Optimal Static Mutation Strength Distributions for the $(1 + \lambda)$ Evolutionary Algorithm on OneMax

Maxim Buzdalov, Carola Doerr

is the operator associated to the 1-point distribution assigning all probability mass to mutation strength 1.

Our Contribution: We want to analyze in this work how common mutation operators, such as the ones mentioned above or the fast mutation operator suggested in [14], compare to an optimal one. To analyze this question, we introduce the $(1 + \lambda)$ UUSD-EA, the family of all $(1+\lambda)$-type mutation-only algorithms whose mutation operator can be defined via a <u>u</u>nary <u>u</u>nbiased <u>s</u>tatic <u>d</u>istribution (which may hence depend on the problem dimension n and the offspring population size λ, but which may not change during the run). This family comprises all $(1 + \lambda)$ evolutionary algorithms (EAs), their Randomized Local Search (RLS) counterparts (which use the flip_1 operator instead of SBM), the fastGAs, the normalized EAs [28], etc.

We numerically compute for various combinations of n and λ the minimal expected runtime that can be obtained by any $(1+\lambda)$ UUSD-EA on ONEMAX in dimension n, and we compare these runtimes to that of the classically studies $(1+\lambda)$-type algorithms mentioned above. Our lower bounds are constructive, in that we also derive the distributions associated to these optimal $(1 + \lambda)$ UUSD-EAs. This allows us to study the properties of these optimal distributions.

Approach: Since the optimal distributions cannot be determined by exact analytical approaches, we apply the separable CMA-ES [27] to identify them. The CMA-ES shows very good performance, and provides us with distributions that are optimal up to the last few digits in the available machine precision. This is also one of the cases where the separable version of CMA-ES is not only computationally faster, but also yields results of same or better quality than the default CMA-ES implementations.

Main result: Our numerical results show, among other things, that the flip_1 operator is optimal when λ is small, whereas the conditional binomial distribution $\text{Bin}_{>0}(n, 1/2)$ is optimal when λ is very large – with this distribution, the $(1+\lambda)$ UUSD-EA performs a uniform random search. The optimal distributions in the middle regime show a rather complex behavior, for which we can identify a few patterns, but for which we also observe a few phenomena that may look counter-intuitive at first glance. For instance, the first mutation strength different from one-bit flip that gets nonzero probability when λ grows appears to be either n or $n - 1$.

Relationship to black-box complexity and to parameter control: Our work can be seen as a continuation of black-box complexity theory for λ-parallel [4] elitist [16] unary unbiased [24] black-box algorithms with static configuration. A key advantage of lower bounds such as ours is that it allows to rigorously quantify the impact of individual decisions. In our case, the driving motivation behind our analyses is a rigorous quantification of the difference between static and dynamic algorithm configurations. Put differently, we aim at quantifying the gap between algorithms using parameter

---

Optimal Static Mutation Strength Distributions for the $(1 + \lambda)$ Evolutionary Algorithm on OneMax

Maxim Buzdalov
ITMO University
Saint Petersburg, Russia

Carola Doerr
Sorbonne Université, CNRS, LIP6
Paris, France

ABSTRACT

Most evolutionary algorithms have parameters, which allow a great flexibility in controlling their behavior and adapting them to new problems. To achieve the best performance, it is often needed to control some of the parameters during optimization, which gave rise to various parameter control methods. In recent works, however, similar advantages have been shown, and even proven, for sampling parameter values from certain, often heavy-tailed, fixed distributions. This produced a family of algorithms currently known as "fast evolution strategies" and "fast genetic algorithms".

However, only little is known so far about the influence of these distributions on the performance of evolutionary algorithms, and about the relationships between (dynamic) parameter control and (static) parameter sampling. We contribute to the body of knowledge by presenting, for the first time, an algorithm that computes the optimal static distributions, which describe the mutation operator used in the well-known simple $(1 + \lambda)$ evolutionary algorithm on a classic benchmark problem ONEMAX. We show that, for large enough population sizes, such optimal distributions may be surprisingly complicated and counter-intuitive. We investigate certain properties of these distributions, and also evaluate the performance regrets of the $(1 + \lambda)$ evolutionary algorithm using commonly used mutation distributions.

