Simple On-the-Fly Parameter Selection Mechanisms for Two Classical Discrete Black-Box Optimization Benchmark Problems

Carola Doerr
Sorbonne Université, CNRS
Laboratoire d'informatique de Paris 6 (LIP6)
Paris, France

Markus Wagner
The University of Adelaide
Optimisation and Logistics, School of Computer Science
Adelaide, Australia

ABSTRACT

Despite significant empirical and theoretically supported evidence that non-static parameter choices can be strongly beneficial in evolutionary computation, the question *how* to best adjust parameter values plays only a marginal role in contemporary research on discrete black-box optimization. This has led to the unsatisfactory situation in which feedback-free parameter selection rules such as the cooling schedule of Simulated Annealing are predominant in state-of-the-art heuristics, while, at the same time, we understand very well that such time-dependent selection rules can not perform as well as adjustment rules that *do* take into account the evolution of the optimization process. A number of adaptive and self-adaptive parameter control strategies have been proposed in the literature, but did not (yet) make their way to a broader public. A key obstacle seems to lie in their rather complex update rules.

The purpose of our work is to demonstrate that high-performing online parameter selection rules do not have to be very complicated. More precisely, we experiment with a multiplicative, comparison-based update rule to adjust the mutation rate of a (1+1) Evolutionary Algorithm. We show that this simple self-adjusting rule outperforms the best static unary unbiased black-box algorithm on LeadingOnes, achieving an almost optimal speedup of about 18%. We also observe promising performance on the OneMax benchmark problem.

CCS CONCEPTS

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

KEYWORDS

Evolutionary Computation, Parameter Selection, Parameter Control

1 INTRODUCTION

One of the best known randomized black-box optimization techniques is *Simulated Annealing* [20]. Simulated Annealing builds on the *Metropolis* heuristic, a randomized search method that aims to overcome the risk of getting stuck in local optima by allowing the search to continue in points that are worse than the current best solution. The probability to "accept" such an inferior solution depends on the absolute difference in function values and upon a parameter T, which is often referred to as the *temperature* of the system. The improvement of Simulated Annealing over the Metropolis algorithm is a non-static choice of this temperature T. By decreasing T over time, the search algorithm converges from an exploratory behavior to a more and more greedy hill-climber, which exploits the good regions identified during the earlier phases. Numerous successful applications and more than 42,000 citations of [20] witness that this idea to *control* the temperature during the optimization process can have an impressive impact on the performance.

It is today quite well understood that non-static parameter choices can be quite beneficial also for evolutionary computation (EC) methods, cf. [6, 18]. The question *how* to adjust the parameters, however, is largely open. Already the selection of suitable *static* parameter values is a very complex problem that has given rise to a number of sophisticated parameter tuning techniques; it is not for nothing that the parameter selection problem is coined the "Achilles' heel of evolutionary computation" in [14]. Finding methods that automatically detect *and track* optimal parameter values over all stages of the optimization process are the long-term vision of research on *parameter control* techniques.

Adaptive parameter choices are indispensable in continuous optimization, and therefore used in most state-of-the-art heuristics. In *discrete domains*, however, the situation is quite different. A number of different parameter control techniques have been experimented with in the literature, but so far none of them has been able to establish itself as a standard technique. Quite the contrary, the vast majority of research papers on discrete black-box optimization problems use static parameters values [18]. In light of the success story of Simulated Annealing, this situation is quite unsatisfactory.

A potential reason for the discrepancy between the commonly acknowledged benefits of non-static parameter selection schemes and its low degree of utilization may be the complexity of the parameter control techniques that predominate in the EC literature: these are often based on self-adaptation, hyper-heuristics, or principles from machine learning, cf. [18] and [6] for surveys of empirical and theoretical results, respectively.

1.1 Our Results

We analyze in this work one of the simplest ways to incorporate feedback from the optimization process into the selection of suitable parameter values: *success-based multiplicative update mechanisms.*

To abstract away potential inter-dependencies between multiple parameters, we concentrate on the control of a single parameter, the mutation rate of a (1+1) Evolutionary Algorithm (EA) that does not evaluate offspring that are identical to their parents (cf. Section 2 for details and motivation). The update of the mutation rate depends only on whether or not the parent individual is replaced by the offspring. When no improvement is found, the mutation rate p is decreased to bp (with the idea to be more conservative), and it is increased to Ap otherwise (motivated by the desire to make even more progress by searching at a larger distance). A and b are *hyper-parameters* that satisfy $A > 1$ and $0 < b < 1$. This multiplicative control technique is entirely *comparison-based*, a highly desirable property for online parameter selection mechanisms [14].

We investigate the performance of the adaptive (1+1) EA variant, which we call the $(1+1)\,\text{EA}_\alpha$, on OneMax and on LeadingOnes. The experiments on OneMax confirm that, for a very broad range of update strengths, the $(1+1)\,\text{EA}_\alpha$ is capable of *identifying optimal parameter values "on-the-fly"*. This leads to average optimization times that are very close to being optimal among all unary unbiased black-box algorithms.[1]

For OneMax in reasonable problem dimensions $n \leq 20,000$, the relative advantage of non-static parameter choices is only around 2%, and thus not very pronounced. The main task of the $(1+1)\,\text{EA}_\alpha$ is therefore to *identify* good (i.e., in the context of OneMax, low) mutation rates, and to not get distracted by a potential desire to greedily increase the mutation rate during the optimization. In order to investigate the ability of the $(1+1)\,\text{EA}_\alpha$ to not only identify but to also *track optimal parameter values that change quite drastically during the optimization process,* we also regard its performance on the classic LeadingOnes benchmark. For LeadingOnes, the optimal number of bits to flip depends on the fitness of the current-best individual. It is n for x with $\text{Lo}(x) = 0$, $n/2$ for search points of fitness 1, and decreases to 1 for search points of Lo-values $\geq n/2$, cf. Lemma 1. Our empirical results confirm that, again for a broad range of hyper-parameters, the $(1+1)\,\text{EA}_\alpha$ is able to find and to track these optimal mutation rates. About 62% of all 450 tested configurations with $1 < A \leq 2.5$ and $0.4 < b < 1$ and around 39% of all $2,450$ configurations with $1 < A \leq 6$ and $0 < b < 1$ outperform Randomized Local Search, the best unary unbiased black-box algorithm with static mutation rates, by at least 10%. Some configurations achieve an almost optimal advantage of around 18%.

