Algorithmica
https://doi.org/10.1007/s00453-023-01153-9

Self-adjusting Population Sizes for Non-elitist Evolutionary Algorithms: Why Success Rates Matter

Mario Alejandro Hevia Fajardo[1] · Dirk Sudholt[2]

Received: 10 November 2021 / Accepted: 27 June 2023

Abstract

Evolutionary algorithms (EAs) are general-purpose optimisers that come with several parameters like the sizes of parent and offspring populations or the mutation rate. It is well known that the performance of EAs may depend drastically on these parameters. Recent theoretical studies have shown that self-adjusting parameter control mechanisms that tune parameters during the algorithm run can provably outperform the best static parameters in EAs on discrete problems. However, the majority of these studies concerned elitist EAs and we do not have a clear answer on whether the same mechanisms can be applied for non-elitist EAs. We study one of the best-known parameter control mechanisms, the one-fifth success rule, to control the offspring population size λ in the non-elitist $(1, \lambda)$ EA. It is known that the $(1, \lambda)$ EA has a sharp threshold with respect to the choice of λ where the expected runtime on the benchmark function OneMax changes from polynomial to exponential time. Hence, it is not clear whether parameter control mechanisms are able to find and maintain suitable values of λ. For OneMax we show that the answer crucially depends on the success rate s (i. e. a one-$(s+1)$-th success rule). We prove that, if the success rate is appropriately small, the self-adjusting $(1, \lambda)$ EA optimises OneMax in $O(n)$ expected generations and $O(n \log n)$ expected evaluations, the best possible runtime for any unary unbiased black-box algorithm. A small success rate is crucial: we also show that if the success rate is too large, the algorithm has an exponential runtime on OneMax and other functions with similar characteristics.

Keywords Evolutionary algorithms · Parameter control · Theory · Runtime analysis · Non-elitism

✉ Dirk Sudholt
dirk.sudholt@uni-passau.de

[1] University of Sheffield, Sheffield, UK

[2] University of Passau, Passau, Germany

1 Introduction

Evolutionary algorithms (EAs) are general-purpose randomised search heuristics inspired by biological evolution that have been successfully applied to solve a wide range of optimisation problems. The main idea is to maintain a population (multiset) of candidate solutions (also called *search points* or *individuals*) and to create new search points (called *offspring*), from applying genetic operators such as *mutation* (making small changes to a *parent* search point) and/or *recombination* (combining features of two or more parents). A process of *selection* is then applied to form the next generation's population. This process is iterated over many generations in the hope that the search space is explored and high-fitness search points emerge.

Thanks to their generality, evolutionary algorithms are especially helpful when the problem in hand is not well-known or when the underlying fitness landscape can only be queried through fitness function evaluations (black-box optimisation) [1]. Frequently in real-world applications the fitness function evaluations are costly, therefore there is a large interest in reducing the number of fitness function evaluations needed to optimise a function, also called optimisation time or runtime [2–5].

EAs come with a range of parameters, such as the size of the parent population, the size of the offspring population or the mutation rate. It is well known that the optimisation time of an evolutionary algorithm may depend drastically and often unpredictably on their parameters and the problem in hand [6, 7]. Hence, parameter selection is an important and growing field of study.

One approach for parameter selection is to theoretically analyse the optimisation time (runtime analysis) of evolutionary algorithms to understand how different parameter settings affect their performance on different parameter landscapes. This approach has given us a better understanding of how to properly set the parameters of evolutionary algorithms. In addition, owing to runtime analysis we also know that during the optimisation process the optimal parameter values may change, making any static parameter choice have sub-optimal performance [7]. Therefore, it is natural to also analyse parameter settings that are able to change throughout the run. These mechanisms are called parameter control mechanisms.

Parameter control mechanisms aim to identify parameter values that are optimal for the current state of the optimisation process. In continuous optimisation, parameter control is indispensable to ensure convergence to the optimum, therefore, non-static parameter choices have been standard for several decades. In sharp contrast to this, in the discrete domain parameter control has only become more common in recent years. This is in part owing to theoretical studies demonstrating that fitness-dependent parameter control mechanisms can provably outperform the best static parameter settings [8–11]. Despite the proven advantages, fitness-dependent mechanisms have an important drawback: to have an optimal performance they generally need to be tailored to a specific problem, which needs a substantial knowledge of the problem in hand [7].

To overcome this constraint, several parameter control mechanisms have been proposed that update the parameters in a *self-adjusting* manner. The idea is to adapt parameters based on information gathered during the run, for instance whether a generation has led to an improvement in the best fitness (called a *success*) or not. Theoretical studies have proven that in spite of their simplicity, these mechanisms are able to use *good* parameter values throughout the optimisation, obtaining the same or better performance than any static parameter choice.

There is a growing body of research in this rapidly emerging area. Lässig and Sudholt [12] presented self-adjusting schemes for choosing the offspring population size in (1+λ) EAs and the number of islands in an island model. Mambrini and Sudholt [13] adapted the migration interval in island models and showed that adaptation can reduce the communication effort beyond the best possible fixed parameter. Doerr and Doerr [14] proposed a self-adjusting mechanism in the $(1 + (\lambda, \lambda))$ GA based on the one-fifth rule and proved that it optimises the well known benchmark function $\textsc{OneMax}(x) = \sum_{i=1}^{n} x_i$ (counting the number of ones in a bit string $x \in \{0, 1\}^n$ of length n) in $O(n)$ expected evaluations, being the fastest known unbiased genetic algorithm on OneMax. Hevia Fajardo and Sudholt [15] studied modifications to the self-adjusting mechanism in the $(1 + (\lambda, \lambda))$ GA on Jump functions, showing that they can perform nearly as well as the $(1 + 1)$ EA with the optimal mutation rate. Doerr et al. [16] presented a success-based choice of the mutation strength for a randomised local search (RLS) variant, proving that it is very efficient for a generalisation of the OneMax problem to a larger alphabet than $\{0, 1\}$. Doerr, Gießen, Witt, and Yang [17] showed that a success-based parameter control mechanism is able to identify and track the optimal mutation rate in the (1+λ) EA on OneMax, matching the performance of the best known fitness-dependent parameter [8]. Doerr and Doerr give a comprehensive survey of theoretical results [7].

Most theoretical analyses of parameter control mechanisms focus on so-called *elitist EAs* that always reject worsening moves (with notable exceptions that study self-adaptive mutation rates in the $(1, \lambda)$ EA [18] and the (μ, λ) EA [19], and hyper-heuristics that choose between elitist and non-elitist selection mechanisms [20]). The performance of parameter control mechanisms in non-elitist algorithms is not well understood, despite the fact that non-elitist EAs are often better at escaping from local optima [21] and are often applied in practice. There are many applications of non-elitist evolutionary algorithms for which an improved theoretical understanding of parameter control mechanisms could bring performance improvements matching or exceeding the ones seen for elitist algorithms.

We consider the $(1, \lambda)$ EA on OneMax that in every generation creates λ offspring and selects the best one for survival. Rowe and Sudholt [22] have shown that there is a sharp threshold at $\lambda = \log_{\frac{e}{e-1}} n$ between exponential and polynomial runtimes on OneMax. A value $\lambda \geq \log_{\frac{e}{e-1}} n$ ensures that the offspring population size is sufficiently large to ensure a positive *drift* (expected progress) towards the optimum even on the most challenging fitness levels. For easier fitness levels, smaller values of λ are sufficient.

This is a challenging scenario for self-adjusting the offspring population size λ since too small values of λ can easily make the algorithm decrease its current fitness, moving away from the optimum. For static values of $\lambda \leq (1 - \varepsilon) \log_{\frac{e}{e-1}} n$, for any constant $\varepsilon > 0$, we know that the optimisation time is exponential with high probability [22]. Moreover, too large values for λ can waste function evaluations and blow up the optimisation time.

We consider a self-adjusting version of the $(1,\lambda)$ EA that uses a success-based rule. Following the naming convention from [7] the algorithm is called self-adjusting $(1,\{F^{1/s}\lambda, \lambda/F\})$ EA (self-adjusting $(1,\lambda)$ EA). For an update strength F and a success rate s, in a generation where no improvement in fitness is found, λ is increased by a factor of $F^{1/s}$ and in a successful generation, λ is divided by a factor F. If one out of $s+1$ generations is successful, the value of λ is maintained. The case $s=4$ is the famous one-fifth success rule [23, 24].

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

We answer this question in the affirmative if the success rate s is chosen correctly. We show in Sect. 3 that, if s is a constant with $0<s<1$, then the self-adjusting $(1,\lambda)$ EA optimises OneMax in $O(n)$ expected generations and $O(n\log n)$ expected fitness evaluations. The bound on evaluations is optimal for all unary unbiased black-box algorithms [11, 25]. However, if s is a sufficiently large constant, $s\geq 18$, the runtime on OneMax becomes exponential with overwhelming probability (see Sect. 4). The reason is that then unsuccessful generations increase λ only slowly, whereas successful generations decrease λ significantly. This effect is more pronounced during early stages of a run when the current search point is still far away from the optimum and successful generations are common. We show that then the algorithm gets stuck in a non-stable equilibrium with small λ-values and frequent fallbacks (fitness decreases) at a linear Hamming distance to the optimum. This effect is not limited to OneMax; we show that this negative result easily translates to other functions for which it is easy to find improvements during early stages of a run.

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

In Sect. 3.2 we use the potential to bound the expected number of evaluations to increase the best-so-far fitness by $\log n$, reaching a new fitness value denoted by b. The time until this happens is called an *epoch*. During an epoch, the number of evaluations is bounded by arguing that λ is unlikely to increase much beyond a threshold value of $O(1/p^+_{b-1,1})$, where $p^+_{b-1,1}$ is the worst-case improvement probability as long as no fitness of at least b is reached. Since at the start of an epoch the initial value of λ is not known, we provide a tail bound showing that λ is unlikely to attain excessively large values and hence any unknown values of λ contribute to a total of $O(n\log n)$ expected evaluations.

In Sect. 5 we complement our runtime analyses with experiments on OneMax. First we compare the runtime of the self-adjusting $(1,\lambda)$ EA, the self-adjusting $(1+\lambda)$ EA and the $(1,\lambda)$ EA with the best known fixed λ for different problem sizes. Second, we show a sharp threshold for the success rate at $s\approx 3.4$ where the runtime changes from polynomial to exponential. This indicates that the widely used one-fifth rule ($s=4$) is inefficient here, but other success rules achieve optimal asymptotic runtime. Finally, we show how different values of s affect fixed target running times, the growth of λ

over time and the time spent in each fitness value, shedding light on the non-optimal equilibrium states in the self-adjusting $(1, \lambda)$ EA.

An extended abstract containing preliminary versions of our results appeared in [26]. The results in this manuscript have evolved significantly from there. In [26] we bounded the expected number of evaluations by showing that, when the fitness distance to the optimum has decreased below $n/\log^3 n$, the self-adjusting $(1, \lambda)$ EA behaves similarly to its elitist version, a $(1+\{F^{1/s}\lambda, \lambda/F\})$ EA. The expected number of evaluations to reach this fitness distance was estimated using Wald's equation, and a reviewer of this manuscript pointed out a mistake in the application of Wald's equation in [26] as the assumption of independent random variables was not met. We found a different argument to fix the proof and noticed that the new argument simplifies the analysis considerably. In particular, the simplified proof is no longer based on the elitist $(1+\{F^{1/s}\lambda, \lambda/F\})$ EA. (We remark that, independently, the analysis from [26] was also simplified in [27, 28] and extended to the class of monotone functions.)

Other changes include rewriting our results in order to refine the presentation and give a more unified analysis for both our positive and negative results. In particular, the conditions on s have been relaxed from $s < \frac{e-1}{e}$ vs. $s \geq 22$ towards $s < 1$ vs. $s \geq 18$. We also extended our negative results towards other fitness function classes JUMP_k, CLIFF_d, ZEROMAX, TWOMAX and RIDGE (Theorem 4.4).

2 Preliminaries

We study the expected number of generations and fitness evaluations of the self-adjusting $(1, \lambda)$ EA with self-adjusted offspring population size λ to find the optimum of the n-dimensional pseudo-Boolean function $\text{ONEMAX}(x) = \text{OM}(x) := \sum_{i=1}^{n} x^{(i)}$. We define $X_0, X_1, \ldots$ as the sequence of states of the algorithm, where $X_t = (x_t, \lambda_t)$ describes the current search point x_t and the offspring population size λ_t at generation t. We often omit the subscripts t when the context is obvious.

Using the naming convention from [7] we call the algorithm self-adjusting $(1, \{F^{1/s}\lambda, \lambda/F\})$ EA (Algorithm 1). The algorithm behaves as the conventional $(1, \lambda)$ EA: each generation it creates λ offspring by flipping each bit independently with probability $1/n$ from the parent and selecting the fittest offspring as a parent for the next generation. In addition, in every generation it adjusts the offspring population size depending on the success of the generation. If the fittest offspring y is better than the parent x, the offspring population size is divided by the *update strength* $F > 1$, and multiplied by $F^{1/s}$ otherwise, with $s > 0$ being the *success rate*.

The idea of the parameter control mechanism is based on the interpretation of the one-fifth success rule from [24]. The parameter λ remains constant if the algorithm has a success every $s + 1$ generations as then its new value is $\lambda \cdot (F^{1/s})^s \cdot 1/F = \lambda$. In pseudo-Boolean optimisation, the one-fifth success rule was first implemented by Doerr et al. [10], and proved to track the optimal offspring population size on the $(1 + (\lambda, \lambda))$ GA in [14]. Our implementation is closer to the one used in [29], where the authors generalise the success rule, implementing the success rate s as a hyper-parameter.

