Self-adjusting evolutionary algorithms for multimodal optimization

Rajabi, Amirhossein; Witt, Carsten

Published in:
Proceedings of the 2020 Genetic and Evolutionary Computation Conference

Link to article, DOI:
10.1145/3377930.3389833

Publication date:
2020

Document Version
Peer reviewed version

Self-Adjusting Evolutionary Algorithms for Multimodal Optimization

Amirhossein Rajabi
Technical University of Denmark
Kgs. Lyngby, Denmark
amraj@dtu.dk

Carsten Witt
Technical University of Denmark
Kgs. Lyngby, Denmark
cawi@dtu.dk

ABSTRACT

Recent theoretical research has shown that self-adjusting and self-adaptive mechanisms can provably outperform static settings in evolutionary algorithms for binary search spaces. However, the vast majority of these studies focuses on unimodal functions which do not require the algorithm to flip several bits simultaneously to make progress. In fact, existing self-adjusting algorithms are not designed to detect local optima and do not have any obvious benefit to cross large Hamming gaps.

We suggest a mechanism called stagnation detection that can be added as a module to existing evolutionary algorithms (both with and without prior self-adjusting schemes). Added to a simple (1+1) EA, we prove an expected runtime on the well-known *Jump* benchmark that corresponds to an asymptotically optimal parameter setting and outperforms other mechanisms for multimodal optimization like heavy-tailed mutation. We also investigate the module in the context of a self-adjusting (1+λ) EA and show that it combines the previous benefits of this algorithm on unimodal problems with more efficient multimodal optimization. To explore the limitations of the approach, we additionally present an example where both self-adjusting mechanisms, including stagnation detection, do not help to find a beneficial setting of the mutation rate. Finally, we investigate our module for stagnation detection experimentally.

CCS CONCEPTS

• **Theory of computation → Theory of randomized search heuristics**;

KEYWORDS

randomised search heuristics, self-adjusting algorithms, multimodal functions, runtime analysis

ACM Reference Format:
Amirhossein Rajabi and Carsten Witt. 2020. Self-Adjusting Evolutionary Algorithms for Multimodal Optimization. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377930.3389833

1 INTRODUCTION

Recent theoretical research on self-adjusting algorithms in discrete search spaces has produced a remarkable body of results showing that self-adjusting and self-adaptive mechanisms outperform static parameter settings. Examples include an analysis of the well-known (1+(λ, λ)) GA using a 1/5-rule to adjust its mutation rate on OneMax [6], of a self-adjusting (1+λ) EA sampling offspring with different mutation rates [13], matching the parallel black-box complexity of the OneMax function, and a self-adaptive variant of the latter [12]. Furthermore, self-adjusting schemes for algorithms over the search space $\{0,\dots,r\}^n$ for $r > 1$ provably outperform static settings [11] of the mutation operator. Self-adjusting schemes are also closely related to hyper-heuristics which, e. g., can dynamically choose between different mutation operators and therefore outperform static settings [25]. Besides the mutation probability, other parameters like the population sizes may be adjusted during the run of an evolutionary algorithm (EA) and analyzed from a runtime perspective [23]. Moreover, there is much empirical evidence (e. g. [14, 15, 18, 27]) showing that parameters of EAs should be adjusted during its run to optimize its runtime. See also the survey article [7] for an in-depth coverage of parameter control, self-adjusting algorithms, and theoretical runtime results.

A common feature of existing self-adjusting schemes is that they use different settings of a parameter (e. g., the mutation rate) and – in some way – measure and compare the progress achievable with the different settings. For example, the 2-rate (1+λ) EA from [13] samples $\lambda/2$ of the offspring with strength $r/2$ (where we define strength as the expected number of flipping bits, i. e., n times the mutation probability) and the other half with strength $2r$. The strength is afterwards adjusted to the one used by a fittest offspring. Similarly, the 1/5-rule [6] increases the mutation rate if fitness improvements happen frequently and decreases it otherwise. This requires that the algorithm is likely enough to make *some* improvements with the different parameters tried or, at least, that the smallest disimprovement observed in unsuccessful mutations gives reliable hints on the choice of the parameter. However, there are situations where the algorithm cannot make progress and does not learn from unsuccessful mutations either. This can be the case when the algorithm reaches local optima escaping from which requires an unlikely event (such as flipping many bits simultaneously) to happen. Classical self-adjusting algorithms would observe many unsuccessful steps in such situations and suggest to set the mutation rate to its minimum although that might not be the best choice to leave the local optimum. In fact, the vast majority of runtime results for self-adjusting EAs is concerned with unimodal functions that have no other local optima than the global optimum. An exception is the work [4] which considers a self-adaptive EA allowing two different mutation probabilities on a specifically designed multimodal problem. Altogether, there is a lack of theoretical results giving guidance on how to design self-adjusting algorithms that can leave local optima efficiently.

In this paper, we address this question and propose a self-adjusting mechanism called *stagnation detection* that adjusts mutation rates when the algorithm has reached a local optimum. In contrast to previous self-adjusting algorithms this mechanism is likely to increase the mutation rate in such situations, leading to a more efficient escape from local optima. This idea has been mentioned before, e. g., in the context of population sizing in stagnation [17]; also, recent empirical studies of the above-mentioned 2-rate (1+λ) EA, handling of stagnation by increasing the variance was explicitly suggested in [36]. Our contribution has several advantages over previous discussion of stagnation detection: it represents a simple module that can be added to several existing evolutionary algorithms with little effort, it provably does not change the behavior of the algorithm on unimodal functions (except for small error terms), allowing the transfer of previous results, and we provide rigorous runtime analyses showing general upper bounds for multimodal functions including its benefits on the well-known Jump benchmark function.

In a nutshell, our stagnation detection mechanism works in the setting of pseudo-boolean optimization and standard bit mutation. Starting from strength $r = 1$, it increases the strength from r to $r+1$ after a long waiting time without improvement has elapsed, meaning it is unlikely that an improving bit string at Hamming distance r exists. This approach bears some resemblance with variable neighborhood search (VNS) [21]; however, the idea of VNS is to apply local search with a fixed neighborhood until reaching a local optimum and then to adapt the neighborhood structure. There have also been so-called quasirandom evolutionary algorithms [9] that search the set of Hamming neighbors of a search point more systematically; however, these approaches do not change the expected number of bits flipped. In contrast, our stagnation detection uses the whole time an unbiased randomized global search operator in an EA and just adjusts the underlying mutation probability. Statistical significance of long waiting times is used, indicating that improvements at Hamming distance r are unlikely to exist; this is rather remotely related to (but clearly inspired by) the estimation-of-distribution algorithm sig-cGA [8] that uses statistical significance to counteract genetic drift.

