Maximizing Drift is Not Optimal for Solving OneMax

Nathan Buskulic, Carola Doerr

Maximizing Drift is Not Optimal for Solving OneMax

Nathan Buskulic[1] and Carola Doerr[1]

[1]Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, Paris, France

January 18, 2021

Abstract

It seems very intuitive that for the maximization of the OneMax problem $\textsc{Om}(x) := \sum_{i=1}^{n} x_i$ the best that an elitist unary unbiased search algorithm can do is to store a best so far solution, and to modify it with the operator that yields the best possible expected progress in function value. This assumption has been implicitly used in several empirical works. In [Doerr, Doerr, Yang: *Optimal parameter choices via precise black-box analysis*, TCS, 2020] it was formally proven that this approach is indeed almost optimal.

In this work we prove that drift maximization is not optimal. More precisely, we show that for most fitness levels between $n/2$ and $2n/3$ the optimal mutation strengths are larger than the drift-maximizing ones. This implies that the optimal RLS is more risk-affine than the variant maximizing the step-wise expected progress. We show similar results for the mutation rates of the classic (1+1) Evolutionary Algorithm (EA) and its resampling variant, the (1+1) $\text{EA}_{>0}$.

As a result of independent interest we show that the optimal mutation strengths, unlike the drift-maximizing ones, can be even.

1 Introduction

It is well understood that iterative optimization heuristics like local search variants, evolutionary algorithms, estimation of distribution algorithms, etc. can benefit from non-static choices of the parameters that determine their search radius, population size, or selective pressure. The question how to select these parameters dynamically is the subject of *parameter control,* which studies different techniques to achieve a good fit between suggested and optimal parameter values.

Complementing a diverse body of empirical works demonstrating advantages of parameter control mechanisms [KHE15, AM16], there is an increasing interest in proving such benefits by mathematical means [DD20]. Among the significant advances in this direction are, in chronological order (with respect to the conference announcements), the analysis of a success-based adaptation strategy for the choice of the offspring population size λ of the $(1 + \lambda)$ EA in distributed models of computation [LS11], the self-adjusting $(1 + (\lambda, \lambda))$ Genetic Algorithm (GA) using the one-fifth success rule [DD18], a learning-based selection of the search radii in Randomized Local Search [DDY16], and the self-adjusting [DGWY19] and self-adaptive [DWY18a] mutation rates in a $(1 + \lambda)$ and $(1, \lambda)$ Evolutionary Algorithm (EA), respectively. All these references consider the optimization of OneMax, the problem of maximizing the counting-ones function $\textsc{Om} : \{0, 1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} x_i$. Only few theoretical results analyzing algorithms with adaptive parameters consider different functions, e.g., [LOW20, DLOW18, DDK18] (see [DD20] for a complete list of references). OneMax also plays a prominent role in empirical research on parameter control. In both communities, it is argued that the consideration of OneMax

provides a "sterile EC-like environment" [FCSS08], in which the optimal parameter values are well understood.

In light of the existing literature it is interesting to note that most works, implicitly or explicitly, assume that for the considered algorithms the optimal strategy for the maximization of OneMax is a greedy selection of the best so far solution, and the variation of the same by the mutation rate/step size that maximizes the expected gain in function value [Bäc92, Bäc93, FCSS08, FCSS09]. Thierens [Thi09] explicitly argues that a particularly useful property of OneMax, which makes this problem a very suitable benchmark for adaptive operator selection, is the fact that the reward of an operator can be computed exactly. He then proceeds by comparing the step-wise expected fitness gains made by different operators, and ranks operators by this value. He thus uses as underlying assumption that drift-maximization is optimal.

That this widely believed-to-be-optimal *drift-maximizing* strategy is indeed almost optimal was formally proven in [DDY20]. More precisely, it is shown in [DDY20] that the best unary unbiased black-box algorithm for OneMax cannot be better by more than an additive $o(n)$ term than the RLS variant that flips in each iteration the drift-maximizing number of bits in a best-so-far solution. Both algorithms have an expected optimization time $n \ln(n) - cn \pm o(n)$, for a constant c between 0.2539 and 0.2665.

It was conjectured in [DW18, Section 3.1] that the drift-maximizing RLS is not only "almost" optimal, but indeed optimal. As mentioned, this conjecture was also—explicitly or implicitly—made in the empirical works cited above (and several other works on the OneMax function). We show in this work that this conjecture is false. More precisely, we show that maximizing drift is not optimal neither for RLS nor for the (1+1) EA nor for its resampling variant, the $(1+1)$ $\text{EA}_{>0}$, suggested in [CD18b].

We explain where the difference between optimal and drift-maximizing strategies comes from, define precisely how to obtain the optimal mutation rates, numerically compute these for some selected dimensions up to $n = 10{,}000$, and analyze the differences between drift-maximizing and optimal mutation rates. We also compare the performances of optimal and drift-maximizing algorithms, and show that the differences in mutation rates/step sizes—albeit significant—result only in marginal differences in terms of overall running time. Given the above-mentioned results in [DDY20], the last statement is not surprising. The main contribution of our work is therefore not to be found in tremendous performance gains, but in new structural insights for the optimization of OneMax, the arguably most widely used benchmark for parameter control and adaptive operator selection mechanisms.

We note that the argument *why* drift-maximization is not optimal is quite easy to understand. Basically, our result is built upon the observation that the drift-maximizer values a potential fitness progress of i by exactly this gain. More precisely, in the computation of the drift, the probability of creating an offspring y of x is multiplied by the difference $\max\{0, \textsc{Om}(y) - \textsc{Om}(x)\}$, for each possible offspring y. The optimal algorithms, however, value a fitness gain of i by the gain in the expected remaining running time. Since this difference in expected remaining running time is much larger than the fitness difference, the optimal RLS and $(1+1)$ EA variants use mutation rates that are larger than the drift-maximizing ones. Put differently, they trade a smaller expected progress for a slightly larger probability of making a larger fitness gain. That is, the optimal algorithms are more risk-affine than the drift-maximizing ones. This quite intuitive fact seems to have been overlooked in the evolutionary computation (EC) community.

Our work has recently been extended to $(1+\lambda)$-type RLS and EAs [BD20]. In that work, not only the optimal mutation rates are computed, but also the expected remaining running times for sub-optimal mutation rates – information that can be used to identify weak spots of parameter control mechanisms.

Precise Running Time Bounds. While we focus in this work on very precise running

time bounds for concrete problem dimensions, which we compute numerically, we note that there exists a significant body of related theoretical works, which focus on asymptotically optimal mutation rates and running times. In addition to the works mentioned above, which all deal with adaptive parameter schemes, we consider the following ones particularly interesting in the context of our study. For the classic RLS variant, which always flips exactly one bit in each iteration, the expected running time on OneMax was computed very precisely in [DD16]. For the (1+1) EA with static mutation rate $1/n$, the best known bounds are proven in [HPR+18] and in the recent work [HW19], which are precise up to an additive $O(\log(n)/n)$ and $O(\log n)$ term, respectively. For other static mutation rates, the best known results are available in [Wit13].

Online Repository. Codes and details for the here-described algorithms can be found on the GitHub page of this project at https://github.com/NathanBuskulic/OneMaxOptimal. The interested reader can find there not only the performance data, but also the drift-maximizing and optimal step sizes/mutation rates of the algorithms discussed below, for problem dimensions up to $n = 10{,}000$.

2 The OneMax Problem

OneMax, also referred to as counting-ones problem in the early works on evolutionary computation, is the problem of maximizing the function

$$\textsc{Om} : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} x_i,$$

which simply assigns to each bit string the number of ones in it. OneMax is considered to be one of the "easiest" non-trivial benchmark problems, for two reasons. Firstly, a number of results exist that show that for several (classes of) algorithms the expected optimization time on OneMax is not bigger than that on any other unimodal function of the same dimension, cf. [DJW12, Sud13, CHJ+17] for examples. A second reason to declare OneMax as "easy", yet useful, benchmark problem is its (presumably) simple structure, which allows us to understand well the optimization process of classical optimization heuristics. One structural property that is particularly useful in runtime analyses is the perfect fitness-distance correlation; i.e., whenever $\textsc{Om}(x) > \textsc{Om}(y)$ for two search points x and y, then the distance of x to the optimum is strictly smaller than that of y.

For readers wondering about the usefulness of a single benchmark instance, we note that for most evolutionary algorithms (EAs) and local search variants such as Randomized Local Search (RLS), Simulated Annealing, etc. the OneMax problem is identical to the problem of maximizing any of the functions $\textsc{Om}_z : \{0,1\}^n \to \mathbb{R}, x \mapsto H(z,x) := |\{i \in [n] \mid x_i \neq z_i\}|$, since for any $z \in \{0,1\}^n$ the Hamming distance problem $\textsc{Om}_z$ has a fitness landscape that is isomorphic to that of OneMax, and the mentioned algorithms are oblivious of the exact problem representation. That is, OneMax is essentially just one representative of the class of Hamming distance problems.