Note that we regard λ to be a real value, so that changes by factors of $1/F$ or $F^{1/s}$ happen on a continuous scale. Following Doerr and Doerr [14], we assume that, whenever an integer value of λ is required, λ is rounded to a nearest integer. For the sake of readability, we often write λ as a real value even when an integer is required. Where appropriate, we use the notation $\lfloor \lambda \rceil$ to denote the integer nearest to λ (that is, rounding up if the fractional value is at least 0.5 and rounding down otherwise).

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

```
Initialization: Choose $x \in \{0,1\}^n$ uniformly at random (u.a.r.) and $\lambda := 1$
Optimization: for $t \in \{1, 2, \dots\}$ do
    Mutation: for $i \in \{1, \dots, \lambda\}$ do
        Create $y'_i \in \{0,1\}^n$ by copying $x$ and flipping each bit independently with probability $1/n$.
    Selection: Choose $y \in \{y'_1, \dots, y'_\lambda\}$ with $f(y) = \max\{f(y'_1), \dots, f(y'_\lambda)\}$ u.a.r.
    Update:
    if $f(y) > f(x)$ then $x \leftarrow y; \lambda \leftarrow \max\{1, \lambda/F\}$ else $x \leftarrow y; \lambda \leftarrow F^{1/s}\lambda$
```

2.1 Notation and Probability Estimates

We now give notation and tools for all $(1, \lambda)$ EA algorithms.

Definition 2.1 For all $\lambda \in \mathbb{N}$ and $0 \le i < n$ we define:

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

As in [22], we call $\Delta^+_{i,\lambda}$ *forward drift* and $\Delta^-_{i,\lambda}$ *backward drift* and note that they are both at least 1 by definition. We call the event underlying the probability $p^-_{i,\lambda}$ a *fallback*, that is, the event that all offspring have lower fitness than the parent and thus $\mathrm{OM}(x_{t+1}) < \mathrm{OM}(x_t)$. The probability of a fallback, is $p^-_{i,\lambda} = (p^-_{i,1})^\lambda$ since all offspring must have worse fitness than their parent. Now, $p^+_{i,1}$ is the probability of one offspring finding a better fitness value and $p^+_{i,\lambda} = 1 - (1 - p^+_{i,1})^\lambda$ since it is sufficient that one offspring improves the fitness. Along with common bounds and standard arguments, we obtain the following lemma.

Lemma 2.2 *For any* $(1, \lambda)$ *EA on* OneMax*, the quantities from Definition 2.1 are bounded as follows.*

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

$$p_{i,\lambda}^{+} \le 1 - \left(1 - 1.14\left(\frac{n-i}{n}\right)\left(1-\frac{1}{n}\right)^{n-1}\right)^{\lambda} \le 1 - \left(1 - \frac{n-i}{n}\right)^{\lambda} \tag{2}$$

If $0.84n \le i \le 0.85n$ *and* $n \ge 163$, *then* $p_{i,1}^{+} \le 0.069$.

$$\left(\frac{i}{n} - \frac{1}{e}\right)^{\lambda} \le p_{i,\lambda}^{-} \le \left(1 - \frac{n-i}{en} - \left(1-\frac{1}{n}\right)^{n}\right)^{\lambda} \le \left(\frac{e-1}{e}\right)^{\lambda} \tag{3}$$

$$1 \le \Delta_{i,\lambda}^{-} \le \frac{e}{e-1} \tag{4}$$

$$1 \le \Delta_{i,\lambda}^{+} \le \sum_{j=1}^{\infty}\left(1 - \left(1 - \frac{1}{j!}\right)^{\lambda}\right) \tag{5}$$

If $\lambda \ge 5$, *then* $\Delta_{i,\lambda}^{+} \le \lceil \log \lambda \rceil + 0.413$.

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

$$p_{i,1}^{+} \ge \frac{n-i}{n}\left(1-\frac{1}{n}\right)^{n-1} \ge \frac{n-i}{en}. \tag{6}$$

Along with $p_{i,\lambda}^{+} = 1 - (1 - p_{i,1}^{+})^{\lambda}$, this proves one of the lower bounds in Eq. (1) in Lemma 2.2. Additionally, using $(1+x)^r \le \frac{1}{1-rx}$ for all $x \in [-1, 0]$ and $r \in \mathbb{N}$

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

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

$$p_{i,1}^{+} \le \frac{n-i}{n}.$$

Additionally, we use the following upper bound shown in [30]:

$$p_{i,1}^{+} \le \min\left\{1.14\left(\frac{n-i}{n}\right)\left(1-\frac{1}{n}\right)^{n-1}, 1\right\}.$$

Since $1.14\left(\frac{n-i}{n}\right)\left(1-\frac{1}{n}\right)^{n-1} \le 1$ for all problem sizes $n > 1$ and $n = 1$ is trivially solved we omit the minimum from now on.

Along with $p_{i,\lambda}^{+} = 1 - (1 - p_{i,1}^{+})^{\lambda}$, this proves the upper bounds in Eq. (2) in Lemma 2.2.

The additional upper bound for $p^+_{i,1}$ when $0.84n \leq i \leq 0.85n$ uses a more precise bound from [31] of:

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

This implies that, for every $n \geq 163$ and $0.84n \leq i \leq 0.85n$, $p^+_{i,1} \leq 0.069$.

We now calculate $p^-_{i,1}$. For the upper bound we use

$$p^-_{i,1} = 1 - p^+_{i,1} - p^0_{i,1}.$$

Using Eq. (6) and bounding $p^0_{i,1}$ from below by the probability of no bit flipping, that is,

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

we get

$$p^-_{i,1} \leq 1 - \frac{n-i}{en} - \left(1-\frac{1}{n}\right)^n. \tag{7}$$

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

$$\begin{aligned} p^-_{i,1} &\geq \left(1-\left(1-\frac{1}{n}\right)^i\right)\left(1-\frac{1}{n}\right)^{n-i} \\ &= \left(1-\frac{1}{n}\right)^{n-i} - \left(1-\frac{1}{n}\right)^n \\ &\geq 1 - \frac{n-i}{n} - \frac{1}{e} = \frac{i}{n} - \frac{1}{e}. \end{aligned} \tag{8}$$

Using $p^-_{i,\lambda} = (p^-_{i,1})^\lambda$ with Eqs. (7) and (8) we obtain

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

The upper bound is simplified as follows:

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

To prove the bounds on the backward drift from Eq. (4), note that the drift is conditional on a decrease in fitness, hence the lower bound of 1 is trivial.

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

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

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

The lower bound on the forward drift, Eq. (5), is again trivial since the forward drift is conditional on an increase in fitness.

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

$$\Pr(B \ge j) = \binom{n}{j}\left(\frac{1}{n}\right)^j \le \frac{1}{j!}. \tag{9}$$

To bound $\Delta^+_{i,\lambda}$ we use the probability that any of the λ offspring flip at least j bits. Let M_λ denote the maximum of the number of bits flipped in λ independent standard bit mutations, then we have $\Pr(M_\lambda \ge j) = 1 - (1 - \Pr(B \ge j))^\lambda$ and

$$\Delta^+_{i,\lambda} \le \mathrm{E}(M_\lambda) \le \sum_{j=1}^{\infty} \Pr(M_\lambda \ge j) \le \sum_{j=1}^{\infty}\left(1 - \left(1 - \frac{1}{j!}\right)^\lambda\right).$$

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

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

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

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

Lemma 2.3 *Consider the self-adjusting* $(1, \lambda)$ *EA on any unimodal function with an initial offspring population size of* $\lambda_0 \leq eF^{1/s}n^3$*. The probability that, during a run, the offspring population size exceeds* $eF^{1/s}n^3$ *before the optimum is found is at most* $\exp(-\Omega(n^2))$.

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

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

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

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

□

2.2 Drift Analysis and Potential Functions

Drift analysis is one of the most useful tools to analyse evolutionary algorithms [32]. A general approach for the use of drift analysis is to identify a potential function that adequately captures the progress of the algorithm and the distance from a desired target state (e. g. having found a global optimum). Then we analyse the expected changes in the potential function at every step of the optimisation (drift of the potential) and finally translate this knowledge about the drift into information about the runtime of the algorithm.

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

Theorem 2.4 *(Additive Drift* [33]*) Let $(X_t)_{t\geq 0}$ be a sequence of non-negative random variables over a finite state space $S \subseteq \mathbb{R}$. Let T be the random variable that denotes the earliest point in time $t \geq 0$ such that $X_t = 0$. If there exists $c > 0$ such that, for all $t < T$,*

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

then

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

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

Theorem 2.5 *(Negative drift theorem* [34, 35]*) Let X_t, $t \geq 0$, be real-valued random variables describing a stochastic process over some state space. Suppose there exists an interval $[a, b] \subseteq \mathbb{R}$, two constants $\delta, \varepsilon > 0$ and, possibly depending on $\ell{:=}b-a$, a function $r(\ell)$ satisfying $1 \leq r(\ell) = o(\ell/\log(\ell))$ such that for all $t \geq 0$ the following two conditions hold:*

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

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

The following theorem is a variation of Theorem 2.5 in which the second condition on large jumps is relaxed.

Theorem 2.6 *(Negative drift theorem with scaling* [36]*) Let X_t, $t > 0$ be real-valued random variables describing a stochastic process over some state space. Suppose there exists an interval $[a, b] \subseteq \mathbb{R}$ and, possibly depending on $\ell := b - a$, a drift bound $\varepsilon{:=}\varepsilon(\ell) > 0$ as well as a scaling factor $r{:=}r(\ell)$ such that for all $t \geq 0$ the following three conditions hold:*

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

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

For our analysis the first step, that is, finding a good potential function is much more interesting. A natural candidate for a potential function is the fitness of the current individual $\mathrm{OM}(x_t)$. However, the self-adjusting $(1, \lambda)$ EA adjusts λ throughout the optimisation, and the expected change in fitness crucially depends on the current value of λ. Therefore, we also need to take into account the current offspring population size λ and capture both fitness and λ in our potential function. Since we study different behaviours of the algorithm depending on the success rate s we generalise the potential function used in [26] by considering an abstract function $h(\lambda_t)$ of the current offspring population sizes. The function $h(\lambda_t)$ will be chosen differently for different contexts, such as proving a positive result for small success rates s and proving a negative result for large success rates.

Definition 2.7 Given a function $h\colon \mathbb{R} \to \mathbb{R}$, we define the potential function $g(X_t)$ as

$$g(X_t) = \mathrm{OM}(x_t) + h(\lambda_t).$$

We do not make any assumptions on $h(\lambda_t)$ at this stage, but we will choose $h(\lambda_t)$ in the following sections as functions of λ_t that reward increases of λ_t, for small values of λ_t. We note that this potential function is also a generalisation of the potential function used in [31] to analyse the self-adjusting $(1, \lambda)$ EA with a reset mechanism on the CLIFF function. We believe that this approach could be useful for the analysis of a wide range of success-based parameter control mechanisms and it might be able to simplify previous analysis such as [14, 29]. A similar approach has been used before for analysing self-adjusting mutation rates [16, 18] and for continuous domains in [37–39].

For every function $h(\lambda_t)$, we can compute the drift in the potential as shown in the following lemma. For the sake of readability we drop the subscript t in λ_t where appropriate.

Lemma 2.8 *Consider the self-adjusting $(1, \lambda)$ EA. Then for every function $h\colon \mathbb{R} \to \mathbb{R}_0^+$ and every generation t with $\mathrm{OM}(x_t) < n$ and $\lambda_t > F$, $E\left(g(X_{t+1}) - g(X_t) \mid X_t\right)$ is*

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

If $\lambda_t \le F$ then, $E\left(g(X_{t+1}) - g(X_t) \mid X_t\right)$ is

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

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

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

Given that $\lambda \geq 1$ if $\lambda \leq F$ then $h(\lambda/F)$ needs to be replaced by $h(\lambda/\lambda) = h(1)$. □

3 Small Success Rates are Efficient

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

3.1 Bounding the Number of Generations

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

Theorem 3.1 *Let the update strength* $F > 1$ *and the success rate* $0 < s < 1$ *be constants. Then for any initial search point and any initial* λ *the expected number of generations of the self-adjusting* $(1, \lambda)$ *EA on* OneMax *is* $O(n)$.

We note that the self-adjusting mechanism aims to obtain one success every $s + 1$ generations. The intuition behind using $0 < s < 1$ in Theorem 3.1 is that then the algorithm tries to succeed (improve the fitness) more than half of the generations. In order to achieve that many successes the λ-value needs to be large, which in turn reduces the probability (and number) of fallbacks during the run.

We make use of the potential function from Definition 2.7 and define $h(\lambda)$ to obtain the potential function used in this section as follows.

Definition 3.2 We define the potential function $g_1(X_t)$ as

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

The definition of $h(\lambda)$ in this case is used as a penalty term that grows linearly in $\log_F \lambda$ (since $-\log_F \left(\frac{enF^{1/s}}{\lambda_t}\right) = -\log_F(enF^{1/s}) + \log_F(\lambda_t)$). That is, when λ

increases the penalty decreases and vice-versa. The idea behind this definition is that small values of λ may lead to decreases in fitness, but these are compensated by an increase in λ and a reduction of the penalty term.

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

Lemma 3.3 *For all generations t, the fitness and the potential are related as follows:* $\mathrm{OM}(x_t) - \frac{2s}{s+1}\log_F(enF^{1/s}) \le g_1(X_t) \le \mathrm{OM}(x_t)$. *In particular,* $g_1(X_t) = n$ *implies* $\mathrm{OM}(x_t) = n$.