1 INTRODUCTION

Many evolutionary algorithms are composed of operators that are used within in several algorithms. In the context of bit string representations, a very popular example of such an operator is *standard bit mutation (SBM)*, the variation operator that takes as input a point x (the "parent"), modifies it by changing the entry in each position with some positive probability p (independently of all other decisions), and outputs the so-obtained "offspring" y. SBM is a *global* search operator, since the probability that it generates a particular point y is positive, regardless of the input x. Another common mutation operator is the flip_1 operator, which creates the offspring by changing the entry in exactly one uniformly selected position. This search operator is a *local* one. Both mutation operators are *unbiased* in the sense proposed by Lehre and Witt [24], i.e., they treat all positions and all possible values identically.

By a characterization derived in [13], unary unbiased operators are exactly the ones that can be described by a distribution D over the integers $\{0, 1, \ldots, n\}$. To apply them, one first samples a *mutation strength* ℓ from this distribution and then changes the input by applying the flip_ℓ operator, which flips the entry in ℓ uniformly chosen, pairwise different positions. That SBM is unbiased now easily follows from the observation that it can be exactly characterized by the binomial distribution $\text{Bin}(n, p)$ with n trials (one for each position) and success probability p (success=bit flip). Likewise, flip_1

control and such that do not. In contrast to classical runtime and black-box complexity results, our work focuses on an exact numeric evaluation of this gap for concrete problem dimensions.

Impact: While the results of our concrete analysis may mostly appeal to theoreticians, our work invites to take a different view on mutation operators by defining them via distributions over the possible mutation strengths. This alternative view makes it substantially easier to generalize concepts such as SBM, flip_1, etc., an advantage that can be leveraged, for example, for designing smooth convergence from global to local search behavior. But the design principle can also lead to performance gains in the static setting. A first strong result in this context is the fastGA [14], which has become the new state-of-the-art in several applications [25]. Our work indicates that there is quite some untapped potential in the design of new mutation operators, and we hope that our work inspires new work in this direction.

Based on the findings of our work, we formulate to conjectures:
Conjecture 1: We conjecture that for each n there exists a threshold $\lambda_1(n)$ such that for all $\lambda \leq \lambda_1(n)$ the 1-point distribution is optimal. Our guess on the particular dependency is $\lambda_1(n) = \Theta(\log n)$. However, we currently do not have a formal proof for this.

Conjecture 2: We conjecture that for each n and each arbitrarily small ε there exists another threshold $\lambda_b(n, \varepsilon)$ such that, for all $\lambda \geq \lambda_b(n, \varepsilon)$, the optimal distribution is closer than ε, in any imaginable measure – such as the maximum of differences between probabilities across all mutation strengths – to the conditional binomial distribution $\text{Bin}_{>0}(n, 1/2)$. This is equivalent to stating that the uniform random search is arbitrarily close to being optimal for large enough λ. Again, we do not yet have a formal proof.

Related Work: Our study continues the recent works [6] and [7], which provide optimal dynamic configurations for $(1+1)$ and $(1+\lambda)$-type algorithms, respectively. While their works are restricted to specific mutation operators (variants of SMB and the k-bit flips flip_k), we study in this work in a generalization to arbitrary unary unbiased variation operators. In contrast to [6, 7] we focus on *static* configurations, with the idea to build a rigorous baseline against which we can compare dynamic parameter control methods.

For $\lambda = 1$, the work [13] quantifies the asymptotic advantage of the best unary unbiased algorithm with dynamic distributions against the best static one (RLS). To extend this work to $(1+\lambda)$-type algorithms, a rigorous bound on the best static case is needed - a baseline that we provide in this work for various combinations of n and λ, with the hope that the insights generated by our examples can be leveraged to rigorously prove certain characteristics of the optimal static unary unbiased operators.

For $\lambda > 1$, related works can be found in the context of the parallel black-box complexity model [4, 5, 23] and for the $(1 + \lambda)$ EA [18, 19]. All these works, however, are either less interested in exact runtime bounds (and focus on asymptotic runtime guarantees instead) or they concern specific mutation operators only. For generalized mutation operators, concrete examples can be found in the mentioned works [14, 28]. We are not aware, however, of previous works explicitly studying optimal unary mutation operators.

Availability of Code and Data: All project source code and data are available for public use at [1].

