The (1+λ) Evolutionary Algorithm with Self-Adjusting Mutation Rate

Benjamin Doerr
Laboratoire d'Informatique (LIX)
École Polytechnique
Palaiseau, France

Christian Gießen
DTU Compute
Technical University of Denmark
Kgs. Lyngby, Denmark

Carsten Witt
DTU Compute
Technical University of Denmark
Kgs. Lyngby, Denmark

Jing Yang
Laboratoire d'Informatique (LIX)
École Polytechnique
Palaiseau, France

ABSTRACT

We propose a new way to self-adjust the mutation rate in population-based evolutionary algorithms. Roughly speaking, 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 to the rate used in that subpopulation which contains the best offspring.

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 shows that this performance is best-possible among all λ-parallel mutation-based unbiased black-box algorithms.

This result shows that the new way of adjusting the mutation rate can find optimal dynamic parameter values on the fly. Since our adjustment mechanism is simpler than the ones previously used for adjusting the mutation rate and does not have parameters itself, we are optimistic that it will find other applications.

CCS CONCEPTS

•Theory of computation → Optimization with randomized search heuristics;

KEYWORDS

Runtime Analysis; Mutation; Dynamic Parameter Control

ACM Reference format:
Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. 2017. The (1+λ) Evolutionary Algorithm with Self-Adjusting Mutation Rate. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071279

GECCO '17, Berlin, Germany

978-1-4503-4920-8/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3071178.3071279

1 INTRODUCTION

Evolutionary algorithms (EAs) have shown a remarkable performance in a broad range of applications. However, it has often been observed that this performance depends crucially on the use of the right parameter settings. Parameter optimization and parameter control are therefore key topics in EA research. Since these have very different characteristics in discrete and continuous search spaces, we discuss in this work only evolutionary algorithms for discrete search spaces.

Theoretical research has contributed to our understanding of these algorithms with mathematically founded runtime analyses, many of which show how the runtime of an EA is determined by its parameters. 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. This problem has been solved by theoretical means for a small number of easy benchmark problems, but it is highly unclear how to find such functional relations in the general case.

A more designer-friendly way to work with dynamic parameters is to modify the parameters based on simple rules taking into account the recent performance. A number of recent results shows 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. For example, good results have been obtained by increasing or decreasing a parameter depending on whether the current iteration improved the best-so-far solution or not, e.g., in a way resembling the 1/5-th rule from continuous optimization.

Such success-based self-adjusting parameter settings can work well when there is a simple monotonic relation between success and parameter value, e.g., when one speculates that increasing the size of the population in an EA helps when no progress was made.

For parameters like the mutation rate, it is not clear what a success-based rule can look like, since a low success rate can either stem from a too small mutation rate (regenerating the parent with high probability) or a destructive too high mutation rate. In [10], a relatively complicated learning mechanism was presented that tries to learn the right mutation strength by computing a time-discounted average of the past performance stemming from different parameter values. This learning mechanism needed a careful trade-off between exploiting the currently most profitably mutation strength and experimenting with other parameter values and a careful choice of the parameter controlling by how much older experience is taken less into account than more recent observations.

1.1 A New Self-Adjusting Mechanism for Population-Based EAs

In this work, we propose an alternative way to adjust the mutation rate on the fly for algorithms using larger offspring populations. It aims at overcoming some of the difficulties of the learning mechanism just described. 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 rate to leave the interval $[2/n, 1/4]$, so that the rates used in the subpopulations are always in the interval $[1/n, 1/2]$.

In this first work proposing this mechanism, we shall not try to optimize it, but rather show in a proof-of-concept manner that it can find the optimal mutation rate. 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.

While we shall not discuss such fine-tunings, we do want to 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 the basin of attraction of a locally optimal mutation rate. Observe that with probability $\Theta(1/n^2)$, a sequence of $\log_2 n$ random modification 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).

1.2 Runtime Analysis for the Self-Adjusting (1+λ) EA on OneMax

To prove that the self-adjusting mechanism just presented can indeed find good dynamic mutation rates, we analyse it in the purest possible setting, namely in the optimization of the classic OneMax test function via the (1+λ) EA (see Algorithm 1 below for the pseudocode of this algorithm).

The runtime of the (1+λ) EA with fixed mutation rates on OneMax is well-understood [13, 14]. In particular, Gießen and Witt [14] 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 [1] 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 $\Theta(\log \log \lambda)$. 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 find on the fly a mutation rate that is sufficiently close to the one proposed in [1] to achieve asymptotically the same expected runtime.

Theorem 1.1. *Let $\lambda = \omega(1)$ and $\lambda = n^{O(1)}$. Let T denote the number of generations of the (1+λ) EA with self-adjusting mutation rate on OneMax. Then,*

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

This corresponds to an expected number of functions 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 [14] does not yield the asymptotically optimal number of generations unless λ is so small that the $n \log n$-term dominates.

Lemma 1.2. *Let $\lambda = \omega(1)$ 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 functions evaluations of $O(\frac{\lambda n}{\log \lambda} + \sqrt{\lambda} n \log n)$.