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

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

Lemma 3.4 *Consider the self-adjusting* $(1, \lambda)$ *EA as in Theorem 3.1. Then for every generation t with* $\mathrm{OM}(x_t) < n$,

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

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

Proof Given that $h(\lambda_t) = -\frac{2s}{s+1}\log_F\left(\max\left(\frac{enF^{1/s}}{\lambda_t}, 1\right)\right)$ is a non-decreasing function, if $\lambda \le F$ then $h(1) \ge h(\lambda/F)$. Hence, by Lemma 2.8, for all λ, $\mathrm{E}\left(g_1(X_{t+1}) - g_1(X_t) \mid X_t\right)$ is at least

$$\left(\Delta^+_{i,\lambda} + h(\lambda/F) - h(\lambda F^{1/s})\right) p^+_{i,\lambda} + h(\lambda F^{1/s}) - h(\lambda) - \Delta^-_{i,\lambda} p^-_{i,\lambda}. \qquad (10)$$

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

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

By Lemma 2.2 $\Delta^+_{i,\lambda} \geq 1$, hence $\mathrm{E}\left(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t < en\right) \geq \frac{2}{s+1} - p^+_{i,\lambda} - \Delta^-_{i,\lambda} p^-_{i,\lambda}$. Using $\frac{2}{s+1} = \frac{s+1+1-s}{s+1} = 1 + \frac{1-s}{s+1}$ yields

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

By Lemma 2.2 this is at least

$$\begin{aligned}
&\frac{1-s}{s+1} + \left(1 - 1.14\left(\frac{n-i}{n}\right)\left(1-\frac{1}{n}\right)^{n-1}\right)^{\lfloor\lambda\rceil} - \left(\frac{e}{e-1}\right)\left(1 - \frac{n-i}{en} - \left(1-\frac{1}{n}\right)^{n}\right)^{\lfloor\lambda\rceil} \\
&\geq \frac{1-s}{s+1} + \left(1 - \frac{1.14}{e\left(1-\frac{1}{n}\right)}\left(\frac{n-i}{n}\right)\right)^{\lfloor\lambda\rceil} - \left(\frac{e}{e-1}\right)\left(1 - \frac{n-i}{en} - \frac{1}{e}\left(1-\frac{1}{n}\right)\right)^{\lfloor\lambda\rceil} \\
&= \frac{1-s}{s+1} + \left(1 - \frac{1.14}{e}\left(\frac{n-i}{n-1}\right)\right)^{\lfloor\lambda\rceil} - \left(\frac{e}{e-1}\right)\left(\frac{e-1}{e} - \frac{n-i-1}{en}\right)^{\lfloor\lambda\rceil}. \qquad (11)
\end{aligned}$$

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

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

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

The terms y_1 and y_2 can be described by linear equations $y_1 = m_1(n-i) + b_1$ and $y_2 = m_2(n-i) + b_2$ with $m_1 = -\frac{1.14}{e(n-1)}$, $b_1 = 1$, $m_2 = -\frac{1}{n\sqrt{e^2-e}}$ and $b_2 = \sqrt{\frac{e-1}{e}} + \frac{1}{n\sqrt{e^2-e}}$. Since $m_2 < m_1$ for all $n \geq 11$, the difference $y_1 - y_2$ is minimised for $n - i = 1$. When $n - i = 1$, then $y_1 = 1 - \frac{1.14}{e(n-1)} > \left(\frac{e-1}{e}\right)^{1/2} = y_2$ for all $n > 3$, therefore $y_1 > y_2$ for all $0 \leq i < n$.

When $\lfloor\lambda\rceil = 1$, from Eq. (11) $\mathrm{E}\left(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t \leq 1.5\right) \geq \frac{1-s}{s+1} - \frac{1.14}{e}\left(\frac{n-i}{n-1}\right) + \frac{n-i-1}{(e-1)n}$ which is monotonically decreasing for $0 < i < n-1$ when $n > e/(1.14 - 0.14e) \approx 3.58$, hence $\mathrm{E}\left(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t \leq 1.5\right) \geq \frac{1-s}{s+1} - \frac{1.14}{e(n-1)}$ which is bounded by $\frac{1-s}{2e}$ for large enough n since $s < 1$.

Finally, for the case $\lambda_t \geq en$, in an unsuccessful generation the penalty term is capped, hence $h(\lambda F^{1/s}) = 0$. We note that $h(\lambda/F) \geq h(\lambda) - \frac{2s}{s+1}$ and $h(\lambda) \leq 0$ for all λ. Then by Eq. (10), $\mathrm{E}\left(g_1(X_{t+1}) - g_1(X_t) \mid X_t, \lambda_t \geq en\right)$ is at least

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

By definition Lemma 2.2, $\lambda_t \geq en$ implies $p^+_{i,\lambda} \geq 1 - \left(1 - \frac{1}{en}\right)^{en} \geq 1 - \frac{1}{e}$ and $p^-_{i,\lambda} \Delta^-_{i,\lambda} \leq \left(\frac{e-1}{e}\right)^{en} \frac{e}{e-1} = \left(\frac{e-1}{e}\right)^{en-1}$. Together,

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

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

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

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

With this constant lower bound on the drift of the potential, the proof of Theorem 3.1 is now quite straightforward.

Proof of Theorem 3.1 We bound the time to get to the optimum using the potential function $g_1(X_t)$. Lemma 3.4 shows that the potential has a positive constant drift whenever the optimum has not been found, and by Lemma 3.3 if $g_1(X_t) = n$ then the optimum has been found. Therefore, we can bound the number of generations by the time it takes for $g_1(X_t)$ to reach n.

To fit the perspective of the additive drift theorem (Theorem 2.4) we switch to the function $\overline{g_1}(X_t) := n - g_1(X_t)$ and note that $\overline{g_1}(X_t) = 0$ implies that

$g_1(X_t) = \mathrm{OM}(x_t) = n$. The initial value $\overline{g_1}(X_0)$ is at most $n + \frac{2s}{s+1} \log_F \left(enF^{1/s}\right)$ by Lemma 3.3. Using Lemma 3.4 and the additive drift theorem, the expected number of generations is at most

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

□

3.2 Bounding the Number of Evaluations

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

Theorem 3.5 *Let the update strength* $F > 1$ *and the success rate* $0 < s < 1$ *be constants. The expected number of function evaluations of the self-adjusting* $(1, \lambda)$ *EA on* OneMax *is* $O(n \log n)$.

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

In the elitist $(1 + \{F^{1/s}\lambda, \lambda/F\})$ EA, small values of λ are not an issue since there are no fallbacks. In our non-elitist algorithm, small values of λ can lead to decreases in fitness, and then the same fitness level can be visited multiple times.

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

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

We solve this problem as follows. We consider the best-so-far fitness $f_t^* = \max\{\mathrm{OM}(x_{t'}) \mid 0 \leq t' \leq t\}$ at time t (as a theoretical concept, as the self-adjusting $(1, \lambda)$ EA is non-elitist and unaware of the best-so-far fitness). We then divide the run into fitness intervals of size $\log n$ that we call *blocks*, and bound the time for the best-so-far fitness to reach a better block. To this end, we reconsider the potential function used to bound the expected number of generations in Theorem 3.1 and refine our arguments to obtain a bound on the expected number of *generations* to increase the best-so-far fitness by $\log n$ (see Lemma 3.6 below). Denoting by b the target fitness of a better block, in the current block the fitness is at most $b-1$. To bound the number of *evaluations*, we show that the offspring population size is likely to remain in $O(1/p_{b-1,1}^+)$, where $p_{b-1,1}^+$ is the worst-case improvement probability for a single offspring creation in the current block. An application of Wald's equation bounds the total expected number of evaluations in all generations until a new block is reached.

At the time a new block i is reached, the current offspring population size $\lambda^{(i)}$ is not known, yet it contributes to the expected number of evaluations during the new block. We provide tail bounds on $\lambda^{(i)}$ to show that excessively large values of $\lambda^{(i)}$ are unlikely. This way we bound the total contribution of $\lambda^{(i)}$'s across all blocks i by $O(n \log n)$.

Lemma 3.6 *Consider the self-adjusting* $(1, \lambda)$ *EA as in Theorem 3.5. For every $a, b \in \{0, \dots, n\}$, the expected number of generations to increase the current fitness from a value at least a to at least $b > a$ is at most*

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

For $b = a + \log n$, this bound is $O(\log n)$.

Proof We use the proof of Theorem 3.1 with a revised potential function of $\overline{g_1}'(X_t) := \max(\overline{g_1}(X_t) - (n-b), 0)$ and stopping when $\overline{g_1}'(X_t) = 0$ (which implies that a fitness of at least b is reached) or a fitness of at least b is reached beforehand. Note that the maximum caps the effect of fitness improvements that jump to fitness values larger than b. As remarked in Lemma 3.4, the drift bound for $g_1(X_t)$ still holds when only considering fitness improvements by 1. Hence, it also holds for $\overline{g_1}'(X_t)$ and the analysis goes through as before. □

In our preliminary publication [26] we introduced a novel analysis tool that we called *ratchet argument*. We considered the best-so-far fitness $f_t^* = \max\{\mathrm{OM}(x_{t'}) \mid 0 \leq t' \leq t\}$ at time t (as mentioned before, as a theoretical concept) and used drift analysis to show that, with high probability, the current fitness never drops far below f_t^*, that is, $\mathrm{OM}(x_t) \geq f_t^* - r \log n$ for a constant $r > 0$. We called this a *ratchet argument*[1]: if the best-so-far fitness increases, the lower bound on the current fitness increases as well. The lower bound thus works like a ratchet mechanism that can only move in one direction. Our revised analysis no longer requires this argument. We still

[1] This name is inspired by the term "Muller's ratchet" from biology [40] that considers a ratchet mechanism in asexual evolution, albeit in a different context.

present the following lemma since (1) it might be of interest as a structural result about the typical behaviour of the algorithm, (2) it has found applications in follow-up work [31] of [26] and it makes sense to include it here for completeness and (3) the basic argument may prove useful in analysing other non-elitist algorithms. In fact, a very similar argument was used in recent work on the $(1, \lambda)$ EA without self-adjustment [41]. Lemma 3.7 also shows that with high probability the fitness does not decrease when $\lambda \geq 4 \log n$. A proof is given in the appendix.

Lemma 3.7 *Consider the self-adjusting* $(1, \lambda)$ *EA as in Theorem 3.5. Let* $\mathrm{OM}^*_t := \max_{t' \leq t} \mathrm{OM}(x_{t'})$ *be its best-so-far fitness at generation* t *and let* T *be the first generation in which the optimum is found. Then with probability* $1 - O(1/n)$ *the following statements hold for a large enough constant* $r > 0$ *(that may depend on* s*).*

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

In [26] we divided the optimisation in blocks of $\log n$ fitness levels and with the help of the ratchet argument shown in Lemma 3.7 and other helper lemmas we showed that each block is typically optimised efficiently. Adding the time spent in each block, we obtained that the algorithm optimises OneMax in $O(n \log n)$ evaluations with high probability. It is straightforward to derive a bound on the number of expected evaluations of the same order.

In this revised analysis we still divide the optimisation in blocks of length $\log n$, but use simpler and more elegant arguments to compute the time spent in a block and the total expected runtime. To bound the time spent optimising a block, first we divide each block on smaller chunks called *phases* and bound the time spent in each phase. This is shown in the following lemma.

Lemma 3.8 *Consider the self-adjusting* $(1, \lambda)$ *EA as in Theorem 3.5. Fix a fitness value* b *and denote the current offspring population size by* λ_0*. Define* $\overline{\lambda_b} := C F^{1/s} / p^+_{b-1,1}$ *for a constant* $C > 0$ *that may depend on* F *and* s *that satisfies*

$$\left(\frac{s+1}{s} \cdot e^{1-C}\right)^{\frac{s}{s+1}} \leq \frac{F^{-1/s}}{2}. \tag{12}$$

Define a phase *as a sequence of generations that ends in the first generation where* λ *attains a value of at most* $\overline{\lambda_b}$ *or a fitness of at least* b *is reached. Then the expected number of evaluations made in that phase is* $O(\lambda_0)$.

We note that a constant $C > 0$ meeting inequality (12) exists since the left-hand side converges to 0 when C goes to infinity, while the right-hand side remains a positive constant. Additionally, given that F and s are constants $\overline{\lambda_b} = O(1/p^+_{b-1,1})$.

Proof of Lemma 3.8 If $\lambda_0 F^{1/s} \leq \overline{\lambda_b}$ or if the current fitness is at least b then the phase takes only one generation and λ_0 evaluations as claimed. Hence we assume in the following that $\overline{\lambda_b} F^{-1/s} < \lambda_0$ and that the current fitness is less than b.

We use some ideas from the proof of Theorem 9 in [14][2] that bounds the expected number of evaluations in the self-adjusting (1+(λ,λ)) GA. Let Z denote the random number of iterations in the phase and let T denote the random number of evaluations in the phase. Since λ_t can only grow by $F^{1/s}$, $\lambda_i \le \lambda_0 \cdot F^{i/s}$ for all $i \in \mathbb{N}_0$. If $Z = z$, the number of evaluations is bounded by

$$\mathrm{E}\left(T \mid Z = z\right) \le \lambda_0 \cdot \sum_{i=1}^{z} F^{i/s} = \lambda_0 \cdot \frac{F^{\frac{z+1}{s}} - F^{1/s}}{F^{1/s} - 1} \le \lambda_0 \cdot \frac{F^{\frac{z+1}{s}}}{F^{1/s} - 1}.$$

While $\lambda_t \ge \overline{\lambda_b} F^{-1/s}$ and the current fitness is $i < b$, the probability of an improvement is at least

$$1 - \left(1 - p^{+}_{i,1}\right)^{\lambda_t} \ge 1 - \left(1 - p^{+}_{b-1,1}\right)^{\overline{\lambda_b} F^{-1/s}} \ge 1 - e^{-\overline{\lambda_b} F^{-1/s} \cdot p^{+}_{b-1,1}} = 1 - e^{-C}.$$

If during the first z iterations we have at most $z \cdot \frac{s}{s+1}$ unsuccessful iterations, this implies that at least $z \cdot \frac{1}{s+1}$ iterations are successful. The former steps increase $\log_F(\lambda)$ by $1/s$ each, and the latter steps decrease $\log_F(\lambda)$ by 1 each. In total, we get $\lambda_z \le \lambda_0 \cdot (F^{1/s})^{z \cdot s/(s+1)} \cdot (1/F)^{Z/(s+1)} = \lambda_0$ and thus $\lambda_z \le \lambda_0 \le \overline{\lambda_b}$. We conclude that having at most $z \cdot \frac{s}{s+1}$ unsuccessful iterations among the first z iterations is a sufficient condition for ending the phase within z iterations.

