HAL open science

Offspring Population Size Matters when Comparing Evolutionary Algorithms with Self-Adjusting Mutation Rates

Anna Rodionova, Kirill Antonov, Arina Buzdalova, Carola Doerr

To cite this version:

Anna Rodionova, Kirill Antonov, Arina Buzdalova, Carola Doerr. Offspring Population Size Matters when Comparing Evolutionary Algorithms with Self-Adjusting Mutation Rates. Genetic and Evolutionary Computation Conference, Jul 2019, Prague, Czech Republic. pp.855-863, 10.1145/3321707.3321827 . hal-02439392

HAL Id: hal-02439392

https://hal.sorbonne-universite.fr/hal-02439392v1

Submitted on 14 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Offspring Population Size Matters when Comparing Evolutionary Algorithms with Self-Adjusting Mutation Rates

Anna Rodionova
ITMO University, Saint Petersburg, Russia

Kirill Antonov
ITMO University, Saint Petersburg, Russia

Arina Buzdalova
ITMO University, Saint Petersburg, Russia
abuzdalova@gmail.com

Carola Doerr
Sorbonne Université, CNRS, LIP6, Paris, France
Carola.Doerr@sorbonne-universite.fr

ABSTRACT

We analyze the performance of the 2-rate $(1 + \lambda)$ Evolutionary Algorithm (EA) with self-adjusting mutation rate control, its 3-rate counterpart, and a $(1 + \lambda)$ EA variant using multiplicative update rules on the OneMax problem. We compare their efficiency for offspring population sizes ranging up to $\lambda = 3,200$ and problem sizes up to $n = 100{,}000$.

Our empirical results show that the ranking of the algorithms is very consistent across all tested dimensions, but strongly depends on the population size. While for small values of λ the 2-rate EA performs best, the multiplicative updates become superior for starting for some threshold value of λ between 50 and 100. Interestingly, for population sizes around 50, the $(1 + \lambda)$ EA with static mutation rates performs on par with the best of the self-adjusting algorithms.

We also consider how the lower bound $p_{\min}$ for the mutation rate influences the efficiency of the algorithms. We observe that for the 2-rate EA and the EA with multiplicative update rules the more generous bound $p_{\min} = 1/n^2$ gives better results than $p_{\min} = 1/n$ when λ is small. For both algorithms the situation reverses for large λ.

CCS CONCEPTS

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

1 INTRODUCTION

A key driver for the success of evolutionary algorithms (EAs) is their *global search behavior*, i.e., their capability of searching the whole decision space without getting stuck in local optima. This feature distinguishes EAs from other well-known heuristics such as Simulated Annealing and other local hill climbers, which search the decision space only within a small neighborhood. The global search behavior of EAs, however, comes at the cost of a less focused search when the optimization converges, which can result in performances losses in the final parts of the optimization process. The question how to most effectively combine the best of both worlds is the driving force behind research on *parameter control* [1, 18, 24], *adaptive operator selection* [20, 28], and *hyper-heuristics* [6], which are the most prominent umbrella terms for adjusting the structure of the search behavior to the current needs of an iterative optimization process.

Interestingly, research on parameter control has shown that in many applications the search behavior can be adjusted very efficiently by quite simple update rules, see the above-cited surveys for details. The considerable performance gains observed in practice have inspired a whole series of theoretical works on non-static parameter choices. In the last years, an increasing number of results appeared which rigorously quantify the advantages of parameter control, see [10] for a summary and classification of the mechanisms.

We will focus in this work on two different self-adjusting approaches previously shown to yield excellent performances on the OneMax benchmark problem: a 2-rate success control and a generalized one-fifth success rule. We analyze their efficiency when implemented in a $(1 + \lambda)$ EA framework. Our key research question concerns the scalability of performance with respect to the offspring population size λ and with respect to the problem dimension n. A summary of findings will be presented in Section 1.2.

1.1 Self-Adjusting Algorithms

The One-Fifth Success Rule Applied to the $(1 + \lambda)$ EA. The *one-fifth success rule* originally stems from theoretical observations made by Rechenberg for the optimal step-size adaptation in the (1+1) Evolution Strategy [29]. An interpretation of this rule which is suitable also for the adaptation of other parameters, such as the mutation rate in discrete EAs, was presented in [25]. The rule itself is simple: if after one iteration of an algorithm a strictly better offspring has been found, the parameter under consideration is multiplied with some constant F, and it is multiplied with $F^{-1/4}$ otherwise. With this setting it holds that after 5 iterations the parameter is the same as in the first if exactly one out of the five iterations was "successful" (i.e., found a better solution), and it is increased or decreased otherwise, depending on the sign of $1 - F$ and the number of successful iterations. The one-fifth success rule was shown to yield very efficient optimization times for the (1 +

(λ, λ)) Genetic Algorithm (GA), first by empirical means [11] and later by rigorous mathematical running time analysis [9].

Since the one-fifth success rule had been derived only for the (1+1) ES and only for particular benchmark problems (the sphere and a corridor function [29]), it seems natural to generalize the update mechanisms to other success rules. In fact, multiplying the parameters by 2 and 1/2 in the case of a successful and an unsuccessful iteration, respectively, has been experimented with in various context. It has also been rigorously proven to be efficient for the optimization of different benchmark functions with a $(1+\lambda)$ EA variant that uses a self-adjusting offspring population size λ [26]. In [16] an empirical study analyzed the impact of the update factors on the performance of a (1+1) EA variant. This algorithm samples in each iteration one offspring y from the current best solution x and updates the mutation rate p to bp (with $b < 1$ being some constant) if y is at least as good as its parent (i.e., in our setting with *maximization* as objective, if and only if $f(y) \geq f(x)$). In this situation x is replaced by y. If y is strictly worse than x (i.e., if $f(y) < f(x)$) the mutation rate is increased to Ap, where $A > 1$ is again some constant. The results presented in [16] show that this self-adjusting $(1+1)$ EA(A, b) is very efficient on two classic benchmark problems OneMax and LeadingOnes, and this holds for broad ranges of update strengths A and b. In the very recent work [12] it was rigorously proven that this algorithm, for suitably chosen hyper-parameters A and b, achieves optimal optimization time on LeadingOnes, up to lower order terms.

In this work, we will extend the $(1+1)$ EA(A, b) to the $(1+\lambda)$ EA, which samples λ offspring per each iteration. Since the probability of creating individuals which are equally good, but not strictly better than the parent, increases considerably for large λ, we have to refine the interpretation of "success" for this context. Taking into account that the fraction of offspring y satisfying $f(y) \geq f(x)$ showed promising performances in initial experiments, in our $(1+\lambda)$ EA(A, b) we therefore discriminate with respect to whether or not at least 5% of the offspring are at least as good as their parent.