This paper is structured as follows: In Section 2, we introduce the concrete mechanism for stagnation detection and employ it in the context of a simple, static (1+1) EA and the already self-adjusting 2-rate (1+λ) EA. Moreover, we collect tools for the analysis that are used in the rest of the paper. Section 3 deals with concrete runtime bounds for the (1+1) EA and (1+λ) EA with stagnation detection. Besides general upper bounds, we prove a concrete result for the Jump benchmark function that is asymptotically optimal for algorithms using standard bit mutation and outperforms previous mutation-based algorithms for this function like the heavy-tailed EA from [10]. Elementary techniques are sufficient to show these results. To explore the limitations of stagnation detection and other self-adjusting schemes, we propose in Section 4 a function where these mechanisms provably fail to set the mutation rate to a beneficial regime. As a technical tool, we use drift analysis and analyses of occupation times for processes with strong drift. To that purpose, we use a theorem by Hajek [20] on occupation times that, to the best of the knowledge, was not used for the analysis of randomized search heuristics before and may be of independent interest. Finally, in Section 5, we add some empirical results, showing that the asymptotically smaller runtime of our algorithm on Jump is also visible for small problem dimensions. We finish with some conclusions. Due to space restrictions, several proofs had to be omitted from this paper and have been replaced by proof sketches, but note that these proofs can be found in the preprint [26].

2 PRELIMINARIES

We shall now formally define the algorithms analyzed and present some fundamental tools for the analysis.

2.1 Algorithms

We are concerned with pseudo-boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$ that w. l. o. g. are to be maximized. A simple and well-studied EA studied in many runtime analyses (e. g., [16]) is the (1+1) EA displayed in Algorithm 1. It uses a standard bit mutation with strength r, where $1 \leq r \leq n/2$, which means that every bit is flipped independently with probability r/n. Usually, $r = 1$ is used, which is the optimal strength on linear functions [35]. Smaller strengths lead to less than 1 bit being flipped in expectation, and strengths above $n/2$ in binary search spaces are considered "ill-natured" [1] since a mutation at a bit should not be more likely than a non-mutation.

Algorithm 1 (1+1) EA with static strength r

```
Select $x$ uniformly at random from $\{0,1\}^n$
for $t \leftarrow 1, 2, \dots$ do
    Create $y$ by flipping each bit in a copy of $x$ independently with probability $\frac{r}{n}$.
    if $f(y) \geq f(x)$ then
        $x \leftarrow y$.
```

The *runtime* (also called *optimization time*) of the (1+1) EA on a function f is the first point of time t where a search point of maximal fitness has been created; often the expected runtime, i. e., the expected value of this time, is analyzed. The (1+1) EA with $r = 1$ has been extensively studied on simple unimodal problems like

$$\textsc{OneMax}(x_1, \dots, x_n) := |x|_1,$$

and

$$\textsc{LeadingOnes}(x_1, \dots, x_n) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$$

but also on the multimodal $\textsc{Jump}_m$ function with gap size m defined as follows:

$$\textsc{Jump}_m(x_1, \dots, x_n) = \begin{cases} m + |x|_1 & \text{if } |x|_1 \leq n - m \text{ or } |x|_1 = n \\ n - |x|_1 & \text{otherwise} \end{cases}$$

The classical (1+1) EA with $r = 1$ optimizes these functions in expected time $\Theta(n \log n)$, $\Theta(n^2)$ and $\Theta(n^m + n \log n)$, respectively (see, e. g., [16]).

The first two problems are unimodal functions, while Jump for $m \geq 2$ is multimodal and has a local optimum at the set of points where $|x|_1 = n - m$. To overcome this optimum, m bits have to flip simultaneously. It is well known [10] that the time to leave this optimum is minimized at strength m instead of strength 1 (see below for a more detailed exposition of this phenomenon). Hence, the (1+1) EA would benefit from increasing its strength when sitting at the local optimum. The algorithm does not immediately know that it sits at a local optimum. However, if there is an improvement at Hamming distance 1 then such an improvement has probability at least $(1-1/n)^{n-1}/n \geq 1/(en)$ with strength 1, and the probability of not finding it in $en \ln n$ steps is at most

$$(1 - 1/(en))^{en \ln n} \leq 1/n.$$

Similarly, if there is an improvement that can be reached by flipping k bits simultaneously and the current strength equals k, then the probability of not finding it within $((en)^k/k^k)\ln n$ steps is at most

$$\left(1 - \frac{k^k}{(en)^k}\right)^{((en)^k/k^k)\ln n} \leq \frac{1}{n}.$$

Hence, after $((en)^k/k^k)\ln n$ steps without improvement there is high evidence for that no improvement at Hamming distance k exists.

We put this ideas into an algorithmic framework by counting the number of so-called unsuccessful steps, i. e., steps that do not improve fitness. Starting from strength 1, the strength is increased from r to $r+1$ when the counter exceeds the threshold $2((en)^r/r^r)\ln(nR)$ for a parameter R to be discussed shortly. Both counter and strength are reset (to 0 and 1 respectively) when an improvement is found, i. e., a search point of strictly better fitness. In the context of the (1+1) EA, the stagnation detection (SD) is incorporated in Algorithm 2. We see that the counter u is increased in every iteration that does not find a strict improvement. However, search points of equal fitness are still accepted as in the classical (1+1) EA. We note that the strength stays at its initial value 1 if finding an improvement does not take longer than the corresponding threshold $2en\ln(Rn)$; if the threshold is never exceeded the algorithm behaves identical to the (1+1) EA with strength 1 according to Algorithm 1.

The parameter R can be used to control the probability of failing to find an improvement at the "right" strength. More precisely, the probability of not finding an improvement at distance r with strength r is at most

$$\left(1 - \frac{r^r}{(en)^r}\right)^{(2(en)^r/r^r)\ln(nR)} \leq \frac{1}{(nR)^2}.$$

As shown below in Theorem 3.3, if R is set to the number of fitness values of the underlying function f, i. e., $R = |\mathrm{Im}(f)|$, then the probability of ever missing an improvement at the right strength is sufficiently small throughout the run. We recommend at least $R = n$ if nothing is known about the range of f, resulting in a threshold of at least $4((en)^r/r^r)\ln(n)$ at strength r.

We also add stagnation detection to the (1+λ) EA with self-adjusting mutation rate defined in [13] (adapted to maximization of the fitness function), where half of the offspring are created with strength $r/2$ and the other half with strength $2r$; see Algorithm 3. Unsuccessful mutations are counted in the same way as in Algorithm 2, taking into account that λ offspring are used. The algorithm can be in two states. Unless the counter threshold is reached and a strength increase is triggered, the algorithm behaves the same as the self-adjusting (1+λ) EA from [13] (State 2). If, however, the counter threshold $2en\ln(nR)/\lambda$ is reached, then the algorithm changes to the module that keeps increasing the strength until a strict improvement is found (State 1). Since it does not make sense to decrease the strength in this situation, all offspring use the same strength until finally an improvement is found and the algorithm changes back to the original behavior using two strengths for the offspring. The boolean variable g keeps track of the state.

From the discussion of these two algorithms, we see that the stagnation detection consisting of counter for unsuccessful steps, threshold, and strength increase also can be added to other algorithms, while keeping their original behavior unless the counter threshold it reached.

