-u} \geq -\frac{6\delta^2}{2Fu-u} \geq -\frac{\delta^2}{10u}, \\ (\tilde{u}+2\delta)\ln\left(1 - \frac{2\delta+1}{\tilde{u}+2\delta}\right) &\geq -(2\delta+1) - \frac{(2\delta+1)^2}{\tilde{u}+2\delta} \geq -2\delta - \frac{4\delta^2}{u} - 3, \\ \frac{B(\tilde{u}+2\delta)}{B(\tilde{u})} &\geq \sqrt{\frac{\tilde{u}}{\tilde{u}+2\delta}} \exp\left(-\frac{41\delta^2}{10u} - 3\right) \geq \sqrt{\frac{1}{3}} e^{-3} \exp\left(-\frac{41\delta^2}{10u}\right).\end{aligned} \tag{5.4}$$

where the last inequality used that $\tilde{u}/(\tilde{u}+2\delta) \geq 1/3$ since $\delta \leq \tilde{u}$. Using $n/k \leq 20/7$ and $r \leq n^2 \ln\lambda/(12(n-2k)^2)$, we obtain

$$\begin{aligned}
\frac{41\delta^2}{10u} &= \frac{41n}{10k} \cdot \frac{\lceil (1+\alpha)((n-2k)/n)r + \alpha \min\{\ln\lambda, r\}\rceil^2}{r} \\
&\leq \frac{41n}{10k} \cdot \frac{((1+\alpha)((n-2k)/n)r + \alpha \min\{\ln\lambda, r\} + 1)^2}{r} \\
&\leq \frac{41n}{10k} \cdot \left(\frac{((1+\alpha)((n-2k)/n)r + \alpha \min\{\ln\lambda, r\})^2}{r} + 2(1+\alpha)\frac{n-2k}{n} + 2\alpha + \frac{1}{r} \right) \\
&\leq \frac{82}{7} \cdot \left((1+\alpha)^2 \left(\frac{n-2k}{n}\right)^2 r + 2(1+\alpha)\frac{n-2k}{n}\alpha\ln\lambda + \alpha^2\ln\lambda + 1 \right) \\
&\leq \frac{82}{7} \cdot \left(\frac{(1+\alpha)^2\ln\lambda}{12} + \frac{3(1+\alpha)\alpha}{5}\ln\lambda + \alpha^2\ln\lambda + 1 \right) < 0.978\ln\lambda + \frac{82}{7}.
\end{aligned}$$

Plugging the last estimate into inequality (5.4), we obtain $B(\tilde{u}+2\delta)/B(\tilde{u}) = \omega(\lambda^{-0.98})$. Thus $F(u+\delta)/(F(u)-F(u+\delta)) = \omega(\lambda^{-0.98})$ and $F(u+\delta) = \omega(\lambda^{-0.98})$ which proves the statement in this lemma. □

For $k < 7n/20$, we need a more careful analysis, where we will estimate the expected progress on fitness averaged over the random rates the algorithm may have at a time. Hence, we assume a fixed current fitness but a random current rate and compute the average drift of fitness with respect to the distribution on the rates. This approach is similar to the one by Jägerskküpper [Jäg11], who computes the average drift of the Hamming distance to the optimum when the (1+1) EA is optimizing a linear function, where the average is taken with respect to a distribution on all search points with a certain Hamming distance.

Of course, we want to exploit that a rate yielding near-optimal fitness progress is used most of the time such that too high (or too low) rates do not have a significant impact. To this end, Lemma 2.12 about occupation probabilities will be crucial.

We now define two fitness dependent bounds $r_l(k)$ and $r_u(k)$. We show in Lemma 5.6 that for any rate, if r/F or Fr is within the bounds, then the algorithm has logarithmic drift on fitness.

Definition 5.5. *Let $n/\ln\lambda < k < n/2$ and $F = 32$. We define*

$$r_u(k) := \begin{cases} n^2\ln(\lambda)/(12(n-2k)^2) & \text{if} \quad 7n/20 \leq k < n/2, \\ 10U(k)\ln(\lambda)/9 & \text{if} \quad n/\ln\lambda < k < 7n/20. \end{cases}$$

$$r_l(k) := \begin{cases} L(k)\ln(\lambda)/2 & \text{if} \quad n/\ln\lambda \leq k < n/2, \\ F & \text{if} \quad n/\lambda < k < n/\ln\lambda. \end{cases}$$

where $L(k)$ and $U(k)$ are defined as in Definition 5.2.

We notice that Lemma 5.3 can be applied to all $r > r_u$ or $r < r_l$ because for all $7n/20 \leq k < n/2$, we have $r_u/(U(k)\ln\lambda) = 22n/(12(2n-k)) \geq 22/(12(2-0.35)) = 10/9$. For $k < n/\ln\lambda$, we set r_l to the minimal possible value of r. Finally note that r_u is non-decreasing in k due to the monotonicity of $n^2/(n-2k)^2$ and $U(k)$.

Lemma 5.6. *Let $n/\lambda < k < n/2$ with $F = 32$. Let $\Delta(k,r)$ denote the fitness gain of the best offspring using rate in $\{r/F, Fr\}$.*

1. *The negative drift of fitness for too high rates $r \geq Fr_u$ is bounded by*
$$E(\Delta(k,r)) \geq -\Big(1+o(1)\Big)\frac{n-2k}{n}\frac{r}{F}.$$
2. *When $k \geq 7n/20$ the positive drift of fitness for good rate $r \leq Fr_u$ is bounded by*
$$E(\Delta(k,r)) \geq \Big(1-o(1)\Big)\cdot 10^{-4}\left(\frac{n-2k}{n}\cdot\frac{r}{F}+\min\left\{\ln\lambda,\frac{r}{F}\right\}\right).$$
3. *When $n/\lambda < k < 7n/20$ the positive drift of fitness for good rate $r \leq Fr_u$ is bounded by*
$$E(\Delta(k,r)) \geq \Big(1-o(1)\Big)\min\left\{\frac{r}{F},\frac{\ln\lambda}{F\ln(en/k)}\right\}.$$

Proof. The probability of using rate r/F is $1/2$. Thus with probability at least $1-(1/2)^\lambda = 1-o(1/n^3)$, at least one offspring uses rate r/F. For this offspring, the expected loss is $(n-2k)r/(Fn)$. If the complementary event (hereinafter called failure) of probability $o(1/n^3)$ happens, we estimate $\Delta(k,r)$ pessimistically by $-n$. This proves the first statement.

To prove the second item, we take $i = 10^{-4}((n-2k)r/(Fn)+\min\{\ln\lambda, r/F\})$. According to Lemma 5.4, the probability that an offspring uses rate r/F and achieves progress of i or more is at least $\lambda^{-0.98}/2$. Thus for λ offspring, we obtain $\Pr(\Delta(k,r)\geq i) \geq 1-(1-\lambda^{-0.98}/2)^\lambda = 1-O(\exp(-\lambda^{0.02}/2)) = 1-o(1)$. If the failure event happens, we estimate $\Delta(k,r)$ pessimistically by $-(n-2k)r/(Fn) = O(i)$. Thus the statement holds.

For the third item, we take $i := \min\{r, \ln(\lambda)/\ln(en/k)\}/F$. Notice that for $k < 7n/20$ we have $r_u(k) < r_u(7n/20) = (25/27)\ln\lambda < 0.93\ln\lambda$. Applying Lemma 2.143 with $r/F \leq r_u(k) = o(\sqrt{n})$ we obtain $(1-r/(Fn))^n \geq (1-o(1))e^{-r/F}$. Therefore the probability that one offspring using rate $r/F < 0.93\ln\lambda$ makes a progress of at least i is lower bounded by (assuming n large enough)

$$\begin{aligned}\binom{k}{i}\left(\frac{r}{Fn}\right)^i\left(1-\frac{r}{Fn}\right)^n &\geq \left(\frac{k}{i}\cdot\frac{r}{Fn}\right)^i\left((1-o(1))e^{-\frac{r}{F}}\right)\\ &> \left(\frac{k}{en}\right)^i e^{-0.94\ln\lambda} \geq \lambda^{-1/F-0.94} > \lambda^{-0.98}.\end{aligned}$$

Thus for λ offspring, we obtain $\Pr(\Delta(k,r)\geq i) \geq 1-(1-\lambda^{-0.98}/2)^\lambda = 1-o(1/\ln(\lambda))$. If the failure event happens we estimate $\Delta(k,r)$ pessimistically by $-(n-2k)r/(Fn) = O(\ln\lambda)$. The contribution of failure events is $o(1)$ which is also $o(i)$. Therefore the third statement holds. □

As discussed, our aim is to show that r_t/F or Fr_t stays in the right range frequently enough such that the overall average drift is still logarithmic. We notice that small rates $r_t < r_l$ intuitively do not have a negative effect, therefore we focus on the probability that $r_t < Fr_u$. Since r_u monotonically decreases when k decreases, we need to analyze whether r still stays in the right range if there are large jumps in fitness distance k. Intuitively, the speed at which the mutation rate is decreased is much higher than than the decrease of fitness distance. To make this rigorous, we first look at the probability of large jumps, as detailed in the following lemma.

Lemma 5.7. *Assume $r \leq n/2$ and let $Z(k,r)$ denote the fitness-distance increase when applying standard bit mutation with probability $p = r/n$ to an individual with k ones. Then*

$$\Pr\left(Z(k,r) \leq (n-2k)r/n - \Delta\right) \leq \exp\left(\frac{-\Delta^2}{2(1-p)(r+\Delta/3)}\right),$$
$$\Pr\left(Z(k,r) \geq (n-2k)r/n + \Delta\right) \leq \exp\left(\frac{-\Delta^2}{2(1-p)(r+\Delta/3)}\right).$$

Proof. Without loss of generality, we assume that the individual has k leading ones and $n-k$ trailing zeros. Let random variables $Z_1, \ldots, Z_n$ be the contribution to fitness distance increase in each position after standard bit mutation. Then

$$\Pr(Z_i = -1) = p \text{ and } \Pr(Z_i = 0) = 1-p \text{ for all } 1 \leq i \leq k;$$
$$\Pr(Z_i = 1) = p \text{ and } \Pr(Z_i = 0) = 1-p \text{ for all } k < i \leq n.$$

The random variables $Z_1, \ldots, Z_n$ are independent and $Z(k,r) = \sum_{i=1}^n Z_i$. Similarly as in the proof of Lemma 5.3 2, we have $E(Z(k,r)) = -kp + (n-2k)p = (n-2k)p$ and $\mathrm{Var}(Z(k,r)) = \sum_{i=1}^n \mathrm{Var}(Z_i) = np(1-p) = (1-p)r$. To apply Bernstein's inequality (Theorem 2.8), we construct $\tilde{Z}_i$ such that $\tilde{Z}_i = Z_i + p$ for all $1 \leq i \leq k$ and $\tilde{Z}_i = Z_i - p$ for all $k < i \leq n$. Therefore $E(\tilde{Z}_i) = 0$ and $\mathrm{Var}(\tilde{Z}_i) = \mathrm{Var}(Z_i)$.

$$\Pr(\tilde{Z}_i = -1+p) = p \text{ and } \Pr(\tilde{Z}_i = p) = 1-p \text{ for all } 1 \leq i \leq k;$$
$$\Pr(\tilde{Z}_i = 1-p) = p \text{ and } \Pr(\tilde{Z}_i = -p) = 1-p \text{ for all } k < i \leq n.$$

By assuming $r \leq n/2$, we have $p \leq 1/2$ and thus $p - 1 \leq \tilde{Z}_i \leq 1-p$ for all $1 \leq i \leq n$. Using the fact that $\sum_{i=1}^n \tilde{Z}_i = Z(k,r) - E(Z(k,r))$, Theorem 2.8 yields with $b := 1-p$ and $\sigma^2 := (1-p)pn = (1-p)r$ that

$$\Pr\left(\sum_{i=1}^n Z(k,r) - E(Z(k,r)) \geq \Delta\right) \leq \exp\left(\frac{-\Delta^2}{2(1-p)(r+\Delta/3)}\right).$$

Similarly the lower tail bound holds. □

We now use Lemma 5.7 to show that once $r_t \geq F r_u(k_t)$, there will be a strong drift for $r_t/r_u(k_t)$ to decrease down to 1.