The paper is structured as follows: In Section 2 we give a short overview over previous analyses of the (1+λ) EA and of self-adjusting parameter control mechanism in EAs from a theoretical perspective. In Section 3 we give the algorithm and the mutation scheme.

For convenience, we also state some key theorems that we will frequently use in the rest of the paper. The next three sections deal with the runtime analysis of 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 1.2 is then given in Section 7. Finally, we conclude in Section 8. Due to space restrictions, some proofs had to be omitted from this paper. They can be found in the preprint [11].

2 RELATED WORK

Since this is a theoretically oriented work on how a dynamic parameter choice speeds up the runtime of the (1+λ) EA on the test function OneMax, let us briefly review what is known about the theory of this EA and dynamic parameter choices in general.

The first to conduct a rigorous runtime analysis of the (1+λ) EA were Jansen, De Jong, and Wegener [15]. They proved, among other results, that when optimizing OneMax a linear speed-up exists up to a population size of $\Theta(\log(n)\log\log(n)/\log\log\log(n))$, that is, for $\lambda = O(\log(n)\log\log(n)/\log\log\log(n))$, finding the optimal solution takes an expected number of $\Theta(n\log(n)/\lambda)$ generations, whereas for larger λ at least $\omega(n\log(n)/\lambda)$ generations are necessary. This picture was completed in [13] with a proof that the expected number of generations taken to find the optimum is $\Theta(\frac{n\log n}{\lambda} + \frac{n\log\log\lambda}{\log\lambda})$. The implicit constants were determined in [14], giving the bound of $(1 \pm o(1))(\frac{1}{2}\frac{n\ln\ln\lambda}{\ln\lambda} + \frac{e^r}{r}\frac{n\ln n}{\lambda})$, for any constant r, as mentioned in the introduction.

Aside from the optimization behavior on OneMax, not too much is known for the (1+λ) EA, or is at least not made explicit (it is easy to see that waiting times for an improvement which are larger than λ reduce by a factor of $\Theta(\lambda)$ compared to one-individual offspring populations). Results made explicit are the $\Theta(n^2/\log(n) + n)$ expected runtime (number of generations) on LeadingOnes [15], the worst-case $\Theta(n + n\log(n)/\lambda)$ expected runtime on linear functions [13], and the $O(m^2(\log n + \log w_{\max})/\lambda)$ runtime estimate for minimum spanning trees valid for $\lambda \le m^2/n$ [21].

While it is clear the 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 advantages of dynamic choices could be demonstrated by mathematical means (for discrete optimization problems; in continuous optimization, step size adaptions are obviously necessary to approach arbitrarily closely a target point).

In the first work in this direction, Böttcher, Doerr, and Neumann [2] showed that when using a *fitness-dependent* mutation rate of $1/(\text{LeadingOnes}(x) + 1)$, then the runtime of the (1+1) EA improves to roughly $0.68n^2$ compared to a time of $0.86n^2$ stemming from the mutation rate $1/n$ or a runtime of $0.77n^2$ stemming from the asymptotically optimal rate of $1.59/n$.

For the $(1 + (\lambda, \lambda))$ GA, a fitness-dependent offspring population size of order $\lambda = \Theta(\sqrt{n/d(x)})$ was suggested in [7], 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 [5] to $O(n)$. Since in this self-adjusting 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. [1], a fitness-dependent mutation rate of $\max\{\frac{\ln\lambda}{n\ln(en/d(x))}, \frac{1}{n}\}$ was shown to improve the classic runtime of $O(\frac{n\log\log\lambda}{\log\lambda} + \frac{n\log n}{\lambda})$ to $O(\frac{n}{\log\lambda} + \frac{n\log n}{\lambda})$. In [10], the (1+1) EA using a k-bit flip mutation operator together with a fitness-dependent choice of k was shown to give a performance on OneMax that is very close to the theoretical optimum (among all unary unbiased black-box algorithms), however, this differs only by lower order terms from the performance of the simple randomized local search heuristic (RLS).

While all these results show an advantage of a dynamic parameter setting, it remains questionable if an algorithm user would be able to find such a functional dependence of the parameter on the fitness. This difficulty can be overcome via *self-adjusting* parameter choices, where the parameter is modified according to a simple rule often based on the success or progress of previous iterations, or via *self-adaptation*, 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 [3], however, is promising and shows examples where self-adaptation can lead to significant speed-ups for non-elitist evolutionary algorithms.

In contrast to this, the last years have produced a profound understanding of self-adjusting parameter choices. The first to perform a mathematical analysis were Lässig and Sudholt [18], 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 [6] it was shown that the fitness-dependent choice of λ for the $(1 + (\lambda, \lambda))$ GA described above 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. 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 [8].