OneMax is often termed the "drosophila of EC", because of the vast amount of literature studying this problem, both in empirical and in theoretical works. In the context of our study in particular the works [Bäc92, Bäc93, FCSS08, FCSS09, Thi09, BLS14, DD18, DDY20, DDY16, DGWY19, DWY18a, dPdLDD15, DW18] are worth mentioning, as they all study the benefits of non-static parameter choices on this problem, for different local search variants and evolutionary algorithms. Among these works, the empirical ones focus on operators that maximize the expected progress ("drift") per each round, either without further justifying it, or explicitly mentioning that drift-maximization is optimal (an assumption that we will refute in Section 4).

Algorithm 1: Blueprint for elitist (1+1) unbiased black-box algorithms maximizing a function $f : \{0,1\}^n \to \mathbb{R}$

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and compute $f(x)$;
Optimization: **for** $t = 1, 2, 3, \dots$ **do**
  Sample $k \sim D(n, f(x))$;
  $y \leftarrow \text{flip}_k(x)$;
  evaluate $f(y)$;
  if $f(y) \geq f(x)$ **then** $x \leftarrow y$;

Among the theoretical works, most are interested in deriving asymptotic results only, with the only exception of [DDY20, DDY16], where very precise bounds for the optimization time of two adaptive RLS variants are proven. Most relevant to our work is the mentioned result from [DDY20] which proves that the drift-maximizing strategy mentioned above is indeed almost optimal. When we show in the next sections that the best possible RLS variant is not the drift-maximizing one, we know by the result from [DDY20] that the gain in expected optimization time cannot be more than an additive $O(n^{2/3} \log^9 n)$ term.

3 Elitist (1+1) Unbiased Algorithms

We are concerned in this work with algorithms following the blueprint given in Algorithm 1. These algorithms start the optimization in a randomly chosen solution x. In each iteration exactly one offspring y is sampled by first copying x and then flipping the entries of k randomly chosen, pairwise different positions $i_1, i_2, \dots, i_k$. The *parent* x is replaced by its *offspring* y if and only if $f(y) \geq f(x)$, i.e., if and only if the offspring is at least as good as y. Algorithms adhering to this scheme are referred to in the theory of EA literature as *elitist unary unbiased black-box algorithms* [DL17].

Elitist unary unbiased black-box algorithms differ only in the choice of the *mutation strength* k. The two most commonly studied classes of algorithms are **Randomized Local Search (RLS)** variants, which use a *deterministic* choice of k, and **(1+1) Evolutionary Algorithms (EAs)**, which sample k from $\text{Bin}(n,p)$, i.e., from a binomial distribution with n trials and success rate p. We note that traditionally *constant* choices, $k = 1$ for RLS and $p = 1/n$ for the $(1+1)$ EA, are studied, but here in this work we focus on *non-static* mutation strengths k and *mutation rates* $0 \leq p \leq 1$. More precisely, we study fitness-dependent choices $k(\ell)$ and $p(\ell)$, which take into account the function value (*fitness*) $\ell = \textsc{Om}(x)$ of the current-best solution. In the terminology proposed in [DD20] such parameter control schemes classify as *state-dependent,* since the parameter value depends only on the current-best solution but not on any other information about the optimization process. The objective of our work is to identify the functions $\ell \mapsto k(\ell)$ and $\ell \mapsto p(\ell)$ that minimize the expected running time of RLS and the $(1+1)$ EA, respectively, when optimizing OneMax.

We add to our investigation the $(1+1)$ $\mathbf{EA}_{>0}$, which samples k from a conditional binomial distribution $\text{Bin}_{>0}(n,p)$, which is defined by $\text{Bin}_{>0}(n,p)(0) = 0$ and $\text{Bin}_{>0}(n,p)(i) = \text{Bin}(n,p)(i)/(1-(1-p)^n) = \binom{n}{i} p^i (1-p)^{n-i}/(1-(1-p)^n)$ for $i \in [n]$. That is, the probability of the $(1+1)$ $\text{EA}_{>0}$ to flip i bits equals that of the $(1+1)$ EA conditional on flipping at least one bit. The $(1+1)$ $\text{EA}_{>0}$ was suggested in [CD18b] as an algorithm that more closely resembles common implementations of the $(1+1)$ EA, cf. also discussions in [CD18a]. The $(1+1)$ $\text{EA}_{>0}$ can be seen as an intermediate algorithm between the RLS variant always flipping one bit and the (unconditional) $(1+1)$ EA, since for p converging to 0 the distribution $\text{Bin}_{>0}(n,p)$ concen-

trates on 1, so that for small p the behavior of the $(1+1)$ $\text{EA}_{>0}$ "converges" against that of RLS.

We note that other elitist unary unbiased black-box algorithms have been recently introduced. The *fast Genetic Algorithm (GA)* suggested in [DLMN17] samples the mutation strength k from a power-law distribution, and k is sampled from a normal distribution $N(\mu, \sigma^2)$ in the *normalized EA* studied in [YDB19]. We will nevertheless focus in this work on RLS and (1+1) EA variants only, simply because they are still the most commonly studied algorithms in evolutionary computation. We note though that an extension of our work in particular to results covering the normalized EAs would be interesting, since this algorithm class can be seen as a meta-model between the class of RLS algorithms and the class of $(\mu + \lambda)$ EAs.

4 Maximizing Drift is Not Optimal

As mentioned in Section 3, our main interest is in identifying the functions $k_{\text{opt}} : [0..n-1] \to [0..n]$, $p_{\text{opt}} : [0..n-1] \to [0,1]$, and $p_{>0,\text{opt}} : [0..n-1] \to [0,1]$ for which the following three algorithms have a best possible expected optimization time:

- **RLS_{opt},** the RLS variant flipping in each iteration exactly $k_{\text{opt}}(\text{OM}(x))$ bits (i.e., using the deterministic mutation strength $k_{\text{opt}}(\text{OM}(x))$),
- $(1+1)$ **EA_{opt},** the $(1+1)$ EA variant using standard bit mutation with mutation rate $p_{\text{opt}}(\text{OM}(x))$ (i.e., the algorithm sampling the mutation strength from the binomial distribution $\text{Bin}(n, p_{\text{opt}}(\text{OM}(x)))$), and
- (1+1) **$\text{EA}_{>0,\text{opt}}$,** the (1+1) $\text{EA}_{>0}$ variant using conditional standard bit mutation flipping at least one bit with mutation rate $p_{>0,\text{opt}}(\text{OM}(x))$ (i.e., sampling the mutation strength from the conditional binomial distribution $\text{Bin}_{>0}(n, p_{>0,\text{opt}}(\text{OM}(x)))$.

Note that, formally, we should write $k_{\text{opt}}(n)$, $p_{\text{opt}}(n)$, and $p_{>0,\text{opt}}(n)$, since these functions depend on the dimension. However, we shall often omit the explicit mention of the dimensions in order to ease the notation. The same applies to the corresponding functions $k_{\text{drift}}(n)$, $p_{\text{drift}}(n)$, and $p_{>0,\text{drift}}(n)$.

It may be surprising that, after so many years of research on the OneMax problem, none of the three algorithms above has been explicitly computed. As mentioned in the introduction, there are two main reasons explaining this situation. Firstly, it is widely believed that the functions k_{drift}, p_{drift}, and $p_{>0,\text{drift}}$, which maximize in each step the expected fitness gain (*drift*) of flipping $k = k_{\text{drift}}(\text{OM}(x))$, $k \sim \text{Bin}(n, p_{\text{drift}}(\text{OM}(x)))$, and $k \sim \text{Bin}_{>0}(n, p_{>0,\text{drift}}(\text{OM}(x)))$ bits, respectively, are optimal. As already discussed, such claims can be quite frequently found in the literature [Bäc92, FCSS08, DW18]. We will show in this section that these claims are not correct, by presenting examples which demonstrate that better expected optimization times can be achieved by choosing $k_{\text{opt}} \neq k_{\text{drift}}$, $p_{\text{opt}} \neq p_{\text{drift}}$, and $p_{>0,\text{opt}} \neq p_{>0,\text{drift}}$, respectively. In Section 5 we will quantify the discrepancies between drift-maximizing and optimal (i.e., time-minimizing) functions for dimensions up to $n = 10{,}000$. Section 6 discusses the impact of these differences on the overall running time.