Disclaimer. A common critique of fundamental research on OneMax and similar benchmark problems is that such "sterile" environments are not very representative for typical applications of EAs. Based on the results presented in this work, we, of course, do not know to what extend the advantages of the success-based multiplicative update rule applies to more complex optimization problems. It has been argued, however, that results for OneMax

can serve, at the very least, to verify if some important properties of parameter control techniques are satisfied [13, 26]. In this sense, our results can be seen as a "proof of concept" for the largely unexplored potential of parameter control.

We also wish to point out that multiplicative update rules have of course been studied much prior to this work, e.g., [1, 3, 7, 15, 19, 21]. The appealing aspect of our study lies in the simplicity of the algorithm and problems, which help to nicely illustrate the working principles of this promising parameter update rule. We hope that the convincing and detailed empirical evaluation serves as a motivation to experiment with parameter control techniques. The case of Simulated Annealing has shown that paradigm changes are possible, and we feel that it is time for EC methods to overcome static and feedback-free parameter selection mechanisms.

2 ALGORITHMS AND BENCHMARKS

Our study aims at quantifying the positive effects of online parameter selection. To remove any unwanted side effects, we therefore remove the population size and selective pressure as parameters, and concentrate on adapting the mutation rate in the two classical black-box optimization algorithms Randomized Local Search (RLS) and the (1+1) Evolutionary Algorithm (EA). We use this section to describe the algorithms and benchmark problems studied in our work. For the hasty reader, we wish to point out that we regard a variant of the (1 + 1) EA in which we ensure that an offspring does not equal its direct parent (i.e., we do not allow offspring to be copies of their parent's genotype).

Notation. The description of the algorithms assumes the *maximization* of a pseudo-Boolean function $f : \{0, 1\}^n \to \mathbb{R}$ as optimization task. By $[n]$ we abbreviate the set $\{1, 2, \ldots, n\}$, and we let $[0..r] := \{0\} \cup [r]$.

2.1 RLS and the Resampling (1+1) EA

RLS and the (1 + 1) EA are (1+1) schemes. That is, they always maintain one previously queried solution x, sample from it exactly one offspring y, and use elitist selection; that is, y replaces x if and only if $f(y) \geq f(x)$. The difference of RLS and the (1 + 1) EA lies in the generation of y. While RLS creates the *offspring* y by flipping exactly one bit in x that is chosen uniformly at random, the (1+1) EA creates y by *standard bit mutation*. That is, $y = (y_1, \ldots, y_n)$ is selected by first copying x and then flipping each bit with some positive probability p, independently of all other bits. The parameter p is referred to as the *mutation rate*. A standard choice for p is $1/n$, which results in an expected number of one bit flip per iteration. Put differently, an average iteration of the (1 + 1) EA with mutation rate $1/n$ behaves like an RLS iteration, with the difference that standard bit mutation is a *global variation operator:* in every iteration, every search point $x \in \{0, 1\}^n$ has a positive probability to be sampled. Our main interest is in studying adaptive choices of p, but before we discuss our adaptation rules, we recall one important observation about standard bit mutation.

It is not very difficult to see that standard bit mutation can be identically defined by first choosing a *step size* (also known as *mutation strength*) ℓ from the binomial distribution $\text{Bin}(n, p)$ and then applying the flip_ℓ variation operator (Algorithm 1), which samples

---

[1] We recall that unary unbiased algorithms are those that sample all search points uniformly at random from the whole search space or from unbiased distributions that depend on exactly one previously evaluated search point. A distribution is *unbiased* if it does not discriminate between bit positions, not between bit values [22].

Algorithm 1: flip_ℓ chooses ℓ different positions and flips the entries in these positions.

---

1 **Input:** $x \in \{0,1\}^n$, $\ell \in \mathbb{N}$;
2 Select ℓ different positions $i_1, \ldots, i_\ell \in [n]$ u.a.r.;
3 $y \leftarrow x$;
4 **for** $j = 1, \ldots, \ell$ **do** $y_{i_j} \leftarrow 1 - x_{i_j}$;

ℓ pairwise different indices uniformly at random and creates an offspring y from x by flipping the bits in these ℓ positions and copying the values from x elsewhere. Note that with this description, RLS is the algorithm that uses in each iteration the operator flip_1, i.e., it chooses $\ell = 1$ deterministically.

It was noted in [17, 24, 27] that the literate implementation of standard bit mutation is inadequate for most practical purposes, since the probability that an offspring is identical to its direct parent equals $\text{Bin}(n,p)(0) = (1-p)^n$, which for $p = 1/n$ converges very quickly to $1/e \approx 36.8\%$. Unless we are dealing with very noisy function evaluations, which is not the situation regarded here, such offspring do not advance the optimization process, as they do not carry any new information about the problem instance. An efficient implementation of the $(1+1)$ EA would therefore avoid to generate such offspring. This is easily possible, as all we need to do is to re-sample the step size ℓ from $\text{Bin}(n,p)$ until we get a non-zero value. This is identical to sampling ℓ from the conditional distribution $\text{Bin}_{>0}(n,p)$, which assigns probability 0 to the step size 0 and probability $\text{Bin}_{>0}(n,p)(k) = \binom{n}{k}p^p(1-p)^{n-k}/(1-(1-p)^n)$ to any positive step size $k > 0$. As argued in [24], this resampling strategy seems to be a common implementation of standard bit mutation in plus strategies like the $(1+1)$ EA. To distinguish this interpretation of the $(1+1)$ EA from the one classically regarded in the theory of evolutionary computation literature, it is named $(1+1)$ EA$_{>0}$ in [24].

2.2 Self-Adaptive Mutation Rates

As mentioned in the introduction, our main interest is in the study of performance gains that can be achieved by a non-static choice of the mutation rate p. To this end, we investigate the following simple update rule. If an iteration was successful, i.e., if it produced an offspring y that replaces x, we increase the mutation rate p by a constant multiplicative factor $A \geq 1$. That is, we replace p by Ap if $f(y) \geq f(x)$ holds. If, on the other hand, y is discarded ($f(y) < f(x)$), we decrease p to bp, where $b \leq 1$ is again some fixed constant. We cap the value of p to ensure that it is always greater than $1/n^2$ and at most $1/2$. The initial value of p is set to p_0, for some constant $p_0 > 0$. This gives the $(1+1)$ EA$_\alpha(A, b, p_0)$, which we summarize in Algorithm 2. Note that the $(1+1)$ EA$_\alpha(A = 1, b = 1, p_0)$ is the $(1+1)$ EA$_{>0}$ with static mutation rate $p = p_0$.

In our experiments we compare the performance of the $(1+1)$ EA$_\alpha$ with a variant of RLS that uses a non-static choice of the step size. This variant will be described in Section 3.