We define independent random variables $\{Y_t\}_{t \ge t^*}$ such that $Y_t \in \{0, 1\}$, $\Pr\left(Y_t = 0\right) = 1 - e^{-C}$ and $\Pr\left(Y_t = 1\right) = e^{-C}$. Denote $Y := \sum_{i=1}^{z} Y_i$ and note that $\mathrm{E}\left(Y\right) = z \cdot e^{-C}$. Using classical Chernoff bounds (see, e. g. Theorem 10.1 in [42]),

$$\begin{aligned}
\Pr\left(Z = z\right) \le \Pr\left(Z \ge z\right) &\le \Pr\left(Y \ge z \cdot \frac{s}{s+1}\right) \\
&= \Pr\left(Y \ge \mathrm{E}\left(Y\right) \cdot \left(\frac{s}{s+1} \cdot e^{C}\right)\right) \\
&\le \left(\frac{e^{\frac{s}{s+1} \cdot e^{C} - 1}}{\left(\frac{s}{s+1} \cdot e^{C}\right)^{\frac{s}{s+1} \cdot e^{C}}}\right)^{z \cdot e^{-C}} \\
&\le \left(\frac{e^{\frac{s}{s+1} \cdot e^{C}}}{\left(\frac{s}{s+1} \cdot e^{C}\right)^{\frac{s}{s+1} \cdot e^{C}}}\right)^{z \cdot e^{-C}} = \left(\left(\frac{s+1}{s} \cdot e^{1-C}\right)^{\frac{s}{s+1}}\right)^{z}.
\end{aligned}$$

By assumption on C this is at most $(F^{-1/s}/2)^z = F^{-z/s} \cdot 2^{-z}$.

[2] Said theorem only holds for values of $F > 1$ that can be chosen arbitrarily small. We generalise the proof to work for arbitrary constant F. This requires the use of a stronger Chernoff bound.

Putting things together,

$$\begin{aligned} \mathrm{E}(T) &= \sum_{z=1}^{\infty} \Pr(Z=z) \cdot \mathrm{E}(T \mid Z=z) \\ &\leq \sum_{z=1}^{\infty} F^{-z/s} \cdot 2^{-z} \cdot \lambda_0 \cdot \frac{F^{\frac{z+1}{s}}}{F^{1/s}-1} \\ &= \lambda_0 \cdot \frac{F^{1/s}}{F^{1/s}-1} \cdot \sum_{z=1}^{\infty} 2^{-z} = \lambda_0 \cdot \frac{F^{1/s}}{F^{1/s}-1}. \end{aligned}$$

□

We now use Lemma 3.8 and Wald's equation to compute the expected number of evaluations spent in a block.

Lemma 3.9 *Consider the self-adjusting* $(1, \lambda)$ *EA as in Theorem 3.5. Starting with a fitness of a and an offspring population size of* λ_0*, the expected number of function evaluations until a fitness of at least b is reached for the first time is at most*

$$E(\lambda_0 + \cdots + \lambda_t \mid \lambda_0) \leq O(\lambda_0) + O(b - a + \log n) \cdot \frac{1}{p^+_{b-1,1}}.$$

Proof We use the variable $\overline{\lambda_b}$ and the definition of a *phase* from the statement of Lemma 3.8. In the first phase, the number of evaluations is bounded by $O(\lambda_0)$ by Lemma 3.8. Afterwards, we either have a fitness of at least b or a λ-value of at most $\overline{\lambda_b}$. In the former case we are done. In the latter case, we apply Lemma 3.8 repeatedly until a fitness of at least b is reached. In every considered phase the expected number of evaluations is at most $O(\overline{\lambda_b}) = O(1/p^+_{b-1,1})$. Note that all these applications of Lemma 3.8 yield a bound that is irrespective of the current fitness and the current offspring population size. Hence these upper bounds can be thought of as independent and identically distributed random variables.

By Lemma 3.6 the expected number of generations to increase the current fitness from a value at least a to a value at least b is $O(b - a + \log n)$. The number of generations is clearly an upper bound for the number of phases required. The previous discussion allows us to apply Wald's equation to conclude that the expected number of evaluations in all phases but the first is bounded by $O(b - a + \log n) \cdot 1/p^+_{b-1,1}$. Together, this implies the claim. □

We note that for $b - a = \log n$ the bound given by Lemma 3.9 depends on the initial offspring population size λ_0, the gap of fitness to traverse $b - a$ and the probability of finding an improvement at fitness value $b - 1$. If we could ensure that λ is sufficiently small at the start of the optimisation of every block we could easily compute the total expected optimisation. Unfortunately, the previous lemmas allow for the value of λ at the end of a block and hence at the start of a new block to be any large value. We solve this in Lemma 3.10 by denoting a generation where λ is excessively large as an *excessive* generation. In the proof of Lemma 3.10 we show that with high probability the algorithm finds the optimum without having an excessive generation. Hence, the expected number of evaluations needed for the algorithm to either find the optimum or have an excessive generation is asymptotically the same as the runtime of the algorithm.

Lemma 3.10 *Call a generation t* excessive *if, for a current search point with fitness i, at the end of the generation λ is increased beyond $5F^{1/s}\ln(n)/p^+_{i,1}$. Let T denote the expected number of function evaluations before a global optimum is found. Let $\overline{T}$ denote the number of evaluations made before a global optimum is found or until the end of the first excessive generation. Then*

$$E(T) \le E\left(\overline{T}\right) + O(1).$$

Proof The proof uses different thresholds for increasingly "excessive" values of λ and we are numbering the corresponding variables for the number of evaluations and generations, respectively. Let $T^{(1)} = \overline{T}$ and let $G^{(1)}$ be the number of *generations* until a global optimum or an excessive generation is encountered. We denote the former event by $B^{(1)}$, that is, the event that an optimum is found before an excessive generation. Let $T^{(2)}$ denote the worst-case number of function evaluations made until λ exceeds $\lambda^{(2)} := F^{1/s}n^3$ or the optimum is found, when starting with a worst possible initial fitness and offspring population size $\lambda \le \lambda^{(2)}$. Let $B^{(2)}$ denote the latter event and let $G^{(2)}$ be the corresponding number of generations. Let $T^{(3)}$ denote the worst-case number of evaluations until the optimum is found, when starting with a worst possible population size $\lambda \le \lambda^{(2)}F^{1/s}$. Then the expected optimisation time is bounded as follows.

$$\mathrm{E}(T) \le \mathrm{E}\left(T^{(1)}\right) + \Pr\left(\overline{B^{(1)}}\right)\left(\mathrm{E}\left(T^{(2)}\right) + \Pr\left(\overline{B^{(2)}}\right)\cdot \mathrm{E}\left(T^{(3)}\right)\right).$$

Note that $T^{(1)} \le T$ since $T^{(1)}$ is a stopping time defined with additional opportunities for stopping, thus the number of generations $G^{(1)}$ for finding the optimum or exceeding $\lambda^{(1)}$ satisfies $\mathrm{E}\left(G^{(1)}\right) = O(n)$ by Lemma 3.6.

In every generation $t \le G^{(1)}$ with offspring population size λ_t and current fitness i, if $\lambda_t \le 5\ln(n)/p^+_{i,1}$, we have $\lambda_{t+1} \le 5F^{1/s}\ln(n)/p^+_{i,1}$ with probability 1, that is, the generation is not excessive. If $5\ln(n)/p^+_{i,1} < \lambda_t \le 5F^{1/s}\ln(n)/p^+_{i,1}$, we have an excessive generation with probability at most

$$(1-p^+_{i,1})^{\lambda_t} \le (1-p^+_{i,1})^{5\ln(n)/p^+_{i,1}} \le e^{-5\ln(n)} = n^{-5}.$$

Thus, the probability of having an excessive generation in the first $G^{(1)}$ generations is bounded, using a union bound, by

$$\Pr\left(\overline{B^{(1)}}\right) \le \sum_{t=1}^{\infty} t \cdot n^{-5} \cdot \Pr\left(G^{(1)} = t\right) = n^{-5} \cdot \mathrm{E}\left(G^{(1)}\right) = O(n^{-4}).$$

We also have $\mathrm{E}\left(G^{(2)}\right) = O(n)$ by Lemma 3.6 (this bound applies for all initial fitness values and all initial offspring population sizes). In all such generations $t \leq G^{(2)}$, we have $\lambda_t \leq \lambda^{(2)}$, thus $\mathrm{E}\left(T^{(2)}\right) \leq \mathrm{E}\left(G^{(2)}\right) \cdot \lambda^{(2)} = O(n^4)$.

As per the above arguments, the probability of exceeding $\lambda^{(2)}$ is either 0 (for $\lambda_t \leq \lambda^{(2)} F^{-1/s}$) or (for $\lambda^{(2)} F^{-1/s} < \lambda_t \leq \lambda^{(2)}$) bounded by

$$(1 - p_{i,1}^{+})^{\lambda_t} \leq (1 - p_{n-1,1}^{+})^{\lambda^{(2)} F^{-1/s}} \leq e^{-\Omega(n^2)}.$$

Taking a union bound,

$$\Pr\left(\overline{B^{(2)}}\right) \leq \sum_{t=1}^{\infty} t \cdot e^{-\Omega(n^2)} \cdot \Pr\left(G^{(2)} = t\right) = e^{-\Omega(n^2)} \cdot \mathrm{E}\left(G^{(2)}\right) = e^{-\Omega(n^2)}.$$

Finally, we bound $\mathrm{E}\left(T^{(3)}\right) \leq n^n$ using the trivial argument that a global optimum is created with every standard bit mutation with probability at least $(1/n)^n$. Putting this together yields

$$\mathrm{E}(T) \leq \mathrm{E}\left(T^{(1)}\right) + O(n^{-4})\left(O(n^4) + e^{-\Omega(n^2)} \cdot n^n\right) = \mathrm{E}\left(T^{(1)}\right) + O(1).$$

□

Owing to Lemma 3.10 we can compute $\mathrm{E}\left(\overline{T}\right)$ without worrying about large values of λ and at the same time obtain the desired bound on the total expected number of evaluations to find the optimum.

Proof of Theorem 3.5 By Lemma 3.10, it suffices to bound $\mathrm{E}\left(\overline{T}\right)$ from above. In particular, we can assume that no generations are excessive as otherwise we are done.

We divide the distance to the optimum in *blocks* of length $\log n$ and use this to divide the run into *epochs*. For $i \in \{0, \dots, \lceil n/\log n \rceil - 1\}$, epoch i starts in the first generation in which the current search point has a fitness of at least $n - (i+1)\log n$ is reached and it ends as soon as a search point of fitness at least $n - i \log n$ is found. Let T_i denote the number of evaluations made during epoch i. Note that after epoch i, once a fitness of at least $n - i \log n$ has been reached, the algorithm will continue with epoch $i - 1$ (or an epoch with an even smaller index, in the unlikely event that a whole block is skipped) and the goal of epoch 0 implies that the global optimum is found. Consequently, the total expected number of evaluations is bounded by $\sum_{i=0}^{\lceil n/\log n \rceil - 1} \mathrm{E}(T_i)$.

Let $\lambda^{(i)}$ denote the offspring population size at the start of epoch i. Applying Lemma 3.9 with $a := n - (i+1)\log n$, $b := n - i \log n$ and $\lambda_0 := \lambda^{(i)}$,

$$\mathrm{E}(T_i) \leq O(\lambda^{(i)}) + O(\log n) \cdot \frac{1}{p_{n-i\log(n)-1,1}^{+}}.$$

Since we assume that no generation is excessive and the fitness is bounded by $n - i\log(n) - 1$ throughout the epoch, we have $\lambda^{(i)} \leq 5F^{1/s}\ln(n)/p^{+}_{n-i\log(n)-1,1}$. Plugging this in, we get

$$\mathrm{E}(T_i) \leq O(\log n) \cdot \frac{1}{p^{+}_{n-i\log(n)-1,1}} \leq O(\log n) \cdot \frac{en}{1+i\log n} = O(n\log n) \cdot \frac{1}{1+i\log n}.$$

Then the expected optimisation time is bounded by

$$\begin{aligned}\sum_{i=0}^{\lceil n/\log n\rceil -1} \mathrm{E}(T_i) &\leq O(n\log n) \cdot \sum_{i=0}^{\lceil n/\log n\rceil -1} \frac{1}{1+i\log n}\\ &\leq O(n\log n) \cdot \left(1 + \sum_{i=1}^{\lceil n/\log n\rceil -1} \frac{1}{i\log n}\right)\\ &= O(n\log n) \cdot \left(1 + \frac{H_{\lceil n/\log n\rceil -1}}{\log n}\right) = O(n\log n)\end{aligned}$$

using $H_{\lceil n/\log n\rceil -1} \leq H_n \leq \ln(n) + 1$ in the last step. □

4 Large Success Rates Fail

In this section, we show that the choice of the success rate is crucial as when s is a large constant, the runtime becomes exponential.

Theorem 4.1 *Let the update strength $F \leq 1.5$ and the success rate $s \geq 18$ be constants. With probability $1 - e^{-\Omega(n/\log^4 n)}$ the self-adjusting $(1, \lambda)$ EA needs at least $e^{\Omega(n/\log^4 n)}$ evaluations to optimise* OneMax.