Lemma 5.8. *Let $k_t < n/2$ and $F = 32$. Let $\tau := \log_F(3/\sqrt{10})$ and $X_t := \log_F(r_t/r_u(k_t)) - \tau$ with $r_u(k_t)$ defined in Definition 5.5, we have*

$$\Pr\left(X_{t+1} - X_t \geq a \mid X_t > 1\right) \leq \lambda^{-\Omega(a+1)} \text{ for all } a \geq -1/2,$$
$$\Pr\left(X_{t+1} - 1 \geq a \mid X_t \leq 1\right) \leq \lambda^{-\Omega(a+1)} \text{ for all } a > 0.$$

Proof. Using the fact that $r_{t+1} \in \{F r_t, r_t/F\}$, we see that

$$X_{t+1} - X_t \in \left\{1 + \log_F\left(\frac{r_u(k_t)}{r_u(k_{t+1})}\right), -1 + \log_F\left(\frac{r_u(k_t)}{r_u(k_{t+1})}\right)\right\}.$$

According to the monotonicity that $r_u(k)$ increases with respect to k, we notice that $k_t \geq k_{t+1}$ is a necessary condition for $X_{t+1} - X_t \geq 1$. We also notice that $X_t \geq \tau$ is equivalent to $r_t/r_u(k_t) \geq 3/\sqrt{10}$, which is sufficient to apply Lemma 5.32 since $r_u(k) \geq (10/9)U(k)\ln\lambda$ as defined in Definition 5.5.

We first consider the case $k_{t+1} \geq k_t$ (equivalent to $r_u(k_{t+1}) \geq r_u(k_t)$). In this case $X_{t+1} - X_t \leq 1$ thus $\Pr(X_{t+1} \geq 1 \cap k_{t+1} \geq k_t \mid X_t < 0) = 0$ and $\Pr(X_{t+1} - 1 \geq 1 \cap k_{t+1} \geq k_t \mid X_t \leq 1) = 0$. It remains to consider

$$\Pr(X_{t+1} - X_t \geq a \cap k_{t+1} \geq k_t \mid X_t > 1) \text{ with } -1/2 \leq a \leq 1, \text{ and}$$
$$\Pr(X_{t+1} - 1 \geq a \cap k_{t+1} \geq k_t \mid 0 \leq X_t \leq 1) \text{ with } 0 < a < 1.$$

If $r_{t+1} = r_t/F$ then $X_{t+1} - X_t \leq -1$. Clearly $X_{t+1} - X_t \geq a \geq -1/2$ is impossible. It also makes $X_{t+1} \geq 1$ with $0 \leq X_t \leq 1$ impossible. Thus, the two probabilities above are bounded by $\Pr(r_{t+1} = Fr_t \cap k_{t+1} \geq k_t \mid X_t \geq \tau) \leq \Pr(r_{t+1} = Fr_t \mid X_t \geq \tau) = \lambda^{-\Omega(1)}$ according to Lemma 5.32.

It remains to consider $k_{t+1} < k_t$ (equivalent to $r_u(k_{t+1}) < r_u(k_t)$). We make a case distinction based on the value of $(n - 2k_t)^2$.

Case 1: $(n - 2k_t)^2 < 2Fn \ln \lambda$. In this case, $r_u(k_t) = n^2 \ln \lambda / (12(n - 2k_t)^2) \geq n/(24F)$ which means that $X_t < 1$ for all rates $r \leq n/(2F)$. Thus $\Pr(X_{t+1} - X_t < a \cap k_{t+1} < k_t \mid X_t > 1) = 0$. When computing $\Pr(X_{t+1} - 1 \geq a \cap k_{t+1} < k_t \mid X_t \leq 1)$, we notice that $X_{t+1} \geq 1 + a$ implies $\log_F((n/2F)/r_u(k_{t+1})) \geq 1 + a + \tau$. Furthermore,

$$\frac{n/2F}{r_u(k_{t+1})} = \frac{12(n - k_{t+1})^2}{(2F)n \ln \lambda} \geq F^{1+a+\tau} = \frac{3F^{1+a}}{\sqrt{10}} \text{ if and only if } (n - k_{t+1})^2 \geq \frac{16F^{1+a} n \ln \lambda}{\sqrt{10}}.$$

Therefore a necessary condition for $X_{t+1} \geq 1 + a$ while $X_t \leq 1$ and $(n - k_t)^2 \leq 2Fn \ln \lambda$ is $k_t - k_{t+1} \geq ((4F^{(1+a)/2}/10^{1/4} - \sqrt{2F})/2)\sqrt{n \ln \lambda} > (6F^{a/2} - 4)\sqrt{n \ln \lambda}$. We notice that $E(k_{t+1} - k_t) > 0$, applying Lemma 5.7 and using a union bound we obtain for $\Delta := (6F^{a/2} - 4)\sqrt{n \ln \lambda} > 2\sqrt{n \ln \lambda}$ that

$$\begin{aligned}
\Pr(k_t - k_{t+1} > \Delta \mid X_t \leq 1) &= \Pr(k_{t+1} - k_t < -\Delta \mid X_t \leq 1) \\
&< \Pr(k_{t+1} - k_t < \mathrm{E}(k_{t+1} - k_t) - \Delta \mid X_t \leq 1) \\
&< \lambda \exp\left(\frac{-\Delta^2}{2(n/2 + \Delta/3)}\right) < \lambda \exp\left(\frac{-\Delta^2}{n + \Delta}\right) = \lambda^{-\Omega(1+a)}.
\end{aligned}$$

Therefore $\Pr(X_{t+1} - 1 \geq a \cap k_{t+1} < k_t \mid X_t \leq 1) = \lambda^{-\Omega(1+a)}$.

Case 2: $(n - 2k_t)^2 \geq 2Fn \ln \lambda$. Let

$$\sigma_t^2 \coloneqq r_u(k_t)/r_u(k_{t+1}) = (n - 2k_{t+1})^2/(n - 2k_t)^2,$$

then $X_{t+1} - X_t \in \{1 + \log_F(\sigma_t^2), -1 + \log_F(\sigma_t^2)\}$. We rewrite for $X_t > 1$ and $a \geq -1/2$

$$\begin{aligned}
&\Pr(X_{t+1} - X_t \geq a \cap k_{t+1} < k_t \mid X_t) \\
&\leq \Pr\left(r_{t+1} = r_t/F \cap \sigma_t^2 \geq F^{a+1} \mid X_t\right) + \Pr\left(r_{t+1} = Fr_t \cap \sigma_t^2 \geq F^{a-1} \mid X_t\right) \\
&\leq \Pr\left(\sigma_t^2 \geq F^{a+1} \mid X_t\right) + \Pr\left(\sigma_t^2 \geq F^{a-1} \mid X_t\right) \mathbb{1}_{a>2} + \Pr(r_{t+1} = Fr_t \mid X_t)\, \mathbb{1}_{a \leq 2},
\end{aligned} \tag{5.5}$$

as well as for $X_t \leq 1$ and $a > 0$

$$\begin{aligned}
&\Pr\left(X_{t+1} - 1 \geq a \cap k_{t+1} < k_t \mid X_t\right) = \Pr\left(X_{t+1} - X_t \geq 1 + a - X_t \cap k_{t+1} < k_t \mid X_t\right) \\
&\leq \Pr\left(r_{t+1} = r_t/F \cap \sigma_t^2 \geq F^{a+2-X_t} \mid X_t\right) + \Pr\left(r_{t+1} = Fr_t \cap \sigma_t^2 \geq F^{a-X_t} \mid X_t\right) \\
&\leq \Pr\left(\sigma_t^2 \geq F^{a+1} \mid X_t\right) + \Pr\left(r_{t+1} = Fr_t \cap \sigma_t^2 \geq F^{a-X_t} \mid X_t\right), \qquad (5.6)
\end{aligned}$$

where the second item in the above inequality (5.6) is furthermore bounded in (5.7) by making a distinction between $X_t \geq \tau \wedge a \leq 2$ and the remaining cases.

$$\begin{aligned}
&\Pr\left(r_{t+1} = Fr_t \cap \sigma_t^2 \geq F^{a-X_t} \mid X_t\right) \\
&\leq \Pr\left(\sigma_t^2 \geq F^{a-X_t} \mid X_t\right) \mathbb{1}_{X_t < \tau \vee a > 2} + \Pr\left(r_{t+1} = Fr_t \mid X_t\right) \mathbb{1}_{X_t \geq \tau \wedge a \leq 2}. \qquad (5.7)
\end{aligned}$$

Applying Lemma 5.32 we see that both $\Pr\left(r_{t+1} = Fr_t \mid X_t\right) \mathbb{1}_{a \leq 2}$ from (5.5) and $\Pr\left(r_{t+1} = Fr_t \mid X_t\right) \mathbb{1}_{X_t \geq \tau \wedge a \leq 2}$ from (5.7) are of order $\lambda^{-\Omega(1)}$. This $\Omega(1)$ exponent is sufficient to prove the lemma for $a \leq 2$. We also notice that the event $\sigma_t^2 \geq F^{a-\tau}$ subsumes all the other remaining events in inequalities (5.5), (5.6), and (5.7). Therefore it remains to validate $\Pr\left(\sigma_t^2 \geq F^{a-\tau} \mid X_t\right) \leq \lambda^{-\Omega(a+1)}$ for $a \geq 0$. To ease representation, let $s := F^{(a-\tau)/2} - 1 \geq F^{-\tau/2} - 1 = (10/9)^{1/4} - 1 > 1/40$. Since $s = \Omega(1 + a)$, proving $\Pr(\sigma_t \geq 1 + s \mid X_t) = O\left(\lambda^{-\Omega(s)}\right)$ is sufficient to conclude the analysis of this case and therefore the lemma. We rewrite

$$\begin{aligned}
\Pr\left(\sigma_t \geq 1 + s \mid X_t\right) &= \Pr\left(\frac{n - 2k_{t+1}}{n - 2k_t} \geq 1 + s \mid X_t\right) \\
&= \Pr\left(k_t - k_{t+1} \geq s(n - 2k_t)/2 \mid X_t\right).
\end{aligned}$$

Let $\Delta := (s/2 + p)(n - 2k_t)$ for $0 < p \leq 1/2$. Applying Lemma 5.7 and using a union bound we obtain

$$\begin{aligned}
&\Pr\left(\sigma_t \geq 1 + s \mid X_t\right) < \lambda \exp\left(\max_{0<p\leq 1/2}\left\{\frac{-\Delta^2}{2(1-p)(pn + \Delta/3)}\right\}\right) \\
&< \lambda \exp\left(\max_{0<p\leq 1/2}\left\{\frac{-\Delta}{2(1+1/3)}\mathbb{1}_{pn\leq\Delta} + \frac{-\Delta^2}{2(1-p)(pn)(1+1/3)}\mathbb{1}_{pn>\Delta}\right\}\right) \\
&< \lambda \exp\left(-\min_{0<p\leq 1/2}\left\{\frac{\Delta}{3}\mathbb{1}_{pn\leq\Delta} + \frac{\Delta^2}{3(1-p)(pn)}\mathbb{1}_{pn>\Delta}\right\}\right)
\end{aligned}$$

We notice that $\Delta \geq (s/2)\sqrt{2Fn\ln(\lambda)} = 4s\sqrt{n\ln(\lambda)}$ and $(s/2 + p)^2/((1-p)p)$ attains the minimal value $s(2 + s) > 2s$ when $p = s/(2(s + 1))$. Using the fact that $(n - 2k_t)^2/n \geq 2F\ln(\lambda)$ and $s > 1/40$,

$$\begin{aligned}
&\Pr\left(\sigma_t \geq 1 + s \mid X_t\right) < \lambda \exp\left(-\min\left\{s\sqrt{n\ln\lambda}\mathbb{1}_{pn\leq\Delta} + \frac{(2s)2F\ln\lambda}{3}\mathbb{1}_{pn>\Delta}\right\}\right) \\
&< \lambda \exp\left(-\min\left\{s\sqrt{n\ln(\lambda)}\mathbb{1}_{pn\leq\Delta} + 42s\ln(\lambda)\mathbb{1}_{pn>\Delta}\right\}\right) = \lambda^{-\Omega(s)}.
\end{aligned}$$

□

We finally use Lemma 5.8 and Lemma 2.12 to obtain a logarithmic drift on average. After this major effort, it is a matter of a relatively straightforward drift analysis of fitness distance to obtain the following bound on the time to leave the far region.