2-Rate and 3-Rate Update Rules. An alternative way to control the mutation rate was proposed and analyzed in [14]. Their 2-rate $(1+\lambda)$ EA, which we refer to as 2-rate $(1+\lambda)$ EA$_{r/2,2r}$ in the following, uses two different mutation rates in each iteration. Half the offspring are created with mutation rate $p/2$ and the other $\lambda/2$ offspring are sampled with mutation rate $2p$. The mutation rate is parametrized as $p = r/n$ in the 2-rate $(1+\lambda)$ EA$_{r/2,2r}$. The value of r is updated after each iteration, by a random decision which favors the rate by which the best offspring has been created (details will be presented in Section 2). It was proven in [14] that the 2-rate $(1+\lambda)$ EA$_{r/2,2r}$ with $\lambda \geq 45$ and $\lambda = n^{O(1)}$ is not only more efficient for the optimization of OneMax than any $(1+\lambda)$ EA with static mutation rates, but, from a comparison with a lower bound proven in [5], their result also implies that the 2-rate $(1+\lambda)$ EA$_{r/2,2r}$ even achieves asymptotically optimal expected optimization time, which is $\Theta\left(\frac{n}{\log \lambda} + \frac{n \log n}{\lambda}\right)$ when measured in terms of generations.

We also study a 3-rate variant of 2-rate $(1+\lambda)$ EA$_{r/2,2r}$, which creates one third of the offspring with mutation rate $c_1 r/n$, r/n, and $c_2 r/n$, respectively, where $0 < c_1 < 1$ and $1 < c_2$ are hyper-parameters set by the user. In the original work [14] a similar 3-rate variant has been studied by empirical means. We extend their preliminary study by considering 100 different pairs of (c_1, c_2), but also by considering its performance for a broader range of population sizes λ and much larger dimensions n.

1.2 Summary of Results

We investigate the efficiency of the three above-mentioned algorithms (i.e., the $(1+\lambda)$ EA(A, b), the 2-rate $(1+\lambda)$ EA$_{r/2,2r}$, and the 3-rate $(1+\lambda)$ EA$_{r/2,r,2r}$) on OneMax, for various population sizes λ up to 3,200 and problem sizes up to $n = 100{,}000$. Their performances are compared among each other and to the traditional $(1+\lambda)$ EA using static mutation rates.

Interestingly, the ranking of the algorithms is very consistent across all tested dimensions, and only depends on the offspring population size λ. More precisely, we show that for small population sizes $\lambda \leq 50$, the 2-rate $(1+\lambda)$ EA$_{r/2,2r}$ performs best, while for $\lambda \geq 100$ the $(1+\lambda)$ EA(A, b) is the most efficient among the four algorithms. It is also worth mentioning that for λ of about 50, the conventional $(1+\lambda)$ EA with static mutation rates performs on par with the best performing self-adjusting algorithm, which is 2-rate $(1+\lambda)$ EA$_{r/2,2r}$.

All our algorithms are implemented with the *shift mutation operator* discussed in [8]. This operator, unlike the unconditional standard bit mutation operator traditionally studied in the theory of evolutionary computation, ensures that offspring differ from their parent by at least one bit. This is achieved as follows. The operator first draws the *mutation strength* ℓ (i.e., the number of bits that are flipped to create the offspring) from the binomial distribution $\mathrm{Bin}(n, p)$, where the parameter p denotes the mutation rate. While the traditional standard bit mutation operator allows $\ell = 0$, it is easily seen that, for $(1+\lambda)$ EAs, which only evolve a single solution, an offspring that is identical to its parent cannot advance the search. The shift mutation operator therefore interprets $\ell = 0$ as a vote for a small search radius, and flips exactly one bit, i.e., effectively using $\ell = 1$. Put differently, this operator always flips at least one bit, and its probability of flipping exactly one bit equals $\mathrm{Bin}(n, p)(0) + \mathrm{Bin}(n, p)(1)$.

When the mutation rate p converges to zero, the shift mutation operator converges against the operator using mutation strength one deterministically, thus effectively reducing the global search property of standard bit mutation to a purely local search. As explained above, we would like to avoid performing a purely random search only, and therefore cap the mutation rate at a lower bound $p_{\min}$. This lower bound can have a significant impact on the performance, as our further experimental results demonstrate. More precisely, we observe that for smaller population sizes a lower bound of $1/n^2$ seems to work better than a lower bound of $1/n$. Interestingly, the situation reverses for larger population sizes. These observations are true for all considered self-adjusting algorithms, but with different values of λ. For 2-rate $(1+\lambda)$ EA$_{r/2,2r}$, the transition population size is between $\lambda = 100$ and $\lambda = 200$, while for $(1+\lambda)$ EA(A, b) it is between $\lambda = 400$ and $\lambda = 800$.

In light of the consistent behavior across all tested dimensions, we are confident that our findings will inspire future rigorous theoretical analyses for the self-adjusting algorithms investigated herein.

1.3 Related Work

Our work can, to some extent, be seen as an extension of [17], where different $(1+\lambda)$ EA variants have been studied on the two benchmark problems OneMax and LeadingOnes. Note though that we consider here much larger offspring population sizes λ (up to 3, 200) and much larger dimensions (up to 100,000), whereas the results presented in [17] are restricted to settings with $n \leq 4000$, and $\lambda \in \{1, 2, 5, 10, 50\}$. As discussed above, the ranking of the algorithms strongly depends on the size of λ, so that our work gives a much more complete picture than the results presented in [17].

1.4 Structure of the Paper

The rest of the paper is structured as follows. In Section 2 we describe the considered algorithms in more detail. Section 3 gives a general overview of how each algorithm performs across different dimensions and population sizes. In Section 4 the influence of the lower bound for the mutation probability is studied, and conclusions about the ranking of the self-adjusting algorithms are made. Section 5 provides insights into the anytime performance of the algorithms from the fixed-budget perspective. In Section 6 we compare the 2-rate $(1+\lambda)$ $\mathrm{EA}_{r/2,2r}$ with its 3-rate counterpart. Conclusions and avenues for future work are presented in Section 7.

1.5 Additional Plots and Statistics

Exact figures for the plots presented in this paper, additional comparative plots, as well as information about the standard deviations of the optimization times are available in [30].

2 ONEMAX AND THREE $(1+\lambda)$ EA VARIANTS

In this section we describe the baseline $(1+\lambda)$ $\mathrm{EA}_{0\to1}$ algorithm and its two self-adjusting variants. The description of the algorithms assumes the *maximization* of a pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$ as the optimization task. All empirical results in the further sections are obtained for the OneMax benchmark problem $\mathrm{Om} : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^n x_i$. OneMax is the most widely studied benchmark problem in the theory of EAs [2, 21], but also serves as a recurring benchmark problem in empirical studies, and in particular in the context of parameter control [3, 4, 16, 19, 22]. A discussion of properties that make OneMax a suitable test problem for adaptive algorithms was offered by Thierens in [31]. Apart from the better comparison with existing results, the availability of theoretical performance limits of adaptive and static evolutionary algorithms makes OneMax a particularly appealing benchmark problem; see [15] for a survey of such *black-box complexity* bounds. In the context of our work, the asymptotic $\Theta(\frac{n}{\log\lambda} + \frac{n\log n}{\lambda})$ bound for all λ-parallel EAs proven in [5] and the precise $n\ln(n) - cn \pm o(n)$ bound for any unary unbiased black-box algorithm from [13] are the probably most relevant results. As mentioned at the end of Section 1.2, we are also confident that further advances in running time analysis will allow us to convert our empirical findings into rigorous mathematical statements.