2 FROM MUTATION OPERATORS TO MUTATION STRENGTH DISTRIBUTIONS

We are concerned in this work with a generalizing view on unary unbiased variation operators, often referred to in the evolutionary computation context as *"mutation"*. In a nutshell, a mutation operator takes as input a search point $x \in \mathcal{S}$, $\mathcal{S}$ denoting the search space, and creates from it an offspring $y \in \mathcal{S}$. More formally, a mutation operator is a family $(D(x))_{x \in \mathcal{S}}$ of unary distributions over the search space $\mathcal{S}$. When fed with an input x, a new search point y is sampled from $D(x)$.

One of the most common mutation operators is standard bit mutation (SBM). In the context of pseudo-Boolean optimization (i.e., the maximization of a function $f : \{0, 1\}^n \to \mathbb{R}$) – which is the setting that we assume for the remainder of the paper – SBM is often explained as follows: to obtain an offspring y from x, we first create a copy of x and then we decide for each bit position $i \in [n] := \{1, \ldots, n\}$ whether the entry shall be updated to $1 - x_i$ ("bit flip") or whether the current entry is maintained. The bit flip decisions are made independently of each other. The probability $p \in (0, 1]$ to flip an entry is referred to as the *mutation rate*.

SBM is a prime example of a *unary unbiased mutation operator* in the sense proposed by Lehre and Witt in [24]. By a characterization proven in [13, Lemma 1], this class subsumes all variation operators that are fully described by a distribution D over the possible mutation strengths $\ell \in [0..n] := [n] \cup \{0\}$. When applied to a search point x, the operator first samples a mutation strength $\ell \in [0..n]$ from its operator-specific distribution D and then creates the offspring y by flipping the entries in ℓ pairwise different, uniformly selected entries.

It is not difficult to see that the operator-specific distribution of SBM is the binomial distribution $\text{Bin}(n, p)$ with n trials and success probability p. Another common operator is the 1-bit-flip operator 1pt, which is used, for example, within Randomized Local Search (RLS). 1pt creates the offspring by flipping exactly one uniformly chosen bit. Its operator-specific distribution over the mutation strengths $[0..n]$ is hence the 1-point distribution that assigns all probability mass to $\ell = 1$. Likewise, the $k$pt operator flips k pairwise different, uniformly selected bits, and its operator-specific distribution is the 1-point distribution giving all probability mass to $\ell = k$. Other common unary unbiased mutation operators include the "shift" SBM, $\text{SBM}_{0 \to 1}$, which is similar to SBM but which assign all probability weight from $\ell = 0$ to $\ell = 1$ and the "resampling" SBM, $\text{SBM}_{>0}$, which assigns the probability weight of sampling mutation strength 0 proportionally to all positive mutation strengths $1 \leq \ell \leq n$ by assigning to each of these values probability $\text{Bin}(n, p)(\ell)/(1-(1-p)^n)$; see [8, 9] for a discussion and motivation of these two latter variants. Motivated by the observation that infrequent large "jumps" can be beneficial in evolutionary algorithm behavior, B. Doerr et al. introduced in [14] the *fast genetic algorithm (GA)*, which samples the mutation strength from the heavy-tailed, power-law distribution $\mathcal{P}[\ell = k] = (C_{n/2}^\beta)^{-1} k^{-\beta}$ with $C_{n/2}^\beta = \sum_{i=1}^{n/2} i^{-\beta}$ and β being some constant, often set as $\beta = 1.5$. Finally, in [28] a normalized mutation operator was suggested, which samples the mutation strength from a normal distribution $\mathcal{N}(\mu, \sigma^2)$. In contrast to the examples discussed above, this operators allows

Algorithm 1: The $(1+\lambda)$ unary unbiased static distribution EA (UUSD-EA) maximizing a function $f : \{0, 1\}^n \rightarrow \mathbb{R}$.

---

1 **Initialization:** Sample $x \in \{0, 1\}^n$ uniformly at random and evaluate $f(x)$;

2 **Optimization: for** $t = 1, 2, 3, \ldots$ **do**

3   **for** $i = 1, \ldots, \lambda$ **do**

4   Sample $\ell(i) \sim D(n, \lambda)$;

5          $y^{(i)} \leftarrow \text{flip}_{\ell(i)}(x)$;

6   evaluate $f(y^{(i)})$;

7      $y \leftarrow \text{select}\left(\arg\max\{f(y^{(i)}) \mid i \in [\lambda]\}\right)$;

8   **if** $f(y) \geq f(x)$ **then** $x \leftarrow y$;

---

to scale the mean and the variance of the distribution independently of each other.