2.3 OneMax and LeadingOnes

As benchmark problems, we select OneMax and LeadingOnes, since for these two problems we understand quite well how the

---

Algorithm 2: The $(1+1)$ EA$_\alpha$ with update strengths A and b and initial mutation rate $p_0 \in [1/n^2, 1/2]$ for the maximization of a pseudo-Boolean function $f : \{0,1\}^n \rightarrow \mathbb{R}$

---

1 **Initialization:** Sample $x \in \{0,1\}^n$ uniformly at random and compute $f(x)$;
2 Set $p = p_0$;
3 **Optimization: for** $t = 1, 2, 3, \ldots$ **do**
4   Sample ℓ from $\text{Bin}_{>0}(n,p)$;
5      $y \leftarrow \text{flip}_\ell(x)$;
6   evaluate $f(y)$;
7   **if** $f(y) \geq f(x)$ **then**
8          $x \leftarrow y$ and $p \leftarrow \min\{A \cdot p, 1/2\}$
9   **else**
10          $p \leftarrow \max\{b \cdot p, 1/n^2\}$

---

optimal mutation strengths depend on the state of the optimization process, so that we have a solid baseline against to which we can compare the performance of the $(1+1)$ EA$_\alpha$. OneMax and LeadingOnes are both problems with a unique global optimum $z \in \{0,1\}^n$.

OneMax. For every *target string* $z \in \{0,1\}^n$, the OneMax function Om_z assigns to each search point $x \in \{0,1\}^n$ the number of positions in which x and z agree, i.e., $\text{Om}_z(x) := |\{i \in [n] \mid x_i = z_i\}|$. Maximizing Om_z corresponds to minimizing the Hamming distance between x and z.

It is well known that every mutation-based algorithm (in the unary unbiased sense promoted in [22]) needs $\Omega(n \log n)$ function evaluations, on average, to optimize OneMax. In this asymptotic sense, all RLS and all static $(1+1)$ EA variants considered in this work are optimal, since they are all unary unbiased algorithms and they all achieve a $\Theta(n \log n)$ expected optimization time. We will nevertheless see that the actual (i.e., non-asymptotic) running time can differ substantially for the different algorithms. We will discuss more precise running time statements in Section 3 below.

LeadingOnes is the problem of optimizing an unknown function of the type $\text{Lo}_{z,\sigma} : \{0,1\}^n \rightarrow \mathbb{R}, x \mapsto \max\{i \in [0..n] \mid \forall j \in [i] : x_{\sigma(i)} = z_{\sigma(i)}\}$, where z is an unknown length-n bit string and $\sigma : [n] \rightarrow [n]$ an unknown permutation (one-to-one map) of the positions. That is, $\text{Lo}_{z,\sigma}(x)$ is the length of the longest common prefix between z and x in the order determined by σ.

Every (1+1) elitist [11] and every unary unbiased [22] black-box algorithm needs $\Omega(n^2)$ function evaluations, on average, to optimize LeadingOnes. This bound is matched by RLS and the $(1+1)$ EA, as we shall discuss in the next section.

3 THEORETICAL PERFORMANCE LIMITS

To establish bounds against which we can compare the $(1+1)$ EA$_\alpha$, we now take a closer look at the best possible performance that any mutation-based algorithm can achieve on OneMax and LeadingOnes. In both cases, this performance is obtained by a variant of RLS that replaces the static choice $\ell = 1$ classically used by RLS by a fitness-dependent step size ℓ. More precisely, it is known that, for OneMax, such an RLS variant has a performance that can not be worse than an optimal unary unbiased black-box algorithm by

more than an additive $o(n)$ term [9]. For LeadingOnes a similar statement can be derived from the methods introduced in [2, 9], cf. Section 3.2.

3.1 OneMax

As mentioned above and summarized in [9], we know quite well how RLS and the $(1 + 1)$ EA perform on the OneMax problem. From the known bounds, we can compute theoretical performance limits of the $(1 + 1)$ EA$_\alpha$. This is the focus of this section.

For **static parameter values,** i.e., for arbitrary $p_0 \in [1/n^2, 1/2]$ and $A = b = 1$, the expected optimization time of the $(1 + 1)$ EA$_\alpha$ on OneMax cannot be better than that of RLS, which is equal to $n \ln(n) + (\gamma - \ln(2)) + o(1) \approx n \ln n - 0.1159n$ [4] ($\gamma = 0.5772 \ldots$ denotes the Euler-Mascheroni constant). Likewise, for **adaptive parameter values** (i.e., for arbitrary values of A, b, and p_0) the expected optimization time of the $(1+1)$ EA$_\alpha$ is bounded from below by the performance of a best unary unbiased black-box algorithm, which satisfies $n \ln(n) - \alpha n \pm o(n)$ for a constant α that is between 0.2539 and 0.2665 [9].

The above-mentioned values are asymptotically optimal running times. In order to obtain *absolute performance limits for concrete problem dimensions,* we regard the *drift-maximizing* RLS variant studied in [9]. Although it cannot be formally proven that this algorithm is indeed optimal, the result in [9] states that its performance cannot be much worse than that of the best possible unary unbiased (i.e., mutation-based) black-box algorithm. We even conjecture that the drift-maximizing RLS, which we call RLS$_{\text{opt,OM}}$, *is* indeed optimal within this class.

RLS$_{\text{opt,OM}}$ is the RLS variant that modifies the best-so-far solution x by applying to it the variation operator flip$_\ell$ for a value of ℓ that maximizes the expected progress that can be obtained in one iteration. This expected progress is often referred to as *drift,* hence the name "drift maximizer". It is not difficult to see that the expected progress $\mathbb{E}[\max\{\text{Om}(\text{flip}_\ell(x)) - \text{Om}(x), 0\}]$ of flip$_\ell$ applied to x equals

$$\sum_{i=\lceil \ell/2 \rceil}^{\ell} \frac{\binom{n-\text{Om}(x)}{i}\binom{\text{Om}(x)}{\ell-i}(2i-\ell)}{\binom{n}{\ell}}. \quad (1)$$

This expression depends only on the problem dimension n and the function value $\text{Om}(x)$, but not on the structure of the search point x. For every n and every possible function value $f \in [0..n]$, we can therefore abbreviate the progress-maximizing choice of ℓ by $k_{\text{opt,OM}}(n, f)$. With this abbreviation, RLS$_{\text{opt,OM}}$ is Algorithm 3.