Before we describe our algorithms, we recall from [22] that for the optimization of OneMax the best static offspring population size is $\lambda = 1$. All larger values result in worse expected optimization times. However, the *parallel optimization time*, measured by the number of generations needed to find the optimum, of a $(1+\lambda)$ EA with $\lambda > 1$ can (and typically is) smaller than that of the $(1+1)$ EA.

Algorithm 1: The $(1+\lambda)$ $\mathrm{EA}_{0\to1}$ with mutation rate $0 < p < 1$ for the maximization of $f : \{0,1\}^n \to \mathbb{R}$

Initialization: Sample $x \in \{0,1\}^n$ u.a.r. and evaluate $f(x)$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  for $i = 1, \ldots, \lambda$ **do**
    Sample $\ell^{(i)}$ from $\mathrm{Bin}_{0\to1}(n,p)$, sample $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(x)$ and evaluate $f(y^{(i)})$;
  Sample x^* from $\arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ u.a.r.;
  if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;

The basic algorithm for our study is the $(1+\lambda)$ $\mathrm{EA}_{0\to1}$. Its description is given in Algorithm 1. The $(1+\lambda)$ $\mathrm{EA}_{0\to1}$ samples λ solution candidates in every iteration, but only the best one of these *offspring* survives – ties broken uniformly at random (u.a.r.). Each offspring is created by the *shift mutation operator*, which first samples a mutation strength λ from the $\mathrm{Bin}_{0\to1}(n,p)$ distribution and then flips the entries in ℓ randomly chosen, pairwise different bit positions. Here and in the following, the distribution $\mathrm{Bin}_{0\to1}(n,p)$ assigns to each integer $2 \leq k \leq n$ the value $\mathrm{Bin}_{0\to1}(n,p)(k) = \mathrm{Bin}(n,p)(k)$, sets $\mathrm{Bin}_{0\to1}(n,p)(0) = 0$, and $\mathrm{Bin}_{0\to1}(n,p)(1) = \mathrm{Bin}(n,p)(1) + \mathrm{Bin}(n,p)(0)$.

We implemented sampling from $\mathrm{Bin}_{0\to1}(n,p)$ as follows. First, each bit of a string of length n is inverted with probability p. Then it is checked whether at least one bit was inverted. If not, one random bit position is chosen uniformly at random and the corresponding bit is inverted.

We note without going into much detail that apart from the "shift" strategy $0 \to 1$, which assigns the probability mass of sampling $\ell = 0$ to $\ell = 1$, a second approach to deal with 0-bit flips was suggested in [8]. This alternative approach, coined "resampling strategy" and denoted by a subscript > 0 in [8] and its follow-up works [16, 17], distributes the probability mass $\mathrm{Bin}(n,p)(0)$ proportionally to all integers $1 \leq \ell \leq n$. Initial experiments with this resampling strategy indicate similar finding as those presented for the shift strategy used here in this work. A detailed examination is left for future work.

Next, we consider two self-adjusting variants of the $(1+\lambda)$ EA. The first one is the 2-rate $(1+\lambda)$ $\mathrm{EA}_{r/2,2r}$ proposed in [14]. This algorithm is summarized in Algorithm 2. It creates one half of the offspring with mutation rate $2r/n$ and the other half of the offspring with mutation rate $r/(2n)$, respectively. At the end of each iteration, the 2-rate $(1+\lambda)$ $\mathrm{EA}_{r/2,2r}$ updates the mutation rate based on which value was used when the best offspring was obtained. Note here that with probability $1/2$ a random decision is made whether to update the mutation rate to $2r/n$ or to $r/(2n)$, thus effectively assigning a $3/4$ chance to update to the value by which the best offspring has been created. The reason not to update to this rate deterministically was explained in [14] by the fact that the constant probability to update the mutation rate in the seemingly unfavorable direction can be useful in the optimization of non-unimodal functions; see [14, Section 3.1] for a more detailed discussion.

Algorithm 2: The 2-rate $(1+\lambda)$ $\mathrm{EA}_{r/2,2r}$ with adaptive mutation rates proposed in [14]

Initialization: Sample $x \in \{0,1\}^n$ u.a.r. and evaluate $f(x)$;
Initialize $r \leftarrow r^{\mathrm{init}}$; // Following [14] we use $r^{\mathrm{init}} = 2$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  for $i = 1, \ldots, \lfloor \lambda/2 \rfloor$ **do**
    Sample $\ell^{(i)}$ from $\mathrm{Bin}_{0\to 1}(n, r/(2n))$, create $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  for $i = \lfloor \lambda/2 \rfloor + 1, \ldots, \lambda$ **do**
    Sample $\ell^{(i)}$ from $\mathrm{Bin}_{0\to 1}(n, 2r/n)$, create $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  $x^* \leftarrow \arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ (ties broken u.a.r.);
  if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;
  Perform one of the following two actions with prob. $1/2$:
    - replace r with the mutation rate that x^* has been created with;
    - replace r with either $2r$ or $r/2$ equiprobably.
  $r \leftarrow \min\{\max\{2, r\}, n/4\}$;

We complement our study by a performance analysis for an extension of the $(1+1)$ $\mathrm{EA}(A, b)$ studied in [16]. In this extension we adapt the success-based multiplicative update rule used in the $(1+1)$ $\mathrm{EA}(A, b)$ to the case that more than one offspring is generated per each iteration. We call our extension the $(1+\lambda)$ $\mathrm{EA}(A, b)$. Algorithm 3 provides its pseudo-code. In the original algorithm from [16], the mutation rate is updated based on whether or not the offspring replaces its parent. In our case, we have λ offspring, and we refine the update rule as follows. First, we compute the number of "good" offspring, which are at least as good as the parent (see line 6 of Algorithm 3). If the share of good offspring is at least 5%, the mutation rate is multiplied by the factor $A > 1$. It is decreased to bp otherwise, where the factor b is some constant satisfying $0 < b < 1$. In this work we set $A = 2$ and $b = 1/2$.

One may wonder why we use the 5% threshold. This is based on some preliminary experiments with $\lambda = 1{,}600$, where we observed good performances for this value. It is likely that this threshold depends on the population size λ. A detailed study is left for future work.

3 ALGORITHM'S PERFORMANCE BY DIMENSION

In a first step, we analyze how each algorithm performs across different problem dimensions. To this end, we compute the average optimization time (i.e., the number of function evaluations needed to find an optimal solution) of each of the algorithms described in Section 2 for 100 independent runs. In Figure 1 we show these values for the algorithm $(1+\lambda)$ $\mathrm{EA}(A, b)$, for different values of λ. We also show the average parallel optimization time, measured by the number of generations needed to find an optimal solution, for the $(1+\lambda)$ $\mathrm{EA}(A, b)$ in Figure 2. The parallel optimization time equals the average optimization time divided by the offspring population size λ. This performance measure is useful when the fitness evaluations within one generation are made in parallel.