Algorithm 2 (1+1) EA with stagnation detection (SD-(1+1) EA)

```
Select $x$ uniformly at random from $\{0,1\}^n$ and set $r_1 \leftarrow 1$.
$u \leftarrow 0$.
for $t \leftarrow 1, 2, \dots$ do
    Create $y$ by flipping each bit in a copy of $x$ independently with probability $\frac{r_t}{n}$.
    $u \leftarrow u + 1$.
    if $f(y) > f(x)$ then
        $x \leftarrow y$.
        $r_{t+1} \leftarrow 1$.
        $u \leftarrow 0$.
    else if $f(y) = f(x)$ and $r_t = 1$ then
        $x \leftarrow y$.
    if $u > 2\left(\frac{en}{r_t}\right)^{r_t}\ln(nR)$ then
        $r_{t+1} \leftarrow \min\{r_t + 1, n/2\}$.
        $u \leftarrow 0$.
    else
        $r_{t+1} \leftarrow r_t$.
```

2.2 Mathematical Tools

We now collect frequently used mathematical tools. The first one is a simple summation formula used to analyze the time spent until the strength is increased to a certain value.

Lemma 2.1. *For $m < n$, we have $\sum_{i=1}^{m}\left(\frac{en}{i}\right)^i < \frac{n}{n-m}\left(\frac{en}{m}\right)^m$.*

Proof. For all $i < m$, we have

$$\left(\frac{en}{m-i}\right)^{m-i} = \left(\frac{m}{en}\right)^i \left(\frac{m}{m-i}\right)^{m-i} \left(\frac{en}{m}\right)^m$$

, so

$$\sum_{i=1}^{m}\left(\frac{en}{i}\right)^i = \sum_{i=0}^{m-1}\left(\frac{en}{m-i}\right)^{m-i} = \left(\frac{en}{m}\right)^m \sum_{i=0}^{m-1}\left(\frac{m}{en}\right)^i \left(1 + \frac{i}{m-i}\right)^{m-i}$$

$$< \left(\frac{en}{m}\right)^m \sum_{i=0}^{m-1}\left(\frac{m}{n}\right)^i < \frac{n}{n-m}\left(\frac{en}{m}\right)^m.$$

□

Algorithm 3 (1+λ) EA with two-rate standard bit mutation and stagnation detection (SASD-(1+λ) EA)

```
Select $x$ uniformly at random from $\{0,1\}^n$ and set $r_1 \leftarrow r^{\text{init}}$.
$u \leftarrow 0$.
$g \leftarrow$ False (boolean variable indicating stagnation detection)
for $t \leftarrow 1, 2, \ldots$ do
    $u \leftarrow u + 1$.
    if $g$ = True then
State 1 – Stagnation Detection
        for $i \leftarrow 1, \ldots, \lambda$ do
            Create $x_i$ by flipping each bit in a copy of $x$ independently with probability $\frac{r_t}{n}$.
        $y \leftarrow \arg\max_{x_i} f(x_i)$ (breaking ties randomly).
        if $f(y) > f(x)$ then
            $x \leftarrow y$.
            $r_{t+1} \leftarrow r^{\text{init}}$.
            $g \leftarrow$ False.
            $u \leftarrow 0$.
        else
            if $u > 2\left(\frac{en}{r_t}\right)^{r_t} \ln(nR)/\lambda$ then
                $r_{t+1} \leftarrow \min\{r_t + 1, n/2\}$.
                $u \leftarrow 0$.
            else
                $r_{t+1} \leftarrow r_t$.
    else (i. e., $g$ = False)
State 2 – Self-Adjusting (1+$\lambda$) EA
        for $i \leftarrow 1, \ldots, \lambda$ do
            Create $x_i$ by flipping each bit in a copy of $x$ independently with probability $\frac{r_t}{2n}$ if $i \le \lambda/2$ and with probability $2r_t/n$ otherwise.
        $y \leftarrow \arg\min_{x_i} f(x_i)$ (breaking ties randomly).
        if $f(y) \ge f(x)$ then
            if $f(y) > f(x)$ then
                $u \leftarrow 0$.
            $x \leftarrow y$.
        Perform one of the following two actions with prob. 1/2:
            – Replace $r_t$ with the strength that $y$ has been created with.
            – Replace $r_t$ with either $r_t/2$ or $2r_t$, each with probability 1/2.
        $r_{t+1} \leftarrow \min\{\max\{2, r_t\}, n/4\}$.
        if $u > 2\left(\frac{en}{r_t}\right)^{r_t} \ln(nR)/\lambda$ then
            $r_{t+1} \leftarrow 2$.
            $g \leftarrow$ True.
            $u \leftarrow 0$.
```

The following result due to Hajek applies to processes with a strong drift towards some target state, resulting in decreasing occupation probabilities with respect to the distance from the target. On top of this occupation probabilities, the theorem bounds *occupation times*, i. e., the number of steps that the process spends in a non-target state over a certain time period.

Theorem 2.2 (Theorem 3.1 in [20]). *Let X_t, $t \ge 0$, be a stochastic process adapted to a filtration $\mathcal{F}_t$ on $\mathbb{R}$. Let $a \in \mathbb{R}$. Assume for $\Delta_t = X_{t+1} - X_t$ that there are $\eta > 0, \delta < 1$ and $D > 0$ such that that*

(a) $\mathrm{E}\left(e^{\eta\Delta} \mid \mathcal{F}_t; X_t > a\right) \le \rho$

(b) $\mathrm{E}\left(e^{\eta\Delta} \mid \mathcal{F}_t; X_t \le a\right) \le D$

If additionally X_0 is of exponential type (i. e., $\mathrm{E}\left(e^{\lambda X_0}\right)$ is finite for some $\lambda > 0$) then for any constant $\epsilon > 0$ there exist absolute constants $K \ge 0, \delta < 1$ such that for all $b \ge a$ and $T \ge 1$

$$\Pr\Big(\frac{1}{T}\sum_{t=1}^{T} \mathbb{1}_{X_t \le b} \le 1 - \epsilon - \frac{1-\epsilon}{1-\rho} D e^{\eta(a-b)}\Big) \le K\delta^T$$

3 ANALYSIS OF SD-(1+1) EA

In this section, we study the SD-(1+1) EA from Algorithm 2 in greater detail. We show general upper and lower bounds on multimodal functions and then analyze the special case of Jump more precisely. We also show the important result that on unimodal functions, the SD-(1+1) EA with high probability behaves in the same way as the classical (1+1) EA with strength 1, including the same asymptotic bound on the expected optimization time.

3.1 Expected Times to Leave Local Optima

In the following, given a fitness function $f\colon \{0,1\}^n \to \mathbb{R}$, we call the *gap* of the point $x \in \{0,1\}^n$ the minimum hamming distance to points with strictly larger fitness function value. Formally,

$$\mathrm{gap}(x) := \min\{H(x,y) : f(y) > f(x), y \in \{0,1\}^n\}.$$

Obviously, it is not possible to improve fitness by changing less than gap(x) bits of the current search point. However, if the algorithm creates a point of gap(x) distance from the current search point x, we can make progress with a positive probability. Note that gap$(x) = 1$ is allowed, so the definition also covers points that are not local optima.