Theorem 5.9. *The (1,λ) EA with self-adapting mutation rate reaches a* OneMax*-value of $k \leq n/\lambda$ within an expected number of $O(n/\log\lambda)$ iterations, regardless of the initial mutation rate. Furthermore, with probability at least $1-o(1)$, it holds $k_{t'} \leq 2n/\lambda$ and $r_{t'} \leq (7/9)\ln\lambda$ for some $t' = O(n/\log\lambda)$.*

Proof. We first argue that within an expected number of $O(\sqrt{n})$ generations we will have $k_t < n/2$. Consider the case that $k_t \geq n/2$ and let the independent random variables X and Y denote the number of flips in k_t one-bits and $(n-k_t)$ zero-bits, respectively, in an offspring using rate $p = r/n$. Referring to Lemma 2.16 for $p \in [2/n, 1/2]$ we obtain, using similar arguments in the proof Lemma 5.32 that $\Pr(X \geq \mathrm{E}(X)+1) = \Theta(1)$ and $\Pr(Y \leq \mathrm{E}(Y)) = \Theta(1)$. Then $\Pr(X - Y \geq \mathrm{E}(X) - \mathrm{E}(Y) + 1) = \Theta(1)$. Since $\mathrm{E}(X) \geq E(Y)$, the probability that an offspring choose rates $\tilde{r} \in \{r_t/F, Fr_t\}$ with $2 \leq \tilde{r} \leq n/2$ and have $X - Y \geq 1$ is at least $1/2 \cdot \Theta(1) = \Theta(1)$. Since the best of $\lambda = \Omega(\ln n)$ offspring is selected, the probability that $k_{t+1} \leq k_t - 1$ holds is at least $1 - \exp(-\Theta(\lambda)) = 1 - o(1/n^2)$. By an additive drift theorem, it takes $O(\max\{k_0 - n/2, 0\}) = O(\sqrt{n})$ iterations from the initial random search point to reach a parent with fitness distance less than $n/2$.

Without loss of generality, we can now assume $k_0 < n/2$. Consider the number of one-bits flips X and zero-bits flips Y in a parent with fitness distance $k_t < n/2$ and rate $2 \leq r < n/2$. As argued above $\Pr(X - Y \geq E(X) - E(Y) + 1) = \Theta(1)$. Since $k_t - (\mathrm{E}(X) - \mathrm{E}(Y)) = k_t - (k_t - (n-k_t))r/n = k_t(1 - r/n) + (n-k_t)(r/n) < n/2$ for all $r < n/2$, the probability that an offspring has fitness distance at most $n/2 - 1$ is $\Theta(1)$. Thus for $\lambda = \Omega(\ln n)$ offspring, we have $\Pr(k_{t+1} < n/2) \geq 1 - \exp(-\Theta(\lambda)) = 1 - o(1/n^2)$. Since that we aim at proving a hitting time of $O(n/\ln\lambda)$ and only consider phases of this length, we may furthermore assume $k_t < n/2$ for all $t \geq 0$, which only introduces an $o(1)$ error term by a union bound.

Define random variables $X_t := \log_F(r_t/r_u(k_t)) - \tau$ with $\tau = \log_F(3/\sqrt{10}) < 0$. We notice that when $(n-2k_t)^2 \leq 2Fn\ln\lambda$ we have $X_t < 1$. If $r_t \geq Fr_u(k_t)$, according to Lemma 5.32, with probability $1 - o(1)$ we have $r_{t-1} = r_t/F$. Therefore within $O(\ln n)$ iterations we will obtain $X_t \leq 1$.

The idea of the remaining proof is to compute an average drift for any fixed distance using the distribution of mutation rates, and then to apply the variable drift theorem to obtain a runtime bound. Applying Lemma 5.8 and Lemma 2.12 to the X_t, we see that

$$\Pr(r_t \geq F^{1+a+\tau} r_u(k_t)) \leq \lambda^{-\Omega(a)} \text{ for all } a > 0.$$

Let $r^{(i)}, i \in \mathbb{Z}$, denote the rate between $(F^i r_u(k), F^{i+1} r_u(k)]$ corresponding to fitness distance k. Thus, for all $i \geq 1$, we obtain

$$\Pr\left(r^{(i)}\right) \leq \Pr\left(r_t > F^i r_u(k_t)\right) \leq \Pr\left(r_t \geq F^{1+(i-1-\tau)+\tau} r_u(k_t)\right) \leq \lambda^{-\Omega(i-1-\tau)}.$$

According to Lemma 5.6, $E(\Delta(k, r^{(i)})) \geq -(1+o(1))(n-2k)r^{(i)}/n)$ for $i \geq 1$ and $E(\Delta(k, r^{(0)})) \geq \Omega((n-2k)r^{(0)}/n)$. The contribution of the negative drift is a lower order term compared to the contribution of the positive drift. Let $\Delta^{(k)}$ denote the average drift at distance k. We obtain

$$\Delta^{(k)} = \sum_{i \in \mathbb{Z}} E(\Delta(k, r^{(i)})) \Pr(r^{(i)}) \geq (1 - o(1)) \sum_{i \leq 0} E(\Delta(k, r^{(i)})) \Pr(r^{(i)}).$$

We notice that $\sum_{i\leq 0}\Pr(r^{(i)}) = 1-o(1)$ and $E(\Delta(k,r^{(i)})) > 0$ for all $i \leq 0$. According to Lemma 5.31, with at least constant probability $r_t = \Omega(r_l(k_t))$. Since for any rate $r = \Omega(r_l(k))$ and $r \leq Fr_u$ the drift is $\mathrm{E}(\Delta(k,r)) \geq \Theta(\ln(\lambda)/\ln(n/k_t))$ according to Lemma 5.6, the average drift satisfies

$$\Delta^{(k)} \geq \Theta(\ln(\lambda)/\ln(n/k)).$$

Using the variable drift theorem (Theorem 2.4) and the fact that

$$\int_{n/\lambda}^{n/2} \frac{\ln(n/k)}{\ln(\lambda)}\,\mathrm{d}k = \frac{\Big(k\ln(n) - k\ln(k) + k\Big)\Big|_{n/\lambda}^{n/2}}{\ln\lambda} = \frac{\Theta(n)}{\ln\lambda},$$

the expected time to reduce the fitness distance to at most n/λ conditioning on the assumption that $k_t < n/2$ for some $t = O(\sqrt{n})$ and $k_{t'} < n/2$ for all $t \leq t' = O(n/\log\lambda)$ is then $\Theta(n/\log\lambda)$. Thus the expected runtime of $O(n/\log\lambda)$ holds with probability $\Omega(1)$ due to Markov's inequality. Using a restart argument we then obtain the claimed expected runtime since the expected number of repetition of a phase of length $O(n/\log\lambda)$ is $O(1)$.

To prove the second statement of the theorem, we notice that the corresponding upper bound on the rate for $k_t = o(n)$ is $r_u(k_t) \leq (10/9)(U(k_t))\ln\lambda = ((10/9)(2/22) + o(1))\ln\lambda < (1/9)\ln\lambda$ and the occupation probability satisfies $\Pr(r_t \leq (7/9)F\ln\lambda \mid k_t = o(n)) \geq 1-\lambda^{-\Omega(1)} = 1-o(1)$. Therefore with probability $1-o(1)$, the first iteration such that $k_t \leq 2n/\lambda$ has rate $r_t \leq (7/9)\ln\lambda$. We then argue for this iteration that with high probability it satisfies $r_{t+1} = r_t/F$ and $k_{t+1} \leq 2n/\lambda$. The probability of being no worse than parent using mutation probability $p \leq (7/9)\ln(\lambda)/n$ is at least $(1-p)^n \geq (1-o(1))\lambda^{-7/9} > \lambda^{-8/9}$. Therefore,

$$\Pr(k_{t+1} \leq k_t \mid r_t \leq (7/9)F\ln\lambda) \geq 1-\Big(1-\lambda^{-8/9}/2\Big)^{\lambda} = 1-o(1).$$

Furthermore $\Pr(r_{t+1} = Fr_t \mid k_t = o(n), r_t \geq (7/9)\ln\lambda) \leq \lambda^{1-(23/22)7} = o(1)$. Then we obtain an iteration with $k_t \leq 2n/\lambda$ and $r_t \leq (7/9)\ln\lambda$ with probability $1-o(1)$. □

5.3 The Near Region

We now analyze the drift in rate and fitness distance in the regime in which the fitness distance satisfies $k = k_t = O(n/\lambda)$, the so-called near region. Informally speaking, this region is responsible for the $(n\log n)/\lambda$ term in the expected number of generations since here the probability of an improvement is only $O(1/\lambda)$ and the offspring population can boost this probability by a factor of $\Theta(\lambda)$, assuming constant rate.

We start with a preparatory lemma determining the probability of making progress in one mutation and similar events.

Lemma 5.10. *Let $0 < k \leq 3n/\lambda$, and $r = o(\lambda^{1/4})$. Let $x \in \{0,1\}^n$ with fitness distance $f(x) = k$. Let $y \in \{0,1\}$ be obtained from x by flipping each bit independently*

with probability r/n. Consider the probabilities

$$\begin{aligned} p_-(r) &:= \Pr(f(y) < f(x)), \\ p_0(r) &:= \Pr(f(y) = f(x)), \\ p'(r) &:= \Pr(\forall i \in [1..n] : x_i = 0 \implies y_i = 0), \end{aligned}$$

that is, the probabilities that the offspring is better than the parent, that is is equally good, and that none of the 0-bits of the parent were flipped in the generation of the offspring.

Then

$$\begin{aligned} (1-o(1))\tfrac{kr}{n}e^{-r} < p_-(r) < (1+o(1))\tfrac{kr}{n}e^{-r}, \\ (1-o(1))e^{-r} < p_0(r) < (1+o(1))e^{-r}, \\ (1-o(1))e^{-r} < p'(r) < (1+o(1))e^{-r}. \end{aligned}$$

Proof. We regard the number X of flips in the k one-bits ("good flips" which reduce the fitness distance) and the number Y of flips in the $(n-k)$ zero-bits of the parent ("bad flips" which increase the fitness distance). Then $p_-(r)$ is at least

$$p_-(r) \geq \Pr(X = 1, Y = 0) = \frac{kr}{n}\left(1-\frac{r}{n}\right)^{n-1} \geq (1-o(1))\frac{kr}{n}e^{-r},$$

where the last estimate uses Lemma 2.14 3.