The reason why the algorithm takes exponential time is that now $F^{1/s}$ is small and λ only increases slowly in unsuccessful generations, whereas successful generations decrease λ by a much larger factor of F. This is detrimental during early parts of the run where it is *easy* to find improvements and there are frequent improvements that decrease λ. When λ is small, there are frequent fallbacks, hence the algorithm stays in a region with small values of λ, where it finds improvements with constant probability, but also has fallbacks with constant probability. We show, using another potential function based on Definition 2.7, that it takes exponential time to escape from this equilibrium.

Definition 4.2 We define the potential function $g_2(X_t)$ as

$$g_2(X_t) := \mathrm{OM}(x_t) + 2.2\log_F^2 \lambda_t.$$

While $g_1(X_t)$ used a (capped) linear contribution of $\log_F(\lambda_t)$ for $h(\lambda_t)$, here we use the function $h(\lambda_t) := 2.2\log_F^2(\lambda_t)$ that is convex in $\log_F(\lambda_t)$, so that changes in λ_t

have a larger impact on the potential. We show that, in a given fitness interval, the potential $g_2(X_t)$ has a negative drift.

Lemma 4.3 *Consider the self-adjusting* $(1, \lambda)$ *EA as in Theorem* 4.1. *Then there is a constant* $\delta > 0$ *such that for every* $0.84n + 2.2\log^2(4.5) < g_2(X_t) < 0.85n$,

$$E\left(g_2(X_{t+1}) - g_2(X_t) \mid X_t\right) \leq -\delta.$$

Proof We abbreviate $\Delta_{g2} := \mathrm{E}\left(g_2(X_{t+1}) - g_2(X_t) \mid X_t\right)$. Given that for all $\lambda \geq 1$

$$h(\lambda/F) = 2.2\log_F^2(\lambda/F) = 2.2\left(\log_F(\lambda) - 1\right)^2 \geq 0 = h(1)$$

then by Lemma 2.8, for all λ, Δ_{g2} is at most

$$\begin{aligned}
&\left(\Delta^+_{i,\lambda} + h(\lambda/F) - h(\lambda F^{1/s})\right) p^+_{i,\lambda} + h(\lambda F^{1/s}) - h(\lambda) - \Delta^-_{i,\lambda} p^-_{i,\lambda}\\
&= \left(\Delta^+_{i,\lambda} + 2.2\log_F^2(\lambda/F) - 2.2\log_F^2(\lambda F^{1/s})\right) p^+_{i,\lambda}\\
&\quad + 2.2\log_F^2(\lambda F^{1/s}) - 2.2\log_F^2(\lambda) - \Delta^-_{i,\lambda} p^-_{i,\lambda}\\
&= \left(\Delta^+_{i,\lambda} + 2.2(\log_F(\lambda) - 1)^2 - 2.2(\log_F(\lambda) + 1/s)^2\right) p^+_{i,\lambda}\\
&\quad + 2.2(\log_F(\lambda) + 1/s)^2 - 2.2\log_F^2(\lambda) - \Delta^-_{i,\lambda} p^-_{i,\lambda}\\
&= \left(\Delta^+_{i,\lambda} - \left(1 + \frac{1}{s}\right) \cdot 4.4\log_F(\lambda) + 2.2 - \frac{2.2}{s^2}\right) p^+_{i,\lambda}\\
&\quad + \frac{4.4\log_F \lambda}{s} + \frac{2.2}{s^2} - \Delta^-_{i,\lambda} p^-_{i,\lambda}.
\end{aligned}$$

The terms containing the success rate s add up to

$$(1 - p^+_{i,\lambda})\left(\frac{4.4\log_F \lambda}{s} + \frac{2.2}{s^2}\right).$$

This is non-increasing in s, thus we bound s by the assumption $s \geq 18$, obtaining

$$\Delta_{g2} \leq \left(\Delta^+_{i,\lambda} - \frac{19}{18} \cdot 4.4\log_F \lambda + 2.2 - \frac{2.2}{324}\right) p^+_{i,\lambda} + \frac{4.4\log_F \lambda}{18} + \frac{2.2}{324} - \Delta^-_{i,\lambda} p^-_{i,\lambda}. \tag{13}$$

We note that in Eq. (13), $\lambda \in \mathbb{R}_{\geq 1}$ but since the algorithm creates $\lfloor \lambda \rceil$ offspring, the forward drift and the probabilities are calculated using $\lfloor \lambda \rceil$. In the following in all the computations the last digit is rounded up if the value was positive and down otherwise to ensure the inequalities hold. We start taking into account only $\lfloor \lambda \rceil \geq 5$, that is, $\lambda \geq 4.5$ and later on we will deal with smaller values of λ. With this constraint on λ we use the simple bound $p^-_{i,\lambda} \geq 0$. Bounding $\Delta^+_{i,\lambda} \leq \lceil \log \lambda \rceil + 0.413$ using Eq. (5) in

Lemma 2.2,

$$\Delta_{g_2} \le \left(2.613 + \lceil \log \lambda \rceil - \frac{19}{18} \cdot 4.4 \log_F \lambda - \frac{2.2}{324}\right) p^+_{i,\lambda} + \frac{4.4 \log_F \lambda}{18} + \frac{2.2}{324}.$$

For all $\lambda \ge 1$, $\mathrm{OM}(x_t) \ge 0.85n$ implies $g_2(X_t) \ge 0.85n$. By contraposition, our precondition $g_2(X_t) < 0.85n$ implies $\mathrm{OM}(x_t) < 0.85n$. Therefore, using Eq. (1) in Lemma 2.2 with the worst case $\mathrm{OM}(x_t) = 0.85n$ and $\lfloor \lambda \rceil = 5$ we get $p^+_{i,\lambda} \ge 1 - \frac{e}{e+0.15\lfloor\lambda\rceil} \ge 1 - \frac{e}{e+5\cdot 0.15} > 0.216$. Substituting these bounds we obtain

$$\begin{aligned}\Delta_{g_2} &\le \left(2.613 + \lceil \log \lambda \rceil - \frac{19}{18} \cdot 4.4 \log_F \lambda - \frac{2.2}{324}\right) 0.216 + \frac{4.4 \log_F \lambda}{18} + \frac{2.2}{324} \\ &\le 0.5562 + 0.216\lceil \log \lambda \rceil - 0.7587 \log_F \lambda\end{aligned}$$

The assumption $F \le 1.5$ implies that $\log_F \lambda \ge \log \lambda$. Using this and $\lceil \log \lambda \rceil \le \log(\lambda) + 1$ yields

$$\begin{aligned}\Delta_{g_2} &\le 0.5562 + 0.216(\log(\lambda) + 1) - 0.7587 \log \lambda \\ &= 0.7722 - 0.5427 \log \lambda \\ &\le 0.7722 - 0.5427 \log 4.5 \le -0.4054.\end{aligned}$$

Up until now we have proved that $\Delta_{g_2} \le -0.4058$ for all $\mathrm{OM}(x_t) < 0.85n$ and $\lfloor \lambda \rceil \ge 5$. Now we need to consider $\lfloor \lambda \rceil < 5$. For $\lfloor \lambda \rceil < 5$, that is, $\lambda < 4.5$, the precondition $g_2(X_t) > 0.84n + 2.2 \log^2(4.5)$ implies that $\mathrm{OM}(x_t) > 0.84n$. Therefore, the last part of this proof focuses only on $0.84n < \mathrm{OM}(x_t) < 0.85n$ and $\lfloor \lambda \rceil < 5$. For this region we use again Eq. (13), but bound it in a more careful way now. By Eq. (3) in Lemma 2.2, $p^-_{i,\lambda} \ge \left(\frac{\mathrm{OM}(x_t)}{n} - \frac{1}{e}\right)^{\lfloor\lambda\rceil} \ge \left(0.84 - \frac{1}{e}\right)^{\lfloor\lambda\rceil}$ and bounding $\Delta^+_{i,\lambda}$ and $\Delta^-_{i,\lambda}$ using Eqs. (5) and (4) in Lemma 2.2 yields:

$$\begin{aligned}\Delta_{g_2} \;\le\; &\underbrace{\left(\sum_{j=1}^{\infty}\left(1-\left(1-\frac{1}{j!}\right)^{\lfloor\lambda\rceil}\right) - \frac{19}{18}\cdot 4.4\log_F \lambda + 2.2 - \frac{2.2}{324}\right)}_{\alpha} p^+_{i,\lambda} \\ &+ \frac{4.4 \log_F \lambda}{18} + \frac{2.2}{324} - \left(0.84 - \frac{1}{e}\right)^{\lfloor\lambda\rceil}. \qquad (14)\end{aligned}$$

We did not bound $p^+_{i,\lambda}$ in the first term yet because the factor α in brackets preceding it can be positive or negative. We now calculate precise values for $\sum_{j=1}^{\infty}\left(1-\left(1-\frac{1}{j!}\right)^{\lfloor\lambda\rceil}\right)$ giving $e-1, 2.157, 2.4458$ and 2.6511 for $\lfloor\lambda\rceil = 1, 2, 3, 4$, respectively. Given that $F \leq 1.5$ the factor α is negative for all $1.5 \leq \lambda < 4.5$, because

$$\begin{aligned}
&\left(\sum_{j=1}^{\infty}\left(1-\left(1-\frac{1}{j!}\right)^{\lfloor\lambda\rceil}\right)-\frac{19}{18}\cdot 4.4\log_F\lambda+2.2-\frac{2.2}{324}\right)\\
&\leq\left(\sum_{j=1}^{\infty}\left(1-\left(1-\frac{1}{j!}\right)^{\lfloor\lambda\rceil}\right)-4.4\log_F(\lambda)+2.2\right)\\
&\leq\begin{cases}4.8511-4.4\log_{1.5}(3.5)=-8.74 & 3.5\leq\lambda<4.5\\ 4.6458-4.4\log_{1.5}(2.5)=-5.29 & 2.5\leq\lambda<3.5\\ 4.357-4.4\log_{1.5}(1.5)=-0.043 & 1.5\leq\lambda<2.5\end{cases}
\end{aligned}$$

On the other hand, for $\lambda < 1.5$ and $\lfloor\lambda\rceil = 1$, α is positive when $\lambda < F^{\gamma}$ for $\gamma = \frac{1933}{7524} + \frac{45e}{209} \approx 0.8422$ and negative otherwise. With this we evaluate different ranges of λ separately using Eq. (14). For $1 \leq \lambda < F^{\gamma}$, we get $\lfloor\lambda\rceil = 1$ and by Lemma 2.2 if $0.84n \leq i \leq 0.85n$ and $n \geq 163$ then $p^{+}_{i,\lambda} \leq 0.069$, thus

$$\begin{aligned}
\Delta_{g2} &\leq \left(e+1.2-\frac{19}{18}\cdot 4.4\log_F(\lambda)-\frac{2.2}{324}\right)0.069+\frac{4.4}{18}\log_F(\lambda)\\
&\quad+\frac{2.2}{324}-\left(0.84-\frac{1}{e}\right)\\
&\leq -0.076\log_F(\lambda)-0.195\leq -0.195.
\end{aligned}$$

For $F^{\gamma} \leq \lambda < 1.5$, by Lemma 2.2 we bound $p^{+}_{i,\lambda} \geq \frac{n-\mathrm{OM}(x_t)}{en} \geq 0.0551$:

$$\begin{aligned}
\Delta_{g2} &\leq \left(e+1.2-\frac{19}{18}\cdot 4.4\log_F(\lambda)-\frac{2.2}{324}\right)0.0551+\frac{4.4}{18}\log_F(\lambda)\\
&\quad+\frac{2.2}{324}-\left(0.84-\frac{1}{e}\right)\\
&\leq -0.0114\log_F(\lambda)-0.2498\leq -0.2498.
\end{aligned}$$

For $1.5 \leq \lambda < 2.5$, by Lemma 2.2 we bound $p^{+}_{i,\lambda} \geq 1-\frac{e}{e+0.3} \geq 0.0993$

$$\begin{aligned}
\Delta_{g2} &\leq \left(4.357-\frac{19}{18}\cdot 4.4\log_F(\lambda)-\frac{2.2}{324}\right)0.0993+\frac{4.4}{18}\log_F(\lambda)\\
&\quad+\frac{2.2}{324}-\left(0.84-\frac{1}{e}\right)^2\\
&\leq 0.2159-0.2167\log_F(\lambda)\\
&\leq 0.2159-0.2167\log_{1.5}(1.5)\leq -0.0008.
\end{aligned}$$

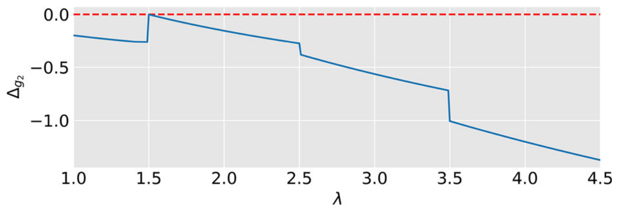

Fig. 1 Bounds on Δ_{g2} with a maximum of -0.0008 for $\lambda = 1.5$

For $2.5 \leq \lambda < 3.5$ we use $p^+_{i,\lambda} \geq 1 - \frac{e}{e+0.45} = 0.142$,

$$\begin{aligned}\Delta_{g2} &\leq \left(4.6458 - \frac{19}{18} \cdot 4.4 \log_F(\lambda) - \frac{2.2}{324}\right) 0.142 + \frac{4.4}{18} \log_F(\lambda) \\ &\quad + \frac{2.2}{324} - \left(0.84 - \frac{1}{e}\right)^3 \\ &\leq 0.5612 - 0.415 \log_F(\lambda) \\ &\leq 0.5612 - 0.415 \log_{1.5}(2.5) \leq -0.376.\end{aligned}$$

Finally for $3.5 \leq \lambda < 4.5$ we use $p^+_{i,\lambda} \geq 1 - \frac{e}{e+0.6} = 0.1808$,

$$\begin{aligned}\Delta_{g2} &\leq \left(4.8511 - \frac{19}{18} \cdot 4.4 \log_F(\lambda) - \frac{2.2}{324}\right) 0.1808 + \frac{4.4}{18} \log_F(\lambda) \\ &\quad + \frac{2.2}{324} - \left(0.84 - \frac{1}{e}\right)^4 \\ &\leq 0.832 - 0.5952 \log_F(\lambda) \\ &\leq 0.832 - 0.5952 \log_{1.5}(3.5) \leq -1.006.\end{aligned}$$

With these results we can see that the potential is negative with $\lambda \in [1, 4.5)$ and $0.84n < \mathrm{OM}(x_t) < 0.85n$. Hence, for every $0.84n + \log^2(4.5) < g_2(X_t) < 0.85n$, and $\delta = 0.0008$, $\Delta_{g_2} \leq -\delta$. □

Finally, with Lemmas 4.3 and 2.3, we now prove Theorem 4.1.