Hereinafter, T_x denotes the number of steps of SD-(1+1) EA to find an improvement point when the current search point is x. Let phase r consists of all points of time where strength r is used in the algorithm with stagnation counter. Let E_r be the event of **not** finding the optimum by the end of phase r, and U_r be the event of not finding the optimum during phases 1 to $r-1$ and finding in phase r. In other words, $U_r = E_1 \cap \cdots \cap E_{r-1} \cap \overline{E_r}$.

The following lemma will be used throughout this section. It shows that the probability of not finding a search point with larger fitness value in phases of larger strength than the real gap size is small; however, by definition phase $n/2$ is not finished before the algorithm finds an improvement. In the statement of the lemma, recall that the parameter R controls the threshold for the number of unsuccessful steps in stagnation detection.

Lemma 3.1. *Let $x \in \{0,1\}^n$ be the current search point of the SD-(1+1) EA on a pseudo-boolean fitness function $f\colon \{0,1\}^n \to \mathbb{R}$ and let $m = \mathrm{gap}(x)$. Then*

$$\Pr(E_r) \le \begin{cases} \frac{1}{(nR)^2} & \text{if } m \le r < n/2 \\ 0 & \text{if } r = n/2. \end{cases}$$

Proof. The algorithm spends $2e^r n^r/r^r \ln(nR)$ steps at strength r until it increases the counter. Then, the probability of not improving at strength $r \ge m$ is at most

$$\Pr(E_r) = \left(1 - \left(1 - \frac{r}{n}\right)^{n-m}\left(\frac{r}{n}\right)^m\right)^{2e^r n^r/r^r \ln(nR)} \le \frac{1}{(nR)^2}.$$

During phase $n/2$, the algorithm does not increase the strength, and it continues to mutate each bit with probability of $1/2$. As each point on domain is accessible in this phase, the probability of eventually failing to find the improvement is 0. □

We turn the previous observation into a general lemma on improvement times.

Theorem 3.2. *Let $x \in \{0,1\}^n$ be the current search point of the SD-(1+1) EA on a pseudo-boolean function $f\colon \{0,1\}^n \to \mathbb{R}$. Define T_x as the time to create a strict improvement and $L_{x,k} := \mathrm{E}(T_x)$ if $\mathrm{gap}(x) = k$. Then, using $m = \min\{k, n/2\}$, we have for all x with $\mathrm{gap}(x) = k$ that*

$$\left(\frac{en}{m}\right)^m\left(1-\frac{m^2}{n-m}\right) < L_{x,k} \le 2\left(\frac{en}{m}\right)^m\left(1+\frac{5m}{n}\ln(nR)\right).$$

Proof. Using the law of total probability with respect to the events U_i defined above, we have

$$\mathrm{E}(T_x) = \sum_{i=1}^{n/2} \mathrm{E}(T_x \mid U_i)\Pr(U_i). \tag{1}$$

Note that the algorithm does not increase the strength to more than $n/2$. By assuming that the algorithm pessimistically does not find a better point for $r < m$, we can bound the formula (1) as follows:

$$\mathrm{E}(T_x) < \underbrace{\mathrm{E}(T_x \mid U_m)}_{=:S_1} + \underbrace{\sum_{i=m+1}^{n/2}\mathrm{E}(T_x \mid U_i)\Pr(U_i)}_{=:S_2}$$

Regarding S_1, it takes $\sum_{i=1}^{m-1} 2(en/i)^i \ln(nR)$ steps until the SD-(1+1) EA increases the strength to m. When the mutation probability is m/n, within an expected number of $((m/n)^m(1-m/n)^{n-m})^{-1}$ steps, a better point will be found. Thus, by using Lemma 2.1, we have

$$\begin{aligned}\mathrm{E}(T_x \mid U_m) &\le \sum_{i=1}^{m-1} 2\left(\frac{en}{i}\right)^i \ln(nR) + \frac{1}{(m/n)^m(1-m/n)^{n-m}}\\ &< 2\frac{n}{n-m+1}\left(\frac{en}{m-1}\right)^{m-1}\ln(nR) + \left(\frac{en}{m}\right)^m\\ &< \left(\frac{en}{m}\right)^m\left(1+\frac{5m}{en}\left(1+\frac{1}{m-1}\right)^{m-1}\ln(nR)\right)\\ &\le \left(\frac{en}{m}\right)^m\left(1+\frac{5m}{n}\ln(nR)\right).\end{aligned}$$

In order to estimate S_2, if $m = n/2$, the value of S_2 equals zero. Otherwise, by using Lemma 3.1, $\Pr(U_i) < \prod_{j=m}^{i-1}\Pr(E_j) < n^{-2(i-m)}$ for $i \ge m+1$ since $R \ge 1$. We compute

$$\begin{aligned}\sum_{i=m+1}^{n/2}\mathrm{E}(T_x \mid U_i)\Pr(U_i) &\le \sum_{i=m+1}^{n/2} O\left(\left(\frac{en}{i}\right)^i \ln(nR)\right) n^{-2(i-m)}\\ &= \ln(nR)\sum_{i=m+1}^{n/2} O\left(\left(\frac{e}{i}\right)^i n^{2m-i}\right) = o((en/m)^m).\end{aligned}$$

Altogether, we have

$$\mathrm{E}(T_x) \le \left(\frac{en}{m}\right)^m\left(1+\frac{5m}{n}\ln(nR)\right) + o((en/m)^m)$$

.

Moreover, the expected number of iterations for finding an improvement is at least $p^{-m}(1-p)^{-(n-m)}$ for any mutation rate p. Using the same arguments as in the analysis of the (1+1) EA on Jump in [10], since $\frac{m}{n}$ is the unique minimum point in the interval $[0,1]$,

$$\mathrm{E}(T_x) \ge (m/n)^{-m}(1-m/n)^{-(n-m)} \ge (en/m)^m\left(1-\frac{m^2}{n-m}\right).$$

□

We now present the above-mentioned important "simulation result" implying that on unimodal functions, the stagnation detection of SD-(1+1) EA is unlikely ever to trigger a strength increase during its run. Moreover, for a wide range of runtime bounds obtained via the fitness level method [30], we show that these bounds transfer to the SD-(1+1) EA up to vanishingly small error terms. The proof carefully estimates the probability of the strength ever exceeding 1.