These results indicate that success-based dynamics work well for adjusting parameters when a monotonic relation like "if progress is difficult, then increase the population size" holds. For adjusting a parameter like the mutation rate, it is less obvious how to do this. Both a too large mutation rate (creating a stronger drift towards a Hamming distance of $n/2$ from the optimum) and a too small mutation rate (giving a too small radius of exploration) can be detrimental. For this reason, to obtain a self-adjusting version of their result on the optimal number k to optimize OneMax via k-bit flips [10], in [9] a learning mechanism was proposed that from the medium-term past estimates the efficiency of different parameter values. As shown there, this does find the optimal mutation strength

sufficiently well to obtain essentially the runtime stemming from the fitness-dependent mutation strength exhibited before.

In the light of these works, our result from the methodological perspective shows that some of the difficulties of the learning mechanism of [9], 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 the mechanism proposed in this work. 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 [9] to intentionally use parameter settings which appear to be slightly off the current optimum to gain additional insight.

3 PRELIMINARIES

3.1 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 1.

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 by either replacing it with the mutation rate that the best individual (i. e. the one with the lowest fitness) was created with or by random decision using a fair coin, i. e. the adjustment of the rate is to some degree randomized. The two adjustment variations have the same probability. In particular, adjusting the mutation rate occurs in every generation, independent of the selection of the next parent individual. Note that the algorithm starts with an initial mutation rate r^{init}. The only assumption on r^{init} is to be greater or equal than 1. Furthermore, the mutation rate is capped below 2 and above $n/4$ because in the case of $r = 2$, a subpopulation with rate 1 is generated. Rates less than 1 flip less than one-bit in expectation. At rate $n/4$, a subpopulation with rate $n/2$ is created, i. e. each offspring in that subpopulation is sampled uniformly at random. The (1+λ) EA with our mutation scheme is given as pseudocode in Algorithm 1.

Algorithm 1 (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 $\min\{\max\{2, r_t\}, n/4\}$.

Designing such a mutation scheme is not straightforward. Instead of standard bit mutation sampling for example from a hypergeometric distribution to create offspring seems reasonable as well. But this can be problematic, for instance if only one one-bit is left and the mutation rate is even, no progress could be made with a hypergeometric approach; standard bit mutation guarantees a certain variance in the number of flipped bits. Furthermore, besides the fitness-guided adjustments we use a random adjustment every other generation in expectation. We consider such a behavior natural for an EA in the sense that this behavior can prevent the algorithm from being stuck in local optima with respect to this operation. Lastly, our choice of doubling and halving the rate is arbitrary, we chose a multiplicative strategy in order to keep the subpopulations apart and make the analysis simpler.

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 $\textsc{OneMax}(x_1, \ldots, x_n) := x_1 + \cdots + x_n$ 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 paper, all asymptotic notation will be with respect to the problem size n. Furthermore, we consider λ to be superconstant, i. e. $\lambda = \omega(1)$ throughout the paper. This assumption is mostly for convenience in the proofs, and we think that our result would also hold for $\lambda = O(1)$.

3.2 Drift Theorems

Our results are obtained by drift analysis, which is also used in previous analyses of the (1+λ) EA without self-adaptation on OneMax and other linear functions [13, 14].

The first theorems stating upper bounds on the hitting time using variable drift go back to [16, 20]. We take a formulation from [19] but simplify it to Markov processes for notational convenience.

Theorem 3.1 (Variable Drift, Upper Bound). *Let $(X_t)_{t\geq 0}$, be random variables describing a Markov process over a finite state space $S \subseteq \{0\} \cup [x_{\min}, x_{\max}]$, where $x_{\min} > 0$. Let $h(x) : [x_{\min}, x_{\max}] \to \mathbb{R}^+$ be a monotone increasing function such that $1/h(x)$ is integrable on $[x_{\min}, x_{\max}]$ and $E(X_t - X_{t+1} \mid X_t) \geq h(X_t)$ if $X_t \geq x_{\min}$. Then it holds for the first hitting time $T := \min\{t \mid X_t = 0\}$ that*

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

The variable drift theorem is often applied in the special case of *additive drift* in discrete spaces: assuming $E(X_t - X_{t+1} \mid X_t; X_t > 0) \geq \epsilon$ for some constant ϵ, one obtains $E(T \mid X_0) \leq X_0/\epsilon$.

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 [12]. Again, this is implied by the previous variable drift theorem.

Theorem 3.2 (Multiplicative Drift [12]). *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 $P(X_t = x) > 0$ we have*

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

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

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

3.3 Chernoff Bounds

For reasons of self-containedness and as a courtesy to the reader, we state two well-known multiplicative Chernoff bounds and a lesser known additive Chernoff bound that is also known in the literature as Bennett's inequality.