Algorithm 3: The $(1+\lambda)$ $\mathrm{EA}(A, b)$ with adaptive mutation rates and update strengths $A > 1$, $0 < b < 1$

1 **Initialization:** Sample $x \in \{0,1\}^n$ u.a.r. and evaluate $f(x)$;
2 Initialize $p \leftarrow 1/n$;
3 **Optimization: for** $t = 1, 2, 3, \ldots$ **do**
4 **for** $i = 1, \ldots, \lambda$ **do**
5 Sample $\ell^{(i)}$ from $\mathrm{Bin}_{0\to 1}(n, p)$, create $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
6 $N \leftarrow |\{i \in [\lambda] \mid f(x^{(i)}) \geq f(x)\}|$;
7 **if** $N \geq \lceil 0.05\lambda \rceil$ **then** $p \leftarrow \min\{1/2, Ap\}$ **else** $p \leftarrow \max\{1/n, bp\}$;
8 $x^* \leftarrow \arg\max\{f(x^{(1)}), \ldots, f(x^{(\lambda)})\}$ (ties broken u.a.r.);
9 **if** $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;

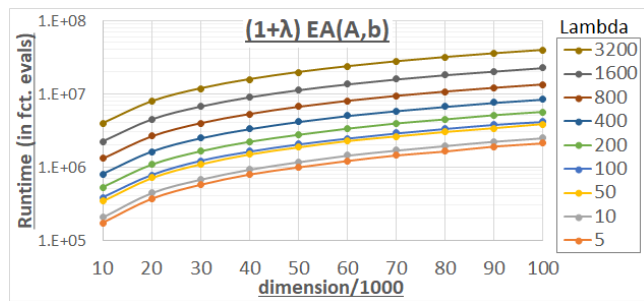


Figure 1: Average optimization times for 100 independent runs of the $(1+\lambda)$ EA(A, b) for different values $5 \leq \lambda \leq 3{,}200$ and different problem dimensions $10^4 \leq n \leq 10^5$.

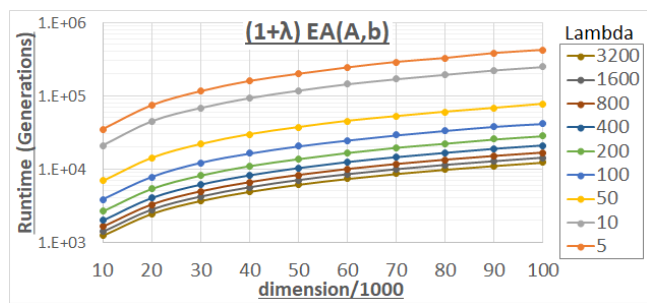
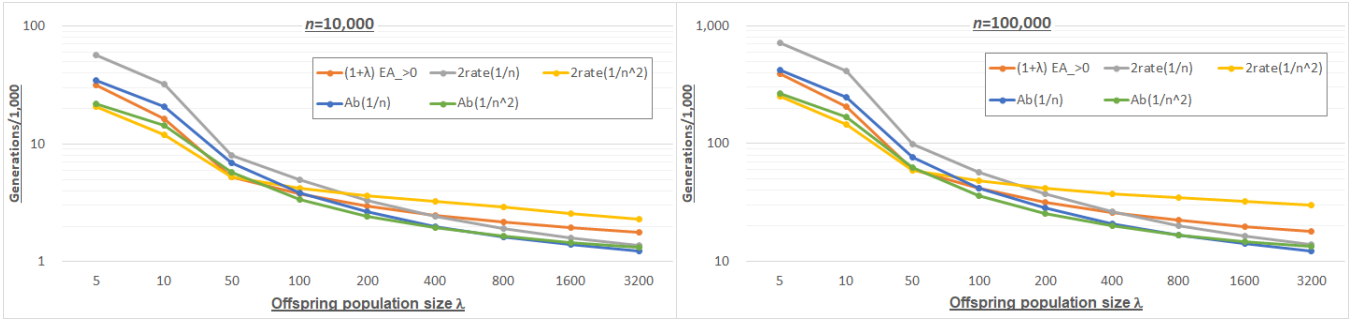


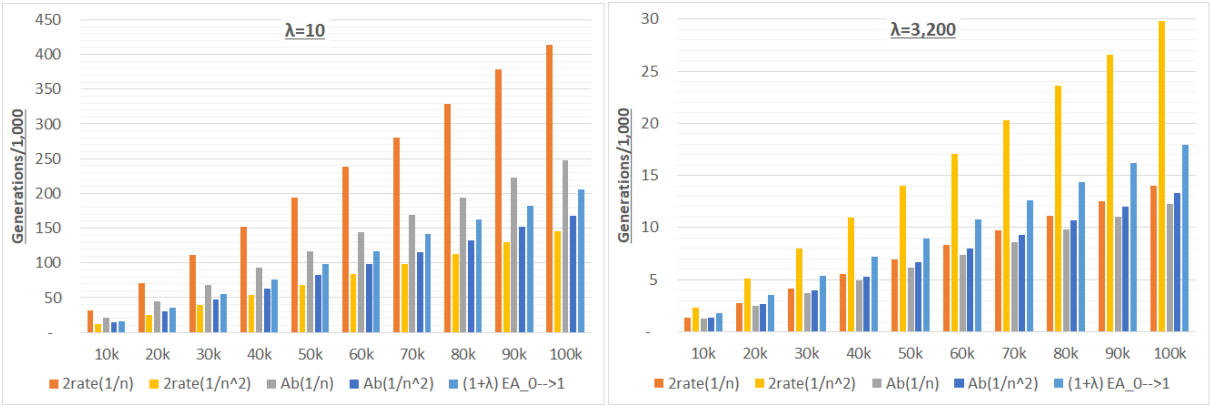
Figure 2: Average parallel optimization times for 100 independent runs of the $(1+\lambda)$ EA(A, b) for different values $5 \leq \lambda \leq 3{,}200$ and different problem dimensions $10^4 \leq n \leq 10^5$.

We note without details that the plots for the 2-rate $(1+\lambda)$ $\mathrm{EA}_{r/2,2r}$ and $(1+\lambda)$ $\mathrm{EA}_{0\to 1}$ look very similar, but with different values. The relative standard deviation is small for all algorithms: it starts at about 10-15% for $\lambda = 5$ and decreases to just 0.5-1.5% for $\lambda = 3200$, so we can draw significant conclusions from the average optimization time and the average parallel optimization time plots. Exact figures and standard deviations for all plots shown in this paper are available in our full report [30].



(a) (b)

Figure 3: Average parallel optimization times for the different $(1+\lambda)$ EA variants to find an optimal solution (a) on the 10,000-dimensional and (b) on the 100,000-dimensional OneMax problem. The averages are for 100 independent runs each.



(a) (b)

Figure 4: Average parallel optimization times for (a) the different (1+10) EA variants and (b) the different (1+3,200) EA variants to find an optimal solution on the n-dimensional OneMax problem, with $10^4 \le n \le 10^5$. The averages are for 100 independent runs each.