Since $r = o(\lambda^{1/4})$, we have $kr/n = o(1)$, $kr^2/n = o(1)$, and $(kr^2/n)^{1.5} = o(kr/n)$. This allows to bound $p_-(r)$ from above by

$$\begin{aligned} p_-(r) &< \Pr(X \in \{1,2\}, Y = 0) + \sum_{i=3}^{2k-1} \Pr(X + Y = i, X > Y) \\ &< \frac{kr}{n}\left(1-\frac{r}{n}\right)^{n-1} + \frac{k^2r^2}{2n^2}\left(1-\frac{r}{n}\right)^{n-2} \\ &\quad + \sum_{i=3}^{2k-1}(i-1)\left(\frac{r}{n}\right)^i\left(1-\frac{r}{n}\right)^{n-i}\binom{k}{\lceil i/2\rceil}\binom{n-k}{\lfloor i/2\rfloor} \\ &< \frac{kr}{n}\left(1-\frac{r}{n}\right)^{n-2}\left(1-\frac{r}{n}+\frac{kr}{2n}\right) + \sum_{i=3}^{2k-1}\left(\frac{r}{n}\right)^i\left(1-\frac{r}{n}\right)^{n-i}(kn)^{i/2} \\ &< (1+o(1))\frac{kr}{n}\left(1-\frac{r}{n}\right)^{n} + \sum_{i=3}^{2k-1}\left(\frac{kr^2}{n}\right)^{i/2}\left(1-\frac{r}{n}\right)^{n-i} \\ &< (1+o(1))\frac{kr}{n}e^{-r}. \end{aligned}$$

Similarly for $p_0(r)$ we have

$$p_0(r) > \Pr(X = Y = 0) = \left(1-\frac{r}{n}\right)^n \geq (1-o(1))e^{-r}.$$

Using again the fact that $kr^2/n = o(1)$, we have

$$\begin{aligned} p_0(r) &= \Pr(X = Y = 0) + \sum_{i=1}^{k} \Pr(X = Y = i) \\ &= \left(1 - \frac{r}{n}\right)^n + \sum_{i=1}^{k} \binom{k}{i}\binom{n-k}{i}\left(\frac{r}{n}\right)^{2i}\left(1 - \frac{r}{n}\right)^{n-2i} \\ &< e^{-r} + \sum_{i=1}^{k}\left(\frac{kr^2}{n}\right)^i e^{-r} < (1 + o(1))e^{-r}. \end{aligned}$$

Finally, for $p'(r)$ we compute $p'(r) = \Pr(Y = 0) = (1 - \frac{r}{n})^{n-k} = (1 \pm o(1))e^{-r}$. □

Lemma 5.11. *Consider one iteration of the self-adaptive (1,λ) EA starting with an individual of fitness distance k and rate $r = o(\lambda^{-1/4})$. Then the probability that there is an offspring which uses rate r/F and which inherits all 0-bits from the parent (and thus is at least as good as the parent), is at least $1 - \exp(-\frac{1}{2}\lambda(1 - o(1))e^{-r/F})$.*

Proof. We compute

$$1 - (1 - \tfrac{1}{2}p'(\tfrac{r}{F}))^\lambda \geq 1 - (1 - \tfrac{1}{2}(1 - o(1))e^{-r/F})^\lambda \geq 1 - \exp(-\tfrac{1}{2}\lambda(1 - o(1))e^{-r/F}).$$

□

The following lemma is the counterpart of Lemma 5.3 2, where now the optimal rate is the smallest possible value F. Again, we regard the event that all best offspring are created with the higher rate, since—due to our tie-breaking rule—only this leads to an increase of the rate. Different from Lemma 5.3 2, now the probability of making a rate-increasing step is no $o(1)$ in general. If $k_t = \Theta(n/\lambda)$ and $r_t = O(1)$, we still have a small constant probability of increasing the rate.

Lemma 5.12. *Let $0 < k \leq 3n/\lambda$ and $F = 32$. The probability that all best offspring have been created with rate Fr is at most $(1 + o(1))\frac{\lambda kFr}{n}e^{-Fr}$ when $r < \ln\lambda$ and it is at most $\exp(-9r)$ for all r.*

Proof. Let first $r < \ln\lambda$. According to Lemma 5.10,

$$p_-(Fr) \leq (1 + o(1))\frac{Fkr}{n}e^{-Fr} \quad \text{and} \quad p_0(r/F) \geq (1 - o(1))e^{-r/F}.$$

Therefore with probability at least $1 - \lambda p_-(Fr) = 1 - (1 + o(1))\frac{\lambda Fkr}{n}e^{-Fr}$, no offspring of rate Fr is better than its parent. Furthermore, by Lemma 5.11, with probability at most $\exp(-(1 - o(1))\frac{1}{2}\lambda\exp(-r/F)) \leq \exp(-(1 - o(1))\frac{1}{2}\lambda^{1-1/F})$ there is no offspring using rate r/F and being equally good as its parent. Hence, the probability that a best offspring has been created with rate r/F is more than

$$1 - (1 + o(1))\frac{\lambda Fkr}{n}e^{-Fr} - \exp(-(1 - o(1))\tfrac{1}{2}\lambda^{1-1/F}) > 1 - (1 + o(1))\frac{\lambda Fkr}{n}e^{-Fr}.$$

Note that for $r < \ln\lambda$, the second bound follows from the first. If $r \geq \ln\lambda$, then the second bound follows from applying Lemma 5.3 to $U(k) = 1/11 + o(1)$. □

We shall use the lemma above twice, first to bound the probability to have a certain rate (which will be needed to estimate the negative fitness drift) and second

to estimate that a suitable two-dimensional drift is of the right order. We start with the occupation probability argument for the rate values.

Lemma 5.13. *Consider a run of the self-adaptive (1,λ) EA started with some search point of fitness distance $k_0 \leq 2n/\lambda$ and rate $r_0 = F$. While the current search point of the algorithm has a fitness distance of at most $3n/\lambda$, the probability that the current rate is F^i is at most $\exp(-8F^{i-1})$ for all $i \in \mathbb{N}_{\geq 2}$.*

Proof. If the current search point has fitness distance at most $3n/\lambda$ and the current rate is r, then by Lemma 5.12 the rate in the next iteration is Fr with probability at most $\exp(-9r)$; note that this estimate is not affected by a possible cap of the rate at $r_{\max}$.

Consequently, the random process describing the rates is such that from rate F^i, $i \in [1..\log_F(r_{\max})]$, we go to rate F^{i+1} with probability at most $p_i = \exp(-9F^i)$. Otherwise, we go to rate F^{i-1} if $i \geq 2$ and stay at rate F if $i = 1$. By Lemma 2.13, note that we obviously have $p_i/(1-p_i) \leq p_{i-1}$, in each iteration (such that the fitness distance has never gone above $3n/\lambda$) and for each $i \geq 2$ the probability q_i that the current rate is F^i is at most

$$q_i \leq \prod_{j=1}^{i-1} \frac{p_j}{1-p_j} \leq \frac{p_{i-1}}{1-p_{i-1}} \leq \exp(-8F^{i-1}).$$

□

We use these occupation probabilities to estimate the drift away from the optimum ("negative drift"). From this we derive the statement that with high probability, the fitness distance does not increase to above $3n/\lambda$ in $n\lambda$ iterations.

Lemma 5.14. *In the situation of Lemma 5.13, the probability that the process within the first $n\lambda$ iterations reaches a search point (as parent individual) with fitness distance more than $3n/\lambda$, is $o(1)$.*

Proof. Consider a run of the self-adjusting (1,λ) EA starting in the situation of Lemma 5.13. Denote by X_t the fitness distance at time t. We start by bounding the negative drift $E(\max\{0, X_t - X_{t-1}\})$ of the X process while it is at most $3n/\lambda$. If the current rate is r, then by Lemma 5.11 with probability at least $1 - \exp(-\frac{1}{2}\lambda(1-o(1))e^{-r/F})$ there is an individual that used rate r/F and that did not flip any zero-bit into a one-bit. Let us call this event "A" and note that, naturally, under this event the drift cannot be negative as the individual without flipped zeroes has at an least as good fitness as the parent.

We now analyze the case that A does not hold. Consider an individual conditional on that it uses rate r/F and at least one zero-bit was flipped into a one-bit. The number of such bad bits follows a distribution $(X \mid X \geq 1)$ with $X \sim \mathrm{Bin}(n - X_{t-1}, r/Fn)$ and has expectation at most $1 + r/F$ by Lemma 2.16. For an individual using rate rF, the expected number of bad flips is $(n-k)\frac{rF}{n} \leq rF$. Consequently, noting that $1 + r/F \leq rF$ when $r \geq F$ and $F \geq \sqrt{2}$, the expected number of bad flips in all individuals (conditional on not A) is at most λrF and this is an upper bound on the negative drift.

In summary, in an iteration starting with rate r, the negative drift is at most

$$\lambda rF \exp(-\tfrac{1}{2}\lambda(1-o(1))e^{-r/F}). \tag{5.8}$$

With Lemma 5.13, we can estimate the probability to have a certain rate. Hence the expected negative drift is

$$\begin{aligned} E(\max\{0, X_t - X_{t-1}\}) &\leq \sum_{i=1}^{\log_F r_{\max}} \Pr(r = F^i)\lambda F^i F \exp(-\tfrac{1}{2}\lambda(1-o(1))e^{-F^i/F}) \\ &\leq \sum_{i=2}^{\infty} \exp(-8F^{i-1})\lambda F^{i+1} \exp(-\tfrac{1}{2}\lambda(1-o(1))e^{-F^{i-1}}) \\ &\quad + \lambda F^2 \exp(-\tfrac{1}{2}\lambda(1-o(1))e^{-1}). \end{aligned}$$

Note that [1] for $i \geq \lceil \log_F(\ln \lambda) + 1 - \frac{1}{5} \rceil = i^*$, we have $\lambda \leq \exp(2F^{i-1})$ and thus $\exp(-8F^{i-1})\lambda F^{i+1} = \exp(-(1-o(1))8F^{i-1})\lambda \leq \exp(-(1-o(1))6F^{i-1})$. Naturally, $\exp(-\frac{1}{2}\lambda(1-o(1))e^{-F^{i-1}}) \leq 1$. Hence

$$\begin{aligned} \sum_{i=i^*}^{\infty} \exp(-8F^{i-1})\lambda F^{i+1} \exp(-\tfrac{1}{2}\lambda(1-o(1))e^{-F^{i-1}}) &\leq \sum_{i=i^*}^{\infty} \exp(-(1-o(1))6F^{i-1}) \\ &\leq \exp(-(1-o(1))6F^{i^*-1}) \leq \lambda^{-3(1-o(1))}. \end{aligned}$$

For $i < \log_F(\ln \lambda) + 1 - \frac{1}{5}$, we have $\exp(-\frac{1}{2}\lambda(1-o(1))e^{-F^{i-1}}) \leq \exp(-\frac{1}{2}(1-o(1))\lambda^{1/2})$ and $\exp(-8F^{i-1})\lambda F^{i+1} = O(\lambda)$. Hence

$$\begin{aligned} &\sum_{i=1}^{i^*-1} \exp(-8F^{i-1})\lambda F^{i+1} \exp(-\tfrac{1}{2}\lambda(1-o(1))e^{-F^{i-1}}) \\ &\qquad \leq O(\log\log\lambda)O(\lambda)\exp(-\tfrac{1}{2}(1-o(1))\lambda^{1/2}) = o(\lambda^{-3}). \end{aligned}$$

Consequently, $E(\max\{0, X_t - X_{t-1}\}) \leq \lambda^{-3(1-o(1))}$.

Define inductively $Y_0 = 0$ and $Y_t = Y_{t-1} + \max\{0, X_t - X_{t-1}\}$, if $\max\{X_s \mid s \in [0..t-1]\} \leq 3n/\lambda$ and $Y_t = Y_{t-1}$ otherwise. In other words, the Y process collects all the moves of the X process that go away from the optimum until the X process goes above $3n/\lambda$.