Theorem 3.3 (Bennett's inequality, Chernoff Bounds [4, Theorem 1.12, Theorem 1.10]). *Let $X_1, \dots, X_n$ be independent random variables and let $X = \sum_{i=1}^n X_i$. Furthermore, let b such that $X_i \leq E(X_i) + b$ for all $i = 1, \dots, n$ and $\sigma^2 = \frac{1}{n}\sum_{i=1}^n \mathrm{Var}(X_i)$. Then, for all $\gamma > 0$*

$$\Pr(X \geq E(X) + \gamma) \leq \left(-\frac{\gamma}{b}\left(\left(1 + \frac{n\sigma^2}{b\gamma}\right)\ln\left(1 + \frac{b\gamma}{n\sigma^2}\right) - 1\right)\right) \ .$$

Moreover, if the X_i for all $i = 1, \dots, n$ take values in $[0, 1]$ then for $\delta \in [0, 1]$:

- $\Pr(X \leq (1-\delta)E(X)) \leq \exp(-\delta^2 E(X)/2)$,
- $\Pr(X \geq (1+\delta)E(X)) \leq \exp(-\delta^2 E(X)/3)$.

3.4 Occupation Probabilities

As mentioned above, we will be analyzing two depending stochastic processes: the random decrease of fitness 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 fitness 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 apply the following theorem from [17]. Note that in the paper a slightly more general version including a self-loop probability is stated, which we do not need here.

Theorem 3.4 (Theorem 7 in [17]). *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 $b \in \mathbb{R}_0^+$,*

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

We can readily apply this theorem in the following lemma that will be used throughout the paper to bound the rate r_t.

Lemma 3.5. *If there is a point $c \geq 4$ such that $\Pr(r_{t+1} < r_t \mid r_t > c) \geq 1/2 + \epsilon$ for some constant $\epsilon > 0$, then for all $t' \geq \min\{t \mid r_t \leq c\}$ and all $b \geq 4$ it holds $\Pr(r_{t'} \geq c + 2^b) \leq 2e^{-2b\epsilon/3}$.*

4 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. It is n for $k \geq n/2 + \omega(\sqrt{n\log\lambda})$, $n/2$ for $k = n/2 \pm o(\sqrt{n\log\lambda})$, and then quickly drops to $r = \Theta(\log n)$ 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 4.1. *Let $c_1(k) = (2\ln(en/k))^{-1}$ and $c_2(k) = 4n^2/(n-2k)^2$ with $0 < k < n/2$ and n large enough.*

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

Proof. Let $Q(k, i, r)$ be the probability of decreasing the number of ones by at least i using standard bit mutation with probability $p = r/n$. Assume that we flip x ones to zeros among k ones and y zeros to ones among $n - k$ zeros, respectively. We obtain

$$Q(k, i, r) = \sum_{x-y\geq i}^{k,k} \binom{k}{x}\binom{n-k}{y} p^{x+y}(1-p)^{n-x-y}.$$

By comparing each component in $Q(k, i, r/2)$ and $Q(k, i, 2r)$ we obtain

$$Q(k, i, 2r)/Q(k, i, r/2) \geq 4^i \frac{(1-2r/n)^n}{(1-r/(2n))^n} \geq \big(1 - o(1)\big)4^i e^{-1.5r}.$$

Here we notice that $\ln(1-x) \geq -x - x^2$ for all $0 \leq x \leq 1/2$. Then $(1-2r/n)^n/(1-r/(2n))^n = (1-1.5r/(n-r/2))^n \geq \exp(-1.5rn/(n-r/2) - (1.5r)^2 n/(n-r/2)^2) \geq (1-o(1))\exp(-1.5r)$. Let ϵ denote $(3/4)^{c_1(k)\ln\lambda}$ and let i^* be the largest i such that $Q(k, i, 2r) \geq 2\ln(1/\epsilon)/\lambda$. We will then have $i^* \geq 1.5r$ because

$$Q(k, 1.5r, 2r) \geq \binom{k}{1.5r}(2p)^{1.5r}(1-2p)^n \geq \left(\frac{k}{1.5r}\cdot\frac{2r}{n}\cdot\frac{1}{e}\right)^{2r}$$
$$\geq \left(\frac{4k}{3en}\right)^{2c_1(k)\ln\lambda} = \left(\frac{4}{3}\right)^{2c_1(k)\ln\lambda}\frac{1}{\lambda} \geq \frac{1/\epsilon^2}{\lambda} \geq \frac{2\ln(1/\epsilon)}{\lambda}.$$

The second inequality which involves $(1-2p)^n$ uses the fact that for $p = o(1/\sqrt{n})$ we have $(1-2p)^n \geq \exp(-2pn - 4p^2 n) \geq (1 - o(1))\exp(-2r)$. The $(1-o(1))$ factor is compensated by decreasing $(2p)^{1.5r}$ to $(2p)^{2r}$. Note that $i^* = \Theta(\log(\lambda)/\log\log\lambda)$ when $r = O(1)$. Since $Q(k, i, r)$ is increasing in r, we obtain $i^* \geq \Theta(\log(\lambda)/\log\log\lambda)$ for all r, which results in $4^{i^*}e^{-1.5r} = \exp(i^*\ln 4 - 1.5r) \geq \exp(i^*/4) = \Theta(\lambda^{1/\log\log\lambda})$. We now look for an upper bound on $Q(k, i^*, 2r)$. Let $q(k, i, r) = Q(k, i, r) - Q(k, i+1, r)$ be the probability of decreasing the number of ones by exactly i. Consider the terms in $q(k, i, r)$ where $x - y = i$. If x and y both increase by 1, the terms change by a factor of $k/x \cdot (n-k)/y \cdot p^2 \leq r^2/(xy)$. If we consider $y > 2i$