As expected, we see that for each algorithm and each dimension the average optimization times strictly increase with increasing λ. Therefore, small values of λ are optimal in terms of optimization time. However, in terms of parallel optimization time, the situation is reversed: the algorithms with larger population sizes λ require fewer iterations to find the optimum. Therefore, both small and large values of λ are worth being used under different circumstances.

4 THE IMPACT OF THE LOWER BOUND

One of the key advantages of sampling the mutation rates from the distribution $\mathrm{Bin}_{0\to 1}(n,p)$ is that it allows to transition from a classical $(1+\lambda)$ EA to a $(1+\lambda)$ variant of RLS that creates each offspring by flipping exactly one uniformly chosen bit. This transition is achieved by reducing p beyond $1/n$, see [7] for a discussion. In the algorithms studied above, we had enforced a lower bound of $1/n$ for p. We now relax this lower bound and only require that $p \ge 1/n^2$. We study the impact of this relaxed lower bound on the performance of the $(1+\lambda)$ EA(A,b) and the 2-rate $(1+\lambda)$ EA$_{r/2,2r}$ algorithm. The algorithms with these relaxed lower bounds are denoted by $(1+\lambda)$ EA$(A,b,1/n^2)$ and 2-rate $(1+\lambda)$ EA$_{r/2,2r}(1/n^2)$, respectively.

In Figures 3a and 3b we show the average parallel optimization times of the different algorithms for $n = 10{,}000$ and $n = 100{,}000$, respectively. Recall that the average parallel optimization time equals the average number of generations. Note also that we use a logarithmic scale, to ease the comparison. Note that the standard deviation is still relatively small for the algorithms using $p_{\min} = 1/n^2$ as well. The only difference is that for 2-rate $(1+\lambda)$ EA$_{r/2,2r}(1/n^2)$ it does not decrease with increasing λ and is always around 5-10%.

The picture for $n = 10{,}000$ is quite similar to that for $n = 100{,}000$, and, in fact, for all tested dimensions. To demonstrate this, we plot in Figures 4a and 4b the average parallel optimization times for fixed $\lambda = 10$ and $\lambda = 3{,}200$, respectively, which confirm a stable ranking of the different algorithms. We use the parallel optimization

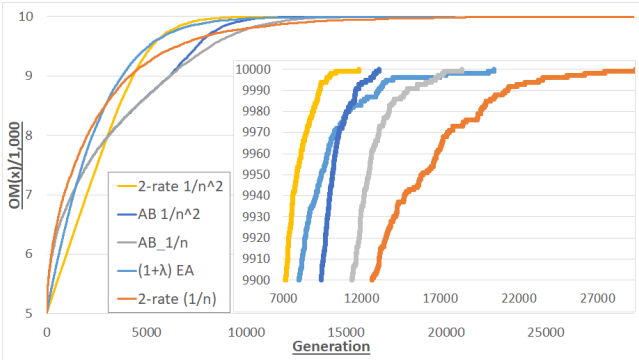


Figure 5: Evolution of the average function value over 100 independent runs of different $(1 + 10)$ EA variants on the 10,000-dimensional OneMax problem, in dependence of the generation. The small plot in the lower right corner zooms into the region in which all algorithms have found function values $\geq 9,900$.

time again as performance measure, to ease the comparison with Figures 3a and 3b.

From Figures 3a and 3b we can derive threshold values for λ at which the ranking of the algorithms changes.

- For $5 \leq \lambda \leq \lambda_1$ with λ_1 being a threshold value between 50 and 100, the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}(1/n^2)$ performs best.
- For $\lambda_1 \leq \lambda \leq \lambda_2$ with $400 \leq \lambda_2 < 800$ the $(1 + \lambda)$ $\mathrm{EA}(A, b, 1/n^2)$ shows best performance.
- For $\lambda_2 \leq \lambda \leq 3,200$ the $(1 + \lambda)$ $\mathrm{EA}(A, b)$ minimizes the expected optimization time.

We see that the ranking of algorithms strongly depends on the population size λ. For small values of λ, the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}(1/n^2)$ performs the best, while for medium values the $(1 + \lambda)$ $\mathrm{EA}(A, b, 1/n^2)$ wins, and then for large values of λ the $(1 + \lambda)$ $\mathrm{EA}(A, b)$ is superior. Interestingly, for $\lambda = 50$, the $(1 + \lambda)$ $\mathrm{EA}_{0\to1}$, which uses a static mutation rate, performs on par with the best self-adjusting algorithm.

Generally, for both self-adjusting algorithms, it is more preferable to use $1/n^2$ as lower bound for the mutation probability when the population size is small, while for large population sizes a lower bound of $1/n$ seems to work better. For the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}$, the transition population size is some value λ_3 satisfying $100 \leq \lambda_3 < 200$, while for the $(1 + \lambda)$ $\mathrm{EA}(A, b)$ it is some λ_2 with $400 \leq \lambda_2 < 800$. At the same time, while the performance of the 2-rate $(1+\lambda)$ $\mathrm{EA}_{r/2,2r}$ and the 2-rate $(1+\lambda)$ $\mathrm{EA}_{r/2,2r}(1/n^2)$ drastically depends on the population size and the mutation probability lower bound, the $(1+\lambda)$ $\mathrm{EA}(A, b, 1/n^2)$ is more stable and performs rather good for all considered population sizes. Note that the $(1 + \lambda)$ $\mathrm{EA}(A, b, 1/n^2)$ is the only self-adjusting algorithm which is never significantly worse than the baseline $(1 + \lambda)$ EA.

5 FIXED-BUDGET PERFORMANCE

In the sections above, we have only regarded the average optimization times. This measure, albeit very commonly used in the runtime analysis community, does not give any information about how the algorithms perform in an *anytime* sense. This criticism was addressed in [8, 23], where the benefits of *fixed-budget* and *fixed-target* analyses are advertised, respectively. We note that, of course, both performance measures have played an important role in empirical evaluations ever since – the contribution of the mentioned papers is rather to be seen in discussing why these measures provide important insights for theoreticians. Figures 5, 6a, 6b show anytime performance plots for selected $(1 + 10)$ EAs, $(1 + 400)$ EAs, and $(1 + 1600)$ EAs, respectively, on the $n = 10,000$-dimensional OneMax instance.

We see that for $\lambda = 10$ (Figure 5) the algorithms demonstrate quite different performance over time, while for larger values of λ (Figures 6a, 6b) the performance of the most efficient algorithms seems to be rather stable.