Lemma 3.3. *Let $f\colon \{0,1\}^n \to \mathbb{R}$ be a unimodal function and consider the SD-(1+1) EA with $R \ge |\mathrm{Im}(f)|$. Then, with probability $1 - o(1)$, the SD-(1+1) EA never increases the strength and behaves stochastically like the (1+1) EA before finding an optimum of f.*

Denote by T_{sd} and $T_{classic}$ the runtime of the SD-(1+1) EA and the classical (1+1) EA with strength 1 on f, respectively. If U is an upper bound on $\mathrm{E}(T_{T_{classic}})$ obtained by summing up worst-case expected waiting times for improving over all fitness values in $\mathrm{Im}(f)$, then

$$\mathrm{E}(T_{sd}) \le U + o(1).$$

The same statements hold with SD-(1+1) EA replaced with SASD-(1+λ) EA, and (1+1) EA replaced with the self-adjusting (1+λ) EA without stagnation detection.

3.2 Analysis on Jump

It is well known that strength 1 for the (1+1) EA leads to an expected runtime of $\Theta(n^m)$ on Jump_m if $m \ge 2$ [16]. The asymptotically dominating term comes from the fact that m bits must flip simultaneously to leave the local optimum at $n - m$ one-bits. To minimize the time for such an escaping mutation, mutation rate m/n is optimal [10], leading to an expected time of $(1+o(1))(n/m)^m(1-m/n)^{m-n}$ to optimize Jump, which is $\Theta((en/m)^m)$ for $m = o(\sqrt{n})$. However, a static rate of m/n cannot be chosen without knowing the gap size m. Therefore, different heavy-tailed mutation operators have been proposed for the (1+1) EA [10, 19], which most of the time choose strength 1 but also use strength r, for arbitrary $r \in \{1, \dots, n/2\}$ with at least polynomial probability. This results in optimization times on Jump of $\Theta((en/m)^m \cdot p(n))$ for some small polynomial $p(n)$ (roughly, $p(n) = \omega(\sqrt{m})$ in [10] and $p(n) = \Theta(n)$ in [19]). Similar polynomial overheads occur with hypermutations as used in artificial immune systems [3]; in fact such overheads cannot be completely avoided with heavy-tailed mutation operators, as proved in [10]. We also remark that Jump can be optimized faster than $O((en/m)^m)$ if crossover is used [29, 31], by simple estimation-of-distribution algorithms [5] or specific black-box algorithms [2]. All of this is outside the scope of this study that concentrates on mutation-only algorithms.

We now state our main result, implying that the SD-(1+1) EA achieves an asymptotically optimal runtime on Jump_m for $m =$

$o(\sqrt{n})$, hence being faster than the heavy-tailed mutations mentioned above. Recall that this does not come at a significant extra cost for simple unimodal functions like OneMax according to Lemma 3.3.

Theorem 3.4. *Let $n \in \mathbb{N}$. For all $2 \le m = o(n)$, the expected runtime $\mathrm{E}(T)$ of the SD-(1+1) EA on Jump$_m$ satisfies*

$$\Omega\left(\left(\frac{en}{m}\right)^m\left(1-\frac{m^2}{n-m}\right)\right) \le \mathrm{E}(T) \le O\left(\left(\frac{en}{m}\right)^m\right).$$

Proof. It is well known that the (1+1) EA with mutation rate $1/n$ finds the optimum of the n-dimensional OneMax function in an expected number of at most $en\ln n - O(n)$ iterations.

Until reaching the plateau consisting of all points of $n-m$ one-bits, Jump is equivalent to OneMax; hence, according to Lemma 3.3, the expected time until SD-(1+1) EA reaches the plateau is at most $O(n\ln n)$ (noting that this bound was obtained via the fitness level method).

Every plateau point x with $n-m$ one-bits satisfies $\mathrm{gap}(x) = m$ according to the definition of Jump. Thus, using Theorem 3.2, the algorithm finds the optimum within expected time

$$\Omega\left(\left(\frac{en}{m}\right)^m\left(1-\frac{m^2}{n-m}\right)\right) \le \mathrm{E}(T_x) \le O\left(\left(\frac{en}{m}\right)^m\right).$$

This dominates the expected time of the algorithm before the plateau point.

Finally,

$$\Omega\left(\left(\frac{en}{m}\right)^m\left(1-\frac{m^2}{n-m}\right)\right) \le \mathrm{E}(T) \le O\left(\left(\frac{en}{m}\right)^m\right).$$

□

It is easy to see (similarly to the analysis of Theorem 3.4) that for all $m = \Theta(n)$, the expected runtime $\mathrm{E}(T)$ of the SD-(1+1) EA on Jump$_m$ satisfies $\mathrm{E}(T) = O\left(\left(\frac{en}{m}\right)^m \ln n\right)$.

3.3 General Bounds

The Jump function only has one local optimum that usually has to be overcome on the way to the global optimum. We generalize the previous analysis to functions that have multiple local optima of possibly different gap sizes. As a special case, we can asymptotically recover the expected runtime on the LeadingOnes function in Corollary 3.6.

Theorem 3.5. *The expected runtime of the SD-(1+1) EA on a pseudo-Boolean fitness function f is at most*

$$\mathrm{E}(T \mid V_1, \dots, V_n) = O\left(\sum_{k=1}^{n} V_k L_k\right),$$

where V_k is the number of points x of $\mathrm{gap}(x) = k$ visited by the algorithm and $L_k := \max\{L_{x,k} \mid x \in \{0,1\}^n \wedge \mathrm{gap}(x) = k\}$ with $L_{x,k}$ as defined in Theorem 3.2. Moreover,

$$\mathrm{E}(T) = O\left(\sum_{k=1}^{n} \mathrm{E}(V_k)\, L_k\right),$$

Proof. The SD-(1+1) EA visits a random trajectory of search points $\{x_0, x_1, x_2, \dots, x_m = x^*\}$ in order to find an optimum point x^*.

For any search point x with $\mathrm{gap}(x) = k$, the expected time to find a better search point when $r \le m$ is $\mathrm{E}(T_x) = L_k$ according to Theorem 3.2.

Also, we have $T = T_{x_1} + T_{x_2} + \dots + T_{x_m} = \sum_{k=1}^{n} V_k \cdot (T_x \mid \mathrm{gap}(x) = k)$. Therefore, as the strength r is reset to 1 after each improvement, we have

$$\mathrm{E}(T \mid V_1, \dots, V_n) = O\left(\sum_{k=1}^{n} V_k L_k\right),$$

which proves the first statement of this theorem. The second follows by the law of total expectation. □

Corollary 3.6. *The expected runtime of the SD-(1+1) EA on LeadingOnes is at most $O(n^2)$.*

Proof. On LeadingOnes, there are at most n points of gap size 1, so according to Theorem 3.5, the expected runtime is $O(n^2)$. □

Corollary 3.6 can be also inferred from Lemma 3.3 since LeadingOnes is unimodal and the $O(n^2)$ bound was inferred via the fitness level method.

We finally specialize Theorem 3.5 into a result for the well-known Trap function [16] that is identical for OneMax except for the all-zeros string that has optimal fitness $n+1$. We obtain a bound of $2^{\Theta(n)}$ instead of the $\Theta(n^n)$ bound for the classical (1+1) EA. The base of our result is somewhat larger than for the fast GA from [10]; however, it is still close to the 2^n bound that would be obtained by uniform search.