and $i > r$ then $xy > 6r^2$ and the factor $r^2/(xy)$ is less than $1/6$. The sum of these factors for all $y > 2i$ is less than the geometric series with ratio $1/6$. Therefore, when $i > r$ the sum from $0 \le y \le 2i$ contributes to at least $4/5$ of the total sum $q(k, i, r)$. Consequently, if we look at the first $2i^*$ terms in $q(k, i^*, 2r)$ and $q(k, i^* + 1, 2r)$ we have

$$\frac{q(k, i^* + 1, 2r)}{q(k, i^*, 2r)} = \Theta\left(\frac{kp}{i^*}\right) \ge \Theta\left(\frac{\log\log\lambda}{\log^2\lambda}\right) \ge \frac{1}{\ln^2\lambda}.$$

Since $q(k, i^* + 1, 2r) \le Q(k, i^* + 1, 2r)$ we obtain

$$\frac{Q(k, i^* + 1, 2r)}{q(k, i^*, 2r)} \ge \frac{1}{\ln^2\lambda} \text{ and } \frac{Q(k, i^* + 1, 2r)}{Q(k, i^*, 2r)} \ge \frac{1}{1 + \ln^2\lambda}.$$

So finally $Q(k, i^*, 2r) \le 2\ln(1/\epsilon)(1 + \ln^2\lambda)/\lambda$ and

$$Q(k, i^*, r/2) \le \frac{Q(k, i^*, 2r)}{4^{i^*} e^{-1.5r}} \le \Theta\left(\frac{1 + \log^2\lambda}{\lambda \cdot \lambda^{1/\log\log\lambda}}\right) = o(1/\lambda).$$

This means with probability $1-o(1)$ no offspring of rate $r/2$ manages to obtain a progress of i^* or more. However, for rate $2r$, with probability $1 - (1 - 2\ln(1/\epsilon)/\lambda)^{\lambda/2} \ge 1 - \epsilon$, at least one of the $\lambda/2$ offspring of rate $2r$ manages to obtain a progress of i^* or more. This proves the first statement of the lemma.

For the second statement, let $X(k, r)$ denote the random decrease in the number of ones after applying standard bit mutation with probability $p = r/n$ to an individual with k ones. Then $E(X) = kp - (n-k)p = (2k-n)p$. According to Bennett's inequality (Theorem 3.3), for any $\Delta > 0$ we have

$$\Pr(X \ge E(X) + \Delta) \le \exp\left(-\mathrm{Var}(X) \cdot h\left(\frac{\Delta}{\mathrm{Var}(X)}\right)\right)$$

where $h(u) = (1 + u)\ln(1 + u) - u$. We compare $h(u)$ with τu^2 for any constant factor τ. Let $g(u) = h(u) - \tau u^2$ be the difference, then $g(0) = g'(0) = 0$ while $g''(u) = 1/(1 + u) - 2\tau \ge 0$ when $1/(1 + u) \ge 2\tau$. This means $h(u) \ge u^2/(2u + 2)$. We now apply this bound with $X = X(k, 2r)$ and $\Delta = E(X(k, r/2)) - E(X(k, 2r)) = (n - 2k)1.5r/n > 0$. For this rate of $2r$ we have $\mathrm{Var}(X(k, 2r)) = n(2p)(1 - 2p) = 2r(1 - 2r/n)$ and $u = 3/4 \cdot (n - 2k)/(n - 2r)$. Then,

$$\Pr(X(k, 2r) \ge E(X(k, r/2))) \le \exp\left(-\frac{\Delta^2}{(2 + 2u)\mathrm{Var}(X(k, 2r))}\right)$$

$$\le \exp\left(-\frac{9(n - 2k)^2 r}{4n(7n - 8r - 6k)}\right) \le \exp\left(-\frac{9(n - 2k)^2 r}{28n^2}\right) < \frac{1}{\lambda^{9/7}}.$$

We notice that we have $7n-8r-6k > 7n-4n-3n = 0$ in the second inequality. Therefore, with probability $o(1)$ the best offspring of rate $2r$ is better than the expectation of rate $r/2$. For rate $r/2$, let X^+ and X^- be the number of one-bits flipped and zero-bits flipped, respectively. Both X^+ and X^- follow a binomial distribution and $X = X^+ - X^-$. We know $E(X^+) = kr/(2n)$ and $E(X^-) = (n-k)r/(2n) \ge \ln\lambda$. Then $\Pr(X^- \le E(X^-)-1) = \Theta(1)$ and $\Pr(X^+ \ge E(X^+) - 1) = \Theta(1)$. Therefore $\Pr(X(k, r/2) > E(X(k, r/2))) \ge \Theta(1)$. Hence, with probability $1 - (1 - \Theta(1))^{\lambda/2} = 1 - o(1)$, at least one offspring beats its expectation. This proves the second statement of the lemma.