4.1 $\text{RLS}_{\text{opt}} \neq \text{RLS}_{\text{drift}}$ for $n = 3$

We first show that $k_{\text{drift}} \neq k_{\text{opt}}$. That is, we study the drift-maximizing and the time-minimizing variants of RLS, which we call $\text{RLS}_{\text{drift}}$ and RLS_{opt} in the following, and show that they are not identical. Interestingly, it suffices to regard $n = 3$ for an example for which the two functions differ. The following table summarizes for $n = 3$ the functions k_{drift}, k_{opt}, and the expected

remaining running times $\mathbb{E}[T_{\text{drift}}(\ell)]$ and $\mathbb{E}[T_{\text{opt}}(\ell)]$ for $\text{RLS}_{\text{drift}}$ and RLS_{opt}, respectively, when starting in a solution x of fitness $\textsc{Om}(x) = \ell$. In column $p^0(\ell)$ we list the probability that a random initial solution has fitness value ℓ. Since uniform random initialization is used, $p^0(\ell) = \binom{n}{\ell}/2^n$. The last line provides the overall expected optimization time of both algorithms. Note that, by the law of total probability,

$$\mathbb{E}[T] = 1 + \sum_{\ell=0}^{n} p^0(\ell)\mathbb{E}[T(\ell)],$$

where the "+1"-term accounts for the evaluation of the initial solution.

ℓ	$p^0(\ell)$	$k_{\text{drift}}(\ell)$	$\mathbb{E}[T_{\text{drift}}(\ell)]$	$k_{\text{opt}}(\ell)$	$\mathbb{E}[T_{\text{opt}}(\ell)]$
3	1/8	-	0	-	0
2	3/8	1	3	1	3
1	3/8	3	4	2	3
0	1/8	3	1	3	1
$\mathbb{E}[T]$		3.75		3.375	

As we see from the last line, the overall expected running time of RLS_{opt} is 3.375 and thus strictly smaller than that of $\text{RLS}_{\text{drift}}$, which is 3.75. We briefly explain how the entries in this table are computed.

Computation of $\text{RLS}_{\text{drift}}$. We start our explanation with the computation of $k_{\text{drift}}(n) : [0..n-1] \to [0..n]$ and $\mathbb{E}[T_{\text{drift}}(\ell)]$. The function k_{drift} was defined above to be the one that maps each fitness value to the number of bits that need to be flipped in order to maximize the expected progress in fitness value, i.e., $k_{\text{drift}}(n, \ell)$ is defined to be the value of k that maximizes the expression

$$\begin{aligned} &\mathbb{E}[\Delta(n, \ell, k)] := \\ &\mathbb{E}[\max\{\textsc{Om}(y) - \textsc{Om}(x), 0\} \mid \textsc{Om}(x) = \ell, y \leftarrow \text{flip}_k(x)] \\ &= \sum_{i=\ell+1}^{\ell+k} (i-\ell)\mathbb{P}[\textsc{Om}(y) = i \mid \textsc{Om}(x) = \ell, y \leftarrow \text{flip}_k(x)] \\ &= \sum_{i=\lceil k/2 \rceil}^{k} \frac{\binom{n-\ell}{i}\binom{\ell}{k-i}(2i-k)}{\binom{n}{k}}, \end{aligned} \tag{1}$$

where we use in the last line the fact that flipping i of the $n-\ell$ previously incorrect bits implies that we flip $k-i$ of the ℓ previously correct bits, which results in a fitness increase of $i-(k-i) = 2i-k$. This event occurs with probability $\frac{\binom{n-\ell}{i}\binom{\ell}{k-i}}{\binom{n}{k}}$, since there are $\binom{n-\ell}{i}$ different ways of choosing i previously incorrect bits, $\binom{\ell}{k-i}$ ways of choosing $k-i$ previously correct bits, and $\binom{n}{k}$ ways of choosing k pairwise different bit positions. When two or more values k exist that minimize this expression, we follow the convention made in [DDY20] and chose in all our computations below the smallest of these drift-maximizing mutation strengths, i.e., formally, $k_{\text{drift}}(n, \ell) = \min\left\{\arg\max_k \mathbb{E}[\Delta(\ell, k)]\right\}$.[1]

It is easily seen that that for $n = 3$ and $\ell = 2$ flipping one bit is optimal, since this is the only mutation strength yielding positive drift. With this value of $k_{\text{drift}}(n = 3, \ell = 2)$ the expected remaining time $\mathbb{E}[T_{\text{drift}}(n = 3, \ell = 2)]$ to find the optimal solution is 3. For $\ell = 1$, the expected

[1] In light of the results presented in this paper, it seems likely that for $\ell > n/2$ the better choice would be $k_{\text{drift}}(n, \ell) = \max\{\arg\max_k \mathbb{E}[\Delta(\ell, k)]\}$, but given the small discrepancies in the resulting running times (cf. Section 6) we do not investigate this question further.

progress of flip_1, i.e., of flipping one bit, is $\mathbb{P}[\textsc{Om}(y) = 2 \mid \textsc{Om}(x) = 1, y \leftarrow \text{flip}_1(x)] = 2/3$, the expected progress of flip_2 equals $2\mathbb{P}[\textsc{Om}(y) = 3 \mid \textsc{Om}(x) = 1, y \leftarrow \text{flip}_2(x)] = 2/3$ (note here that no fitness gain of one is possible), and the expected progress of flip_3 is 1, since in this case we deterministically obtain an offspring y of fitness $\textsc{Om}(y) = 2$. The best drift is thus obtained by operator flip_3, which implies $k_{\text{drift}}(n = 3, \ell = 1) = 3$. With this choice of the mutation strength, the expected remaining optimization time equals $1 + \mathbb{E}[T_{\text{drift}}(n = 3, \ell = 2)] = 4$. In general, $\mathbb{E}[T_{\text{drift}}(n, \ell)]$ can be computed as

$$1 + \sum_{i=\ell+1}^{n-1} \mathbb{P}[\textsc{Om}(y) = i \mid \mathcal{E}]\mathbb{E}[T_{\text{drift}}(n, i)],$$

where $\mathcal{E}$ is the event that $\textsc{Om}(x) = \ell$ and $y \leftarrow \text{flip}_{k_{\text{drift}}(n,\ell)}(x)$. When $\textsc{Om}(x) = 0$ then flipping all bits, i.e., applying flip_n is optimal, since it directly produces the optimal solution. Note here that, more generally, the function value of the bitwise complement $\bar{x}$ of a solution x equals $\textsc{Om}(\bar{x}) = n - \textsc{Om}(x)$.

With these values, the expected running time of $\text{RLS}_{\text{drift}}$ on $n = 3$ is equal to $1+\frac{3}{8}3+\frac{3}{8}4+\frac{1}{8} = \frac{15}{4} = 3.75$.

Computation of RLS_{opt}. We next discuss how to compute $k_{\text{opt}}(n) : [0..n-1] \to [0..n]$ and $\mathbb{E}[T_{\text{opt}}(\ell)]$. This time, we start our investigation by recalling that, by the law of total probability, the expected optimization time $\mathbb{E}[T(\text{RLS}_{\text{opt}})]$ of RLS_{opt} equals $1 + \sum_{\ell=0}^{n-1} \mathbb{P}[\textsc{Om}(x^0) = \ell]\mathbb{E}[T_{\text{opt}}(\ell)]$. The best-possible RLS algorithm is hence the one using at each fitness level ℓ the mutation strength $k_{\text{opt}}(n, \ell)$ which minimizes the expected remaining optimization time $\mathbb{E}[T_k(n, \ell)]$ of flipping k bits, which is equal to

$$1 + \sum_{i=\ell+1}^{n-1} \mathbb{P}[\textsc{Om}(y) = i \mid \textsc{Om}(x) = \ell, y \leftarrow \text{flip}_k(x)]\mathbb{E}[T_{\text{opt}}(n, i)]. \tag{2}$$

Formally, we set again $k = \arg\min_k \mathbb{E}[T_k(n, \ell)]$. Note that the expression in (2) requires to know the values $\mathbb{E}[T_{\text{opt}}(n, i)]$ for $i > \ell$. In order to compute $k_{\text{opt}}(n, \ell)$ one therefore has to start with fitness level $n - 1$. Once $k_{\text{opt}}(n, n-1)$ and $\mathbb{E}[T_{\text{opt}}(n, n-1)]$ are known, $k_{\text{opt}}(n, n-2)$ and $\mathbb{E}[T_{\text{opt}}(n, n-2)]$ can be computed, and one continues in this way until eventually reaching $\ell = 0$ for which $k_{\text{opt}}(n, 0) = n$ holds.