Proof of Theorem 4.1 We apply the negative drift theorem with scaling (Theorem 2.6). We switch to the potential function $\overline{h}(X_t) := \max\{0, n - g_2(X_t)\}$ in order to fit the perspective of the negative drift theorem. In this case we can pessimistically assume that if $\overline{h}(X_t) = 0$ the optimum has been found.

The first condition of the negative drift theorem with scaling (Theorem 2.6) can be established with Lemma 4.3 for $a = 0.15n$ and $b = 0.16n - 2.2 \log^2(4.5)$. Furthermore, with Chernoff bounds we can prove that at initialization $\overline{h}(X_t) \geq b$ with probability $1 - 2^{-\Omega(n)}$.

To prove the second condition we need to show that the probability of large jumps is small. Starting with the contribution that λ makes to the change in $\overline{h}(X_t)$, we use Lemma 2.3 to show that this contribution is at most $2.2 \log^2(eF^{1/s}n^3) \leq 4.79 +$

$19.8 \log^2 n \leq 20 \log^2 n$ with probability $\exp(-\Omega(n^2))$, where the last inequality holds for large enough n.

The only other contributor is the change in fitness. The probability of a jump in fitness away from the optimum is maximised when there is only one offspring. On the other hand the bigger the offspring population the higher the probability of a large jump towards the optimum. Taking this into account and pessimistically assuming that every bit flip either decreases the fitness in the first case or increases the fitness in the latter we get the following probabilities. Recalling (9),

$$\Pr\left(\mathrm{OM}(x_t) - \mathrm{OM}(x_{t+1}) \geq \kappa\right) \leq \frac{1}{\kappa!}$$
$$\Pr\left(\mathrm{OM}(x_{t+1}) - \mathrm{OM}(x_t) \geq \kappa\right) \leq 1 - \left(1 - \frac{1}{\kappa!}\right)^{\lambda}$$

Given that $\frac{1}{\kappa!} \leq 1 - \left(1 - \frac{1}{\kappa!}\right)^{\lambda}$, and that $\lambda \leq eF^{1/s}n^3$.

$$\begin{aligned}\Pr\left(|\mathrm{OM}(x_{t+1}) - \mathrm{OM}(x_t)| \geq \kappa\right) &\leq 1 - \left(1 - \frac{1}{\kappa!}\right)^{eF^{1/s}n^3}\\ &\leq \frac{eF^{1/s}n^3}{\kappa!}\\ &\leq \frac{e^{\kappa+1}F^{1/s}n^3}{\kappa^{\kappa}}\end{aligned}$$

Joining both contributions, we get

$$\Pr\left(|g_2(X_{t+1}) - g_2(X_t)| \geq \kappa + 20\log^2 n\right) \leq \frac{e^{\kappa+1}F^{1/s}n^3}{\kappa^{\kappa}}. \tag{15}$$

To satisfy the second condition of the negative drift theorem with scaling (Theorem 2.6) we use $r = 21\log^2 n$ and $\kappa = j\log^2 n$ in order to have $\kappa + 20\log^2 n \leq jr$ for $j \in \mathbb{N}$. For $j = 0$ the condition $\Pr\left(|g_2(X_{t+1}) - g_2(X_t)| \geq jr\right) \leq e^0$ is trivial. From Eq. (15), we obtain

$$\Pr\left(|g_2(X_{t+1}) - g_2(X_t)| \geq jr\right) \leq \frac{eF^{1/s}e^{(j\log^2 n)}n^3}{(j\log^2 n)^{j\log^2 n}}$$

We simplify the numerator using

$$e^{(j\log^2 n)}n^3 = e^{(j\log(n)\ln(n)/\ln(2))}n^3 = n^{(3+j\log(n)/\ln(2))}$$

and bound the denominator as

$$(j\log^2 n)^{j\log^2 n} \geq (\log n)^{2j\log^2 n} = n^{2j\log(n)\log\log(n)},$$

yielding

$$\Pr\left(|g_2(X_{t+1}) - g_2(X_t)| \geq jr\right) \leq eF^{1/s} n^{(3+j\log(n)/\ln(2)-2j(\log n)\log\log n)}$$
$$= eF^{1/s} n^{(3+j(\log(n)/\ln(2)-2(\log n)\log\log n))}.$$

For $n \geq 7$, $\log(n)/\ln(2) - 2(\log n)\log\log n \leq -4$, hence

$$\Pr\left(|g_2(X_{t+1}) - g_2(X_t)| \geq jr\right) \leq eF^{1/s} n^{3-4j}.$$

which for large enough n is bounded by e^{-j} as desired.

The third condition is met with $r = 21\log^2 n$ given that $\delta\ell/(132\log((21\log^2 n)/\delta)) = \Theta(n/\log\log n)$, which is larger than $r^2 = \Theta(\log^4 n)$ for large enough n.

With this we have proved that the algorithm needs at least $e^{\Omega(n/\log^4 n)}$ generations with probability $1 - e^{-\Omega(n/\log^4 n)}$. Since each generation uses at least one fitness evaluation, the claim is proved. □

We note that although Theorem 4.1 is applied for OneMax specifically, the conditions used in the proof of Theorem 4.1 and Lemma 4.3 apply for several other benchmark functions. This is because our result only depends on some fitness levels of OneMax and other functions have fitness levels that are symmetrical or resemble these fitness levels. We show this in the following theorem. To improve readability we use $|x|_1 := \sum_{i=1}^n x_i$ and $|x|_0 := \sum_{i=1}^n (1 - x_i)$.

Theorem 4.4 *Let the update strength $F \leq 1.5$ and the success rate $s \geq 18$ be constants. With probability $1 - e^{-\Omega(n/\log^4 n)}$ the self-adjusting $(1, \lambda)$ EA needs at least $e^{\Omega(n/\log^4 n)}$ evaluations to optimise:*

- $\text{Jump}_k(x) := \begin{cases} n - |x|_1 & \textit{if } n - k < |x|_1 < n, \\ k + |x|_1 & \textit{otherwise}, \end{cases}$
  with $k = o(n)$,
- $\text{Cliff}_d(x) := \begin{cases} |x|_1 & \textit{if } |x|_1 \leq d, \\ |x|_1 - d + 1/2 & \textit{otherwise}, \end{cases}$
  with $d = o(n)$,
- $\text{ZeroMax}(x) := |x|_0$,
- $\text{TwoMax}(x) := \max\{|x|_1, |x|_0\}$,
- $\text{Ridge}(x) := \begin{cases} n + |x|_1 & \textit{if } x = 1^i 0^{n-i}, i \in \{0, 1, \ldots, n\}, \\ |x|_0 & \textit{otherwise}. \end{cases}$

Proof For Jump_k and Cliff_d, given that k and d are $o(n)$ the algorithm needs to optimise a OneMax-like slope with the same transition probabilities as in Lemma 4.3 before the algorithm reaches the local optima. Hence, we can apply the negative drift theorem with scaling (Theorem 2.6) as in Theorem 4.1 to prove the statement.

For ZeroMax the algorithm will behave exactly as in OneMax, because it is unbiased towards bit-values. Similarly, for TwoMax, independently of the slope the algorithm is optimising, it needs to traverse through a OneMax-like slope needing at least the same number of function evaluations as in OneMax.

Finally, for Ridge, unless the algorithm finds a search point on the ridge ($x \in 1^i 0^{n-i}$ with $i \in \{0, 1, \dots, n\}$) beforehand, the first part of the optimisation behaves as ZeroMax and similar to Theorem 4.1 by Lemma 4.3 and the negative drift theorem with scaling (Theorem 2.6) it will need at least $e^{Cn/\log^4 n}$ generations with probability $e^{-Cn/\log^4 n}$ to reach a point with $|x|_1 \le 0.15n$ for some constant $C > 0$.

It remains to show that the ridge is not reached during this time, with high probability. We first imagine the algorithm optimising ZeroMax and note that the behaviour on Ridge and ZeroMax is identical as long as no point on the ridge is discovered. Let $x_0, x_1, \dots$ be the search points created by the algorithm on ZeroMax in order of creation. Since ZeroMax is symmetric with respect to bit positions, for any arbitrary but fixed t we may assume that the search point x_t with $d = |x_t|_1$ is chosen uniformly at random from the $\binom{n}{d}$ search points that have exactly d 1-bits. There is only one search point $1^d 0^{n-d}$ that on the function Ridge would be part of the ridge. Thus, for $d \ge 0.15n$ the probability that x_t lies on the ridge is at most

$$\binom{n}{d}^{-1} \le \binom{n}{0.15n}^{-1} \le \left(\frac{n}{0.15n}\right)^{-0.15n} = \left(\frac{20}{3}\right)^{-0.15n}.$$

(Note that these events for t and t' are not independent; we will resort to a union bound to deal with such dependencies.) By Lemma 2.3, during the optimisation of any unimodal function every generation uses $\lambda \le eF^{1/s}n^3$ with probability $1 - \exp(-\Omega(n^2))$. By a union bound over $e^{Cn/\log^4 n}$ generations, for an arbitrary constant $C > 0$, each generation creating at most $eF^{1/s}n^3$ offspring, the probability that a point on the ridge is reached during this time is at most

$$e^{Cn/\log^4 n} \cdot eF^{1/s}n^3 \cdot \left(\frac{20}{3}\right)^{-0.15n} = e^{-\Omega(n)}.$$

Adding up all failure probabilities, the algorithm will not create a point on the ridge before $e^{Cn/\log^4 n}$ generations have passed with probability $1 - e^{-\Omega(n/\log^4 n)}$, and the algorithm needs at least $e^{\Omega(n/\log^4 n)}$ evaluations to solve Ridge with probability $1 - e^{-\Omega(n/\log^4 n)}$. □

5 Experiments

Due to the complex nature of our analyses there are still open questions about the behaviour of the algorithm. In this section we show some elementary experiments to enhance our understanding of the parameter control mechanism and address these unknowns. All the experiments were performed using the IOHProfiler [43].

In Sect. 3 we have shown that both the self-adjusting $(1, \lambda)$ EA and the self-adjusting $(1 + \lambda)$ EA have an asymptotic runtime of $O(n \log n)$ evaluations on

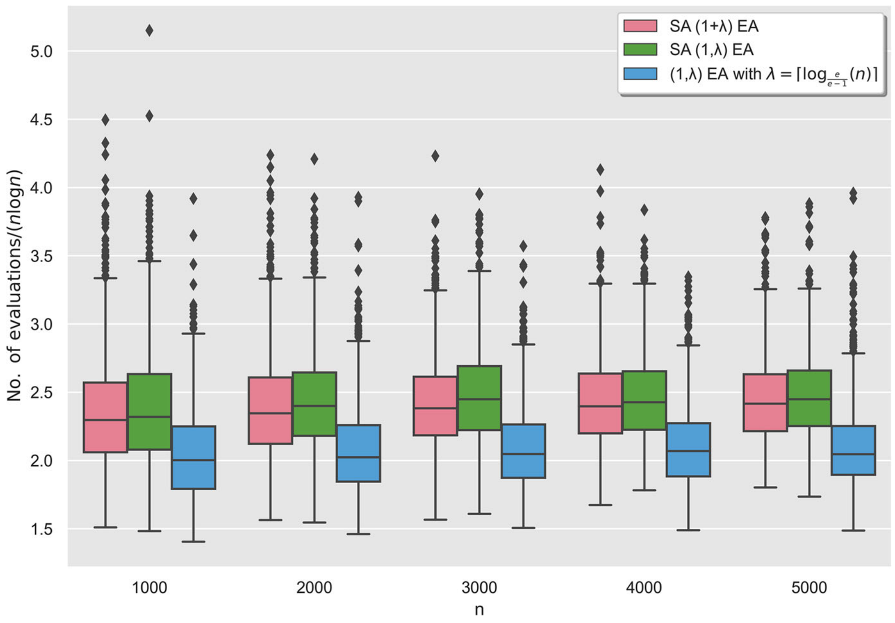

Fig. 2 Box plots of the number of evaluations used by the self-adjusting $(1, \lambda)$ EA, the self-adjusting $(1+\lambda)$ EA with $s = 1$, $F = 1.5$ and the $(1, \lambda)$ EA over 1000 runs for different n on OneMax. The number of evaluations is normalised by $n \log n$

OneMax. This is the same asymptotic runtime as the $(1, \lambda)$ EA with static parameters $\lambda = \lceil \log_{\frac{e}{e-1}}(n) \rceil$ [22]. We remark that very recently the conditions for efficient offspring population sizes have been relaxed to $\lambda \geq \lceil \log_{\frac{e}{e-1}}(cn/\lambda) \rceil$ for any constant $c > e^2$ [44]. However, this only reduces the best known value of λ by 1 or 2 for the considered problem sizes, and so we stick to the simpler formula of $\lambda = \lceil \log_{\frac{e}{e-1}}(n) \rceil$, i. e. the best static parameter value reported in [22]. Unfortunately the asymptotic notation may hide large constants, therefore, our first experiments focus on the comparison of these three algorithms on OneMax.