As proven in [9], the value of $k_{\text{opt,OM}}(n, f)$ equals 1 whenever $f \geq 2n/3$. For general f, however, we do not have a simple to evaluate closed form expression to describe $k_{\text{opt,OM}}(n, f)$. For this reason an approximation of $k_{\text{opt,OM}}(n, f)$ is used in [9]. Since here in this work we are not interested in asymptotic bounds, but rather absolute values for concrete problem dimensions, we do not need to approximate $k_{\text{opt,OM}}(n, f)$ but can work with the exact drift maximizing choice. Using these values, we can evaluate the expected performance of RLS$_{\text{opt,OM}}$ empirically. This is our approach for the results presented in Section 4. These empirical averages are quite close to the above-mentioned asymptotic lower bound presented in [9].

Algorithm 3: The drift-maximizing algorithm RLS$_{\text{opt,OM}}$

---

1 Sample $x \in \{0, 1\}^n$ uniformly at random and compute $\text{Om}(x)$;
2 **for** $t = 1, 2, 3, \ldots$ **do**
3 $\quad$ $\ell \leftarrow k_{\text{opt,OM}}(n, \text{Om}(x))$;
4 $\quad$ $y \leftarrow \text{flip}_\ell(x)$;
5 $\quad$ **if** $f(y) \geq f(x)$ **then** $x \leftarrow y$;

3.2 LeadingOnes

For LeadingOnes the known theoretical bounds are as follows. RLS needs $1 + n^2/2$ function evaluations, on average, for its optimization. This bound is also a lower bound for the $(1 + 1)$ EA$_\alpha$ with **static parameter choices**; i.e., for arbitrary $p_0 \in [1/n^2, 1/2]$ and $A = b = 1$. For the classical $(1 + 1)$ EA, which samples ℓ from the unconditional binomial distribution $\text{Bin}(n, p)$ (and may therefore sample $\ell = 0$), the best static choice is $p \approx 1.59n$, which gives an expected optimization time of about $0.77n^2$, while for the $(1 + 1)$ EA$_{>0}$ it holds that the smaller the mutation rate, the better performance we obtain [16]. More precisely, it holds that the expected running time of the $(1 + 1)$ EA$_{>0}$ converges to $1 + n^2/2$ when the mutation rate p converges to zero.

In [2], also optimal **adaptive mutation rates** have been computed for the classical $(1+1)$ EA. It is shown there that the $(1+1)$ EA using at each point x the optimal mutation rate $(n - \text{Lo}(x))/n$ has an expected running time on LeadingOnes of $0.68n^2 \pm O(n)$.

The adaptive $(1 + 1)$ EA variant from [2] clearly looses performance for iterations in which the offspring equals its parent. It is therefore natural to ask for the best performance that a—possibly adaptive—unary unbiased black-box algorithm can achieve. As far as we know, such a best-possible mutation-based algorithm has not been explicitly reported in the literature. It turns out, however, that we can generalize mathematical statements proven in [2] and [9] to design such an optimal unary unbiased black-box algorithm for LeadingOnes.

Before describing this algorithm in detail, we note that already allowing 1- and 2-bitflips (i.e., flip$_\ell$ with $\ell = 1$ and $\ell = 2$) decreases the optimal $1 + n^2/2$ expected optimization time of static unary unbiased algorithms to about $0.4233n^2$ [23] (the fact that flip$_\ell$ is defined slightly different in [23] has a negligible impact on this result). This running time can be further reduced by allowing larger step sizes. We investigate the limits of this approach in the remainder of this section.

To compute the progress-maximizing variant of RLS, we could, similarly to the OneMax case, compute the expected progress of flip$_\ell$ when applied to a search point x. For LeadingOnes, however, it suffices to maximize the *probability* of making progress [2], which is, in general, much easier than computing the expected progress. When applied to a search point x, the offspring y created from x by flipping ℓ bits satisfies $\text{Lo}(y) > \text{Lo}(x)$ if and only if the $(\text{Lo}(x)+1)$-st bit is flipped but none of the first $\text{Lo}(x)$ bits. The probability of this event, for uniformly chosen bit flips, equals

$$\mathbb{P}\left[\text{Lo}(\text{flip}_\ell(x)) > \text{Lo}(x)\right] = \frac{\ell}{n - \text{Lo}(x)}\binom{n - \text{Lo}(x)}{\ell}\Big/\binom{n}{\ell} \quad (2)$$

$$= \binom{n - \text{Lo}(x) - 1}{\ell - 1}\Big/\binom{n}{\ell}.$$

In line with the notation used for the OneMax case, we abbreviate the value ℓ that maximizes expression (2) by $k_{\text{opt,LO}}(n, \text{Lo}(x))$. The following lemma seems to be well known in the theory of evolutionary computation community, but, as far as we know, it has not been mentioned explicitly.

Lemma 1 ($k_{\text{OPT,LO}}(n, \text{Lo}(x))$). *For all $n \in \mathbb{N}$ and for all $x \in \{0, 1\}^n$ it holds that $k_{opt,LO}(n, \text{Lo}(x)) = \lfloor n/(\text{Lo}(x) + 1) \rfloor$.*

With these values, we can study the expected running time of $\text{RLS}_{\text{opt,LO}}$, which is Algorithm 3 with line 2 replaced by "$\ell \leftarrow k_{\text{opt,LO}}(n, \text{Lo}(x))$". Combining Lemma 1 with the characterization of unary unbiased mutation operators provided in [9, Lemma 1] and an extension of the results proven in [2] to unary unbiased black-box algorithms, it is not difficult to show the following theorem, which, intuitively speaking, states that $\text{RLS}_{\text{opt,LO}}$ is optimal among all mutation-based black-box algorithms for LeadingOnes. Furthermore, this optimality does not only apply to the overall optimization time, but also to all intermediate target values. We thus obtain the following statement about the *fixed-target performance* of $\text{RLS}_{\text{opt,LO}}$.

Theorem 2. *For LeadingOnes, the expected number of function evaluations needed by $\text{RLS}_{opt,LO}$ to obtain a search point of function value at least i equals*

$$T(\text{RLS}_{opt,LO}, Lo, i) := 1 + \frac{1}{2} \sum_{j=0}^{i-1} \binom{n}{k_{opt,LO}(n,j)} \Big/ \binom{n-j-1}{k_{opt,LO}(n,j)-1}$$

$$= 1 + \frac{1}{2} \sum_{j=0}^{i-1} \binom{n}{\lfloor n/(j+1) \rfloor} \Big/ \binom{n-j-1}{\lfloor n/(j+1) \rfloor - 1}.$$

For all n and all i this performance is optimal among all unary unbiased black-box algorithms. That is, for any unary unbiased black-box algorithm A the expected time needed by A to reach a search point of LeadingOnes value $\geq i$ is at least as large as that of $\text{RLS}_{opt,LO}$.