Corollary 3.7. *The expected runtime of SD-(1+1) EA on Trap is at most $O(2.34^n \ln n)$.*

Proof. On Trap, there are one point of gap size n and $O(n)$ points with gap size of 1. So according to Theorem 3.5, the expected runtime is $O\left((2.34)^n \ln n\right)$. □

4 AN EXAMPLE WHERE SELF-ADAPTATION FAILS

While our previous analyses have shown the benefits of the self-adjusting scheme, in particular highlighting stagnation detection on multimodal functions, it is clear that our scheme also has limitations. In this section, we present an example of a pseudo-Boolean function where stagnation detection does not help to find its global optimum in polynomial time; moreover, the function is hard for other self-adjusting schemes since measuring the number of successes does not hint on the location of the global optimum. In fact, the function demonstrates a more general effect where the behavior is very sensitive with respect to choice of the the mutation probability. More precisely, a plain (1+1) EA with mutation probability $1/n$ with overwhelming probability gets stuck in a local optimum from which it needs exponential time to escape while the (1+1) EA with mutation probability $2/n$ and also above finds the global optimum in polynomial time with overwhelming probability. Since the function is unimodal except at the local optimum, our self-adjusting (1+1) EA with stagnation detection fails as well.

To the best of our knowledge, a phase transition with respect to the mutation probability where an increase by a small constant factor leads from exponential to polynomial optimization time has been unknown in the literature of runtime analysis so far and may be of independent interest. We are aware of opposite phase transitions on monotone functions [24] where increasing the mutation rate is detrimental; however, we feel that our function and the general underlying construction principle are easier to understand than these specific monotone functions.

The construction of our function, called NeedHighMut, is based on a general principle that was introduced in [32] to show the benefits of populations and was subsequently applied in [22] to separate a coevolutionary variant of the (1+1) EA from the standard (1+1) EA. Section 5 of the latter paper also beautifully describes the general construction technique that involves creating two differently pronounced gradients for the algorithms to follow. Further applications are given in [33] and [34] to show the benefit of populations in elitist and non-elitist EAs. Also [28] use very similar construction technique for their Balance function that is easier to optimize in frequently changing than slowly changing environments; however, they did not seem to be aware that their approach resembles earlier work from the papers above.

We now describe the construction of our function NeedHighMut. The crucial observation is that strength 1 (i. e., probability $p = 1/n$) makes it more likely to flip exactly one specific bit than strength 2 – in fact strength 1 is asymptotically optimal since the probability of flipping one specific bit is $p(1-p)^{n-1} \approx pe^{-pn}$, which is maximized for $p = 1/n$. However, to flip specific two bits, which has probability $p^2(1-p)^{n-2} \approx p^2e^{-pn}$, the choice $p = 2/n$ is asymptotically optimal and clearly better than $1/n$. Now, given a hypothetical time span of T, we expect approximately $T_1(p) := Tpe^{-p/n}$ specific one-bit and $T_2(p) := Tp^2e^{-p/n}$ specific two-bit flips. Assuming the actual numbers to be concentrated and just arguing with expected values, we have $T_1(1/n) \gg T_2(1/n)$ but $T_2(2/n) \gg T_1(2/n)$, i. e., there will be considerably more two-bit flips at strength 2 than at strength 1 and considerably less 1-bit flips. The fitness function will account for this. It leads to a trap at a local optimum if a certain number of one-bit flips is exceeded before a certain minimum number of two-bit flips has happened; however, if the number of one-bit flips is low enough before the minimum number of two-bit flips has been reached, the process is on track to the global optimum.

We proceed with the formal definition of NeedHighMut, making these ideas precise and overcoming technical hurdles. Since we have at most n specific one-bit flips but a specific two-bit flip is already by a factor of $O(1/n)$ less likely than a one-bit flip, we will work with two-bit flips happening in small blocks of size $\sqrt[4]{n}$, leading to a probability of roughly $n^{-3/2}$ for a two-bit flip in a block. In the following, we will imagine a bit string x of length n as being split into a prefix $a := a(x)$ of length $n-m$ and a suffix $b := b(x)$ of length m, where m still has to be defined. Hence, $x = a(x) \circ b(x)$, where $\circ$ denotes the concatenation.

The prefix $a(x)$ is called *valid* if it is of the form 1^i0^{n-m-i}, i. e., i leading ones and $n-m-i$ trailing zeros. The prefix fitness $\textsc{pre}(x)$ of a string $x \in \{0,1\}^n$ with valid prefix $a(x) = 1^i0^{n-m-i}$ equals just i, the number of leading ones. The suffix consists of $\lceil \frac{2}{3}\xi\sqrt{n} \rceil$, where $\xi \geq 1$ is a parameter of the function, consecutive blocks of $\lceil n^{1/4} \rceil$ bits each, altogether $m \leq \xi\frac{2}{3}n^{3/4} = o(n)$ bits. Such a block is called *valid* if it contains either 0 or 2 one-bits; moreover, it is called *active* if it contains 2 and *inactive* if it contains 0 one-bits. A suffix where all blocks are valid and where all blocks following first inactive block are also inactive is called valid itself, and the suffix fitness $\textsc{suff}(x)$ of a string x with valid suffix $b(x)$ is the number of leading active blocks before the first inactive block. Finally, we call a string $x \in \{0,1\}^n$ valid if both its prefix and suffix are valid.

Our final fitness function is a weighted combination of $\textsc{pre}(x)$ and $\textsc{suff}(x)$. We define for $x \in \{0,1\}^n$, where $x = a \circ b$ with the above-introduced a and b,

$$\textsc{NeedHighMut}_\xi(x) :=$$

$$\begin{cases} n^2\textsc{suff}(x) + \textsc{pre}(x) & \text{if } \textsc{pre}(x) \leq \frac{9(n-m)}{10} \wedge x \text{ valid} \\ n^2m + \textsc{pre}(x) + \textsc{suff}(x) - n - 1 & \text{if } \textsc{pre}(x) > \frac{9(n-m)}{10} \wedge x \text{ valid} \\ -\textsc{OneMax}(x) & \text{otherwise.} \end{cases}$$

We note that all search points in the second case have a fitness of at least $n^2m - n - 1$, which is bigger than $n^2(m-1) + n$, an upper bound on the fitness of search points that fall into the first case without having m leading active blocks in the suffix. Hence, search points x where $\textsc{pre}(x) = n - m$ and $\textsc{suff}(x) = \lceil \frac{2}{3}\xi\sqrt{n} \rceil$ represent local optima of second-best overall fitness. The set of global optima equals the points where $\textsc{pre}(x) = 9(n-m)/10$ and $\textsc{suff}(x) = m$, which implies that $(n-m)/10 = \Omega(n)$ bits have to be flipped simultaneously to escape from the local toward the global optimum.

The parameter $\xi \geq 1$ controls the target strength that allows the algorithm to find the global optimum with high probability. In the simple setting $\xi = 1$, strength 1 usually leads to the local optimum first while strengths above 2 usually lead directly to the global optimum. Using larger ξ increases the threshold for the strength necessary to find the global optimum instead of being trapped in the local one.