Applying these computations to our example with $n = 3$, we first easily obtain $k_{\text{opt}}(n = 3, \ell = 2) = 1$ and $\mathbb{E}[T_{\text{opt}}(n = 3, \ell = 2)] = 3$, as in the drift maximizing case analyzed above. Given that $k_{\text{opt}}(3, 0) = 3$, the only interesting case is fitness level $\ell = 1$. The expected remaining time $\mathbb{E}[T_1(n = 3, \ell = 1)]$ equals $1+\frac{2}{3}\mathbb{E}[T_{\text{opt}}(3, 2)]+\frac{1}{3}\mathbb{E}[T_1(3, 1)]$. Since $\mathbb{E}[T_{\text{opt}}(3, 2)] = 3$, a simple algebraic transformation shows $\mathbb{E}[T_1(3, 1)] = \frac{9}{2}$. When flipping two bits, we either obtain the optimal solution (this happens with probability 1/3) or we remain at the current fitness level, which shows that $\mathbb{E}[T_2(3, 1)] = 1 + \frac{2}{3}\mathbb{E}[T_2(3, 1)]$. Thus, $\mathbb{E}[T_2(3, 1)] = 3$. Finally, we compute that $\mathbb{E}[T_3(3, 1)] = 1 + \mathbb{E}[T_{\text{opt}}(3, 2)] = 4$. We therefore see that $k_{\text{opt}}(n = 3, \ell = 1) = 2$ and $\mathbb{E}[T_{\text{opt}}(3, 1)] = 3$. With these values, we obtain that the expected optimization time of RLS_{opt} on the 3-dimensional OneMax is $1 + \frac{3}{8}\mathbb{E}[T_{\text{opt}}(3, 2)] + \frac{3}{8}\mathbb{E}[T_{\text{opt}}(3, 1)] + \frac{1}{8} = \frac{27}{8} = 3.375$.

Optimal Mutation Strengths Need Not be Uneven. With this example, we not only prove that $\text{RLS}_{\text{opt}} \neq \text{RLS}_{\text{drift}}$, but we also make another interesting observation, which concerns the parity of the values $k_{\text{opt}}(n, \ell)$. It was proven in [DDY20] that k_{drift} takes only odd values, since for every k the drift of flipping $2k$ bits is strictly smaller than that of flipping $2k + 1$ bits.

The example above shows that the situation is different for k_{opt}. More precisely, we have seen that in the situation $n = 3$ and $\ell = 1$ flipping 2 bits is optimal.

4.2 $(1+1)$ $\text{EA}_{\text{opt}} \neq (1+1)$ EA_{drift} for $n = 3$

The only difference between the $(1+1)$ EA and RLS is the random choice of the mutation strength k, which the $(1+1)$ EA samples from a binomial distribution $\text{Bin}(n,p)$. The $(1+1)$ EA_{drift} is defined by choosing the fitness-dependent mutation rate $p_{\text{drift}}(n,\ell)$ which maximizes the expected progress

$$\begin{aligned}
&\mathbb{E}[\Delta(n,\ell,p)] \qquad (3)\\
&:= \sum_{i=\ell+1}^{n} (i-\ell)\mathbb{P}[\textsc{Om}(y) = i \mid \textsc{Om}(x) = \ell, y \leftarrow \text{flip}_k(x), k \sim \text{Bin}(n,p)]\\
&= \sum_{k=1}^{n} \text{Bin}(n,p)(k)\mathbb{E}[\Delta(n,\ell,k)]\\
&= \sum_{k=1}^{n} \left(\binom{n}{k} p^k (1-p)^{n-k} \sum_{i=\lceil k/2 \rceil}^{k} \frac{\binom{n-\ell}{i}\binom{\ell}{k-i}(2i-k)}{\binom{n}{k}} \right),
\end{aligned}$$

where we recall that $\mathbb{E}[\Delta(n,\ell,k)]$ had been defined in equation (1).

Following the same arguments as in the definition of RLS_{opt} in Section 4.1, the $(1+1)$ EA_{opt} is defined by choosing in each fitness level the mutation rate $p_{\text{opt}}(n,\ell)$ which minimizes the expected remaining time, i.e., the expression

$$\begin{aligned}
\mathbb{E}[T_p(n,\ell)] = 1 &+ \mathbb{P}[\textsc{Om}(y) \leq \ell \mid \mathcal{E}]\mathbb{E}[T_p(n,\ell)]\\
&+ \sum_{i=\ell+1}^{n-1} \mathbb{P}[\textsc{Om}(y) = i \mid \mathcal{E}]\mathbb{E}[T_{\text{opt}}(n,i)],
\end{aligned}$$

where we abbreviate by $\mathcal{E}$ the event that $\textsc{Om}(x) = \ell$, $y \leftarrow \text{flip}_k(x)$, and $k \sim \text{Bin}(n,p)$.

As above, we thus need to determine first the values of $p_{\text{opt}}(n,n-1)$ and $\mathbb{E}[T_{\text{opt}}(n,n-1)]$, then progress with the computation of $p_{\text{opt}}(n,n-2)$ and $\mathbb{E}[T_{\text{opt}}(n,n-2)]$, etc.

For $n = 3$ we obtain the following values, which prove that, like for RLS, drift-maximization is also not optimal for the $(1+1)$ EA.

ℓ	$p^0(\ell)$	$p_{\text{drift}}(\ell)$	$\mathbb{E}[T_{\text{drift}}(\ell)]$	$p_{\text{opt}}(\ell)$	$\mathbb{E}[T_{\text{opt}}(\ell)]$
3	1/8	-	0	-	0
2	3/8	1/3	6.75	1/3	3
1	3/8	1	7.75	2/3	3
0	1/8	1	1	1	1
$\mathbb{E}[T]$		6.5625		6.1875	

Another interesting observation that we can make by comparing this table with the corresponding one of RLS (Sec. 4.1) is that $p_{\text{drift}}(3,\ell) = k_{\text{drift}}(3,\ell)/3$ and $p_{\text{opt}}(3,\ell) = k_{\text{opt}}(3,\ell)/3$. We will discuss this effect in more detail in Section 5.2.

4.3 $(1+1)$ $\text{EA}_{>0,\text{opt}} \neq (1+1)$ $\text{EA}_{>0,\text{drift}}$ for $n = 3$

$(1+1)$ $\text{EA}_{>0,\text{opt}}$ and $(1+1)$ $\text{EA}_{>0,\text{drift}}$ are defined by replacing in all definitions in Section 4.2 the binomial distribution $\text{Bin}(n,p)$ by the conditional binomial distribution $\text{Bin}_{>0}(n,p)$, and by replacing the formulas accordingly. We omit a detailed definition for reasons of space. All replacements are straightforward, the only particularity to pay attention to is that both the drift and the expected remaining time may be better for ever smaller values of p. This happens

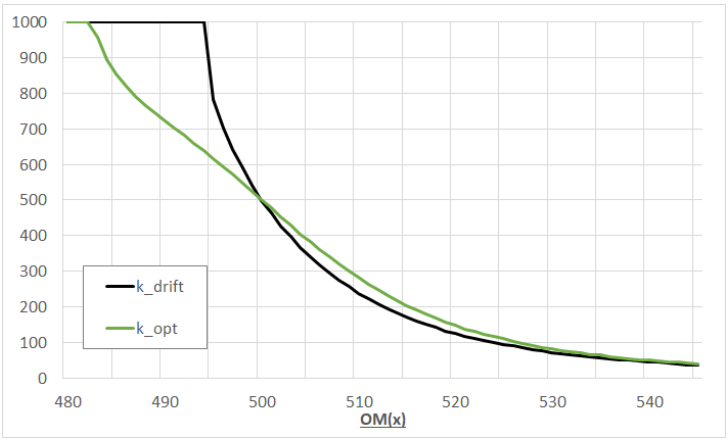

Figure 1: Comparison of $k_{\text{opt}}(n,\ell)$ and $k_{\text{drift}}(n,\ell)$ for $n = 1{,}000$ (zoom into fitness levels $480 \leq \ell \leq 545$).

when flipping one bit deterministically is better in terms of drift or expected running time, respectively, than using standard bit mutation. In this case we can either use the convention that the conditional standard bit mutation with mutation rate $p = 0$ is to be interpreted as the flip_1 operator (i.e., we set $\text{Bin}(n,0)(1) = 1$ and $\text{Bin}(n,0)(k) = 0$ for all $k \neq 1$), or we set a lower bound $p_{\min}$ for the mutation rate. The effects of the lower bound will be discussed in Section 6. When using $p_{\min} = 0$, the situation for the $(1+1)$ $\text{EA}_{>0}$ for OneMax in dimension $n = 3$ is given by the following table.

ℓ	$p^0(\ell)$	$p_{>0,\text{drift}}(\ell)$	$\mathbb{E}[T_{>0,\text{drift}}(\ell)]$	$p_{>0,\text{opt}}(\ell)$	$\mathbb{E}[T_{>0,\text{opt}}(\ell)]$
3	1/8	-	0	-	0
2	3/8	0	3	0	3
1	3/8	1	4	3/4	27/7
0	1/8	1	1	1	1
$\mathbb{E}[T]$		3.75		≈ 3.696	

5 Optimal RLS and (1+1) EA Variants

Using the formulas provided in Section 4 we can compute the optimal RLS, $(1+1)$ EA, and $(1+1)$ $\text{EA}_{>0}$ algorithms, as well as their drift-maximizing counterparts. Note, though, that the numerical evaluation of the binomial coefficients, as well as the optimization required to determine p_{opt} and $p_{>0,\text{opt}}$ is not straightforward. For the latter, we have used the bounded method of the *scipy* optimization module [JOP$^+$]. The overall expected running times are summarized in Table 1, which can be found at the end of this paper.