We did not find an easy to evaluate closed form for $T(\text{RLS}_{\text{opt,LO}}, \text{Lo}, i)$. We can nevertheless evaluate this sum numerically, and obtain that for $n \rightarrow \infty$ the expected optimization time of $\text{RLS}_{\text{opt,LO}}$ seems to converge to around $0.388...n^2$, cf. Table 1. For $n = 10,000$ the value is still around $0.3884n^2$.

4 SUMMARY OF EXPERIMENTAL RESULTS

With the description of the algorithms and benchmark problems in place, we can now report our empirical results for the $(1 + 1) \text{ EA}_\alpha$.

4.1 Grid search

In a first step, we are interested in the sensitivity of the $(1 + 1) \text{ EA}_\alpha$ with respect to the two update strengths A and b. To analyze the influence of these two *hyper-parameters*, we perform a grid search, in which we run the $(1 + 1) \text{ EA}_\alpha$ for various combinations of A and b. For these experiments, we always initialize the mutation rate as $p_0 = 1/n$. In Figures 1a to 1f we report for each configuration the average optimization times of 101 independent runs of these algorithms on OneMax (Figures 1a and 1b) and on LeadingOnes (Figures 1d, 1e, 1f). In these plots, the parameters are chosen as follows. For OneMax and for LeadingOnes with $n \leq 250$, we vary A between 1.0 and 6.0, in multiples of 0.1, and we choose b between

0.00 and 1.00, in multiples of 0.02. For LeadingOnes with $n = 500$ we restrict the values to $1 \leq A \leq 3$ and $0.4 \leq b \leq 1$. That is, Figure 1f is a zoom into the upper left region of the full grid search.

These graphics are to be read as follows. As so-called *heat maps*, we visualize in color the optimization times of the $(1 + 1) \text{ EA}_\alpha$ variants; that is, we plot the average number of fitness evaluations that these algorithms needed in order to locate the global optimum. We use a binning of colors to emphasize the visibility of regions and gradients. For example, the large green regions indicate the configurations for which the $(1 + 1) \text{ EA}_\alpha$ performs best. Also, the quick succession of colors in the top right corner shows that, beyond some threshold values for A and b, small changes in the configuration can cause large changes in the performance.

For OneMax we first observe that the heat maps have a very similar structure across the different dimensions, cf. Figures 1a and 1b. In addition to the results shown in Figure 1 we also computed heat maps for OneMax with problem dimension $n = 100$, $n = 1,000$, and $n = 2,000$ and for each of these heat maps the overall structure is very similar to that plotted in Figures 1a and 1b. Also for LeadingOnes the figures are quite similar across the dimensions, cf. Figures 1d, 1e, and 1f (recall that Figure 1f is a zoom into upper left corner). We also observe that the performance landscapes for OneMax is quite flat; i.e., the bulk of the configurations achieves a very similar performance.

For both problems, OneMax and LeadingOnes, we see that for large values of A and b the average optimization times become worse very quickly. In Figure 1c we therefore zoom into the most interesting region of high-performing configurations and display only results for configurations that achieve an average optimization time that is at most 150,000 (for comparison, the $(1+1) \text{ EA}_{>0}$ needs around 135,700 iterations, on average, on this problem instance, and RLS 125,000 iterations, cf. Table 1). This zoom increases the granularity of the color scheme, and allows to detect more structure within this region.

As a next step, we compare the average running times with those of the $(1+1) \text{ EA}_{>0}$ and RLS. We observe that a significant number of configurations outperform the static $(1+1) \text{ EA}_{>0}$. For LeadingOnes, the average optimization times of different algorithms can be found in Table 1 and for OneMax we note that the average optimization time of the $(1 + 1) \text{ EA}_{>0}$ is around 4,750 for $n = 500$ and about 16,630 for $n = 1,500$. Several configurations also outperform RLS, but for OneMax the 101 runs do not suffice to make a statistically sound comparison, since the advantage of adaptive step sizes is bounded by around 2% for all tested dimensions. For LeadingOnes, however, the advantages over RLS are quite significant, as Figure 2 demonstrates. In this plot, the lowermost two lines illustrate the fraction of all 2,450 configurations with $1 < A \leq 6$ and $0 < b < 1$ that yield a better average performance on LeadingOnes than RLS. In this figure, the x-axis indicates the percentage by which the algorithms are better than RLS, and on the y-axis we plot the fraction of the configurations that outperform RLS by at least this much. That is, we see that among all 2,450 configurations around 67% have an average optimization time below $n^2/2$. Between 37% ($n = 100$) and 41% (n=250) of all configurations are better than RLS by at least 10%.

When we restrict the configurations to those 450 that satisfy $1 < A \leq 2.5$ and $0.4 \leq b < 1$ (three uppermost lines), around 78%
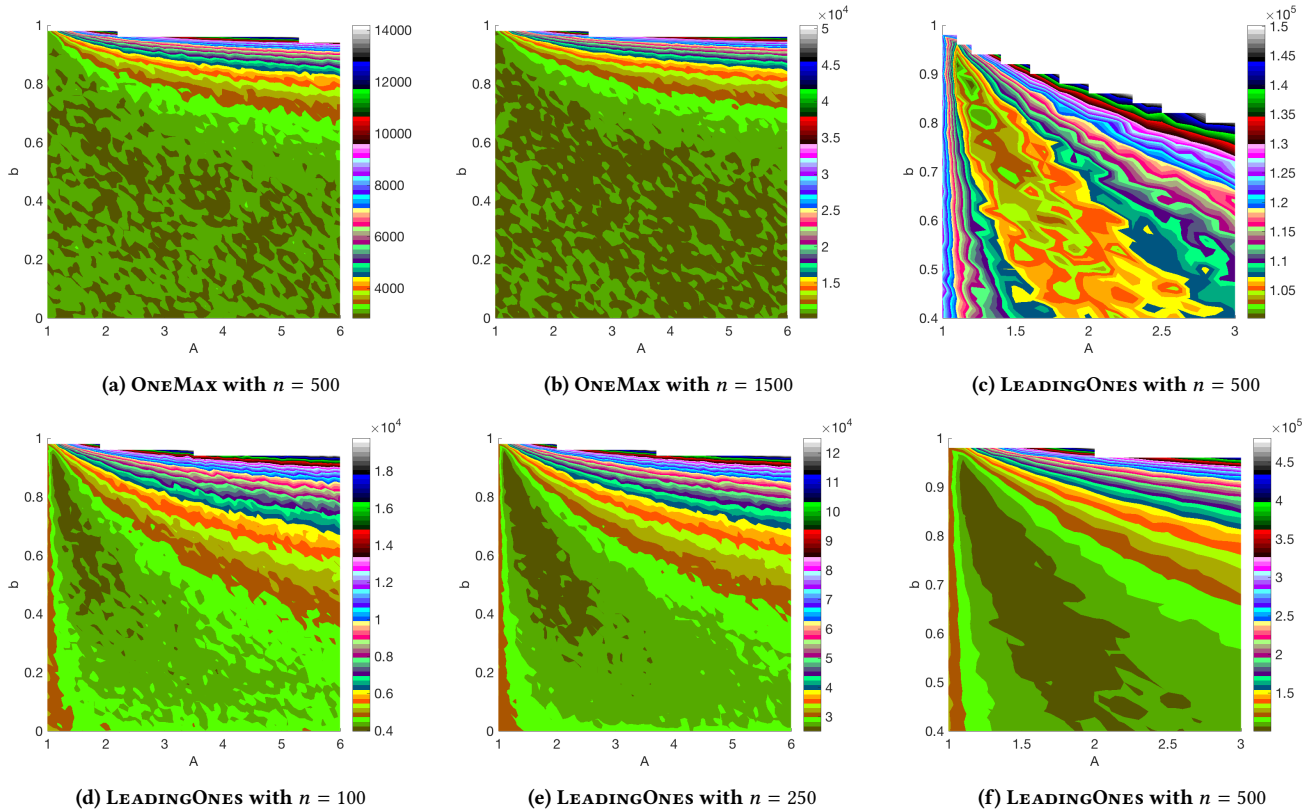
(a) ONEMAX with $n = 500$