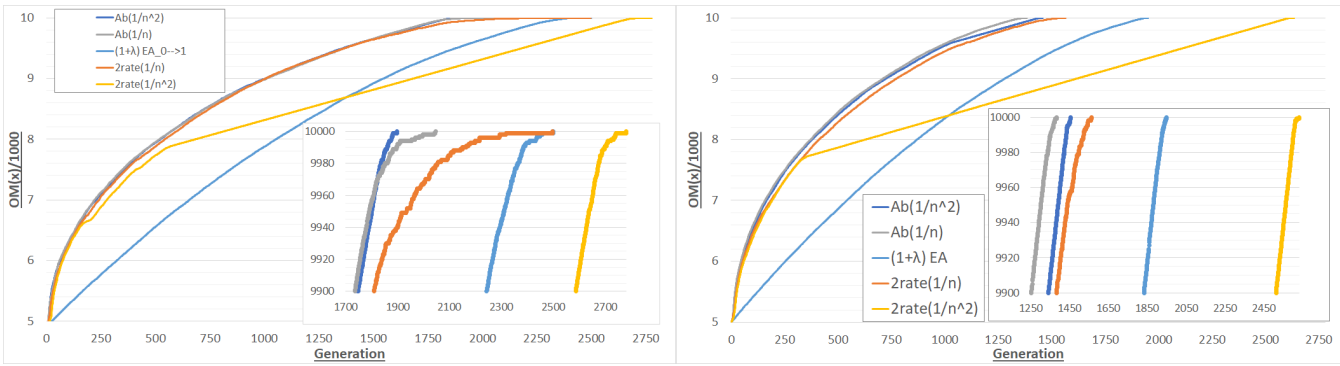
More precisely, the situation is as follows. For generational budgets (note that the budget in terms of fitness evaluations is simply λ times larger than the generational budget, since we are dealing with constant offspring population sizes in this paper) up to around 2,590 the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}$ is the most efficient algorithm; then from 2,950 to 5,615 generations $(1 + \lambda)$ $\mathrm{EA}_{0\to1}$ is the best one, then the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}(1/n^2)$, which remains to be the best performing algorithm for all budgets up to 11,800, at which point it hits the optimum. The runner-up is the $(1 + \lambda)$ $\mathrm{EA}(A, b, 1/n^2)$, third the $(1 + \lambda)$ $\mathrm{EA}(A, b)$, then the $(1 + \lambda)$ $\mathrm{EA}_{0\to1}$ and the worst among the five algorithms is the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}$ (with lower bound $p_{\min} = 1/n$). Therefore, for small population sizes, the ranking of the algorithms depends on the budget.

For $\lambda = 400$ and $\lambda = 1,600$, we see that for all budgets the two algorithms $(1 + \lambda)$ $\mathrm{EA}(A, b, 1/n^2)$ and $(1 + \lambda)$ $\mathrm{EA}(A, b)$ show similar performance, and are the best among the five algorithms. For $\lambda = 400$ the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}$ shows fine performance for budgets up to around 1,700, at which point its performance deteriorates (due to too many offspring sampled with four times the in this case optimal mutation rate). For $\lambda = 1,600$ this effect disappears, and the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}$ shows fine performance. The performance of the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}(1/n^2)$ is seen to be the worst of all algorithms for budgets greater than 1,400 for $\lambda = 400$ and budgets larger than 1,000 for $\lambda = 1,600$.

6 2-RATE VS. 3-RATE ADAPTATION

We next analyze an idea previously communicated in [14], a 3-rate success-based variant of 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}$, which we call the 3-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,r,2r}$. This algorithm creates one third of the offspring with mutation rate $c_1 r/n$, r/n, and $c_2 r/n$, respectively, where $0 < c_1 < 1$ and $1 < c_2$ are hyper-parameters set by the user. The detailed description of the 3-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,r,2r}$ is given in Algorithm 4. In the original work [14] a similar 3-rate variant has been studied. However, there is no pseudocode of this variant, so it is hard to compare the update rule used in the original work and the one we use.

Following [14], *we use the standard bit mutation operator for all algorithms in this section.* It inverts each bit of a string equiprobably with the corresponding mutation rate. So, in contrast to the shift mutation operator used above, it may happen that no bits are flipped in the mutation phase. However, for moderately large λ, the shift mutation operator performs only slightly better than the standard



(a) (b)

Figure 6: Evolution of the average function value over 100 independent runs of different $(1 + 400)$ EA (a) and $(1 + 1600)$ EA (b) variants on the 10,000-dimensional OneMax problem, in dependence of the generation.

one (this empirical observation is also confirmed by the findings made in [17] for the *resampling* strategy, and can be confirmed theoretically; see the full version of [7] for a similar theoretical justification in the context of the $(1 + \lambda)$ EA with static mutation rates), so we believe that the key observations made in this section do not depend on the choice of the mutation operator.

Algorithm 4: The 3-rate $(1 + \lambda)$ $\mathrm{EA}_{c_1 r, r, c_2 r}$ with adaptive mutation rates and three subpopulations

Initialization: Sample $x \in \{0, 1\}^n$ u.a.r. and evaluate $f(x)$;
Initialize $r \leftarrow r^{\mathrm{init}}$; // We use $r^{\mathrm{init}} = 2$;
Initialize $c_1 \geq 1, 0 < c_2 < 1$; // We use $c_1 = 0.7, c_2 = 1.4$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  for $i = 1, \ldots, \lfloor \lambda/3 \rfloor$ **do**
    Sample $\ell^{(i)}$ from $\mathrm{Bin}(n, c_1 r/n)$, create $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  for $i = \lfloor \lambda/3 \rfloor + 1, \ldots, \lfloor 2\lambda/3 \rfloor$ **do**
    Sample $\ell^{(i)}$ from $\mathrm{Bin}(n, r/n)$, create $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  for $i = \lfloor 2\lambda/3 \rfloor + 1, \ldots, \lambda$ **do**
    Sample $\ell^{(i)}$ from $\mathrm{Bin}(n, c_2 r/n)$, create $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  $x^* \leftarrow \arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ (ties broken u.a.r.);
  if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;
  Perform one of the following two actions with prob. $1/2$:
    - replace r with the mutation rate that x^* has been created with;
    - replace r with either $c_1 r$ or $c_2 r$ equiprobably.
  $r \leftarrow \min\{\max\{2, r\}, n/4\}$;

As we have seen in the previous sections, the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2,2r}$ performs worse than the $(1+\lambda)$ $\mathrm{EA}(A, b)$ for large population sizes. To ensure that this is not caused by a poor tuning of the 2-rate $(1+\lambda)$ $\mathrm{EA}_{r/2,2r}$, we consider the 3-rate $(1+\lambda)$ $\mathrm{EA}_{c_1 r, r, c_2 r}$ on $\lambda = 1{,}600$ and check 100 different configurations of (c_1, c_2) hyper-parameters for this version. The tuning procedure and its results are illustrated in Figure 7. First, some random values for c_1 and c_2 were taken. Then we investigated the areas around values which gave better performance. These results indicated that the best configuration are close to $(c_1 = 0.7, c_2 = 1.4)$. It is also worth mentioning that the performance of the 3-rate $(1 + \lambda)$ $\mathrm{EA}_{c_1 r, r, c_2 r}$ obtained with the different hyper-parameters differs by up to around 16%. The relative standard deviation of the tested algorithms is about 1%.

Note that we also plot in Figure 7 the line for which $c_1 = 1/c_2$, because such a functional dependence was used in [14]. According to the results of our procedure, the corresponding line does not fit the whole area of good configurations, albeit it does seem to cross some of the best configurations.