The characterization from [13, Lemma 1], which identifies mutation operators via their distributions over the set $[0..n]$, is classically only used to verify that a certain operator is unbiased. We use it here the other way around, by asking ourselves how different optimal mutation operators are from those commonly studied in the evolutionary computation community.

The $(1 + \lambda)$ UUSD-EA. We study this question in the context of the $(1 + \lambda)$ unary unbiased static distribution EA (UUSD-EA), which is given by Algorithm 1. The $(1 + \lambda)$ UUSD-EA is initialized uniformly at random. In each iteration, it samples λ points, which are all sampled from the same unary unbiased mutation operator. We denote the distribution from which the mutation strength is sampled by $D(n, \lambda)$ to indicate that it may depend on n and λ, but not on any information accumulated during the run of the algorithm. That is, the $(1 + \lambda)$ UUSD-EA allows only *static* mutation operators. For creating the λ search points that shall be evaluated in the current iteration, the $(1 + \lambda)$ UUSD-EA samples for each one of them a mutation strength $\ell(i)$ and then creates the i-th "offspring" by applying the $\text{flip}_{\ell(i)}$ operator, which flips $\ell(i)$ pairwise different, uniformly chosen bits in the input x. The best of these offspring replaces x if it is at least as good as it. It is irrelevant for the context of our work how the ties are broken in line 8, as our results apply to all tie-breaking rules.

ONEMAX: We focus on ONEMAX, i.e., our goal is to determine the optimal static unary unbiased distributions for maximizing the function $\sum_{i=1}^{n} x_i$. In the context of our work, this problem is equivalent to that of minimizing the Hamming distance to an arbitrary bit string $z \in \{0, 1\}^n$ [24].

Expected Runtimes: As common in literature on theory of evolutionary computation, we understand as an optimal distribution the one that minimizes the expected runtime, which we measure here in terms of generations. Since the offspring population size λ is fixed, this does not impact our results: a parallel runtime of T generations corresponds to a runtime of exactly $(T - 1)\lambda + 1$ function evaluations. As *runtime* we therefore denote the number of iterations that the algorithm performs until it evaluates an optimal solution for the first time.

(Non-)Uniqueness of the Optimal Distributions: We note that we do not have any guarantee at the moment that the optimal

distributions are unique. In fact, we observe that for any fixed problem dimension n, there is a certain threshold $\lambda(n)$ before which small differences in the distributions cause measurable effects on the expected running times, so that the optimal distribution seems to be unique from the point of view of our computations. After the threshold, however, differences between the distributions have no measurable effect on the expected runtime, so that our algorithm may consider different ones as optimal, and may hence not always return the same distribution.

Notation: For combinations of n and λ for which the optimal distributions are unique, we denote by $P^*(k \mid n, \lambda)$ the probability that this distribution assigns to flipping k bits, $k \in [n]$. To ease the reading, we sometimes use the same notation also for those distributions which yield expected running times that deviates only negligibly from what appears to be the true optimum.

3 ALGORITHM FOR COMPUTING THE OPTIMAL DISTRIBUTIONS

Our algorithm to compute the optimal static unary unbiased distributions is based on the dynamic programming approach from [7]. We start the description by explaining the principles of dynamic programming that are used to compute the expected running time T_f of the $(1+\lambda)$ UUSD-EA, measured in iterations, assuming it starts at fitness f and the values $T_{f'}$ are known for all $f' > f$. We note then that a (practical) analytic solution of the problem of finding an optimal distribution is quite unlikely to exist even for small values of n, and instead give a black-box minimization scheme with the use of a separable CMA-ES [27], a simplified and more computationally efficient version of the well-known continuous optimizer [20]. We complete with an investigation of convergence properties, which allows us to say, with great confidence, that separable CMA-ES finds a globally optimal distribution in a constant fraction of runs.