(b) ONEMAX with $n = 1500$

(c) LEADINGONES with $n = 500$

(d) LEADINGONES with $n = 100$

(e) LEADINGONES with $n = 250$

(f) LEADINGONES with $n = 500$

Figure 1: Optimization times of the $(1 + 1)$ EA$_\alpha$ for different update strengths A and b, averaged over 101 independent runs

of them are better than RLS, around 62% are better by at least 10%, around 30% outperform RLS by at least 15%, and still almost 20% of the configurations are better by at least 16%. From an algorithm design point of view this is very good news: finding good hyperparameters is not very difficult for this problem. We also see that the numbers are very similar across all three tested dimensions $n = 100, 250, 500$, indicating that this surprisingly good performance might translate to larger dimensions.

As we have proven in Section 3, the best possible running time on LEADINGONES (among all unary unbiased black-box algorithms, and hence in particular among all $(1+1)$ EA and RLS variants) is achieved by RLS$_{\mathrm{opt,LO}}$. For the considered problem dimensions, RLS$_{\mathrm{opt,LO}}$ is better, in expectation, than RLS by around 22.3%, cf. Table 1. This advantage over RLS is of course also the maximal improvement that any $(1+1)$ EA$_\alpha$ variant can achieve over RLS. For all dimensions, we observe that between 2 and 4% of the configurations are better than RLS by at least 18%. No configuration achieves a 20% improvement. Note here that this does not come as a surprise: even if the $(1+1)$ EA chooses in each iteration the for this state optimal mutation rate, its performance still suffers from the random choice of the step sizes. This risk is eliminated in RLS and RLS$_{\mathrm{opt}}$ by the deterministic choice of the mutation strength $\ell = 1$ and $\ell = k_{\mathrm{opt,LO}}(n, \mathrm{LO}(x))$, respectively. We can therefore not expect any configuration, or, more generally, any evolutionary algorithm, to achieve the same performance as RLS$_{\mathrm{opt,LO}}$.
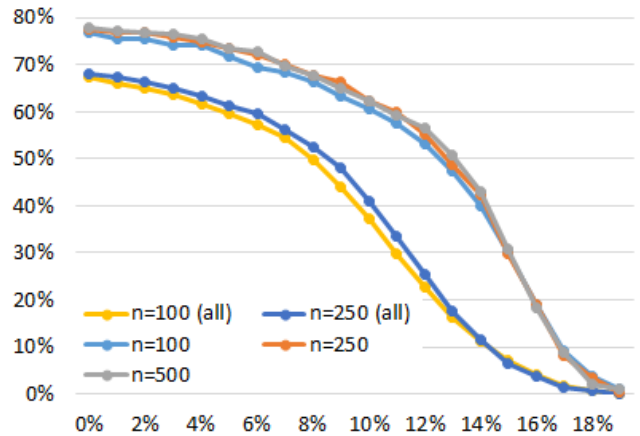

Figure 2: Fraction of all 2,450 (450) configurations with $1 < A \le 6$ and $0 < b < 1$ ($1 < A \le 2.5$ and $0.4 \le b < 1$) that outperform RLS on LEADINGONES by at least $x\%$

With respect to the $(1 + 1)$ EA$_{>0}$, the comparison is even more impressive. About 73% of all 2,450 configurations and around 82% of the 450 restricted configurations with $1 < A \le 2.5$ and $0.4 \le b < 1$ have a better performance on LEADINGONES than the $(1 + 1)$ EA$_{>0}$.

About 64% (76% for the restricted hyper-parameters) are better by at least 10% and still around 14% (47%) are better, on average, by more than 20%. The best improvements over the $(1 + 1)$ EA$_{>0}$ are around 26%, but we should keep in mind here that the numbers are averages for 101 independent runs only. We will therefore do a more thorough investigation of selected configurations in the next subsection, but note here already that we see a quite consistent behavior in Figure 2, raising our confidence that these results are not much skewed by the relatively low number of independent runs.

For OneMax, around 90% of all configurations perform better than the $(1 + 1)$ EA$_{>0}$, and between 64% ($n = 100$) and 83% ($n = 2,000$) are better by at least 30%. The best configurations achieve an improvement of up to around 40%.

4.2 Selected Configurations

To substantiate the comparisons made in Section 4.1 and to investigate how the results translate to larger problem dimensions, we now take a closer look at some selected configurations.

For LeadingOnes, the results of this comparison are summarized in Table 1, where we report both the average optimization times T and the relative values T/n^2. For RLS and RLS$_{\text{opt,LO}}$ the numbers in Table 1 are computed from the exact running time statements (cf. Section 3), which are all indeed very close to the results that we obtain empirically. All other numbers in Table 1 are averages over 1,001 and 101 (starred values) independent runs, respectively.