By our above computation, we have $E(Y_t) \leq t\lambda^{-3(1-o(1))}$. Consequently, by Markov's inequality, we have

$$\Pr(Y_t \geq t\lambda^{-2}) \leq \lambda^{-1+o(1)}$$

for all $t \in \mathbb{N}$. In particular, for $t = n\lambda$, we have $\Pr(Y_t \geq n/\lambda) \leq \lambda^{-1+o(1)}$. Note that $Y_t \leq n/\lambda$ implies $X_s \leq 3n/\lambda$ for all $s \leq t$. □

Lemma 5.15. *In the situation of Lemma 5.13, with probability at least $\frac{3}{4}$ there is a $T^* = O(n\ln(n/\lambda)/\lambda)$ such that $k_{T^*} = 0$.*

Proof. Since we are proving an asymptotic statement, we can assume that n is as large as we find convenient. Consider a run of the self-adjusting (1,λ) EA from our starting position. Let T be the first time that the fitness distance is larger than $3n/\lambda$, if such a time exists, and $T = \infty$ otherwise. Let k_t denote the fitness distance at time t and r_t the rate used in iteration t, if $t \leq T$, and $(k_t, r_t) := (0, F)$ otherwise.

1. In this part of the proof, we use the fact that $F = 32$. This does not mean that for other not too small values of F we would not obtain similar results, but it increases the readability to work with this concrete value.

We show that the process (k_t, r_t) reaches $(0, F)$ in time T^* with probability at least $1 - 1/e^2$.

We use a two-dimensional drift argument. Let $\gamma = 2F$ and define $g : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ by $g(k, r) = k + \gamma(r - F)$ for all k and r. We show that if for some t we have $(k, r) = (k_t, r_t)$, then $(k', r') := (k_{t+1}, r_{r+1})$ satisfies

$$E(g(k', r')) \leq g(k, r)(1 - \tfrac{\lambda}{10n}) \tag{5.9}$$

when assuming n to be sufficiently large.

There is nothing to show in the artificial case when $k > 3n/\lambda$ as we have, by definition, $g(k', r') = 0$ in this case. Among the interesting cases, we consider first that $r = F$. We obtain an improvement in fitness in particular if there is an offspring that uses rate $r/F = 1$, flips exactly one of the k missing bits, and flips no other bit. Hence the probability to make a positive fitness progress is at least

$$1 - (1 - \tfrac{1}{2}(1 - \tfrac{1}{n})^{n-1}\tfrac{k}{n})^\lambda \geq 1 - (1 - \tfrac{k}{2en})^\lambda \geq 1 - \exp(-\tfrac{k\lambda}{2en}) \geq \tfrac{k\lambda}{3en},$$

where we used $(1 - \frac{1}{n})^{n-1} \geq \frac{1}{e}$, $\frac{k\lambda}{2en} \leq \frac{3}{2e} < \frac{3}{2} \cdot \frac{1}{2}$ and Lemma 2.14 4. The expected negative progress is at most $\lambda F^2 \exp(-(1 + o(1))\frac{1}{2e}\lambda)$ as shown in (5.8). This negative drift can be assumed to be $O(n^{-2})$ by taking the implicit constant in the assumption $\lambda = \Omega(\log n)$ large enough. Consequently, $E(k') \leq k - \frac{1}{3e}\frac{\lambda k}{n}$.

Regarding r', we note that by Lemma 5.12 we have $\Pr(r' = F^2) \leq (1 + o(1))\frac{\lambda k}{n} F^2 \exp(-F^2)$ and $r' = F$ otherwise. Hence $E(r') = F + (1 + o(1))(F - 1)F^3\frac{\lambda k}{n}\exp(-F^2)$. Consequently,

$$\begin{aligned} E(g(k, r) - g(k', r')) &\geq \tfrac{1}{3e}\tfrac{\lambda k}{n} - \gamma(1 + o(1))(F - 1)F^3\tfrac{\lambda k}{n}\exp(-F^2) \\ &= \tfrac{\lambda k}{n}(\tfrac{1}{3e} - \gamma(F - 1)F^3\exp(-F^2) - o(1)) \\ &\geq \tfrac{\lambda k}{n}\tfrac{1}{10} = g(k, r)\tfrac{\lambda}{10n}. \end{aligned}$$

Let now be $r > F$. Note that the minimum fitness loss among the offspring is at most the minimum number of bits flipped, which in expectation is at most the number of bits flipped in the first offspring, which is exactly Fr. Consequently, we have $E(k') \leq k + Fr$. For r', we note that by Lemma 5.12, we have $r' = Fr$ with probability at most $\exp(-9r)$ and we have $r' = \frac{r}{F}$ otherwise. Consequently, $E(r') \leq Fr\exp(-9r) + \frac{r}{F}$. This yields

$$\begin{aligned} E(g(k, r) - g(k', r')) &\geq -Fr + \gamma(r - Fr\exp(-9r) - \tfrac{r}{F}) \\ &\geq r(-F + \gamma - F\exp(-9F^2) - \tfrac{1}{F}) \\ &\geq r(-F + \gamma - \tfrac{2}{F}) \geq 31r = r + 30r \geq F^2 + 30r \\ &\geq 32^2 + 30r \geq \tfrac{\lambda}{10n}(k + \gamma r) \geq g(k, r)\tfrac{\lambda}{10n}, \end{aligned}$$

where we used that $\lambda \leq 2n$; note that $\lambda > 2n$ gives $k_0 = 0$.

We have thus shown (5.9) for all (k, r). Since we start the process with a g-potential of at most $g(2n/\lambda, F) = \frac{2n}{\lambda}$, the multiplicative drift theorem gives that after $t = \lceil \frac{10n}{\lambda}(2 + \ln(\frac{2n}{\lambda}))\rceil$ iterations, we have $\Pr(g(k_t, r_t) > 0) \leq \frac{1}{e^2}$. Consequently, with probability $1 - \frac{1}{e^2}$, the potential is zero at time t, which implies $k_t = 0$ or $k_t > \frac{3n}{\lambda}$. By Lemma 5.14, note that $\lambda = \Omega(\log n)$ implies $t = O(n) = o(n\lambda)$, the probability that $k_t > \frac{3n}{\lambda}$ is $o(1)$, hence with probability at least $\frac{3}{4}$, we have indeed $k_t = 0$. □

Theorem 5.16. *Assume $k_0 \leq \frac{2n}{\lambda}$ and $r_0 \leq \frac{7}{9}\ln\lambda$. Then there is a $t = O(n\ln(n/\lambda)/\lambda)$ such that with probability at least $\frac{1}{2}$, we have $k_t = 0$.*

Proof. We first show that with good probability we quickly reach the initial situation of Lemma 5.15. The probability of observing $R^* - 1 := \log_F(r_0) - 1 \leq \log_F(\frac{7}{9}\ln\lambda) - 1$ rate-decreasing steps in a row by Lemma 5.12 is at least

$$\prod_{i=2}^{R^*} \left(1 - \exp(-9F^i)\right) \geq 1 - \sum_{i=2}^{R^*} \exp(-9F^i) \geq 1 - 0.001$$

by the Weierstrass product inequality (Lemma 2.14 5).

The probability of not flipping any zero-bits in at least one offspring, resulting in not increasing fitness distance, is for rate $r \leq \frac{7}{9}\ln\lambda$ at least $1 - \exp(-\frac{1}{2}\lambda(1 - o(1))e^{-r/F})$ by Lemma 5.11. By a union bound over $R^* - 1$ iterations, the probability of decreasing the initial rate to F in $O(\log\log\lambda)$ iterations without losing fitness is at least $1 - 0.001 - (R^* - 1)\exp(-\frac{1}{2}\lambda(1 - o(1))e^{-r/F}) \geq 5/6$ for sufficiently large n.

We can now apply Lemma 5.15 and obtain that with probability at least $\frac{3}{4}$ we have found the optimum within $t = O(n\ln(n/\lambda)/\lambda)$ iterations. This show the claim. □

5.4 Putting Everything Together

We can now put everything together to prove our main result.

Proof. Starting with arbitrary initialization, Theorem 5.9 along with a Markov bound yield that with probability $\Omega(1)$ after $t = O(n/\log\lambda)$ iterations a search point is reached such that $k_t \leq 2n/\lambda$ and $r_t < 0.6(\ln\lambda)$. Assuming this to happen, the assumptions of Theorem 5.16 are satisfied. Hence, after another $O((n\log n)/\lambda)$ iterations the optimum is found with probability at least $1/2$. Altogether, with probability $\Omega(1)$ the optimum is found from an arbitrary initial OneMax-value and rate within $T^* = O(n/\log\lambda + (n\log n)/\lambda)$ iterations. The claimed expected time now follows by a standard restart argument, more precisely by observing that after expected $O(1)$ repetitions of a phase of length T^* the optimum is found. □

5.5 Experimental Results

To gain some insight that cannot be derived from our asymptotic analysis, we performed a few numerical experiments. To this end we implemented the (1,λ) EA in C++11 using the *default random engine* to generate pseudo-random numbers. The runtime is still measured via the number of generations until optimum is found.

We first see in Figure 5.1 how fitness distance and mutation strength evolve in one run for $n = 100$, $\lambda = 12$ and $F = 1.2$. We used this small value of n to increase the readability of the figure, we used larger values for n in the remainder. Given the small value of n, we used a small mutation update factor of 1.2 instead of the value $F = 32$ used in our theoretical analysis. This run uses Algorithm 9 with $r^{\text{init}} = F$. We see that the algorithm prefers large mutation strengths at the beginning and small mutation strengths near the end of the optimization process. We also see that fitness distance can increase occasionally, in particular, when the rate is higher (in the plot, this happened in iteration 52 and iteration 88).

In Figure 5.2, we display the average runtime over 100 runs of different versions of the (1,λ) EA on OneMax for $n = 10^5$ and $\lambda = 100, 200, \ldots, 1000$. For our self-adaptive (1,λ) EA (Algorithm 9), we used the update strengths $F \in \{1.2, 2, 32\}$. We did experiments also for $F = 1.05$, but the results were clearly inferior, so to not overload this figure we do not visualize them. We always set the initial mutation strength to $r^{\text{init}} = F$. We further regard the classic (1,λ) EA using a static mutation rate of

$\frac{1}{n}$ and the (1,λ) EA with fitness-dependent mutation rate $p = \max\{\frac{\ln \lambda}{n \ln(en/d)}, \frac{1}{n}\}$ as presented in [BLS14].

The results clearly show that the update factor of $F = 32$ used in our mathematical analysis gives sub-optimal results for these values of λ and n. Recalling the working principle of the self-adaptive (1,λ) EA, this is not overly surprising. Even using the minimal possible rate $r = F$, the algorithm creates half of the offspring using an incredible large mutation probability of $F^2/n = 1024/n$. It is quite clear that this cannot be overly effective, but this can also be seen from the figure. The runtime of the self-adaptive (1,λ) EA with $F = 32$ is very close to the runtime of the static (1,λ) EA for half the λ-value, suggesting that half the offspring created by the self-adaptive (1,λ) EA, most likely the ones created with a mutation rate of F^2/n, had no impact on the process.

The results in Figure 5.2 also show that the fitness-dependent mutation strength of [BLS14] leads to a very good performance. In principle, of course, it is clear that the best fitness-dependent rate gives better results than any self-regulating rate since the latter needs to use also sub-optimal rates to find out what is the best rate. That the rate suggested in [BLS14], a paper mostly concerned with asymptotic runtimes, shows such good results, is remarkable.

To ease the comparison of the algorithms having a similar performance, we plot in Figure 5.3 these runtimes relative to the one of the classic (1,λ) EA. This shows that in most cases, the EAs using a dynamic mutation rate outperform the classic (1,λ) EA. We also notice that the self-adaptive EA appears to outperform the one using the fitness-dependent rate for sufficiently large values of λ, e.g., for $\lambda \geq 200$ when $F = 1.2$.