3.1 Dynamic Programming on Expected Times

We first explain how to compute T_f for a given distribution $D = (D_k)_{k \in [0..n]}$. We begin with computing the probabilities $S_{k,g}$ of sampling an offspring with fitness g by flipping exactly k bits chosen uniformly without replacement in a solution of fitness f. Note that these quantities depend only on the current fitness f and the problem properties, that is, they depend neither on λ nor on D. It holds from simple combinatorics that $S_{k,g} = \binom{n-f}{g-f+i}\binom{f}{i}/\binom{n}{k}$, where we assume this probability to be zero if one of the binomial coefficient arguments are out of bounds.

Next we compute the probabilities $(Q_g^{(1)})_{g=f+1,f+2,\ldots,n}$ of sampling a *single* offspring of fitness g. For $g > f$ they are derived from $S_{k,g}$ by using the distribution parameters $(D_k)_{k \in [0..n]}$ as follows:

$$Q_g^{(1)} = \sum_{k=1}^{n} D_k S_{k,g}.$$

As the $(1 + \lambda)$ UUSD-EA is an elitist algorithm, and the behavior of the algorithm with different parents having the same fitness value is the same, with the remaining probability the algorithm remains in the same state, which we capture as $Q_f^{(1)} = 1 - \sum_{g=f+1,f+2,\ldots,n} Q_g^{(1)}$.

The probability of each possible fitness improvement after sampling all the λ offspring is then computed as follows:

$$Q_g^{(\lambda)} = \left(\sum_{i=f}^{g} Q_i^{(1)}\right)^{\lambda} - \left(\sum_{i=f}^{g-1} Q_i^{(1)}\right)^{\lambda}. \tag{1}$$

In simple words, the new fitness is g if all offspring have the fitness in $[f..g]$, but not all of them have the fitness in $[f..g-1]$, counting all offspring with fitness smaller than f towards f.

Finally, the expected time to reach the optimum from the fitness f is computed using the following expression:

$$T_f = \frac{1}{1 - Q_f^{(\lambda)}} \left(1 + \sum_{g=f+1}^{n} Q_g^{(\lambda)} T_g\right), \tag{2}$$

by standard arguments as detailed in [6, 7].

The time and memory complexities of such a step is $O(n^2)$ for computing $S_{k,g}$ and $O(n)$ for the other stages. As there are n different fitness values for f, the time complexity of computing all the expected running times for one set of algorithm parameters (λ, D) is $O(n^3)$, whereas the memory complexity is still $O(n^2)$ as the $S_{k,g}$ matrix may be discarded once f changes. However, since $S_{k,g}$ depend only on n and f, one may evaluate up to $O(n)$ different combinations of algorithm parameters (λ, D) in a single run by merging the activities corresponding to each f, which preserves the total time complexity of $O(n^3)$ and the memory complexity of $O(n^2)$. This feature will turn later to be beneficial when a population-based optimizer is applied to find the best possible distribution D.

3.2 Optimization with Separable CMA-ES

When solving the problem of finding an optimal distribution D, one could choose to express each of T_f as a function of $n + 1$ distribution parameters $(D_k)_{k \in [0..n]}$, promote such expressions through dynamic programming, take their weighed sum for the expected running time from a random point and perform analytical optimization using standard analysis approaches. However, the presence of $Q_f^{(\lambda)}$ in the denominator in (2) makes the resulting expression nonlinear even for $\lambda = 1$, and having a λ in the exponent in (1) makes it even harder. The resulting expression appears to be a ratio of polynomials of degree $\Theta(n\lambda)$ with D_k as variables, which makes it infeasible to perform an exact analytical minimization even for small problem sizes.

Such large degree of the polynomials also effectively prevents gradient-based optimization, since the exact numeric computation of the derivatives – although possible – would require considerable computational resources, which furthermore significantly increase with λ. At the same time, evaluating the expected runtime of the $(1 + \lambda)$ UUSD-EA for a given distribution does not depend on λ, assuming the computations are done in machine precision. This makes it possible to apply black-box optimization techniques to identify distributions which minimize the expected runtime. From the large set of possible black-box optimization techniques, we chose the CMA-ES family of algorithms [20] that are well suitable for such kind of optimization. The application of CMA-ES requires a few clarifications, which we list below.