5.1 Optimal Mutation Strengths

We start our comparison by considering the differences between the drift-maximizing and the optimal mutation strengths for RLS. Figure 1 plots the interesting region of $k_{\text{opt}}(n,\ell)$ and $k_{\text{drift}}(n,\ell)$ for $n = 1{,}000$; the overall picture is very similar across all dimensions n. In particular it holds for all n that the curves cross at fitness level $\ell = n/2$. For smaller values, the optimal mutation strengths are smaller or identical to drift-maximizing ones, and the situation is reversed for fitness levels $\ell > n/2$. This can be explained by the formulas given in Section 4. While the drift-maximizer values a potential progress of i by this same value, regardless of the current

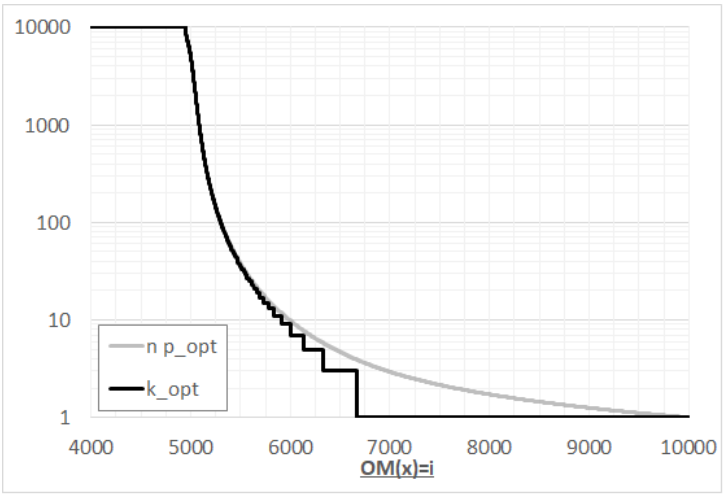

Figure 2: Comparison of $k_{\mathrm{opt}}(n,\ell)$ and $np_{\mathrm{opt}}(n,\ell)$ for $n = 10{,}000$. Note the logarithmic scale.

fitness level, the same potential progress is valued by $\mathbb{E}[T_{\mathrm{opt}}(\ell+i)] - \mathbb{E}[T_{\mathrm{opt}}(\ell)] > i$. $\mathrm{RLS}_{\mathrm{drift}}$ is thus more risk-averse than $\mathrm{RLS}_{\mathrm{opt}}$. Put differently, the latter makes use of the fact that an unlikely large fitness gain results in a larger reduction of the expected remaining optimization time than a more likely small fitness increase. $\mathrm{RLS}_{\mathrm{opt}}$ therefore accepts a smaller probability of an improving move, at the benefit of a potentially larger fitness increase. This observation also explains why the extreme-valued parameter adaptation method proposed in [FCSS08] showed better performance on OneMax than update schemes based on average gains.

It was proven in [DDY20] that an approximated drift-maximizer always flips only one bit when $\ell > 2n/3$. For the actual drift-maximizer this has not been formally proven, but in all our numerical evaluations for dimensions up to 10,000 we have $k_{\mathrm{opt}}(\ell) = k_{\mathrm{drift}}(\ell) = 1$ for $\ell > 2n/3$.

For dimension $n = 1{,}000$, we see from Fig. 1 that $k_{\mathrm{opt}}(\ell) = 1{,}000$ for $\ell \leq 482$ and $k_{\mathrm{drift}}(\ell) = 1{,}000$ for $\ell \leq 494$. In this regime it is thus beneficial to invest one iteration to obtain, deterministically, a search point with function value $n - \ell$.

The difference between the two functions becomes negligible for $\ell > 545$.

We do not plot the comparison of p_{opt} with p_{drift} nor that of $p_{>0,\mathrm{opt}}$ vs. $p_{>0,\mathrm{drift}}$; their curves, however, are similar to those of RLS.

5.2 Comparison of k_{opt} and p_{opt}

We have observed in Section 4.2 that for $n = 3$ the values of p_{opt} were identical to k_{opt}/n. Likewise, we had observed that in this example $p_{\mathrm{drift}} = k_{\mathrm{drift}}/n$. Figure 2 plots $k_{\mathrm{opt}}(n,\ell)$ and $np_{\mathrm{opt}}(n,\ell)$ for $n = 10{,}000$ and Figure 3 plots $k_{\mathrm{drift}}(n,\ell)$ and $np_{\mathrm{drift}}(n,\ell)$ for $n = 1{,}000$; the overall picture is the same for drift-maximizing and optimal functions in both cases.

While Figure 2 gives the global picture, Figures 3 zooms into the region in which k_{drift} is between 3 and 47. We observe that the mutation strength is always smaller, but very close to n times the respective mutation rate. At the points at which k_{opt} and k_{drift} change value the difference between np_{opt} and np_{drift} is smallest.

6 Running Times

We now discuss the impact of the differences in mutation strengths and rates on the overall expected running times.

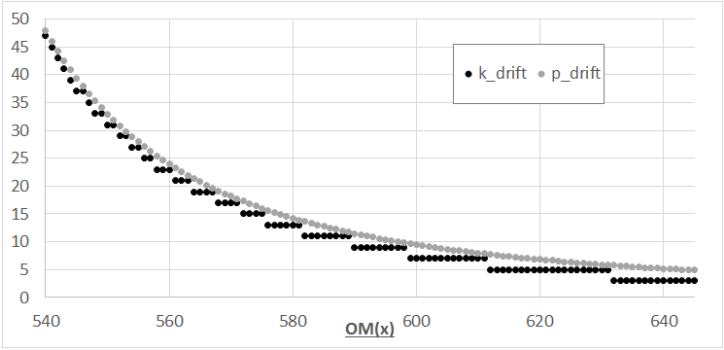

Figure 3: Comparison of $k_{\text{drift}}(n, \ell)$ and $np_{\text{drift}}(n =, \ell)$ for $n = 1{,}000$ (zoom into fitness levels $540 \leq \ell \leq 645$).

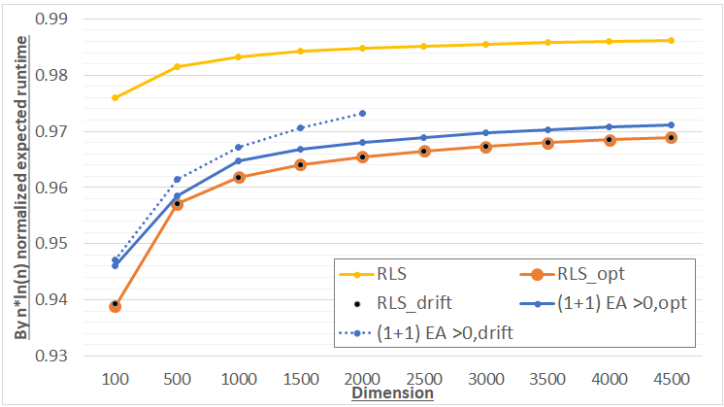

Figure 4: Expected optimization time of different variants of RLS and the $(1 + 1)$ $\text{EA}_{>0,\text{opt}}$, normalized by $n \ln(n)$.

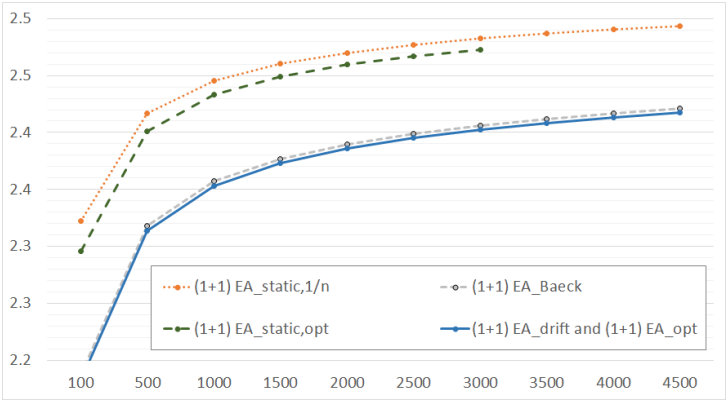

Figure 5: Expected optimization time of different variants of $(1+1)$ EA, normalized by $n\ln(n)$.