The third statement follows from a straightforward application of Bennett's inequality and a union bound and is omitted. □

The lemma above shows that the rate r is attracted to the interval $[c_1(k)\ln n, c_2(k)\ln n]$. Unfortunately, we cannot show that we obtain a sufficient progress in the fitness for exactly this range of r-values. 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 4.3 (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 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, see Theorem 4.4 and its proof.

Let $\Delta := \Delta(\lambda/2, k, r)$ denote the fitness gain after selection among the best of $\lambda/2$ offspring generated with rate r from a parent with fitness distance $k := \textsc{OneMax}(x)$. Let $x^{(i)}, i \in \{1, \dots, \lambda/2\}$, be independent offspring generated from x by flipping each bit independently with probability r/n. Then the random variable Δ is defined by $\Delta := \max\{0, k - \min\{\textsc{OneMax}(x^{(i)}) \mid i \in \{1, \dots, \lambda/2\}\}\}$.

Lemma 4.2. *Let $2n/5 \le k < n/2$ and n be large enough. Let c be such that $c \le \min\{n^2/(50(n - 2k)^2), n/(4\ln\lambda)\}$ and $c \ge 1/2$. Let $r = c\ln\lambda$, then $E(\Delta) \ge 0.03\ln\lambda$.*

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

Lemma 4.3. *Let $n/\ln\lambda \le k < n/2$ and n be large enough. If $r \in [c_1(k)\ln\lambda, c_2(k)\ln(\lambda)/200]$ for $k \ge 2n/5$ and $r \in [c_1(k)\ln\lambda, \ln(\lambda)/2]$ for $k < 2n/5$ with $c_1(k), c_2(k)$ defined as in Lemma 4.1, then $E(\Delta) \ge 0.05\ln(\lambda)/\ln(en/k)$.*

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

Theorem 4.4. *The (1+λ) EA needs $O(n/\log\lambda)$ generations in expectation to reach a OneMax-value of $k \le n/\ln\lambda$ after initialization.*

Proof. We first argue quickly that it takes an expected number of at most $O(\sqrt{n})$ iterations to reach a fitness distance of $k < n/2$. To this end, we note that if $k \ge n/2$, then the probability for k to decrease by at least one is at least $\Theta(1)$ for any $1 < r < n/4$. Then, with probability at least $1 - o(1)$ the fitness k decreases by selecting the best of λ offspring. The initial k deviates from $n/2$ by $\Omega(\sqrt{n})$ in expectation. Hence, it takes $O(\sqrt{n})$ generations to obtain a fitness of $k < n/2$.

Without loss of generality we assume $k < n/2$ for the initial state. Our intuition is that once we begin to use rate r bounded by $c_1(k)\ln\lambda$ and $c_2(k)\ln\lambda$ 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 not far away from the bounds. After we make progress and k decreases to a new level, the corresponding c_1 and c_2 decrease, and the algorithm takes some time to readjust r into new bounds.

We consider the stochastic process $X_t = \log_2(r_t)$ and the current OneMax-value Y_t. According to Lemma 4.1 we have $E(X_t -$

$X_{t+1} \mid X_t; X_t > \log_2(c_2(Y_t)\ln\lambda)) \geq \epsilon = \Omega(1)$ and $E(X_{t+1} - X_t \mid X_t; X_t < \log_2(c_1(Y_t)\ln\lambda)) \geq \epsilon = \Omega(1)$. We pessimistically assume that all the iterations adjusting r_t make no progress. Let $k_0 > k_1 > \cdots > k_N$ be the OneMax-values taken by Y_t until k_N hits $n/\ln\lambda$. According to the additive drift theorem it takes at most $O(\log n)$ iterations to bound $r_t/\ln\lambda$ by $c_1(k_0)$ and $c_2(k_0)$, no matter how we set the initial rate. Consider t_i to be the last time that $Y_t = k_i$. This means r_t makes a progress of $k_i - k_{i+1} > 0$. Referring to Lemma 4.1 we know that $r_t < 2c_2(k_i)\ln\lambda$ with probability $1 - o(1)$. Hence, $r_{t+1} \leq 2r_t \leq 4c_2(k_i)\ln\lambda$ and according to the additive drift theorem, it takes an expected number of at most $O(\log(4c_2(k_i)/c_2(k_{i+1}))$ iterations to reach $r_{t+1} \leq c_2(Y_t)\ln\lambda$. The total number of iterations of the readjustment process for r_t to satisfy the upper bound is

$$\sum_{i=0}^{N-1} O\left(\log\left(\frac{4c_2(k_i)}{c_2(k_{i+1})}\right)\right) = O\left(\log\left(4^N \frac{c_2(k_0)}{c_2(k_N)}\right)\right) = O(N + \log n).$$

We then consider the lower bound. Since $c_1(k_i)$ decreases along with k_i, once $c_1(k_0)$ is hit, the lower bound condition is obtained for all the following k_i.