We now formally show with respect to different algorithms that NeedHighMut is challenging to optimize without setting the right mutation probability in advance. We start with an analysis of the classical (1+1) EA, where we for simplicity only show the negative result for $p = 1/n$ even though it would even hold for ξ/n.

Theorem 4.1. *Consider the plain (1+1) EA with mutation probability p on NeedHighMut$_\xi$ for a constant $\xi \geq 1$. If $p = 1/n$ then with probability $1 - 2^{-\Omega(n)}$, its optimization time is $n^{\Omega(n)}$. If $p = (c\xi)/n$ for any constant $c \geq 2$ then the optimization time is $O(n^2)$ with probability $1 - 2^{-\Omega(\sqrt{n})}$.*

For space reasons, we had to omit the proofs of all theorems in this section. They can be found in the preprint [26]. The underlying idea is to apply Chernoff bounds on the number of improving mutations with respect to prefix and suffix. Recalling the discussion above, we expect more prefix-improving than suffix-improving mutations for mutation strength 1 and the opposite for sufficiently large mutation strength $c\xi$. As the definition of NeedHighMut suggest, the small strength then leads to the local optimum, while the global optimum is reached efficiently in the other case (both with high probability). Some technical difficulties have to be overcome to handle steps that simultaneously improve both prefix and suffix.

This can be transferred to the SD-(1+1) EA with stagnation detection, showing that this mechanism does not help to increase the success probability significantly compared to the plain (1+1) EA with $p = 1/n$. The proof shows that the SD-(1+1) EA with high probability does not behave differently from the (1+1) EA. The only major difference is visible after reaching the local optimum of NeedHighMut, where stagnation detection kicks in. This results in the bound $2^{\Omega(n)}$ in the following theorem, compared to $n^{\Omega(n)}$ in the previous one.

Theorem 4.2. *The SD-(1+1) EA needs at least $2^{\Omega(n)}$ steps to optimize NeedHighMut$_\xi$ for $\xi \geq 1$ with probability at least $1 - O(1/n)$.*

Finally, we also show that the self-adaptation scheme of the SASD-(1+λ) EA does not help to concentrate the mutation rate on the right regime for NeedHighMut$_\xi$ if ξ is a sufficiently large constant and λ is not too large. This still applies in connection with stagnation detection.

Theorem 4.3. *Let ξ be a sufficiently large constant and assume $\lambda = o(n)$ and $\lambda = \omega(1)$. Then with probability at least $1 - O(1/n)$, the SASD-(1+λ) EA with stagnation detection (Algorithm 3) needs at least $2^{\Omega(n)}/\lambda$ generations to optimize NeedHighMut$_1$.*

The proof of this theorem uses more advanced techniques, more precisely Theorem 2.2 to analyze the distribution of strength in the offspring over time. This technique allows us that only a small constant fraction of steps uses strength that are more beneficial for the suffix than the prefix.

5 EXPERIMENTS

Our theoretical results are asymptotic. In this section, we show the results of the experiments[1] we did in order to see how the different algorithms perform in practice for small n. All figures are available in [26].

In the first experiment, we ran an implementation of Algorithms 2 (SD-(1+1) EA) and 3 (SASD-(1+λ) EA) on the Jump fitness function with jump size $m = 4$ and n varying from 40 to 160. We compared our algorithms against (1+1) EA with standard mutation rate 1/n, (1+1) EA with mutation probability m/n, and Algorithm (1+1) FEA$_\beta$ from [10] with three different $\beta = \{1.5, 2, 4\}$.

In Figure 1, we observe that stagnation detection technique makes the algorithm faster than the algorithms with heavy-tailed mutation operator (1+1) FEA$_\beta$. Also, Algorithm SD-(1+1) EA is not much slower than the (1+1) EA with mutation probability $\frac{m}{n}$ even though it does not need the gap size.

In a second experiment, we ran our algorithms and the classic (1+1) EA with different mutation probabilities on NeedHighMut$_\xi$ with $n = \{200, 400, 600, 800, 1000\}$ and $\xi = 3$.

The outcomes support that the theory from Section 4 already holds for small n. In additional experimental data, one can see that for $\xi = 3$, the (1+1) EA with $p = 6/n$ and $8/n$ is much more successful to find global optimum points than the rest of the algorithms.

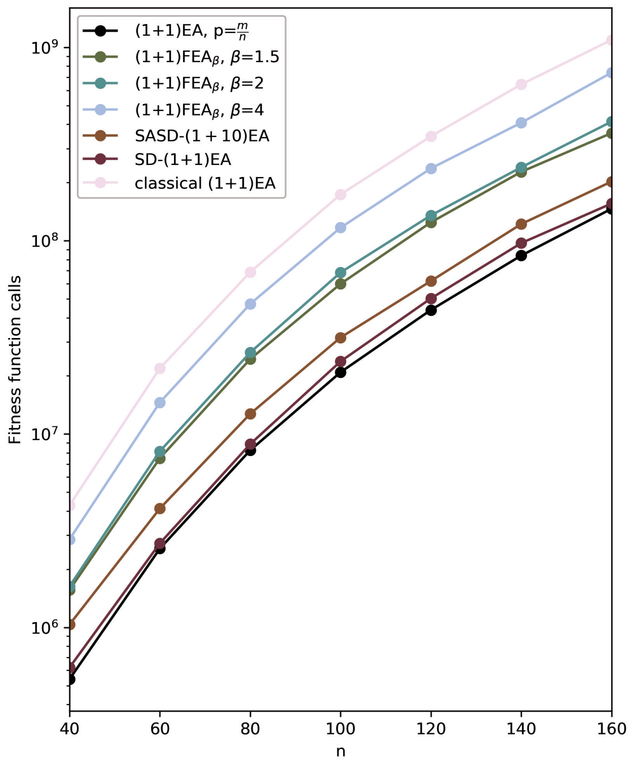

Figure 1: Average number of fitness calls (over 1000 runs) the mentioned algorithms take to optimize Jump$_4$.

CONCLUSIONS

We have designed and analyzed self-adjusting EAs for multimodal optimization. In particular, we have proposed a module called *stagnation detection* that can be added to existing EAs without essentially changing their behavior on unimodal (sub)problems. Our stagnation detection keeps track of the number of unsuccessful steps and increases the mutation rate based on statistically significant waiting times without improvement. Hence, there is high evidence for being at a local optimum when the strength is increased.

Theoretical analyses reveal that the (1+1) EA equipped with stagnation detection optimizes the Jump function in asymptotically optimal time corresponding to the best static choice of the mutation rate. Moreover, we have proved a general upper bound for multimodal functions that can recover asymptotically runtimes on well-known example functions, and we have shown that on unimodal functions, the (1+1) EA with stagnation detection with high probability never deviates from the classical (1+1) EA; also a related statement was proved for the self-adjusting (1+λ) EA from [13]. Finally, to show the limitations of the approach we have presented a function on which all of our investigated self-adjusting EAs provably fail to be efficient.