To understand how our tie-breaking rule influences the performance, we also ran the self-adapting (1,λ) EA without the bias towards smaller rates when breaking ties. In Figure 5.4, we again plot the average runtimes over 100 runs relative to the results of static (1,λ) EA. We use the three update factors 1.2, 2, and 32 and the two tie-breaking rule of preferring the smaller rate in case of ties (as in our theoretical analysis) and random tie-breaking, that is, choosing uniformly at random an offspring with maximal fitness and taking its rate as the new rate of the algorithm. While for the two larger factors $F = 2$ and $F = 32$, no significant differences are visible, we see that for $F = 1.2$ random tie-breaking surpasses biased tie-breaking significantly when λ become larger than 200.

To understand how the tie-breaking rule influences the mutation strength chosen by the algorithm, we plot in Figure 5.5 the mutation strength used at each fitness distance with a setting of $n = 10000$, $\lambda = 500$, and $F = 1.2$. We regarded one exemplary runs of our algorithm with each tie breaking rule. In each of these two experiments, we determined the set of all pairs (d_t, r_t) such that in iteration t, the fitness distance of the parent individual was d_t and its rate was r_t. We then plotted these sets, where to increase the readability we connected the points to polygonal curve. This visualization clearly shows that random tie breaking lets the algorithm pick larger rates more frequently. Together with the better runtimes, it appears that biased tie-breaking has a small negative effect on the choice of the mutation strength.

Finally, we regard the question of how to set the initial rate r^{init}. From the general experience that larger mutation rates are more profitable at the start of the search process, one could guess that it is a good idea to start with the largest possible rate $r_{\max} = F^{\lfloor \log_F(n/(2F)) \rfloor}$ instead of the smallest possible rate $r_{\min} = F$. For the settings used in Figure 5.5, that is, $n = 10000$, $\lambda = 500$, and $F = 1.2$, we obtain (as average of 100 runs) the runtimes given in Table 5.1. So indeed an initialization with a larger rate gives some improvement. Since it might be a particularity of the

Average runtime	Biased ties breaking	Random ties breaking
$r^{\text{init}} = r_{\min}$	2137	2011
$r^{\text{init}} = r_{\max}$	2080	1974

TABLE 5.1: Comparison of the average runtime of 100 runs for different initial mutation rates ($n = 10000$, $\lambda = 500$, and $F = 1.2$)

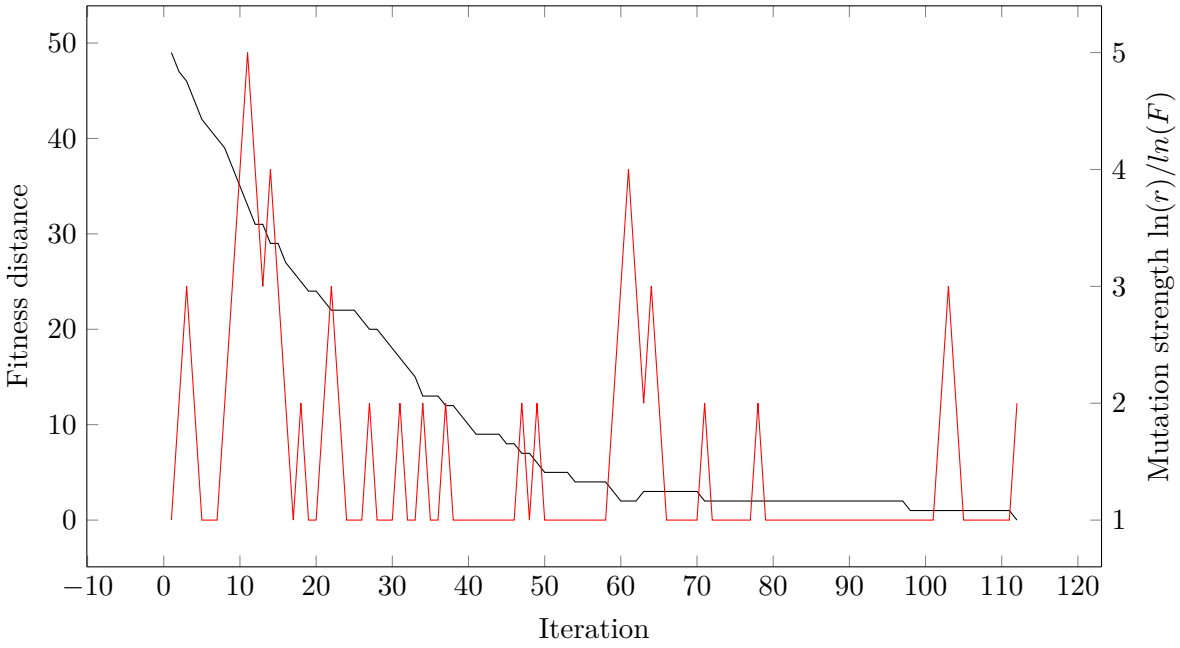

FIGURE 5.1: Development of fitness distance and mutation strength in one run of self-adapting (1,λ) EA on ONEMAX ($n = 100$, $F = 1.2$, $\lambda = 12$)

ONEMAX test function that huge rates are initially beneficial, we would not give out a general recommendation to start with the rate $r_{\max}$, but only state that we observed moderate performance differences from using different initial rates, making the initial rate not the most critical parameter of the algorithm, but still one that can be worth optimizing.

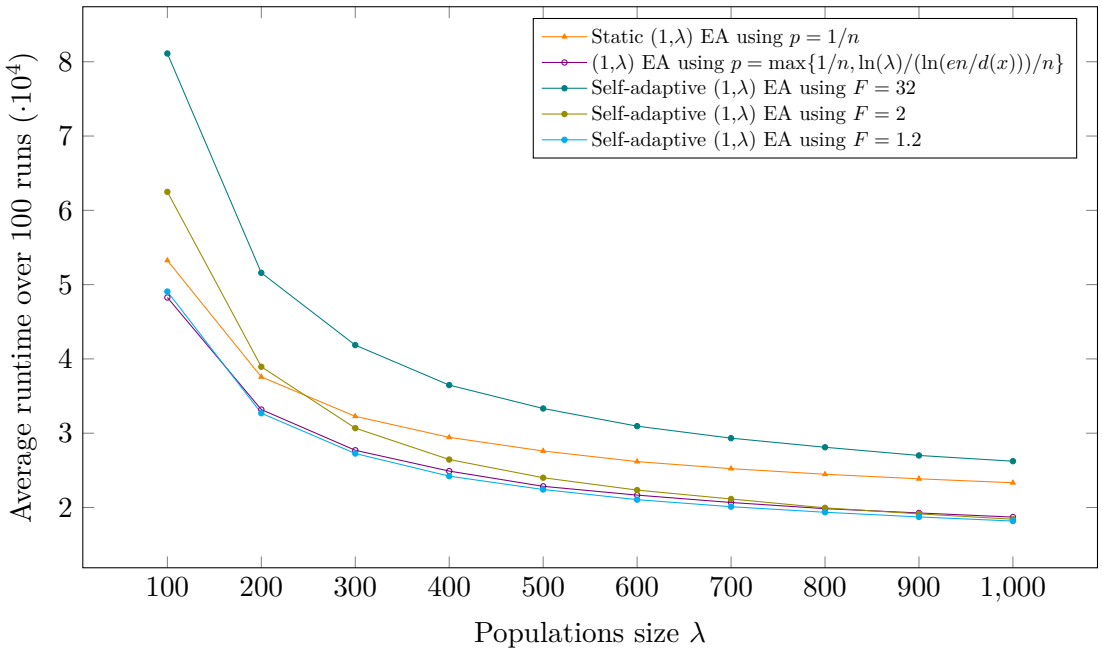

FIGURE 5.2: Average runtime over 100 runs of five variants of the (1,λ) EA on ONEMAX for $n = 10^5$.

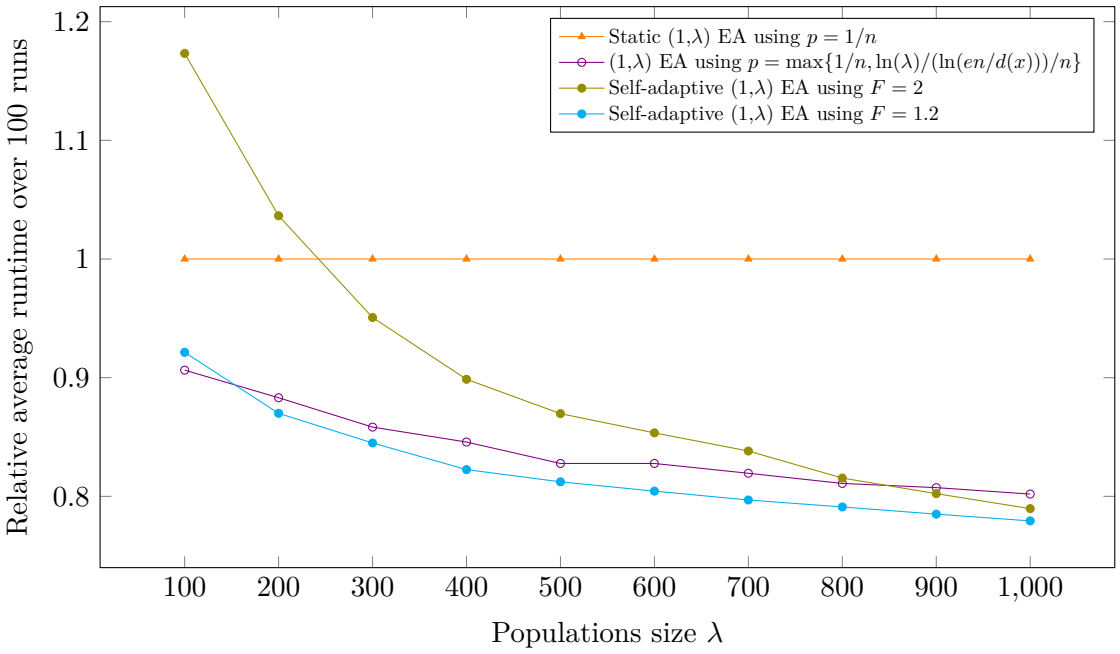

FIGURE 5.3: Average runtime of three dynamic (1,λ) EAs relative to the average runtime of the static (1,λ) EA on ONEMAX ($n = 10^5$)

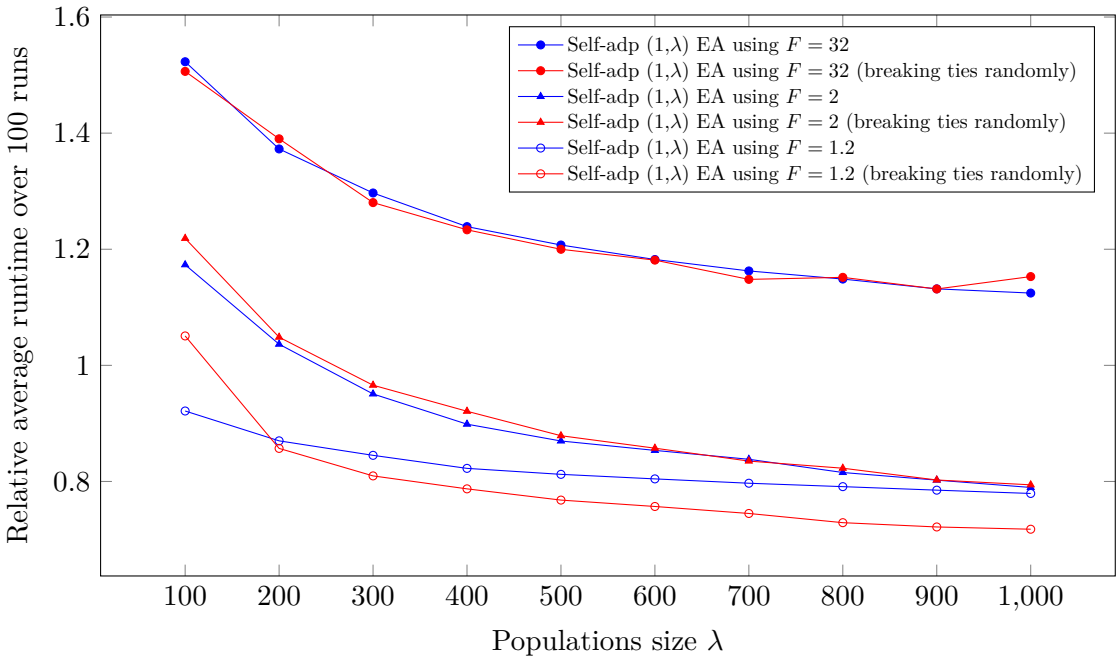

FIGURE 5.4: Relative average runtime of self-adapting (1,λ) EAs with different tie breaking rules on ONEMAX ($n = 10^5$)