- Our implementation of CMA-ES is based on the one from Apache Commons Math, version 3.6.1. The choice of the Java
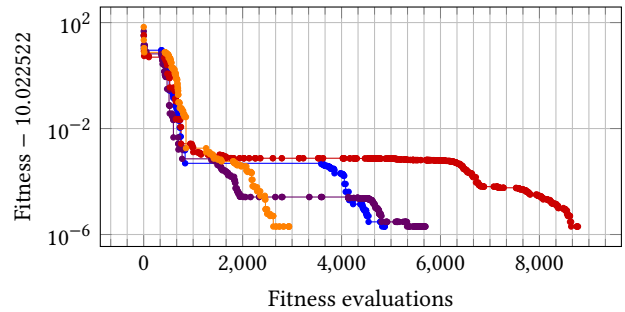

Figure 1: Fitness as a function of the number of evaluations for $n = 16$ and $\lambda = 8$. First four runs are shown.

programming platform was due to the computational complexity of the fitness evaluation. On one hand, it is too expensive to be implemented in Python or Matlab (the languages with reference implementations of CMA-ES maintained by the authors of this algorithm). On the other hand, the costs of inter-process communication are not negligible compared to fitness evaluation, so evaluation of fitness shall happen in the same process as the optimizer.
- To search the distributions, we tune the CMA-ES to respect box constraints (each variable is in $[0, 1]$) and, before fitness evaluation, we normalize variable values in the Lamarckian sense, that is, without updating the individuals themselves, which would clash with the assumptions made by CMA-ES.
- As we perform distribution optimization for a noiseless problem, we can safely set D_0 to zero, which restricts the search to the n-dimensional cube $[0, 1]^n$.

We had to modify the implementation of CMA-ES, as the particular class hierarchy in Apache Commons Math does not allow to evaluate the whole population in a single call, which we benefit from. Besides, after some experimentation, we switched to the separable CMA-ES [27], which is more efficient in terms of computational costs, but also produced better results (likely due to less operations that caused precision loss). We also vastly optimized its implementation, which resulted in an 8× reduction of the wall-clock running time.

The configurable parameters of CMA-ES were: population size 10, initial step size 1.0, random initial guess, computational budget $100n^2$. The optimizer, however, did not reach the computational budget, as, in all runs, it converged to a single point and terminates at one of the degeneration criteria much earlier than that.

3.3 Convergence Analysis

The preliminary experiments showed that CMA-ES typically converges relatively quickly to nearly the same value in most of runs. Fig. 1 shows example runs for $n = 16$ and $\lambda = 8$. Runs for all n and λ demonstrate similar behavior. The rest of the paper is based on the data collected for $n \in \{3, 5, 8, 11, 16, 23, 32, 45, 64, 91, 100\}$ and $\lambda \in [1..7] \cup \{2^i \mid 3 \le i \le 10\}$. For each (n, λ) pair we performed 50 independent runs of the CMA-ES.

We observed that in a constant fraction of runs the algorithm optimized the distribution to produce the same expected running
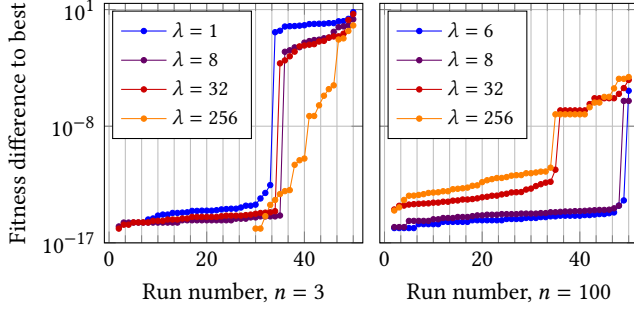
Figure 2: **Relative loss of the expected runtime induced by the result of the 50 CMA-ES runs, when compared against the best expected runtime. Plots are for $n = 3$ (left) and $n = 100$ (right), respectively, values are sorted by increasing loss.**
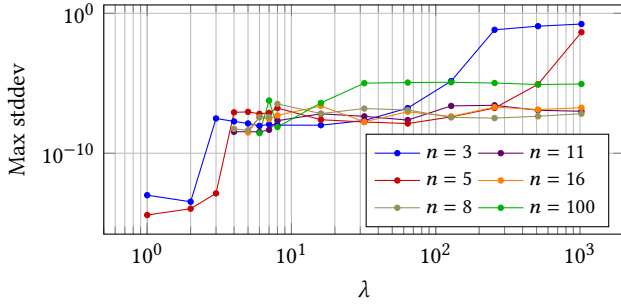