Now we compute the expected number of generations for k_0 to decrease to k_N. We choose b large enough such that $2e^{-2b\epsilon/3} \leq 1/2 - \delta/2$ holds for some positive constant $\delta > 0$ and note that b is constant. Applying Lemma 3.5 we obtain $\Pr(r_t \geq c_2(k)\ln\lambda + 2^b) \leq 2e^{-2b\epsilon/3}$ and $\Pr(r_t \leq c_1(k)\ln\lambda - 2^b) \leq 2e^{-2b\epsilon/3}$. Once r_t is between $c_1(k)\ln\lambda$ and $c_2(k)\ln\lambda$, before k decreases to another bound level, we have that $c_1(k)\ln\lambda - 2^b \leq r_t \leq c_2(k)\ln\lambda + 2^b$ happens with probability at least δ. We see that there are at most $\log_2^{200}$ steps between range $c_1(k)\ln\lambda \leq r \leq c_2(k)\ln\lambda$ and an even smaller range $c_1(k)\ln\lambda \leq r \leq c_2(k)\ln(\lambda)/200$ which is described in Lemma 4.3. If r_t reaches the wider region, it takes at most a constant number of iterations α in expectation to reach the narrow region because our mutation scheme employs a 50% chance to perform a random step of the mutation rate. Based on Lemma 4.3 the narrow region for the rate ensures $0.05\ln(\lambda)/\ln(en/k)$ drift on the fitness at distance k. This contributes to an average drift of at least $0.05\ln(\lambda)/\ln(en/k) \cdot \delta/(1+\alpha) = \Omega(\log(\lambda)/\log(en/k)$ for all random rates at distance k. Applying Theorem 3.1, we can estimate the runtime as

$$O\left(\frac{1}{\log(\lambda)/\log(e\log\lambda)} + \int_{n/\log\lambda}^{n/2} \frac{\mathrm{d}k}{\log(\lambda)/\log(en/k)}\right) = O\left(\frac{n}{\log\lambda}\right).$$

Details about how to compute the above integral can be found in the proof of Theorem 4 of [1]. We notice that N is the number of different k values and N must be bounded by the above runtime. Combining the expected number of $O(N+\log n)$ iterations to adjust r_t and the expected number of $O(\sqrt{n})$ iterations to hit $k < n/2$, the total runtime is $O(n/\log\lambda)$ in expectation. □

5 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 with respect 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 5.1 and apply that result afterwards to estimate the number of generations to cross the region.

Lemma 5.1. *Let $n/\lambda \leq k \leq n/\ln\lambda$ and $1 \leq r \leq \ln(\lambda)/2$. Then $E(\Delta) \geq (1 - o(1))\min\{1/2, \sqrt{\lambda}k/(8n)\}$.*

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 have frequently often $r_t \in [1, \ln(\lambda)/2]$ which provides the drift we need.

Theorem 5.2. *Assume $k \leq n/\ln\lambda$ for the current OneMax-value of the (1+λ) EA. Then the expected number of generations until $k \leq n/\lambda$ is $O(n/\log\lambda)$.*

6 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 6.1. *Let $k \leq n/\lambda$ and $3 \leq r_t \leq \ln(\lambda)/2$. Then, the probability that $r_{t+1} = r_t/2$ is at least 0.51.*

We will also have to show that the r-value will not be much above 3 after it for the first time has become less than or equal to 3. This is achieved in the proof of the following theorem. It shows that the near region allows a speed-up of a factor of $\Theta(\lambda)$ here since every offspring only has a probability of $O(1/\lambda)$ of making progress (see also [13, 14]).

Theorem 6.2. *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 gets into this region, we consider the stochastic process $X_t := \max\{0, \lceil\log_2(r_t/c)\rceil\}$, where $c := 4$, which is even larger than the lower bound on r_t from Lemma 6.1. The lemma gives us the drift $E(X_t - X_{t+1} \mid X_t; X_t > c) \geq 0.02$. 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 $r_t \leq c$. 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 3.5. We choose b large enough such that $2e^{-2b\cdot 0.02/4} \leq 1-\delta$ holds for some positive constant $\delta > 0$ and note that b is constant. We consider two cases for r_t. If $r_t \leq c + 2^b$, which happens with probability at least δ, we have $r_t = O(1)$ and obtain a probability of at least

$$1-\left(1 - \binom{Y_t}{1}\left(\frac{r_t}{n}\right)\left(1-\frac{r_t}{n}\right)^{n-1}\right)^{\lambda} \geq 1-\left(1-\Theta\left(\frac{Y_t}{n}\right)\right)^{\lambda} = \Omega(\lambda Y_t/n),$$

to improve the OneMax-value by 1, using that $Y_t = O(n/\lambda)$. If $r_t > c + 2^b$, 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 straightforward multiplicative drift analysis (Theorem 3.2 using $\delta = \Theta(\lambda/n)$) 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 c, this proves the theorem. □

7 PUTTING EVERYTHING TOGETHER

In this section, we put together the analyses of the different regimes to prove our main result.