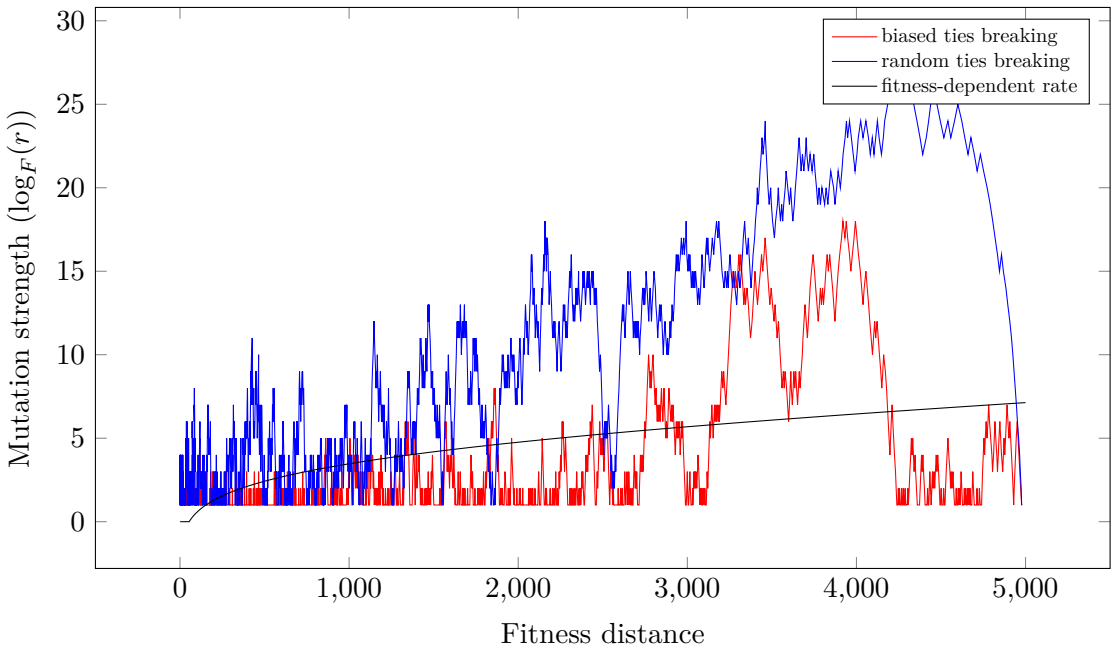

FIGURE 5.5: Mutation strengths used at a certain fitness distance level in two example runs of the self-adapting (1,λ) EA on ONEMAX ($n = 10000$, $\lambda = 500$, $F = 1.2$). For comparison, also the fitness-dependent rate proposed in [BLS14] is plotted. Recall that a mutation strength of r in the self-adapting runs means that in average half the offspring use the rate r/F and half use the rate rF.

Chapter 6

Conclusions

We numerically computed the precise unary unbiased black-box complexity on OneMax problem. We showed that drift-maximization is near-optimal for this problem, thus the precise complexity can be achieved (apart from an additive $o(n)$ term) by a simple hill climber with fitness dependent mutation rate.

After analyzing the suitable mutation strength, we proposed a self-adjusting choice of the mutation strength k for the hill climber. This use of k-bit flips instead of the usually preferred standard bit mutation with its random mutation strength allowed to much better exploit the most effective mutation strength. This self-adjusting choice allowed to find the optimal mutation strength automatically and on-the-fly. By this, also the risk of getting stuck in local optima, the known draw-back of k-bit flips, was overcome.

Based on the learning-inspired parameter control on randomized local search, we proposed and analyzed a new simple self-adjusting mutation scheme for the (1+λ) EA. It consists of creating half the offspring with a slightly larger and the rest with a slightly smaller mutation rate. Based on the success of the subpopulations, the mutation rate is adjusted. This simple scheme overcomes difficulties of previous self-adjusting choices, e.g., the careful choice of the exploration-exploitation balance and the forgetting rate in the learning scheme.

We proved rigorously that this self-adjusting (1+λ) EA optimizes the OneMax test function in an expected number of $O(n\lambda/\log\lambda + n\log n)$ fitness evaluations. This matches the runtime shown in [BLS14] for a careful fitness-dependent choice of the mutation rate, which was also shown to be asymptotically optimal among all λ-parallel black-box optimization algorithms. Hence our runtime result indicates that the self-adjusting mechanism developed in this work is able to find very good mutation rates. To the best of our knowledge, this is the first time that a self-adjusting choice of the mutation rate speeds up a mutation-based algorithm on the OneMax test function by more than a constant factor.

After that, we analyzed the self-adaptive (1,λ) EA using a very simple scheme for mutating the mutation rate and proved that is achieves the expected runtime $O(n\lambda/\log\lambda + n\log n)$ on OneMax, which is optimal for all λ-parallel mutation-based unbiased black-box algorithms. Hence, we have identified a simple and natural example where self-adaptation of strategy parameters in discrete EAs can lead to provably optimal runtimes that beat all known static parameter settings. Moreover, a relatively complicated and partly unintuitive self-adjusting scheme for the mutation rate proposed can be replaced by our simple endogenous scheme.

The analysis of the (1,λ) EA has revealed a highly non-trivial stochastic process where drift happens in two dimensions, namely regarding fitness distance and mutation rate. We have advanced the techniques for the analysis of such two-dimensional drift processes, including a useful lemma about occupation probabilities. Altogether,

we are optimistic that our research helps pave the ground for further analyses of self-adaptive EAs.

Bibliography

[Azu67] Kazuoki Azuma. "Weighted sums of certain dependent variables". In: *Tohoku Mathematical Journal* 19 (1967), pp. 357–367.

[Bäc92] Thomas Bäck. "Self-adaptation in genetic algorithms". In: *Proc. of ECAL '92*. MIT Press, 1992, pp. 263–271.

[BLS14] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. "Unbiased black-box complexity of parallel search". In: *Proc. of PPSN '14*. Springer, 2014, pp. 892–901.

[BDN10] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. "Optimal fixed and adaptive mutation rates for the LeadingOnes problem". In: *Proc. of PPSN '10*. Springer, 2010, pp. 1–10.

[BD18] Nathan Buskulic and Carola Doerr. personal communication. 2018.

[BD17] Maxim Buzdalov and Benjamin Doerr. "Runtime Analysis of the $(1 + (\lambda, \lambda))$ Genetic Algorithm on Random Satisfiable 3-CNF Formulas". In: *Proc. of GECCO '17*. ACM, 2017, pp. 1343–1350.

[DL16] Duc-Cuong Dang and Per Kristian Lehre. "Self-adaptation of mutation rates in non-elitist populations". In: *Proc. of PPSN '16*. Springer, 2016, pp. 803–813.

[Dev72] Luc Devroye. "The compound random search". PhD thesis. Purdue Univ., West Lafayette, 1972.

[Doe11] Benjamin Doerr. "Analyzing randomized search heuristics: tools from probability theory". In: *Theory of Randomized Search Heuristics*. Ed. by Anne Auger and Benjamin Doerr. World Scientific Publishing, 2011, pp. 1–20.

[Doe18a] Benjamin Doerr. "Better Runtime Guarantees via Stochastic Domination". In: *Proc. of EvoCOP '18*. 2018, pp. 1–17.

[Doe18b] Benjamin Doerr. "Probabilistic Tools for the Analysis of Randomized Optimization Heuristics". In: *CoRR* abs/1801.06733 (2018). arXiv: 1801.06733. URL: http://arxiv.org/abs/1801.06733.

[DD15] Benjamin Doerr and Carola Doerr. "Optimal parameter choices through self-adjustment: applying the 1/5-th rule in discrete settings". In: *Proc. of GECCO '15*. ACM, 2015, pp. 1335–1342.

[DD16] Benjamin Doerr and Carola Doerr. "The Impact of Random Initialization on the Runtime of Randomized Search Heuristics". In: *Algorithmica* 75.3 (2016), pp. 529–553.

[DD18a] Benjamin Doerr and Carola Doerr. "Optimal Static and Self-Adjusting Parameter Choices for the $(1 + (\lambda, \lambda))$ Genetic Algorithm". In: *Algorithmica* 80 (2018), pp. 1658–1709.

[DD18b] Benjamin Doerr and Carola Doerr. “Theory of Parameter Control Mechanisms for Discrete Black-Box Optimization: Provable Performance Gains Through Dynamic Parameter Choices”. In: *Theory of Randomized Search Heuristics in Discrete Search Spaces*. Ed. by Benjamin Doerr and Frank Neumann. To appear. Springer, 2018.

[DDE15a] Benjamin Doerr, Carola Doerr, and Franziska Ebel. “From black-box complexity to designing new genetic algorithms”. In: *Theoretical Computer Science* 567 (2015), pp. 87–104.

[DDE15b] Benjamin Doerr, Carola Doerr, and Franziska Ebel. “From black-box complexity to designing new genetic algorithms”. In: *Theoretical Computer Science* 567 (2015), pp. 87 –104.

[DDK16] Benjamin Doerr, Carola Doerr, and Timo Kötzing. “Provably optimal self-adjusting step sizes for multi-valued decision variables”. In: *Proc. of PPSN ’16*. Springer, 2016, pp. 782–791.

[DDY16a] Benjamin Doerr, Carola Doerr, and Jing Yang. “k-Bit mutation with self-adjusting k outperforms standard bit mutation”. In: *Proc. of PPSN ’16*. Springer, 2016, pp. 824–834.

[DDY16b] Benjamin Doerr, Carola Doerr, and Jing Yang. “Optimal parameter choices via precise black-box analysis”. In: *Proc. of GECCO ’16*. ACM, 2016, pp. 1123–1130.

[DFW10] Benjamin Doerr, Mahmoud Fouz, and Carsten Witt. “Quasirandom evolutionary algorithms”. In: *Proc. of GECCO ’10*. ACM, 2010, pp. 1457–1464.

[DFW11] Benjamin Doerr, Mahmoud Fouz, and Carsten Witt. “Sharp bounds by probability-generating functions and variable drift”. In: *Proc. of GECCO ’11*. ACM, 2011, pp. 2083–2090.

[DJW12] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. “Multiplicative Drift Analysis”. In: *Algorithmica* 64 (2012), pp. 673–697.

[DK15] Benjamin Doerr and Marvin Künnemann. “Optimizing linear functions with the $(1+\lambda)$ evolutionary algorithm – different asymptotic runtimes for different instances”. In: *Theoretical Computer Science* 561 (2015), pp. 3–23.

[DWY18] Benjamin Doerr, Carsten Witt, and Jing Yang. “Runtime Analysis for Self-adaptive Mutation Rates”. In: *Proc. of GECCO ’18*. ACM, 2018, to appear.

[Doe+13a] Benjamin Doerr, Thomas Jansen, Carsten Witt, and Christine Zarges. “A method to derive fixed budget results from expected optimisation times”. In: *Proc. of GECCO ’13*. ACM, 2013, pp. 1581–1588.