In Figure 8 we plot the performance of the 2-rate $(1+\lambda)$ $\mathrm{EA}_{c_1 r, c_2 r}$ and the 3-rate $(1 + \lambda)$ $\mathrm{EA}_{c_1 r, r, c_2 r}$ using the original hyper-parameters $(c_1 = 2, c_2 = 0.5)$ and using $(c_1 = 1.4, c_2 = 0.7)$. We compare these results to that of the $(1+\lambda)$ EA and the $(1+\lambda)$ $\mathrm{EA}(A, b)$ for the population size $\lambda = 1{,}600$. For the $(1 + \lambda)$ EA and the $(1 + \lambda)$ $\mathrm{EA}(A, b)$ the relative standard deviation is around 1%, while for the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{c_1 r, c_2 r}$ and the 3-rate $(1 + \lambda)$ $\mathrm{EA}_{c_1 r, r, c_2 r}$ it is around 2.5%.

Figure 8 shows that with the original hyper-parameters the 3-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2, r, 2r}$ performs worse than the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{r/2, 2r}$, while with $c_1 = 1.4, c_2 = 0.7$ they both perform better and very similar to each other. However, the algorithms are still outperformed by the $(1 + \lambda)$ $\mathrm{EA}(A, b)$. Therefore, the careful tuning of the hyper-parameters does not influence the performance of the 2-rate $(1 + \lambda)$ $\mathrm{EA}_{c_1 r, c_2 r}$ and the 3-rate $(1 + \lambda)$ $\mathrm{EA}_{c_1 r, r, c_2 r}$ drastically compared to the standard choice of $(c_1 = 0.5, c_2 = 2)$, which confirms the observations made in [14], where three values of $c2 = 2.0, 1.5, 1.2$ with a corresponding $c_1 = 1/c_2$ were tested.

7 CONCLUSIONS

We have seen in this work that the ranking of different self-adjusting $(1 + \lambda)$ EA variants strongly depends on the population size. Both small population sizes and large ones are of interest, as the former

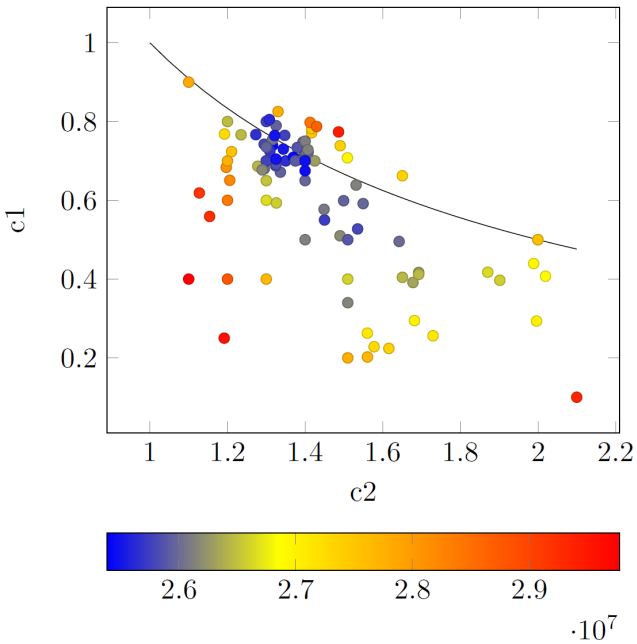


Figure 7: Average number of fitness evaluations (shown as color scale) for different hyper-parameters c_1, c_2 in 3-rate $(1+\lambda)$ EA$_{c_1 r, r, c_2 r}$ for $\lambda = 1{,}600$, $n = 100{,}000$.

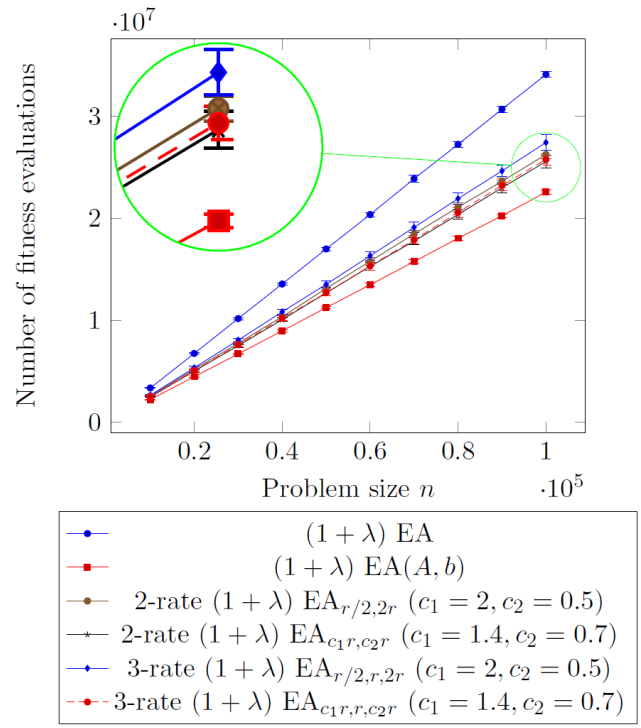


Figure 8: Average number of fitness evaluations and its standard deviation for the 2-rate $(1+\lambda)$ EA$_{r/2,2r}$ and the 3-rate $(1+\lambda)$ EA$_{c_1 r, r, c_2 r}$ with different multipliers compared to the $(1+\lambda)$ EA and the $(1+\lambda)$ EA(A, b) for $\lambda = 1{,}600$.

provide the best performance in terms of number of fitness evaluations, while the latter are more efficient in terms of generation number, which is useful when calculating fitness in parallel.

Based on our results, the following conclusions for the OneMax benchmark problem may be drawn: for small offspring population sizes $5 \le \lambda \le \lambda_1$, λ_1 satisfying $50 \le \lambda_1 < 100$ the 2-rate $(1+\lambda)$ EA$_{r/2,2r}(1/n^2)$ is the most efficient among the tested algorithms, while for medium population sizes $\lambda_1 \le \lambda \le \lambda_2$ with $400 \le \lambda_2 < 800$ the $(1+\lambda)$ EA$(A, b, 1/n^2)$ performs best, and for large offspring population sizes $\lambda_2 \le \lambda \le 3,200$ the $(1+\lambda)$ EA(A, b) with $p_{\min} = 1/n$ as the lower bound for the mutation probability should be used. Furthermore, if the evaluation budget is limited and the population size is small, one should pick the algorithms very carefully, as the ranking strongly depends on the budget in this case.

We also confirmed that although a careful selection of the hyper-parameters can improve the performance of the 2-rate $(1+\lambda)$ EA$_{r/2,2r}$ and the 3-rate $(1+\lambda)$ EA$_{r/2,r,2r}$, the default parameters seem to be quite suitably chosen, so that the overall gain of such hyper-parameter tuning seems to be moderate at best. We plan on extending our studies by investigating if similar conclusions can be drawn for the $(1+\lambda)$ EA(A, b) with different values of A and b. The results in [16] suggest a rather flat dependence of performance on these hyper-parameters for the $(1+1)$ EA, but note that we have considered in our work much larger dimensions and offspring population sizes, so that it is a priori not clear if a similar conclusion holds. We recall from [27] that small changes in the parameter setting of an EA can result in drastic performance differences.