Proof of Theorem 1.1. The lower bound actually holds for all unbiased parallel black-box algorithms, as shown in [1].

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 4.4, Theorem 5.2 and Theorem 6.2, 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 $O(n/\log\lambda + n\log(n)/\lambda)$. □

Proof of Lemma 1.2. We basically revisit the regions of different OneMax-values analyzed in this paper and bound the time spent in these regions under the assumption $r = \ln(\lambda)/2$. In the far region, Lemmas 4.2 and 4.3, 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 4.4.

The middle region is shortened at the lower end. For $k \geq n/\sqrt{\lambda}$, Lemma 5.1 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 6.2, 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. □

8 CONCLUSIONS

We proposed and analyzed a simple self-adjusting mutation scheme for the (1+λ) EA. By using this mutation scheme, the upper bound for the expected optimization time of the (1+λ) EA matches the asymptotic lower bound for every parallel unbiased black-box algorithm. To the best of our knowledge, this is the first time that a natural EA, without any prior knowledge of the objective function achieves a superconstant speed improvement on OneMax. As a corollary from our analyses, we have noted that using a fixed rate of $r = \ln(\lambda)/2$ gives the bound $O(n/\log\lambda + n\log(n)/\lambda^{1/2})$, which is also asymptotically optimal unless λ is very small. However, this setting is far off the usual constant choice of r. We find it noteworthy that our algorithm automatically adjust the rate, first to logarithmic values and then to constant ones.

An open question is how our mutation scheme performs on objective functions beyond OneMax. Future work could characterize function classes where this scheme fails or succeeds.

Acknowledgments. This work was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH, and by a grant by the Danish Council for Independent Research (DFF-FNU 4002–00542).

REFERENCES

[1] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. 2014. Unbiased black-box complexity of parallel search. In *Proc. of PPSN '14*. 892–901.
[2] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. 2010. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Proc. of PPSN '10*. 1–10.
[3] Duc-Cuong Dang and Per Kristian Lehre. 2016. Self-adaptation of mutation rates in non-elitist populations. In *Proc. of PPSN '16*, Vol. 9921. 803–813.
[4] Benjamin Doerr. 2011. Analyzing randomized search heuristics: tools from probability theory. In *Theory of Randomized Search Heuristics*, Anne Auger and Benjamin Doerr (Eds.). World Scientific Publishing, 1–20.
[5] Benjamin Doerr. 2016. Optimal parameter settings for the (1 + (λ, λ)) genetic algorithm. In *Proc. of GECCO '16*. 1107–1114.
[6] Benjamin Doerr and Carola Doerr. 2015. Optimal parameter choices through self-adjustment: applying the 1/5-th rule in discrete settings. In *Proc. of GECCO '15*. 1335–1342.
[7] Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science* 567 (2015), 87–104.
[8] Benjamin Doerr, Carola Doerr, and Timo Kötzing. 2016. Provably optimal self-adjusting step sizes for multi-valued decision variables. In *Proc. of PPSN '16*. 782–791.
[9] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. k-Bit mutation with self-adjusting k outperforms standard bit mutation. In *Proc. of PPSN '16*. 824–834.
[10] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. Optimal parameter choices via precise black-box analysis. In *Proc. of GECCO '16*. 1123–1130.
[11] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. 2017. The (1+λ) evolutionary algorithm with self-adjusting mutation rate. *ArXiv e-prints* (April 2017). arXiv:1704.02191
[12] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. 2012. Multiplicative drift analysis. *Algorithmica* 64, 4 (2012), 673–697.
[13] Benjamin Doerr and Marvin Künnemann. 2015. Optimizing linear functions with the (1+λ) evolutionary algorithm - different asymptotic runtimes for different instances. *Theoretical Computer Science* 561 (2015), 3–23.
[14] Christian Gießen and Carsten Witt. 2016. The interplay of population size and mutation probability in the (1+λ) EA on OneMax. *Algorithmica* (2016).
[15] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. 2005. On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation* 13 (2005), 413–440.
[16] Daniel Johannsen. 2010. *Random combinatorial structures and randomized search heuristics.* Ph.D. Dissertation. Saarland University.
[17] Timo Kötzing, Andrei Lissovoi, and Carsten Witt. 2015. (1+1) EA on generalized dynamic OneMax. In *Proc. of FOGA '15*. 40–51.
[18] Jörg Lässig and Dirk Sudholt. 2011. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proc. of FOGA '11*. 181–192.
[19] Per Kristian Lehre and Carsten Witt. 2014. Concentrated hitting times of randomized search heuristics with variable drift. In *Proc. of ISAAC '14*. 686–697.
[20] Boris Mitavskiy, Jonathan E. Rowe, and Chris Cannings. 2009. Theoretical analysis of local search strategies to optimize network communication subject to preserving the total number of links. *International Journal of Intelligent Computing and Cybernetics* 2, 2 (2009), 243–284.
[21] Frank Neumann and Ingo Wegener. 2007. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science* 378 (2007), 32–40.