[Doe+13b] Benjamin Doerr, Timo Kötzing, Johannes Lengler, and Carola Winzen. “Black-Box Complexities of Combinatorial Problems”. In: *Theoretical Computer Science* 471 (2013), pp. 84–106.

[Doe+13c] Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. “Mutation rate matters even when optimizing monotone functions”. In: *Evolutionary Computation* 21 (2013), pp. 1–21.

[Doe+17] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. "The (1+λ) Evolutionary Algorithm with Self-Adjusting Mutation Rate". In: *Proc. of PPSN '17*. Full version available at http://arxiv.org/abs/1704.02191. ACM, 2017, pp. 1351–1358.

[DW18] Carola Doerr and Markus Wagner. "On the Effectiveness of Simple Success-Based Parameter Selection Mechanisms for Two Classical Discrete Black-Box Optimization Benchmark Problems". In: *Proc. of GECCO '18*. To appear. Preliminary version available at https://arxiv.org/abs/1803.01425. ACM, 2018.

[DJW02] Stefan Droste, Thomas Jansen, and Ingo Wegener. "On the analysis of the (1+1) Evolutionary Algorithm". In: *Theoretical Computer Science* 276 (2002), pp. 51–81.

[DJW06] Stefan Droste, Thomas Jansen, and Ingo Wegener. "Upper and Lower Bounds for Randomized Search Heuristics in Black-box Optimization". In: *Theory of Computing Systems* 39 (2006), pp. 525–544.

[EHM99] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. "Parameter control in evolutionary algorithms". In: *IEEE Transactions on Evolutionary Computation* 3 (1999), pp. 124–141.

[Fia+08] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. "Extreme Value Based Adaptive Operator Selection". In: *Proc. PPSN '08*. Vol. 5199. Lecture Notes in Computer Science. Springer, 2008, pp. 175–184.

[Fia+09] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. "Dynamic Multi-Armed Bandits and Extreme Value-Based Rewards for Adaptive Operator Selection in Evolutionary Algorithms". In: *Proc. LION '09*. Vol. 5851. Lecture Notes in Computer Science. Springer, 2009, pp. 176–190.

[GKS99] Josselin Garnier, Leila Kallel, and Marc Schoenauer. "Rigorous Hitting Times for Binary Mutations". In: *Evolutionary Computation* 7 (1999), pp. 173–203.

[GW17] Christian Gießen and Carsten Witt. "The interplay of population size and mutation probability in the (1+λ) EA on OneMax". In: *Algorithmica* 78 (2017), 587–609.

[GW15] Christian Gießen and Carsten Witt. "Population Size vs. Mutation Strength for the (1+λ) EA on OneMax". In: *Proc. GECCO '15*. ACM, 2015, pp. 1439–1446.

[GW16] Christian Gießen and Carsten Witt. "Optimal Mutation Rates for the (1+λ) EA on OneMax". In: *Proc. of GECCO '16*. 2016, pp. 1147–1154.

[Haj82] Bruce Hajek. "Hitting-time and occupation-time bounds implied by drift analysis with applications". In: *Advances in Applied Probability* 13 (1982), pp. 502–525.

[HY04] Jun He and Xin Yao. "A study of drift analysis for estimating computation time of evolutionary algorithms". In: *Natural Computing* 3 (2004), pp. 21–35.

[Hwa+18] Hsien-Kuei Hwang, Alois Panholzer, Nicolas Rolin, Tsung-Hsi Tsai, and Wei-Mei Chen. "Probabilistic analysis of the (1+1)-evolutionary algorithm". In: *Evolutionary Computation* 26 (2018), pp. 299–345.

[Jäg11] Jens Jägersküpper. "Combining Markov-chain analysis and drift analysis – the (1+1) evolutionary algorithm on linear functions reloaded". In: *Algorithmica* 59 (2011), pp. 409–424.

[Jan13] Thomas Jansen. *Analyzing Evolutionary Algorithms - The Computer Science Perspective*. Natural Computing Series. Springer, 2013. ISBN: 978-3-642-17338-7.

[JW06] Thomas Jansen and Ingo Wegener. "On the analysis of a dynamic evolutionary algorithm". In: *Journal of Discrete Algorithms* 4 (2006), pp. 181–199.

[JZ14] Thomas Jansen and Christine Zarges. "Performance analysis of randomised search heuristics operating with a fixed budget". In: *Theoretical Computer Science* 545 (2014), pp. 39–58.

[Joh10] Daniel Johannsen. "Random combinatorial structures and randomized search heuristics". PhD thesis. Saarland University, 2010. URL: http://scidok.sulb.uni-saarland.de/volltexte/2011/3529/.

[KB80] Rob Kaas and Jan M. Buhrman. "Mean, Median and Mode in Binomial Distributions". In: *Statistica Neerlandica* 34 (1980), pp. 13–18.

[KLW15] Timo Kötzing, Andrei Lissovoi, and Carsten Witt. "(1+1) EA on generalized dynamic OneMax". In: *Proc. of FOGA '15*. ACM, 2015, pp. 40–51.

[LDD15] Axel de Perthuis de Laillevault, Benjamin Doerr, and Carola Doerr. "Money for Nothing: Speeding Up Evolutionary Algorithms Through Better Initialization". In: *Proc. of the GECCO '15*. ACM, 2015, pp. 815–822.

[LS11] Jörg Lässig and Dirk Sudholt. "Adaptive population models for offspring populations and parallel evolutionary algorithms". In: *Proc. of FOGA '11*. ACM, 2011, pp. 181–192.

[LW12] Per Kristian Lehre and Carsten Witt. "Black-Box Search by Unbiased Variation". In: *Algorithmica* 64 (2012), pp. 623–642.

[Len18] Johannes Lengler. "A General Dichotomy of Evolutionary Algorithms on Monotone Functions". In: *CoRR* abs/1803.09227 (2018). arXiv: 1803.09227. URL: http://arxiv.org/abs/1803.09227.

[MRC09] Boris Mitavskiy, Jonathan E. Rowe, and Chris Cannings. "Theoretical analysis of local search strategies to optimize network communication subject to preserving the total number of links". In: *Journal of Intelligent Computing and Cybernetics* 2 (2009), pp. 243–284.

[NW07] Frank Neumann and Ingo Wegener. "Randomized local search, evolutionary algorithms, and the minimum spanning tree problem". In: *Theoretical Computer Science* 378 (2007), pp. 32–40.

[PA02] Paul Fischer Peter Auer Nicoló Cesa-Bianchi. "Finite-time analysis of the multiarmed bandit problem". In: *Machine Learning* 47 (2002), 235–256.

[Rec73] Ingo Rechenberg. In: *Evolutionsstrategie* (1973).

[Rob55] Herbert Robbins. "A Remark on Stirling's Formula". In: *The American Mathematical Monthly* 62 (1955), pp. 26–29.

[RS14] Jonathan E. Rowe and Dirk Sudholt. "The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm". In: *Theoretical Computer Science* 545 (2014), pp. 20–38.

[SS68] Michael A. Schumer and Kenneth Steiglitz. “Adaptive step size random search”. In: vol. 13. IEEE Transactions on Automatic Control, 1968, 270–276.

[Sud13] Dirk Sudholt. “A New Method for Lower Bounds on the Running Time of Evolutionary Algorithms”. In: *IEEE Transactions on Evolutionary Computation* 17 (2013), pp. 418–435.

[Wit13] Carsten Witt. “Tight Bounds on the Optimization Time of a Randomized Search Heuristic on Linear Functions”. In: *Combinatorics, Probability & Computing* 22 (2013), pp. 294–318.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Conception de meilleurs algorithmes évolutionnaires grâce à la théorie de la complexité boîte noire

Mots clés : randomized search heuristics, temps d'exécution, algorithmes d'évolution

Résumé : De nombreux problèmes d'optimisation du monde réel sont trop complexes pour être résolus en temps polynomial. Un moyen de traiter de tels problèmes utilise « randomized search heuristics » (RSHs), qui produisent des solutions plus efficacement en compromettant l'optimalité, l'exhaustivité, l'exactitude ou la précision. Les RSHs initialisent d'abord un point de recherche avec une position aléatoire dans l'espace de recherche, puis résolvent les problèmes de manière itérative jusqu'à ce qu'un critère de terminaison soit rempli. À chaque itération, ils répètent les étapes suivantes: générer un ou plusieurs candidats; évaluer la qualité des nouveaux individus; mettre à jour les informations connues. Puisque les RSHs ne nécessitent aucune information que les valeurs objectives des points de recherche évalués, nous les appelons algorithmes d'optimisation de boîte noire.
Les RSHs sont des algorithmes paramétrés et leurs performances typiquement dépendent de manière cruciale de la valeur de ces paramètres. Dans ce travail, nous contribuons à notre connaissance de la manière de contrôler les paramètres en ligne de diverses façons. Nous étudions la fonction OneMax qui est définie via $\{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^n x_i$. Dans un premier temps, nous prouvons que tout algorithme unaire non-biaisée de type boîte noire nécessite $n\ln(n) - cn \pm o(n)$ itérations, en moyenne, pour trouver la solution optimale à ce problème, où c est une constante entre $0,2539$ et $0,2665$. Ce temps d'exécution peut être obtenu avec un (1+1)-type algorithme simple en utilisant un paramètre fitness-dépendant. Nous montrons également que ce paramètre peut être remplacé par un paramètre d'auto-ajustement sans perte d'efficacité.
Nous étendons ensuite notre stratégie d'auto-ajustement aux algorithmes d'évolution (EAs) basés sur la population, qui utilisent des mécanismes inspirés par l'évolution biologique, tels que la reproduction, la mutation, la recombinaison et la sélection. Grosso modo, l'idée principale des EAs est de créer par itération λ points de recherche en utilisant deux valeurs de paramètre différentes. Le paramètre est ensuite mis à jour en fonction de celui utilisé dans la sous-population qui contient le meilleur point de recherche. Nous proposons une version d'auto-ajustement de $(1+\lambda)$ EA et une version auto-adaptative de $(1,\lambda)$ et nous prouvons qu'ils trouvent tous les deux l'optimum dans un nombre espéré de $O(n\lambda/\log\lambda + n\log n)$ itérations.

Title : From a Complexity Theory of Evolutionary Computation to Superior Randomized Search Heuristics

Keywords : randomized search heuristics, runtime, evolutionary algorithms

Abstract : Many real-world optimization problems are too complex to be solved in polynomial time. One way to deal with such problems is using randomized search heuristics (RSHs), which produce solutions more efficiently by compromising on optimality, completeness, accuracy, or precision. RSHs first initialize a search point with a random position and then solve problems iteratively until a termination criterion is met. In each iteration, they repeat the following steps: generate one or more candidates; evaluate the quality of the new individuals; update the known information. Since RSHs do not require any other information than the objective values of the evaluated search points, we call them black-box optimization algorithms.
RSHs are parametrized algorithms, and their performance typically depends very crucially on the value of these parameters. In this work, we contribute to our knowledge of how to control the parameters online in various ways. We study the OneMax function which is defined via $\{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^n x_i$. In a first step, we prove that unary unbiased black-box algorithm needs $n\ln(n) - cn \pm o(n)$ iterations, on average, to find an optimal solution for this problem, where c is a constant between 0.2539 and 0.2665. This runtime can be achieved with a simple (1+1)-type algorithm using a fitness-dependent parameter setting. We also show that this setting can be replaced by a self-adjusting one without losing efficiency.
We then extend our self-adjusting strategy to population-based evolutionary algorithms (EAs) which use mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Roughly speaking, the main idea of the EAs is to create per iteration λ search points using two different parameter values. The parameter is then updated based on the one used in that subpopulation which contains the best search point. We propose a self-adjusting version of $(1+\lambda)$ and a self-adaptive version of the $(1,\lambda)$ and prove that they both find the optimum in an expected number of best possible $O(n\lambda/\log\lambda + n\log n)$ iterations.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France