We start our comparison with the RLS and the $(1+1)$ $\text{EA}_{>0}$ variants. Figure 4 plots the by $n\ln(n)$ normalized optimization times of five different algorithms for 10 different problem dimensions between 100 and 4,500. We denote here and in the following by RLS the traditional RLS variant using static mutation strength $k = 1$. We see that there is practically no difference between RLS_{opt} and $\text{RLS}_{\text{drift}}$, and this despite the significant differences in the mutation strengths k_{opt} and k_{drift}. While the asymptotic result from [DDY20] guarantees that the absolute difference is bounded by $O(n^{2/3}\log^9(n))$, the absolute difference between the two algorithms is even less than 1 across all tested problem dimensions. The normalized running times of both algorithms increase from around 0.939 for $n = 100$ to around 0.969 for $n = 4{,}500$. As we know from the theoretic result [DDY20] these values converge to 1 for growing dimension n.

The differences between the $(1+1)$ $\text{EA}_{>0,\text{opt}}$ and the $(1+1)$ $\text{EA}_{>0,\text{drift}}$ to RLS_{opt} are very small. The difference between the first two algorithms seems to be more significant than between drift-maximizing and optimal RLS variants, with a numerical difference between (1+1) $\text{EA}_{>0,\text{opt}}$ and $(1+1)$ $\text{EA}_{>0,\text{drift}}$ of around 0.5% for $n = 2{,}000$. We do not have an explanation for this comparatively large difference, but it may be caused by the numerical precision at which the results have been computed. More details about the (1+1) $\text{EA}_{>0}$ will be discussed in Section 6.1.

Our next chart, Figure 5, compares the expected running times of different $(1+1)$ EA variants. We first note that we plot two different static versions, one using the asymptotically optimal static mutation rate $1/n$, and the other one using the optimal static mutation rate per each dimension. The latter is slightly larger than $1/n$, as was already proven in [CWA14]. Since they only computed the optimal static rates for $n \leq 100$, we also had to compute these for larger dimensions (using a direct computation, not the there-suggested matrix-based approach). Alternatively, we could have used the approximations suggested in [GW18], which extend the results of [CWA14] to the $(1+\lambda)$ EA and to larger dimensions. The relative advantage over $1/n$ is not very pronounced, and decreases from around 1.2% for $n = 100$ to around 0.4% for $n = 3{,}000$. The curves of the (1+1) EA_{drift} and the (1+1) $\text{EA}_{>0}$ are practically indistinguishable in this plot. Like for RLS the absolute difference between the expected running time of the two algorithms is less than 1 for all tested dimensions, again despite significant differences in the functions p_{opt} and p_{drift}. We add to this chart a comparison with the $(1+1)$ EA using the fitness-dependent mutation rate $p(\ell) = 1/(2\ell + 2 - n)$ (for $\ell \geq n/2$) suggested in [Bäc93]; we use $p(\ell) = p_{\text{drift}}(\ell)$ for $\ell < n/2$. Bäck obtained this mutation rate from numerical evaluations of p_{drift} in small dimensions $n \leq 100$. His algorithm performs only slightly worse than the true drift-maximizing $(1+1)$ EA_{drift}, and, thus, as the $(1+1)$ EA_{opt}.

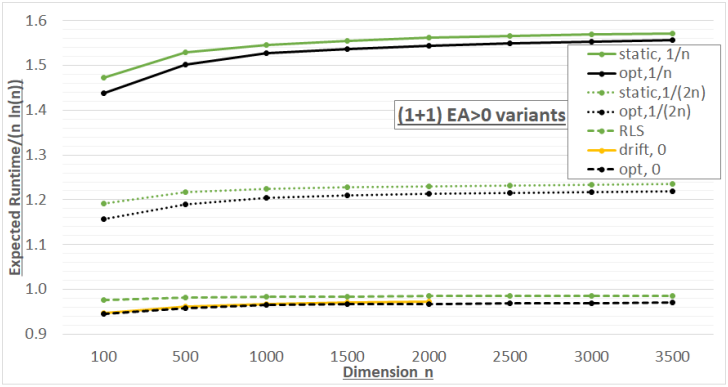

Figure 6: Expected Optimization time of different variants of $(1+1)$ $\text{EA}_{>0}$, normalized by $n \ln(n)$.

6.1 Influence of $p_{\min}$ on the $(1+1)$ $\text{EA}_{>0}$

We have briefly mentioned in Section 4.3 that for the $(1+1)$ $\text{EA}_{>0}$ one needs to specify a lower bound for the mutation probability, since in some situations the optimal mutation rate is zero (when using the convention that $\text{Bin}(n,0)$ deterministically returns one). For practical applications such small mutation rates may be undesirable, e.g., when using multiplicative success-based updates rules as suggested in [DW18]. We therefore investigate the influence of this lower bound on the expected running times. These normalized running times are plotted for six different algorithms in Figure 6. The drift-maximizing variants would be indistinguishable in this plot from the optimal ones, and are therefore omitted, except for the case $p_{\min} = 0$, which we have already discussed in Figure 4. Note that the $(1+1)$ $\text{EA}_{>0}$ with optimal static mutation rate uses $p_{\min} = 0$, and is therefore equal to RLS. The relative disadvantage of increasing $p_{\min}$ to $1/(2n)$ increases from around 22% in dimension $n = 100$ to around 26% in dimension 3,500, both for the static and the adaptive variants. Further increasing $p_{\min}$ to $1/n$ results in a relative disadvantage of $51 - 59\%$ for the static and from $52 - 60\%$ for the dynamic variants.

6.2 Anytime Performance

Fixed-Budget Results. While we have focused above on expected optimization times we will now follow the suggestion made in [DDY20] and provide a more detailed analysis of the *anytime behavior* of the algorithms. More precisely, we regard fixed-budget performance of RLS_{opt}, $\text{RLS}_{\text{drift}}$, and RLS. Only RLS_{opt} and RLS are plotted in Figure 7, the curves of RLS_{opt} and $\text{RLS}_{\text{drift}}$ are practically indistinguishable. Note that the numbers underlying the plot in Figures 7 and 8 (discussed in the next section) are the only ones in this paper that are not derived from theoretical bounds. We have performed a simulation of 500 independent runs of the three algorithms instead, and we used IOHprofiler [DWY+18b] to analyze the runtime data. We show not only the mean value, but also the standard deviation. The curves are well separated even when considering these, for all budgets up to around 3,500. Analyzing the data in more detail, we observe that the relative advantage in average function value decreases from 10% for budget 100 to 1% for budget 2,500. For larger budgets, the average fitness value is less than 1% larger for RLS_{opt} than for RLS. However, as proven to hold in an asymptotic sense for the $\text{RLS}_{\text{drift}}$ in [DDY20], the average distance to the optimum is constantly about $12 - 14\%$ better for RLS_{opt} than for RLS, for budgets up to 3,500. The average function values at this budget (3,500 function evaluations) are slightly smaller than 990 for all three algorithms, RLS, RLS_{opt}, and $\text{RLS}_{\text{drift}}$. For larger budgets, the distance to the optimum is hence very small. This,

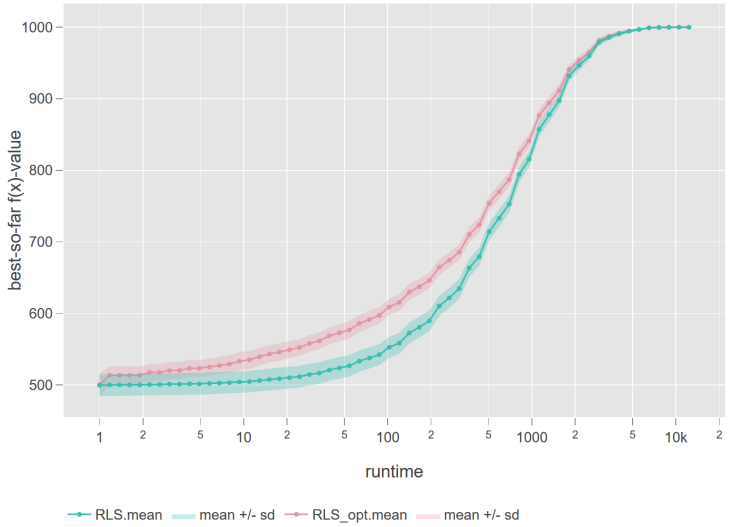

Figure 7: Average fixed-budget results for RLS and RLS_{opt} on OneMax in dimension $n = 1{,}000$ across 500 independent runs.