In the future, we would like to investigate our module for stagnation detection in other EAs and study its benefits on combinatorial optimization problems.

ACKNOWLEDGMENTS

This work was supported by a grant by the Danish Council for Independent Research (DFF-FNU 8021-00260B).

REFERENCES

[1] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. A tight runtime analysis for the (1 + (λ, λ)) GA on LeadingOnes. In *Proc. of FOGA '19*, pages 169–182. ACM Press, 2019.

[1]https://github.com/DTUComputeTONIA/StagnationDetection.

[2] Maxim Buzdalov, Benjamin Doerr, and Mikhail Kever. The unrestricted black-box complexity of jump functions. *Evolutionary Computation*, 24(4):719–744, 2016.

[3] Dogan Corus, Pietro Simone Oliveto, and Donya Yazdani. Fast artificial immune systems. In *Proc. of PPSN '18*, pages 67–78. Springer, 2018.

[4] Duc-Cuong Dang and Per Kristian Lehre. Self-adaptation of mutation rates in non-elitist populations. In *Proc. of PPSN '16*, pages 803–813. Springer, 2016.

[5] Benjamin Doerr. A tight runtime analysis for the cGA on jump functions: EDAs can cross fitness valleys at no extra cost. In *Proc. of GECCO '19*, pages 1488–1496. ACM Press, 2019.

[6] Benjamin Doerr and Carola Doerr. Optimal static and self-adjusting parameter choices for the $(1+(\lambda, \lambda))$ genetic algorithm. *Algorithmica*, 80(5):1658–1709, 2018.

[7] Benjamin Doerr and Carola Doerr. Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In B. Doerr and F. Neumann, editors, *Theory of Evolutionary Computation – Recent Developments in Discrete Optimization*, pages 271–321. Springer, 2020.

[8] Benjamin Doerr and Martin S. Krejca. Significance-based estimation-of-distribution algorithms. In *Proc. of GECCO '18*, pages 1483–1490. ACM Press, 2018.

[9] Benjamin Doerr, Mahmoud Fouz, and Carsten Witt. Quasirandom evolutionary algorithms. In *Proc. of GECCO '10*, pages 1457–1464. ACM Press, 2010.

[10] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Proc. of GECCO '17*, pages 777–784. ACM Press, 2017.

[11] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica*, 80(5): 1732–1768, 2018.

[12] Benjamin Doerr, Carsten Witt, and Jing Yang. Runtime analysis for self-adaptive mutation rates. In *Proc. of GECCO '18*, pages 1475–1482. ACM Press, 2018.

[13] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. The $(1 + \lambda)$ evolutionary algorithm with self-adjusting mutation rate. *Algorithmica*, 81(2):593–631, 2019.

[14] Carola Doerr and Markus Wagner. Sensitivity of parameter control mechanisms with respect to their initialization. In *Proc. of PPSN '18*, pages 360–372. Springer, 2018.

[15] Carola Doerr, Furong Ye, Sander van Rijn, Hao Wang, and Thomas Bäck. Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: Profiling $(1+\lambda)$ EA variants on OneMax and LeadingOnes. In *Proc. of GECCO '18*, page 951–958. ACM Press, 2018.

[16] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[17] A. E. Eiben, Elena Marchiori, and V. A. Valkó. Evolutionary algorithms with on-the-fly population size adjustment. In *Proc. of PPSN '04*, pages 41–50. Springer, 2004.

[18] Mario A. Hevia Fajardo. An empirical evaluation of success-based parameter control mechanisms for evolutionary algorithms. In *Proc. of GECCO '19*, pages 787–795. ACM Press, 2019.

[19] Tobias Friedrich, Francesco Quinzan, and Markus Wagner. Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In *Proc. of GECCO '18*, pages 293–300. ACM Press, 2018.

[20] Bruce Hajek. Hitting and occupation time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 14:502–525, 1982.

[21] Pierre Hansen and Nenad Mladenovic. Variable neighborhood search. In Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende, editors, *Handbook of Heuristics*, pages 759–787. Springer, 2018.

[22] Thomas Jansen and R. Paul Wiegand. The cooperative coevolutionary (1+1) EA. *Evolutionary Computation*, 12(4):405–434, 2004.

[23] Jörg Lässig and Dirk Sudholt. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proc. of FOGA '11*, pages 181–192. ACM Press, 2011.

[24] Johannes Lengler. A general dichotomy of evolutionary algorithms on monotone functions. In *Proc. of PPSN '18*, pages 3–15. Springer, 2018.

[25] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. Simple hyper-heuristics control the neighbourhood size of randomised local search optimally for leadingones. *Evolutionary Computation*, 2020. in print.

[26] Amirhossein Rajabi and Carsten Witt. Self-adjusting evolutionary algorithms for multimodal optimization. *CoRR*, abs/2004.03266, 2020. URL http://arxiv.org/abs/2004.03266.

[27] Anna Rodionova, Kirill Antonov, Arina Buzdalova, and Carola Doerr. Offspring population size matters when comparing evolutionary algorithms with self-adjusting mutation rates. In *Proc. of GECCO '19*, pages 855–863. ACM Press, 2019.

[28] Philipp Rohlfshagen, Per Kristian Lehre, and Xin Yao. Dynamic evolutionary optimisation: an analysis of frequency and magnitude of change. In *Proc. of GECCO '09*, pages 1713–1720. ACM Press, 2009.

[29] Jonathan E. Rowe and Aishwaryaprajna. The benefits and limitations of voting mechanisms in evolutionary optimisation. In *Proc. of FOGA '19*, pages 34–42. ACM Press, 2019.

[30] Ingo Wegener. Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In Ruhul Sarker, Masoud Mohammadian, and Xin Yao, editors, *Evolutionary Optimization*. Kluwer Academic Publishers, 2001.

[31] Darrell Whitley, Swetha Varadarajan, Rachel Hirsch, and Anirban Mukhopadhyay. Exploration and exploitation without mutation: Solving the jump function in $\Theta(n)$ time. In *Proc. of PPSN '18*, pages 55–66. Springer, 2018.

[32] Carsten Witt. Population size vs. runtime of a simple EA. In *Proc. of the Congress on Evolutionary Computation (CEC 2003)*, volume 3, pages 1996–2003. IEEE Press, 2003.

[33] Carsten Witt. Runtime analysis of the $(\mu+1)$ EA on simple pseudo-boolean functions. *Evolutionary Computation*, 14(1):65–86, 2006.

[34] Carsten Witt. Population size versus runtime of a simple evolutionary algorithm. *Theoretical Computer Science*, 403(1):104–120, 2008.

[35] Carsten Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing*, 22: 294–318, 2013.

[36] Furong Ye, Carola Doerr, and Thomas Bäck. Interpolating local and global search by controlling the variance of standard bit mutation. In *Proc. of CEC '19*, pages 2292–2299, 2019.