Figure 2 displays box plots of the number of evaluations over 1000 runs for different problem sizes on OneMax. From Fig. 2 we observe that the difference between both self-adjusting algorithms is relatively small. This indicates that there are only a small number of fallbacks in fitness and such fallbacks are also small. We also observe that the best static parameter choice from [22] is only a small constant factor faster than the self-adjusting algorithms.

In the results of Sects. 3 and 4 there is a gap between $s < 1$ and $s \geq 18$ where we do not know how the algorithm behaves on OneMax. In our second experiment, we explore how the algorithm behaves in this region by running the self-adjusting $(1, \lambda)$ EA on OneMax using different values for s shown in Fig. 3. All runs were stopped once the optimum was found or after $500n$ generations were reached. We found a sharp threshold at $s \approx 3.4$, indicating that the widely used one-fifth rule ($s = 4$) is inefficient here, but other success rules achieve optimal asymptotic runtime.

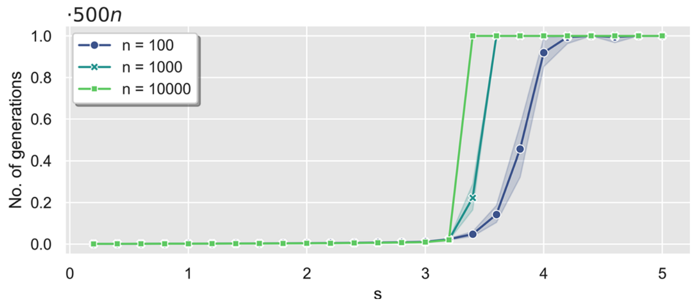

Fig. 3 Average number of generations with 99% bootstrapped confidence interval of the self-adjusting $(1,\lambda)$ EA with $F = 1.5$ in 100 runs for different n, normalised and capped at $500n$ generations

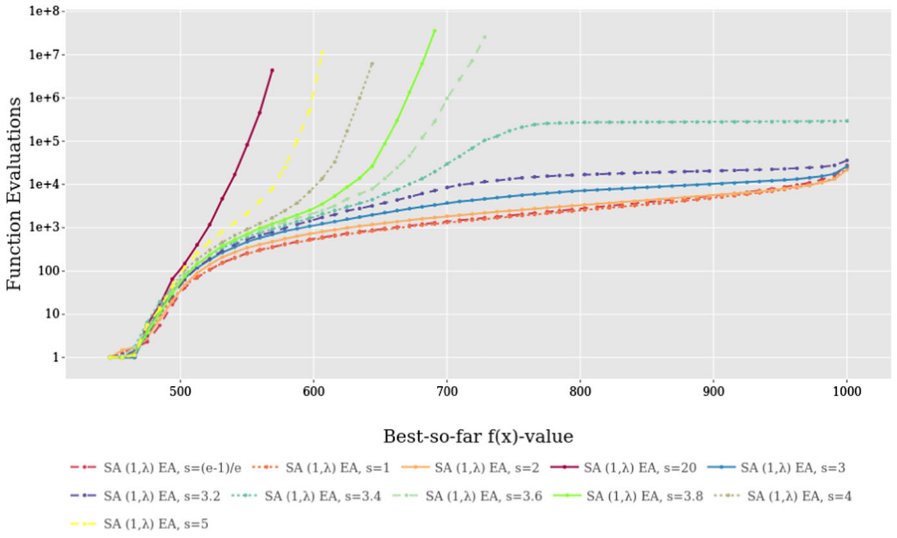

Fig. 4 Fixed target results for the self-adjusting $(1,\lambda)$ EA on OneMax with $n = 1000$ (100 runs)

Additionally, in Fig. 4 we plot fixed target results, that is, the average time to reach a certain fitness, for $n = 1000$ for different s. All runs were stopped once the optimum was found or after $500n$ generations. No points are graphed for fitness values that were not reached during the allocated time. We note that the plots do not start exactly at $n/2 = 500$; this is due to the random effects of initialisation. From here we found that the range of fitness values with negative drift is wider than what we where able to prove in Sect. 4. Already for $s = 3.4$, there is an interval on the scale of number of ones around $0.7n$ where the algorithm spends a large amount of evaluations to traverse this interval. Interestingly, as s increases the algorithm takes longer to reach points farther away from the optimum.

We also explored how the parameter λ behaves throughout the optimisation depending on the value of s. In Fig. 5 we can see the average λ at every fitness value for $n = 1000$. As expected, on average λ is larger when s is smaller. For $s \geq 3$ we can

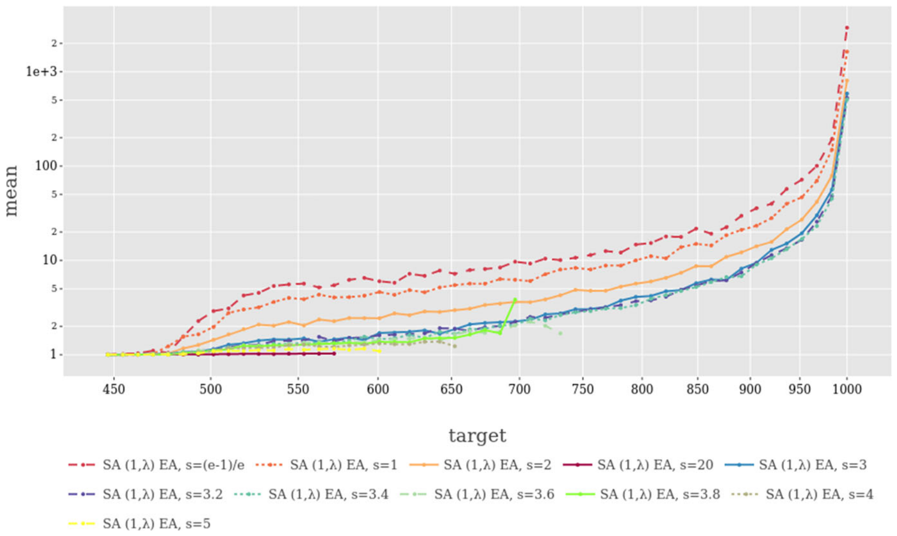

Fig. 5 Average λ values for each fitness level of the self-adjusting $(1, \lambda)$ EA on OneMax with $n = 1000$ (100 runs)

appreciate that on average $\lambda < 2$ until fitness values around $0.7n$ are reached. This behaviour is what creates the non-stable equilibrium slowing down the algorithm.

Finally, to identify the area of attraction of the non-stable equilibrium, in Fig. 6 we show the percentage of fitness evaluations spent in each fitness level for $n = 100$ (100 runs) and different s values near the transition between polynomial and exponential. Runs were stopped when the optimum was found or when 1,500,000 function evaluations were made. The first thing to notice is that for $s = 20$ the algorithm is attracted and spends most of the time near $n/2$ ones, which suggests that it behaves similar to a random walk. When s decreases, the area of attraction moves towards the optimum but stays at a linear distance from it. For $s \leq 3.4$ most of the evaluations are spent near the optimum on the harder fitness levels where λ tends to have linear values.

6 Discussion and Conclusions

We have shown that simple success-based rules, embedded in a $(1, \lambda)$ EA, are able to optimise OneMax in $O(n)$ generations and $O(n \log n)$ evaluations. The latter is best possible for any unary unbiased black-box algorithm [11, 25].

However, this result depends crucially on the correct selection of the success rate s. The above holds for constant $0 < s < 1$ and, in sharp contrast, the runtime on OneMax (and other common benchmark problems) becomes exponential with overwhelming probability if $s \geq 18$. Then the algorithm stagnates in an equilibrium state at a linear distance to the optimum where successes are common. Simulations showed that, once λ grows large enough to escape from the equilibrium, the algorithm is able to maintain

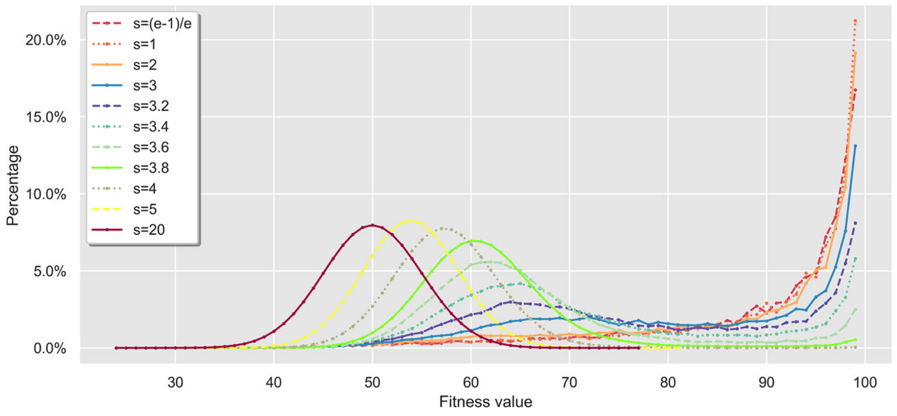

Fig. 6 Percentage of fitness function evaluations used per fitness value for the self-adjusting (1, λ) EA on OneMax with $n = 100$ over 100 runs (runs were stopped when the optimum was found or when 1,500,000 function evaluations were made)

large values of λ until the optimum is found. Hence, we observe the counterintuitive effect that for too large values of s, optimisation is harder when the algorithm is far away from the optimum and becomes easier when approaching the optimum. (To our knowledge, such an effect was only reported before on HotTopic functions [45] and Dynamic BinVal functions [46].)

There is a gap between the conditions $s < 1$ and $s \geq 18$. Further work is needed to close this gap. In our experiments we found a sharp threshold at $s \approx 3.4$, indicating that the widely used one-fifth rule ($s = 4$) is inefficient here, but other success rules achieve optimal asymptotic runtime.

Our analyses focus mostly on OneMax, but we also showed that when s is large the self-adjusting (1, λ) EA has an exponential runtime with overwhelming probability on Jump_k, Cliff_d, ZeroMax, TwoMax and Ridge. We believe that these results can be extended for many other functions: we conjecture that for any function that has a large number of contiguous fitness levels that are easy, that is, that the probability of a successful generation with $\lambda = 1$ is constant, then there is a (large) constant success rate s for which the self-adjusting (1, λ) EA would have an exponential runtime. We suspect that many combinatorial problem instances are easy somewhere, for example problems like minimum spanning trees, graph colouring, Knapsack and MaxSat tend to be easy in the beginning of the optimisation.

Given that for large values of s the algorithm gets stuck on easy parts of the optimisation and that OneMax is the easiest function with a unique optimum for the (1 + 1) EA [47–49] with regards to the expected optimisation time, in our preliminary work [26] we conjectured that any s that is efficient on OneMax would also be a good choice for any other problem. This conjecture was very recently disproved by Kaufmann, Larcher, Lengler and Zou [50], who showed that OneMax is not the easiest function with respect to fitness improvements, and that for a BinaryValue function with dynamically changing weights, improvements are even easier to find. This leads to a parameter setup for which the self-adjusting (1, λ) EA is efficient on

OneMax, but inefficient on dynamic BinVal [50]. The paper [50] concludes that there are two different notions of "easiness" and the ease of finding improvements is the more relevant notion for success-based parameter control mechanisms.

Another open question is to establish sufficient conditions for the self-adjusting $(1, \lambda)$ EA to perform well. The present authors recently made progress in this direction by showing that on all problems on which improvements are always hard to find, called *everywhere-hard* problems, self-adjustment in the self-adjusting $(1, \lambda)$ EA works as intended, for all constant values of the success rate s [51].

Acknowledgements The authors thank an anonymous reviewer as well as the authors of [27, 28] for reporting a bug in [26] and a previous version of this manuscript, and Maxime Larcher and Johannes Lengler in particular for discussions on how to fix this bug.

Funding Open Access funding enabled and organized by Projekt DEAL. This research has been supported by CONACYT (Consejo Nacional de Ciencia y Tecnología) under the Grant No. 739621.

Declarations

Competing Interests The authors have no competing interests to declare that are relevant to the content of this article.

A Proof of Lemma 3.7

This appendix contains the proof of Lemma 3.7 that was omitted from the main part.

Lemma 3.7 *Consider the self-adjusting* $(1, \lambda)$ *EA as in Theorem 3.5. Let* $\mathrm{OM}^*_t := \max_{t' \le t} \mathrm{OM}(x_{t'})$ *be its best-so-far fitness at generation t and let T be the first generation in which the optimum is found. Then with probability* $1 - O(1/n)$ *the following statements hold for a large enough constant* $r > 0$ *(that may depend on s).*

1. *For all $t \le T$ in which $\lambda_t \ge 4 \log n$, we have* $\mathrm{OM}(x_{t+1}) \ge \mathrm{OM}(x_t)$.
2. *For all $t \le T$, the fitness is at least:* $\mathrm{OM}(x_t) \ge \mathrm{OM}^*_t - r \log n$.