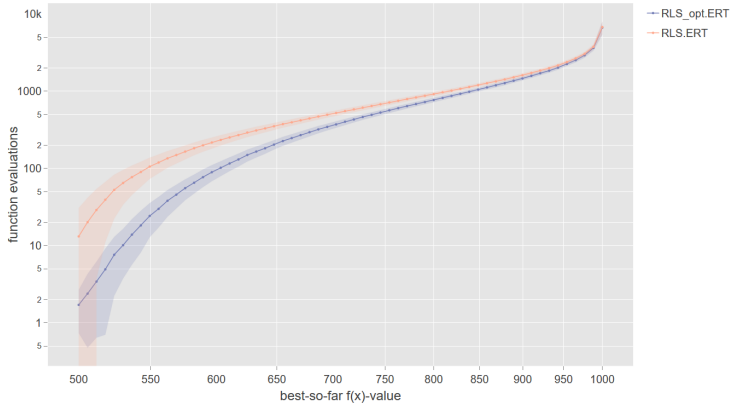

Figure 8: Average fixed-target running time for RLS and RLS_{opt} on OneMax in dimension $n = 1{,}000$ across 500 independent runs.

in combination with the variance of our simulation, results in inconsistent relative advantages in terms of distance to the optimum for budgets greater than 3,500.

Fixed-Target Results. Using the same runtime data for the 500 runs, we can also compute fixed-target results, i.e., the function mapping each fitness level ℓ to the expected time needed to reach a solution x of fitness $\text{Om}(x) \geq \ell$. These values, of course, could also easily be computed theoretically from the results presented in Section 5.1, but we feel that the precision of the simulation suffices to demonstrate the main effects. The results are plotted in Figure 8.

It is not difficult to see that RLS_{opt} is not optimal for minimizing the expected first hitting time of targets $\ell < n$, simply because overshooting the target ℓ are disadvantageous for this optimization goal. For a similar reason, RLS_{opt} is also not optimal in terms of maximizing the expected function value at a given budget of $B < \mathbb{E}[T(\text{RLS}_{\text{opt}})]$, i.e., when the budget is less than the expected overall optimization time of RLS_{opt}.

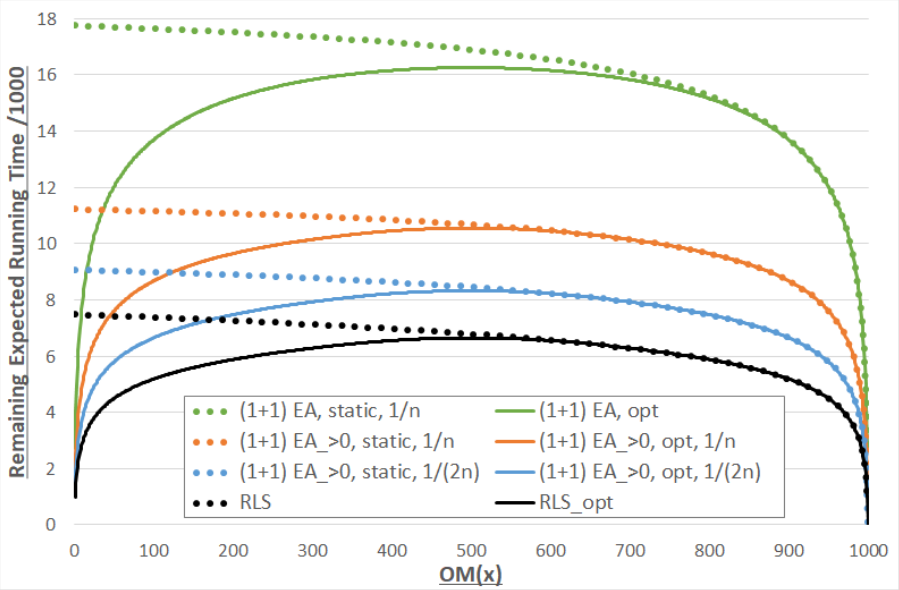

Figure 9: Expected remaining optimization times for OneMax in dimension $n = 1{,}000$.

6.3 Remaining Optimization Times

Finally, we take a look at the evolution of the expected remaining optimization time per each fitness level. These values, derived from our numerical evaluation of the theoretical bounds presented in Section 4, are plotted in Figure 9. While the algorithms with static mutation rates and strength are not able to profit from the fact that $\textsc{Om}(\bar{x}) = n - \textsc{Om}(x)$ for each $x \in \{0,1\}^n$, we see an almost symmetric behavior for the adaptive algorithms. We also see again the influence of the lower bound $p_{\min} \in \{1/n, 1/(2n)\}$ in the $(1+1)$ $\text{EA}_{>0}$ variants, which are quite significant.

From this figure we can also compute the weights $\mathbb{E}[T_{\text{opt}}(\ell+1)] - \mathbb{E}[T_{\text{opt}}(\ell)]$ by which the RLS_{opt} starting in a search point of fitness ℓ values a potential fitness progress of i. We plot in Figure 10 the gradient of the curves $\mathbb{E}[T(\ell)]$ plotted in Figure 9. That is, for every ℓ we plot the values $\mathbb{E}[T(1000,\ell)] - \mathbb{E}[T(1000,\ell-1)]$ for RLS_{opt} and RLS. We recall that $\text{RLS}_{\text{drift}}$ values a potential fitness progress of i by the same value i. We thus clearly see that RLS_{opt} gives much more importance to large fitness gains, and hence uses the already discussed more risky strategy aiming at potentially larger fitness gains, at the cost of a larger probability of creating an offspring that will be discarded.

6.4 Best Unary Unbiased Algorithms for OneMax

Note that plot in Figure 9 also raises the question how much the algorithms lose in performance by being forced to be elitist. Note that slightly better algorithms are possible when allowing them to first decrease the function value to 0 and then inverting the bit string. For the adaptive algorithms, this would clearly bring more flexibility, and a provable positive advantage over the elitist algorithms studied in this work. Put differently, the best unary unbiased black-box algorithm for OneMax is slightly better than RLS_{opt}. The almost perfect symmetric shape of the algorithms in Figure 9, however, indicates that the advantage is very small. A rigorous quantification, which we consider to be of rather philosophical benefit, is left for future work.

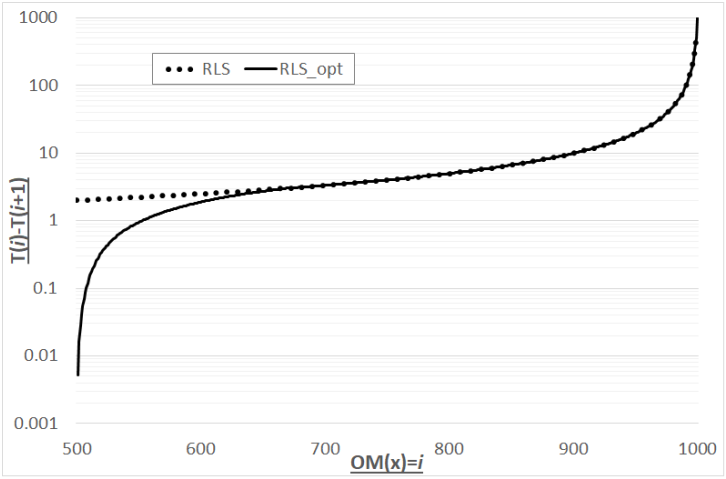

Figure 10: Gradient of expected remaining optimization times for OneMax in dimension $n = 1{,}000$.

7 Discussion

We have shown that the assumption that drift-maximization is optimal for solving the OneMax problem is not correct, neither for RLS, nor the $(1+1)$ EA, nor the $(1+1)$ $\text{EA}_{>0}$. A more risky strategy turns out to be optimal. However, while the differences in the drift-maximizing and the optimal mutation rates are significant (Figure 1), the difference in expected running time is negligibly small already for very small dimensions. The structural findings made here for the OneMax problem also applies in a broader sense to the optimization of non-deceptive problems. Already for linear functions like BinVal, the difference between drift-maximizing and optimal RLS and $(1+1)$ EA variants may be more substantial than for OneMax. We also note that, while we have restricted ourselves to (1+1)-type algorithms, similar effects also hold for population-based EAs.

The computation of the drift-maximizing and time-minimizing mutation strengths and rates are quite tedious and require several days of computing time already for moderate dimension. In order to obtain valid baseline algorithms for larger dimensions, it would be desirable to derive closed formula expressions that approximate these functions sufficiently well. Note that the formula provided by Bäck for the drift-maximizer (cf. discussion in Section 6) seems to allow to derive quite reliable predictions for the drift-maximizing (1+1) EA as seen in Figure 5.

Acknowledgments.

We thank Thomas Bäck for several valuable discussions on the history of adaptive parameter settings.

Our research benefited from the support of the *Paris Ile-de-France Region*, a public grant as part of the Investissement d'avenir project, reference *ANR-11-LABX-0056-LMH*, LabEx LMH, in a joint call with Gaspard Monge Program for optimization, operations research and their interactions with data sciences, and *COST Action CA15140* on 'Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO)' supported by COST (European Cooperation in Science and Technology).

References

[AM16] Aldeida Aleti and Irene Moser. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys*, 49:56:1–56:35, 2016.

[Bäc92] Thomas Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In *Proc. of Parallel Problem Solving from Nature (PPSN'92)*, pages 87–96. Elsevier, 1992.