As mentioned in the main part, the consistent ranking of the algorithms makes us feel confident about an extension of our work to a rigorous theoretical running time analysis. Such results would offer more detailed insights into the working principles underlying the diverse rankings, and may help to identify the cut-off points where performances of two algorithms intersect.

In another important line of research we plan on investigating whether our conclusions for the OneMax problem can be extended to other, more challenging benchmark problems. A promising direction of such extensions is offered by the W-model [32], which allows one to calibrate different features of the optimization problem, such as its ruggedness, the fraction of effective and "dummy" variables, the epistasis, neutrality, separability, etc.

ACKNOWLEDGMENTS

Arina Buzdalova was supported by the Government of Russian Federation (Grant 08-08). Carola Doerr was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, and by the Paris Ile-de-France Region. We also acknowledge support from COST Action CA15140 'Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO)' supported by the European Cooperation in Science and Technology.

REFERENCES

[1] Aldeida Aleti and Irene Moser. 2016. A Systematic Literature Review of Adaptive Parameter Control Methods for Evolutionary Algorithms. *Comput. Surveys* 49 (2016), 56:1–56:35.

[2] Anne Auger and Benjamin Doerr. 2011. *Theory of Randomized Search Heuristics*. World Scientific.

[3] Thomas Bäck. 1992. The Interaction of Mutation Rate, Selection, and Self-Adaptation Within a Genetic Algorithm. In *Proc. of Parallel Problem Solving from Nature (PPSN'92)*. Elsevier, 87–96.

[4] Thomas Bäck. 1993. Optimal Mutation Rates in Genetic Search. In *Proc. of the 5th International Conference on Genetic Algorithms (ICGA'93)*. Morgan Kaufmann, 2–8.

[5] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. 2014. Unbiased Black-Box Complexity of Parallel Search. In *Proc. of Parallel Problem Solving from Nature (PPSN'14) (Lecture Notes in Computer Science)*, Vol. 8672. Springer, 892–901.

[6] Edmund K. Burke, Michel Gendreau, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-Heuristics: a Survey of the State of the Art. *JORS* 64, 12 (2013), 1695–1724. https://doi.org/10.1057/jors.2013.71

[7] Eduardo Carvalho Pinto and Carola Doerr. 2018. A Simple Proof for the Usefulness of Crossover in Black-Box Optimization. In *Proc. of Parallel Problem Solving from Nature (PPSN'18) (Lecture Notes in Computer Science)*, Vol. 11102. Springer, 29–41. https://doi.org/10.1007/978-3-319-99259-4_3 Full version available at http://arxiv.org/abs/1812.00493.

[8] Eduardo Carvalho Pinto and Carola Doerr. 2018. Towards a More Practice-Aware Runtime Analysis of Evolutionary Algorithms. *CoRR* abs/1812.00493 (2018). arXiv:1812.00493 http://arxiv.org/abs/1812.00493 An extended abstract appeared in *Proc. of Artificial Evolution (EA'17)*, pages 298–305.

[9] 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.

[10] 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. Available online at https://arxiv.org/abs/1804.05650.

[11] Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From Black-Box Complexity to Designing New Genetic Algorithms. *Theoretical Computer Science* 567 (2015), 87–104.

[12] Benjamin Doerr, Carola Doerr, and Johannes Lengler. 2019. Self-Adjusting Mutation Rates with Provably Optimal Success Rules. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19)*. ACM. To appear. Full version available online at http://arxiv.org/abs/1902.02588.

[13] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. Optimal Parameter Choices via Precise Black-Box Analysis. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'16)*. ACM, 1123–1130.

[14] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. 2019. The $(1+\lambda)$ Evolutionary Algorithm with Self-Adjusting Mutation Rate. *Algorithmica* 81, 2 (2019), 593–631. https://doi.org/10.1007/s00453-018-0502-x

[15] Carola Doerr. 2018. Complexity Theory for Discrete Black-Box Optimization Heuristics. *CoRR* abs/1801.02037 (2018). arXiv:1801.02037 http://arxiv.org/abs/1801.02037

[16] Carola Doerr and Markus Wagner. 2018. 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)*. ACM, 943–950.

[17] Carola Doerr, Furong Ye, Sander van Rijn, Hao Wang, and Thomas Bäck. 2018. Towards a Theory-Guided Benchmarking Suite for Discrete Black-Box Optimization Heuristics: Profiling $(1+\lambda)$ EA Variants on OneMax and LeadingOnes. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*. ACM, 951–958.

[18] Agoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. 1999. Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation* 3 (1999), 124–141.

[19] Álvaro Fialho, Luís Da Costa, Marc Schoenauer, and Michèle Sebag. 2008. Extreme Value Based Adaptive Operator Selection. In *Proc. of Parallel Problem Solving from Nature (PPSN'08) (Lecture Notes in Computer Science)*, Vol. 5199. Springer, 175–184.

[20] 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. https://doi.org/10.1007/s10472-010-9213-y

[21] Thomas Jansen. 2013. *Analyzing Evolutionary Algorithms—The Computer Science Perspective*. Springer.

[22] 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.

[23] Thomas Jansen and Christine Zarges. 2014. Performance Analysis of Randomised Search Heuristics Operating with a Fixed Budget. *Theoretical Computer Science* 545 (2014), 39–58. https://doi.org/10.1016/j.tcs.2013.06.007

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

[25] 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.

[26] Jörg Lässig and Dirk Sudholt. 2011. Adaptive Population Models for Offspring Populations and Parallel Evolutionary Algorithms. In *Proc. of Foundations of Genetic Algorithms (FOGA'11)*. ACM, 181–192.

[27] Johannes Lengler. 2018. A General Dichotomy of Evolutionary Algorithms on Monotone Functions. In *Proc. of Parallel Problem Solving from Nature (PPSN'18) (Lecture Notes in Computer Science)*, Vol. 11102. Springer, 3–15. https://doi.org/10.1007/978-3-319-99259-4_1

[28] Jorge Maturana, Frédéric Lardeux, and Frédéric Saubion. 2010. Autonomous Operator Management for Evolutionary Algorithms. *Journal of Heuristics* 16 (2010), 881–909. https://doi.org/10.1007/s10732-010-9125-3

[29] Ingo Rechenberg. 1973. *Evolutionsstrategie*. Friedrich Fromman Verlag (Günther Holzboog KG), Stuttgart.

[30] Anna Rodionova, Kirill Antonov, Arina Buzdalova, and Carola Doerr. 2019. Offspring Population Size Matters when Comparing Evolutionary Algorithms with Self-Adjusting Mutation Rates. *arXiv e-prints:1904.08032* (2019). arXiv:1904.08032 https://arxiv.org/abs/1904.08032 Full version containing additional figures and tables.

[31] Dirk Thierens. 2009. On Benchmark Properties for Adaptive Operator Selection. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'09), Companion Material*. ACM, 2217–2218. https://doi.org/10.1145/1570256.1570306

[32] Thomas Weise and Zijun Wu. 2018. Difficult Features of Combinatorial Optimization Problems and the Tunable W-Model Benchmark Problem for Simulating them. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18), Companion Material)*. ACM, 1769–1776. https://doi.org/10.1145/3205651.3208240