In addition to the $(1 + 1)$ EA$_{>0}$, we list 4 different configurations. The $(1 + 1)$ EA$_{\alpha}$ with $A = 1.3, b = 0.75$ and with $A = 1.2, b = 0.85$ are examples for configurations that show a good (but not empirically best) performance in the grid search conducted in Section 4.1. We also add to the comparison the configuration $A = 2, b = 0.5$, a seemingly intuitive configuration used also in different context, e.g., in the adaptive choice of the population size in [21]. The configuration $A = 1.11, b = 0.66$ corresponds to the 1/5-th success rule with update strength 1.5, a very common adaptation rule in continuous domain, cf. [1, 5, 19] and references therein.

We observe that the performances are quite stable over the tested dimensions. The two selected configurations $A = 1.3, b = 0.75$ and $A = 1.2, b = 0.85$ are better by around 18% than RLS, on all tested dimensions. The doubling/halving rule $A = 2, b = 0.5$ achieves a $15 - 17\%$ performance gain over RLS, while the 1/5-th success rule with $A = 1.11, b = 0.66$ achieves an improvement over RLS of about $7 - 10\%$.

As an important consequence, the results of this section suggest that a tuning of the hyper-parameters A and b on smaller dimension is possible. In addition, the stability of the results indicates that similar results as those presented in Section 4.1 are likely to apply also to larger problem dimensions.

For OneMax, we conducted experiments for the same configurations as in Table 1, for problem dimensions up to $n = 3,000$. The results are similar to those for LeadingOnes in that also for OneMax the relative performance gains observed in small dimensions seem to transfer to larger ones. It is remarkable that all the selected configurations, on average over 1,001 independent runs, achieve a performance that is very close to that of RLS, and in some cases even outperform it. The following table summarizes
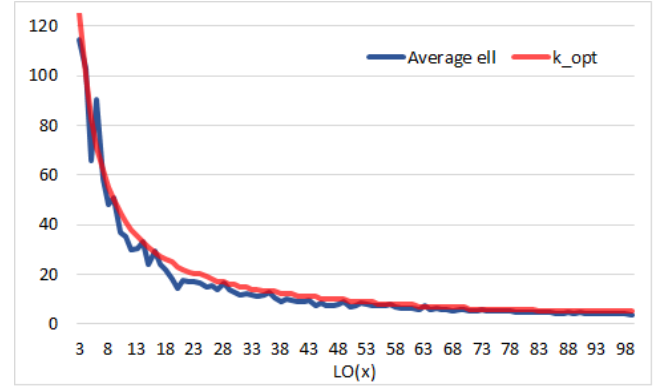

Figure 3: Average and optimal mutation strengths for different Lo(x) values ($n = 500$, 10 independent runs of the $(1 + 1)$ EA$_{\alpha}$ with $A = 1.2, b = 0.85$, and $p_0 = 1/n$)

selected results. For convenience, we also compute for each empirical average T the value $c := 100T/(n \ln n)$. Starred results are for 101 independent runs, all other results are averages over 1,001 runs.

Algorithm	$n = 500$		$n = 1000$		$n = 3000$	
	T	c	T	c	T	c
RLS	3,050	98.16	6,871	99.47	23,814	99.15
RLS$_{\text{opt}}$	2,974	95.72	6,690	96.85	23,507	97.87
$(1+1)$ EA$_{>0}$	4,756	153.06	10,574	153.07	37,256	155.11
$(1+1)$ EA$_{\alpha}$ (A=1.2, b=0.85)	3,059	98.45	6,751	97.73	23,558*	98.08
$(1+1)$ EA$_{\alpha}$ (A=1.3, b=0.75)	3,033	97.61	6,801	98.45	23,715*	98.73
$(1+1)$ EA$_{\alpha}$ (A=2.0, b=0.5)	3,013	96.97	6,753	97.76	23,027*	95.87
$(1+1)$ EA$_{\alpha}$ (A=1.11, b=0.66)	3,039	97.80	6,749	97.70	24,011*	99.97

4.3 Zooming into Typical Runs

We finally want to understand how well the selected mutation rates resemble the optimal ones. Our benchmark problem is the 500-dimensional LeadingOnes function. We store for 10 independent runs of the $(1 + 1)$ EA$_{\alpha}$ with $A = 1.2$ and $b = 0.85$ and for each iteration the Lo-value of a best-so-far solution along with the mutation strength ℓ that has been chosen in this iteration, i.e., the number of bits that have been flipped by the flip$_{\ell}$ operator to create the offspring of this iteration. We then average for each function value Lo(x) over the ℓ-values that have been used in iterations that started with this fitness value. Figure 3 plots these averages for $3 \leq$ Lo$(x) < 100$ (blue, ragged curve), along with the corresponding $k_{\text{opt,LO}}(n, \text{Lo}(x))$ values (smooth red curve). The $(1 + 1)$ EA$_{\alpha}$ seems to sample indeed almost optimal mutation strengths. For Lo$(x) \in \{0, 1, 2\}$ we note that the values are smaller than $k_{\text{opt,LO}}(n, \text{Lo}(x))$, but this is explained by the initialization of the mutation rate with $p_0 = 1/n$, which forces the algorithm to first increase this rate to a close-to-optimal value. This process decreases the average considerably. For values Lo$(x) \geq 100$, the two curves are almost indistinguishable and have therefore been removed from the illustration.

5 CONCLUSIONS

We hope to contribute with our work to a more widespread experimentation and use of non-static parameter selection mechanisms in discrete optimization contexts. We are confident that significant

Algorithm	$n=100$		$n=250$		$n=500$		$n=1000$		$n=1500$	
	T	T/n^2	T	T/n^2	T	T/n^2	T	T/n^2	T	T/n^2
$\text{RLS}_{\text{opt,LO}}$	3,883	38.83%	24,273	38.8368%	97,102	38.8408%	388,427	38.8427%	873,981	38.8436%
RLS	5,001	50.01%	31,251	50.0016%	125,001	50.0004%	500,001	50.0001%	1,125,001	50.0000%
$(1+1)\,\text{EA}_{>0}$	5,401	54.01%	33,817	54.1072%	135,782	54.3128%	544,288	54.4288%	1,216,448*	54.0644%
$(1+1)\,\text{EA}_{\alpha}(\text{A=1.2, b=0.85})$	4,063	40.63%	25,497	40.7952%	101,976	40.7904%	409,820*	40.9820%	921,900*	40.9733%
$(1+1)\,\text{EA}_{\alpha}(\text{A=1.3, b=0.75})$	4,185	41.85%	25,731	41.1696%	102,985	41.1940%	413,518*	41.3518%	931,313*	41.3917%
$(1+1)\,\text{EA}_{\alpha}(\text{A=2.0, b=0.5})$	4,195	41.95%	26,247	41.9952%	104,193*	41.6772%	416,362*	41.6362%	932,791*	41.4574%
$(1+1)\,\text{EA}_{\alpha}(\text{A=1.11, b=0.66})$	4,495	44.95%	28,277	45.2432%	114,814*	45.9256%	448,815*	44.8815%	1,016,830*	45.1924%

Table 1: Average optimization times for LeadingOnes. Exact bounds for RLS and $\text{RLS}_{\text{opt,LO}}$, empirical averages over 1,001 (*=101) independent runs otherwise

performance gains are possible, for a broad number of applications. We have shown in this work that already quite simple parameter control mechanisms can give almost optimal performance. On LeadingOnes, significant performance gains over the best static parameter values were possible for a broad range of multiplicative update rules.