[Bäc93] Thomas Bäck. Optimal mutation rates in genetic search. In *Proc. of the 5th International Conference on Genetic Algorithms (ICGA'93)*, pages 2–8. Morgan Kaufmann, 1993.

[BD20] Maxim Buzdalov and Carola Doerr. Optimal mutation rates for the $(1+\lambda)$ EA on OneMax. In *Proc. of Parallel Problem Solving from Nature (PPSN'20)*, volume 12270 of *LNCS*, pages 574–587. Springer, 2020.

[BLS14] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. Unbiased black-box complexity of parallel search. In *Proc. of Parallel Problem Solving from Nature (PPSN'14)*, volume 8672 of *Lecture Notes in Computer Science*, pages 892–901. Springer, 2014.

[CD18a] Eduardo Carvalho Pinto and Carola Doerr. A simple proof for the usefulness of crossover in black-box optimization. In *Proc. of Parallel Problem Solving from Nature (PPSN'18)*, volume 11102 of *Lecture Notes in Computer Science*, pages 29–41. Springer, 2018. Full version available at http://arxiv.org/abs/1812.00493.

[CD18b] Eduardo Carvalho Pinto and Carola Doerr. Towards a more practice-aware runtime analysis of evolutionary algorithms. *CoRR*, abs/1812.00493, 2018.

[CHJ+17] Dogan Corus, Jun He, Thomas Jansen, Pietro Simone Oliveto, Dirk Sudholt, and Christine Zarges. On easiest functions for mutation operators in bio-inspired optimisation. *Algorithmica*, 78:714–740, 2017.

[CWA14] Francisco Chicano, Darrell Whitley, and Enrique Alba. Exact computation of the expectation surfaces for uniform crossover along with bit-flip mutation. *Theoretical Computer Science*, 545:76–93, 2014.

[DD16] Benjamin Doerr and Carola Doerr. The impact of random initialization on the runtime of randomized search heuristics. *Algorithmica*, 75:529–553, 2016.

[DD18] Benjamin Doerr and Carola Doerr. Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm. *Algorithmica*, 80:1658–1709, 2018.

[DD20] Benjamin Doerr and Carola Doerr. Theory of parameter control mechanisms for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 271–321. Springer, 2020.

[DDK18] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica*, 80:1732–1768, 2018.

[DDY16] Benjamin Doerr, Carola Doerr, and Jing Yang. k-bit mutation with self-adjusting k outperforms standard bit mutation. In *Proc. of Parallel Problem Solving from Nature (PPSN'16)*, volume 9921 of *Lecture Notes in Computer Science*, pages 824–834. Springer, 2016.

[DDY20] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. *Theoretical Computer Science*, 801:1–34, 2020.

[DGWY19] 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.

[DJW12] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. *Algorithmica*, 64:673–697, 2012.

[DL17] Carola Doerr and Johannes Lengler. Introducing elitist black-box models: When does elitist behavior weaken the performance of evolutionary algorithms? *Evolutionary Computation*, 25:587–606, 2017.

[DLMN17] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*, pages 777–784. ACM, 2017.

[DLOW18] Benjamin Doerr, Andrei Lissovoi, Pietro Simone Oliveto, and John Alasdair Warwicker. On the runtime analysis of selection hyper-heuristics with adaptive learning periods. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 1015–1022. ACM, 2018.

[dPdLDD15] Axel de Perthuis de Laillevault, Benjamin Doerr, and Carola Doerr. Money for nothing: Speeding up evolutionary algorithms through better initialization. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'15)*, pages 815–822. ACM, 2015.

[DW18] Carola Doerr and Markus Wagner. On the effectiveness of simple success-based parameter selection mechanisms for two classical discrete black-box optimization benchmark problems. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 943–950. ACM, 2018.

[DWY18a] Benjamin Doerr, Carsten Witt, and Jing Yang. Runtime analysis for self-adaptive mutation rates. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*, pages 1475–1482. ACM, 2018.

[DWY+18b] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *arXiv e-prints:1810.05281*, October 2018. IOHprofiler is available at https://github.com/IOHprofiler.

[FCSS08] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Extreme value based adaptive operator selection. In *Proc. of Parallel Problem Solving from Nature (PPSN'08)*, volume 5199 of *Lecture Notes in Computer Science*, pages 175–184. Springer, 2008.

[FCSS09] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In *Proc. of Learning and Intelligent Optimization (LION'09)*, volume 5851 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2009.

[GW18] Christian Gießen and Carsten Witt. Optimal mutation rates for the (1+λ) EA on OneMax through asymptotically tight drift analysis. *Algorithmica*, 80(5):1710–1731, 2018.

[HPR+18] Hsien-Kuei Hwang, Alois Panholzer, Nicolas Rolin, Tsung-Hsi Tsai, and Wei-Mei Chen. Probabilistic analysis of the (1+1)-evolutionary algorithm. *Evolutionary Computation*, 26, 2018.

[HW19] Hsien-Kuei Hwang and Carsten Witt. Sharp bounds on the runtime of the (1+1) EA via drift analysis and analytic combinatorial tools. In *Proc. of ACM/SIGEVO Conference on Foundations of Genetic Algorithms (FOGA'19)*, pages 1–12. ACM, 2019.

[JOP+] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[KHE15] Giorgos Karafotias, Mark Hoogendoorn, and A.E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19:167–187, 2015.

[LOW20] 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*, 28(3):437–461, 2020.

[LS11] Jörg Lässig and Dirk Sudholt. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Proc. of Foundations of Genetic Algorithms (FOGA'11)*, pages 181–192. ACM, 2011.

[Sud13] Dirk Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17:418–435, 2013.

[Thi09] Dirk Thierens. On benchmark properties for adaptive operator selection. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'09), Companion Material*, pages 2217–2218. ACM, 2009.

[Wit13] Carsten Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing*, 22:294–318, 2013.

[YDB19] Furong Ye, Carola Doerr, and Thomas Bäck. Interpolating local and global search by controlling the variance of standard bit mutation. In *Proc. of IEEE Congress on Evolutionary Computation (CEC'19)*, pages 2292–2299. IEEE, 2019.

Algorithm	problem dimension									
	100	500	1,000	1,500	2,000	2,500	3,000	3,500	4,000	4,500
RLS_{opt}	433.4	2,975.0	6,645.0	10,576.7	14,678.4	18,906.4	23,235.2	27,647.7	32,131.9	36,678.8
$\text{RLS}_{\text{drift}}$	433.6	2,975.3	6,645.2	10,576.9	14,678.6	18,906.7	23,235.5	27,648.0	32,132.2	36,679.0
RLS	450	3,051	6,793	10,797	14,971	19,272	23,673	28,158	32,714	37,333
(1+1) $\text{EA}_{>0,\text{opt}}$	437	2,979	6,665	10,606	14,717	18,954	23,292	27,714	32,207	36,763
(1+1) $\text{EA}_{>0,\text{opt},p_{\min}=1/(2n)}$	534	3,700	8,321	13,270	18,441	23,775	29,239	34,813		
(1+1) $\text{EA}_{>0,\text{opt},p_{\min}=1/n}$	663	4,666	10,548	16,865	23,473	30,298	37,296	44,438		
(1+1) $\text{EA}_{>0,\text{drift}}$	437	2,989	6,682	10,648	14,795					
(1+1) $\text{EA}_{>0,\text{drift},p_{\min}=1/(2n)}$	534	3,711	8,322	13,271	18,442	23,776	29,242	34,814		
(1+1) $\text{EA}_{>0,\text{drift},p_{\min}=1/n}$	664	4,682	10,549	16,865	23,473	30,298	37,297	44,438		
(1+1) $\text{EA}_{>0,\text{static},p=1/(2n)}$	550	3,781	8,458	13,475	18,712	24,113	29,644	35,284		
(1+1) $\text{EA}_{>0,\text{static},p=1/n}$	679	4,751	10,684	17,066	23,740	30,631	37,695	44,902	52,233	59,672
$(1+1)$ EA_{opt}	1,006	7,189	16,254	26,031	36,269	46,850	57,705	68,788	80,065	
$(1+1)$ EA_{drift}	1,006	7,189	16,254	26,031	36,269	46,850	57,706	68,788	80,066	91,513
(1+1) $\text{EA}_{\text{Bäck}}$	1,008	7,203	16,283	26,075	36,328	46,925	57,795	68,893	80,186	91,649
(1+1) $\text{EA}_{\text{static},p=\text{opt}}$	1,058	7,461	16,807	26,867	37,389	48,257	59,398			
(1+1) $\text{EA}_{\text{static},p=1/n}$	1,071	7,510	16,896	26,992	37,550	48,451	59,626	71,028	82,625	94,392

Table 1: Expected Optimization Times of Different Variants of RLS and the (1+1) EA on OneMax for problem dimensions between $n = 100$ and $n = 4{,}500$.