Proof of Lemma 3.7 Let E^t_1 denote the event that $\lambda_t < 4 \log n$ or $\mathrm{OM}(x_{t+1}) \ge \mathrm{OM}(x_t)$. Hence we only need to consider λ_t-values of $\lambda_t \ge 4 \log n \ge 2 \log_{\frac{e}{e-1}} n$ and by Lemma 2.2 Eq. (3) we have

$$\Pr\left(\overline{E^t_1}\right) \le \left(\frac{e-1}{e}\right)^{\lambda_t} \le \left(\frac{e-1}{e}\right)^{2\log_{\frac{e}{e-1}} n} = \frac{1}{n^2}.$$

Given that the event $\overline{E_1^t}$ happens in each step with probability at most $\frac{1}{n^2}$, by a union bound, the probability that this happens in the first T generations, with T being a random variable with $\mathrm{E}(T) < \infty$, is at most $\sum_{t=1}^{\infty} \Pr(T = t) \cdot t/n^2 = \mathrm{E}(T)/n^2$, and by Theorem 3.1 $\mathrm{E}(T)/n^2 = O(1/n)$. Hence, the probability that the first statement holds is $1 - O(1/n)$. For the second statement, let t^* be a generation in which the best-so-far fitness was attained: $\mathrm{OM}(x_{t^*}) = \mathrm{OM}_t^*$. By Lemma 3.3, abbreviating $\alpha := \frac{2s}{s+1} \log_F(enF^{1/s})$, the condition $\mathrm{OM}(x_{t^*}) \geq \mathrm{OM}(x_t) + r \log n$ implies $g_1(X_{t^*}) \geq \mathrm{OM}(x_{t^*}) - \alpha \geq \mathrm{OM}(x_t) - \alpha + r \log n \geq g_1(X_t) - \alpha + r \log n$.

Now define events $E_2^t = (\forall t' \in [t+1, n^2]\colon g_1(X_{t'}) \geq g_1(X_t) + \alpha - r \log(n))$. We apply the negative drift theorem (Theorem 2.5) to bound $\Pr\left(\overline{E_2^t}\right)$ from above. For any $t < n^2$ let $a := g_1(X_t) - r \log n + \alpha$ and $b := g_1(X_t) < n$, where $r > \alpha$ will be chosen later on. We pessimistically assume that the fitness component of g can only increase by at most 1. Lemma 3.4 has already shown that, even under this assumption, the drift is at least a positive constant. This implies the first condition of Theorem 2 in [35]. For the second condition, we need to bound transition probabilities for the potential. Owing to our pessimistic assumption, the current fitness can only increase by at most 1. The fitness only decreases by j if *all* offspring are worse than their parent by at least j. Hence, for all λ, the decrease in fitness is bounded by the decrease in fitness of the *first* offspring. The probability of the first offspring decreasing fitness by at least j is bounded by the probability that j bits flip, which is in turn bounded by $1/(j!) \leq 2/2^j$. The possible penalty in the definition of g changes by at most $\max\left(\frac{se}{e-1}, \frac{se}{e-1} \cdot \frac{1}{s}\right) = \frac{e}{e-1} < 2$. Hence, for all t,

$$\Pr\left(|g_1(X_{t-1}) - g_1(X_t)| \geq j + 2 \mid g_1(X_t) > a\right) \leq \frac{8}{2^{j+2}},$$

which meets the second condition of Theorem 2 in [35]. It then states that there is a constant c^* such that the probability that within $2^{c^*(a-b)/4}$ generations a potential of at most a is reached, starting from a value of at least b, is $2^{-\Omega(a-b)}$. By choosing the constant r large enough, we can scale up $a - b$ and thus make $2^{c^*(a-b)/4} \geq n^2$ and $2^{-\Omega(a-b)} = O(1/n^2)$. This yields $\Pr\left(\overline{E_2^t}\right) = O(1/n^2)$.

Arguing as before, using a union bound we show that the probability that $\overline{E_2^t}$ happens during the first T generations is at most $\sum_{t=1}^{\infty} \Pr(T = t) \cdot t \cdot O(1/n^2) = O(1/n)$. By Markov's inequality, the probability of not finding the optimum in n^2 generations, that is, $\Pr\left(T \geq n^2\right)$ is at most $\mathrm{E}(T)/n^2 = O(1/n)$ as well. Adding up all failure probabilities completes the proof. □

References

1. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing, 2nd edn. Springer, Berlin (2015)
2. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization—Algorithms and Their Computational Complexity. Springer, Berlin, Heidelberg (2010)
3. Jansen, T.: Analyzing Evolutionary Algorithms: The Computer Science Perspective. Springer, Berlin (2013)

4. Auger, A., Doerr, B. (eds.): Theory of Randomized Search Heuristics—Foundations and Recent Developments. Series on Theoretical Computer Science, vol. 1. World Scientific, USA (2011)
5. Doerr, B., Neumann, F. (eds.): Theory of Evolutionary Computation: Recent Developments in Discrete Optimization. Springer, Berlin (2020)
6. Lobo, F.G., Lima, C.F., Michalewicz, Z. (eds.): Parameter Setting in Evolutionary Algorithms. Studies in Computational Intelligence, vol. 54. Springer, Berlin, Heidelberg (2007)
7. Doerr, B., Doerr, C.: Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In: Doerr, B., Neumann, F. (eds.) Theory of Evolutionary Computation: Recent Developments in Discrete Optimization, pp. 271–321. Springer, Cham (2020)
8. Badkobeh, G., Lehre, P.K., Sudholt, D.: Unbiased black-box complexity of parallel search. In: Proc. of Parallel Problem Solving from Nature – PPSN XIII, pp. 892–901. Springer, Cham (2014)
9. Böttcher, S., Doerr, B., Neumann, F.: Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In: Proc. of Parallel Problem Solving from Nature—PPSN XI, vol. 6238, pp. 1–10. Springer, Cham (2010)
10. Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing new genetic algorithms. In: Theoretical Computer Science, vol. 567, pp. 87–104 (2015)
11. Doerr, B., Doerr, C., Yang, J.: Optimal parameter choices via precise black-box analysis. Theoret. Comput. Sci. **801**, 1–34 (2020)
12. Lässig, J., Sudholt, D.: Adaptive population models for offspring populations and parallel evolutionary algorithms. In: Proceedings of the 11th Workshop Proceedings on Foundations of Genetic Algorithms. FOGA '11, pp. 181–192. ACM, New York, NY, USA (2011)
13. Mambrini, A., Sudholt, D.: Design and analysis of schemes for adapting migration intervals in parallel evolutionary algorithms. Evol. Comput. **23**(4), 559–582 (2015)
14. Doerr, B., Doerr, C.: Optimal static and self-adjusting parameter choices for the $(1+(\lambda,\lambda))$ genetic algorithm. Algorithmica **80**(5), 1658–1709 (2018)
15. Hevia Fajardo, M.A., Sudholt, D.: On the choice of the parameter control mechanism in the $(1+(\lambda, \lambda))$ Genetic Algorithm. In: Proceedings of the Genetic and Evolutionary Computation. GECCO '20, pp. 832–840. ACM, New York, NY, USA (2020)
16. Doerr, B., Doerr, C., Kötzing, T.: Static and self-adjusting mutation strengths for multi-valued decision variables. Algorithmica **80**(5), 1732–1768 (2018)
17. Doerr, B., Gießen, C., Witt, C., Yang, J.: The $(1+\lambda)$ evolutionary algorithm with self-adjusting mutation rate. Algorithmica **81**(2), 593–631 (2019)
18. Doerr, B., Witt, C., Yang, J.: Runtime analysis for self-adaptive mutation rates. Algorithmica **83**(4), 1012–1053 (2021)
19. Case, B., Lehre, P.K.: Self-adaptation in nonelitist evolutionary algorithms on discrete problems with unknown structure. IEEE Trans. Evol. Comput. **24**(4), 650–663 (2020)
20. Lissovoi, A., Oliveto, P.S., Warwicker, J.A.: On the time complexity of algorithm selection hyper-heuristics for multimodal optimisation. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 2322–2329 (2019)
21. Jägersküpper, J., Storch, T.: When the plus strategy outperforms the comma strategy and when not. In: Proceedings of the IEEE Symposium on Foundations of Computational Intelligence (FOCI 2007), pp. 25–32 (2007)
22. Rowe, J.E., Sudholt, D.: The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm. Theoret. Comput. Sci. **545**, 20–38 (2014)
23. Rechenberg, I.: Evolutionsstrategie. PhD thesis (1973)
24. Kern, S., Müller, S.D., Hansen, N., Büche, D., Ocenasek, J., Koumoutsakos, P.: Learning probability distributions in continuous evolutionary algorithms: a comparative review. Nat. Comput. **3**(1), 77–112 (2004)
25. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**(4), 623–642 (2012)
26. Hevia Fajardo, M.A., Sudholt, D.: Self-adjusting population sizes for non-elitist evolutionary algorithms: Why success rates matter. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '21, pp. 1151–1159. ACM, New York, NY, USA (2021)
27. Kaufmann, M., Larcher, M., Lengler, J., Zou, X.: Self-adjusting population sizes for the $(1, \lambda)$-EA on monotone functions. ArXiv e-prints (2022) arXiv:2204.00531

28. Kaufmann, M., Larcher, M., Lengler, J., Zou, X.: Self-adjusting population sizes for the (1, λ)-EA on monotone functions. In: Proc. of Parallel Problem Solving from Nature—PPSN XVIII. Lecture Notes in Computer Science, vol. 13399, pp. 569–585. Springer, Cham (2022)
29. Doerr, B., Doerr, C., Lengler, J.: Self-adjusting mutation rates with provably optimal success rules. Algorithmica **83**(10), 3108–3147 (2021)
30. Paixão, T., Heredia, J.P., Sudholt, D., Trubenová, B.: Towards a runtime comparison of natural and artificial evolution. Algorithmica **78**(2), 681–713 (2017)
31. Hevia Fajardo, M.A., Sudholt, D.: Self-adjusting offspring population sizes outperform fixed parameters on the cliff function. In: Proceedings of the 16th Workshop on Foundations of Genetic Algorithms. FOGA '21, pp. 5–1515. ACM, New York, NY, USA (2021)
32. Lengler, J.: Drift analysis. In: Doerr, B., Neumann, F. (eds.) Theory of Evolutionary Computation: Recent Developments in Discrete Optimization, pp. 89–131. Springer, Cham (2020)
33. He, J., Yao, X.: A study of drift analysis for estimating computation time of evolutionary algorithms. Nat. Comput. **3**(1), 21–35 (2004)
34. Oliveto, P.S., Witt, C.: Simplified drift analysis for proving lower bounds in evolutionary computation. Algorithmica **59**(3), 369–386 (2011)
35. Oliveto, P.S., Witt, C.: Erratum: Simplified drift analysis for proving lower bounds in evolutionary computation. ArXiv e-prints (2012) arXiv:1211.7184
36. Oliveto, P.S., Witt, C.: Improved time complexity analysis of the simple genetic algorithm. Theoret. Comput. Sci. **605**, 21–41 (2015)
37. Akimoto, Y., Auger, A., Glasmachers, T.: Drift theory in continuous search spaces: Expected hitting time of the (1 + 1)-ES with 1/5 success rule. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '18, pp. 801–808. ACM, New York, NY, USA (2018)
38. Morinaga, D., Akimoto, Y.: Generalized drift analysis in continuous domain: Linear convergence of (1+1)-ES on strongly convex functions with lipschitz continuous gradients. In: Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms. FOGA '19, pp. 13–24. ACM, New York, NY, USA (2019)
39. Morinaga, D., Fukuchi, K., Sakuma, J., Akimoto, Y.: Convergence rate of the (1+1)-evolution strategy with success-based step-size adaptation on convex quadratic functions. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '21, pp. 1169–1177. ACM, New York, NY, USA (2021)
40. Felsenstein, J.: The evolutionary advantage of recombination. Genetics **78**, 737–756 (1974)
41. Jorritsma, J., Lengler, J., Sudholt, D.: Comma selection outperform plus selection on OneMax with randomly planted optima. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '23). ACM Press, New York, NY, USA (2023). To appear
42. Doerr, B.: Probabilistic tools for the analysis of randomized optimization heuristics. In: Doerr, B., Neumann, F. (eds.) Theory of Evolutionary Computation: Recent Developments in Discrete Optimization, pp. 1–87. Springer, Cham (2020)
43. Doerr, C., Wang, H., Ye, F., Rijn, S., Bäck, T.: IOHprofiler: a benchmarking and profiling tool for iterative optimization heuristics. arXiv e-prints:1810.05281 (2018) arXiv:1810.05281
44. Bossek, J., Sudholt, D.: Do additional optima speed up evolutionary algorithms? In: Proceedings of the 16th ACM/SIGEVO Conference on Foundations of Genetic Algorithms (FOGA 2021), pp. 8–1811. ACM, New York, NY, USA (2021)
45. Lengler, J., Zou, X.: Exponential slowdown for larger populations: the (μ+1)-EA on monotone functions. Theoret. Comput. Sci. **875**, 28–51 (2021)
46. Lengler, J., Riedi, S.: Runtime analysis of the (μ+1)-EA on the dynamic BinVal function. In: Evolutionary Computation in Combinatorial Optimization, pp. 84–99. Springer, Cham (2021)
47. Doerr, B., Johannsen, D., Winzen, C.: Drift analysis and linear functions revisited. In: IEEE Congress on Evolutionary Computation (CEC '10), pp. 1967–1974 (2010)
48. Witt, C.: Tight bounds on the optimization time of a randomized search heuristic on linear functions. Comb. Probab. Comput. **22**(2), 294–318 (2013)
49. Sudholt, D.: A new method for lower bounds on the running time of evolutionary algorithms. IEEE Trans. Evol. Comput. **17**, 418–435 (2013)
50. Kaufmann, M., Larcher, M., Lengler, J., Zou, X.: Onemax is not the easiest function for fitness improvements. In: Pérez Cáceres, L., Stützle, T. (eds.) Evolutionary Computation in Combinatorial Optimization, pp. 162–178. Springer, Cham (2023)

51. Hevia Fajardo, M.A., Sudholt, D.: Hard problems are easier for success-based parameter control. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO ’22. ACM, New York, NY, USA (2022)