Figure 3: **Maximal standard deviations, out of k standard deviations of $P^*(k \mid n, \lambda)$, as functions of λ for various n.**

Figure 4: **Average wall-clock times for various λ together with guesses for asymptotic bounds.**

Figure 5: **Probabilities of flipping k bits in optimal static mutation strength distributions for different population sizes λ for optimizing OneMax in dimension $n = 16$.**

time up to precision of 10^{-12} and better. Fig. 2 shows examples for some values of λ for $n = 3$ and $n = 100$. Note that there is no curve for $\lambda = 1$ in the $n = 100$ plot since in this case all the values were identical. Since the delivered precision is very close to the precision available for the 64-bit floating point machine numbers, we assume that CMA-ES reaches the global optimum of the problem in most of the cases. For further analyses we selected only the runs that are at most $1 + 10^{-9}$ times greater than the obtained minimum.

We have also performed the robustness analysis for the distributions produced by the optimizer. For each n, λ, and k, the standard deviation of the values $P^*(k \mid n, \lambda)$ was computed across all the good enough distributions. The results are presented in Fig. 3, where an intriguing picture appears. First, small λ produce very small (much less than 10^{-10}) maximal standard deviations, which then jump to the region of $[10^{-8}; 10^{-5}]$ and remain there until λ reaches a certain threshold. Above that threshold, the maximal standard deviations experience some sort of phase shift and raise to very high values reaching 0.1 and above. Note that, by our selection of the data that is considered in this computation, the $(1 + \lambda)$ UUSD-EA still shows nearly identical expected running times on such different distributions and values of λ. We show later in the next section that this is, in fact, an expected behavior that corresponds to situations when there is a single global optimum, but a number
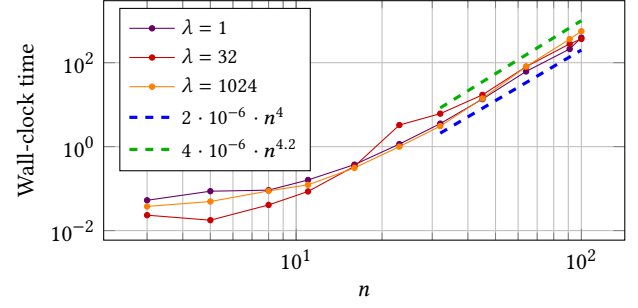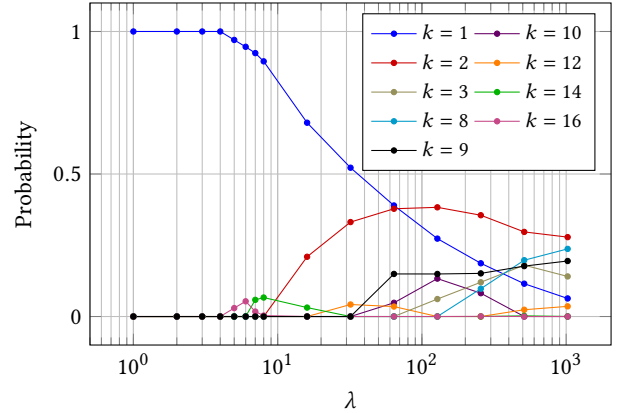
of different distributions coincide in its running time expectation with that global optimum up to the machine precision.

Fig. 4 displays the average wall-clock times required to find the optimal distribution for all available n and few values of λ. The available data suggests that the time complexity scales polynomially with the degree of $4 + \varepsilon$ for some small ε, which, together with the earlier cubic runtime bound for the evaluation of a given distribution, suggests that the CMA-ES requires $O(n^{1+\varepsilon})$ iterations before hitting one of its termination criteria.

4 OPTIMAL DISTRIBUTIONS

We now take a closer look at the distributions returned by the algorithm from Sec. 3. As mentioned, the data is available at [1], and we present here only our main findings.