Much more sophisticated parameter control techniques, including a number of portfolio-based methods inspired by the multi-armed bandit literature have been proposed and analyzed in the literature, by experimental [12, 13, 25] and theoretical [8, 23] means. Developing a rigorous understanding of which update scheme to favor under which circumstances is the ultimate goal of our research. First comparisons with the above-mentioned techniques are quite favorable for the success-based multiplicative update rule; a rigorous comparison is left for future work.

As a more immediate research question, it would be desirable to understand how the performance of the $(1+1)\,\text{EA}_{\alpha}$ scales with very large dimensions, by means of an empirical comparison and/or a mathematical running time analysis.

REFERENCES

[1] Anne Auger. 2009. Benchmarking the (1+1) evolution strategy with one-fifth success rule on the BBOB-2009 function testbed. In *Companion Material GECCO'09*. ACM, 2447–2452.

[2] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. 2010. Optimal Fixed and Adaptive Mutation Rates for the LeadingOnes Problem. In *PPSN'10 (LNCS)*, Vol. 6238. Springer, 1–10.

[3] Benjamin Doerr and Carola Doerr. 2015. Optimal Parameter Choices Through Self-Adjustment: Applying the 1/5-th Rule in Discrete Settings. In *GECCO'15*. ACM, 1335–1342.

[4] Benjamin Doerr and Carola Doerr. 2016. The Impact of Random Initialization on the Runtime of Randomized Search Heuristics. *Algorithmica* 75 (2016), 529–553.

[5] Benjamin Doerr and Carola Doerr. 2018. Optimal Static and Self-Adjusting Parameter Choices for the $(1+(\lambda,\lambda))$ Genetic Algorithm. *Algorithmica* 80 (2018), 1658–1709.

[6] Benjamin Doerr and Carola Doerr. 2018. Theory of Parameter Control Mechanisms for Discrete Black-Box Optimization: Provable Performance Gains Through Dynamic Parameter Choices. In *Theory of Randomized Search Heuristics in Discrete Search Spaces*, Benjamin Doerr and Frank Neumann (Eds.). Springer. To

appear.

[7] Benjamin Doerr, Carola Doerr, and Timo Kötzing. 2016. Provably Optimal Self-Adjusting Step Sizes for Multi-Valued Decision Variables. In *PPSN'16 (LNCS)*, Vol. 9921. Springer, 782–791.

[8] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. k-Bit Mutation with Self-Adjusting k Outperforms Standard Bit Mutation. In *PPSN'16 (LNCS)*, Vol. 9921. Springer, 824–834.

[9] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. Optimal Parameter Choices via Precise Black-Box Analysis. In *GECCO'16*. ACM, 1123–1130.

[10] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. 2017. The $(1+\lambda)$ Evolutionary Algorithm with Self-Adjusting Mutation Rate. In *GECCO'17*. ACM, 1351–1358.

[11] Carola Doerr and Johannes Lengler. 2016. The (1+1) Elitist Black-Box Complexity of LeadingOnes. In *GECCO'16*. ACM, 1131–1138.

[12] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. 2008. Extreme Value Based Adaptive Operator Selection. In *PPSN'08 (LNCS)*, Vol. 5199. Springer, 175–184.

[13] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. 2009. Dynamic Multi-Armed Bandits and Extreme Value-Based Rewards for Adaptive Operator Selection in Evolutionary Algorithms. In *LION'09 (LNCS)*, Vol. 5851. Springer, 176–190.

[14] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. 2010. Analyzing bandit-based adaptive operator selection mechanisms. *Annals of Mathematics and Artificial Intelligence* 60 (2010), 25–64.

[15] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. 2005. On the Choice of the Offspring Population Size in Evolutionary Algorithms. *Evolutionary Computation* 13 (2005), 413–440.

[16] Thomas Jansen and Christine Zarges. 2011. Analysis of evolutionary algorithms: from computational complexity analysis to algorithm engineering. In *FOGA'11*. ACM, 1–14.

[17] Thomas Jansen and Christine Zarges. 2014. Performance analysis of randomised search heuristics operating with a fixed budget. *Theoretical Computer Science* 545 (2014), 39–58.

[18] G. Karafotias, M. Hoogendoorn, and A.E. Eiben. 2015. Parameter Control in Evolutionary Algorithms: Trends and Challenges. *IEEE Transactions on Evolutionary Computation* 19 (2015), 167–187.

[19] Stefan Kern, Sibylle D. Müller, Nikolaus Hansen, Dirk Büche, Jiri Ocenasek, and Petros Koumoutsakos. 2004. Learning probability distributions in continuous evolutionary algorithms - a comparative review. *Natural Computing* 3 (2004), 77–112.

[20] Scott Kirkpatrick, C. D. Gelatt, and Mario P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680.

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

[22] Per Kristian Lehre and Carsten Witt. 2012. Black-Box Search by Unbiased Variation. *Algorithmica* 64 (2012), 623–642.

[23] Andrei Lissovoi, Pietro Simone Oliveto, and John Alasdair Warwicker. 2017. On the runtime analysis of generalised selection hyper-heuristics for pseudo-boolean optimisation. In *GECCO'17*. ACM, 849–856.

[24] Eduardo Carvalho Pinto and Carola Doerr. 2017. Discussion of a More Practice-Aware Runtime Analysis for Evolutionary Algorithms. In *EA'17*. 298–305.

[25] Dirk Thierens. 2005. An Adaptive Pursuit Strategy for Allocating Operator Probabilities. In *GECCO'05*. ACM, 1539–1546.

[26] Dirk Thierens. 2009. On benchmark properties for adaptive operator selection. In *Companion Material GECCO'09*. ACM, 2217–2218.

[27] Jano I. van Hemert and Thomas Bäck. 2002. Measuring the Searched Space to Guide Efficiency: The Principle and Evidence on Constraint Satisfaction. In *PPSN'02 (LNCS)*, Vol. 2439. Springer, 23–32.