We first study the impact of λ on the optimal distributions. To this end, we fix the dimension and analyze how for each possible mutation strength $k \in [1..n]$ its probability of being chosen depends on λ. Fig. 5 illustrates these probabilities for dimension $n = 16$.

The 1-point distribution $\Pr[\ell = 1] = 1$ is optimal when λ is small. For $n = 16$, we see that deterministically flipping one bit ($\Pr[\ell = 1] = 1$) is optimal for $\lambda \leq 4$. Note that for $\lambda = 1$ this distribution defines Randomized Local Search (RLS), an algorithm
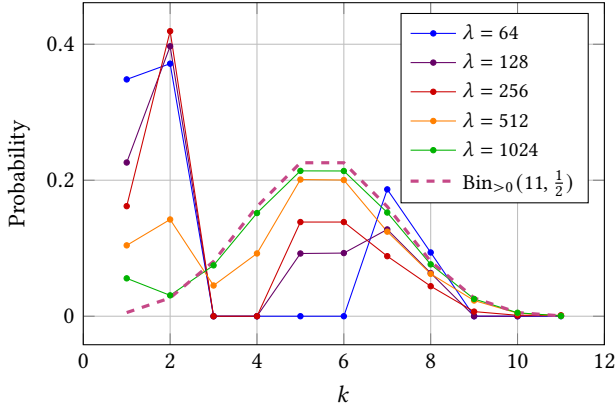
Figure 6: Values $P^*(k \mid n = 11, \lambda)$ for large λ plotted against k. The larger λ, the more the curves resemble the binomial distribution $\text{Bin}_{>0}(11, 1/2)$, which is also plotted.

that is often used as baseline for comparisons, both in empirical [17] and in theoretical [15] research. Our data shows that for $n \in \{8, 11\}$ the generalized $(1 + \lambda)$ RLS is optimal for $\lambda \le 3$ and suboptimal for $\lambda = 4$. Similarly, for $n \in \{16, 23, 32\}$ it is optimal for $\lambda \le 4$, and for $n \in \{64, 91, 100\}$ the threshold is $\lambda = 5$. We are confident that this describes a general trend of a positive correlation between the dimension n and the maximal λ for which the 1-point distribution $\Pr[\ell = 1] = 1$ is optimal.

The conditional binomial distribution $\text{Bin}_{>0}(n, 1/2)$ is arbitrarily close to an optimal one for large λ. We plot in Fig. 6 the optimal probability $P^*(k \mid n = 11, \lambda)$ for large λ. The curves for $\lambda = 512$ and $\lambda = 1\,024$ approximate the conditional binomial distribution $\text{Bin}_{>0}(11, 1/2)$, which we plot as dotted red line. This can be explained as follows: the $(1 + \lambda)$ UUSD-EA with the static mutation strength distribution $\text{Bin}_{>0}(n, 1/2)$ is simply the random search algorithm, which samples all search points, except the parent, uniformly at random. For $\lambda = \Omega(2^n)$, this algorithm has a very good chance of sampling every point $y \in \{0, 1\}^n$, so that it also has a decent chance of hitting the optimum. When λ is not much larger than 2^n, the quality of a distribution is significantly influenced by each of its parameters, so for these cases our computations provide distributions that are almost identical to random search in each independent run. When λ is much bigger than 2^n, however, the situation changes: while the common sense suggests that the truly optimal distribution gets even closer to pure random search, the quality of a distribution quickly loses sensitivity to its parameters, and we obtain different distributions, which all yield practically indistinguishable expected runtimes. We provide an example for $n = 3$ in Tab. 1, where we observe that for $\lambda \le 128$ the average of the computed static mutation strength distributions converge against $\text{Bin}_{>0}(n, 1/2)$. The standard deviation of the independent runs of our optimizer are negligible in this regime. For $\lambda = 256, 512$, and $1\,024$, however, the maximum standard deviation among the three mutation strengths are 0.07, 0.12, and 0.17, respectively. Fig. 7 plots the results of all 50 independent runs for $\lambda = 1\,024$.

$P^*(1 \mid \lambda, n)$ decreases monotonically with increasing λ. For fixed dimension n, the importance of 1-bit flips significantly

Table 1: Average values of the recommended distributions $P^*(k \mid n = 3, \lambda)$ for different λ and all mutation strengths k. For $\lambda \le 128$ the standard deviations are negligible, then grow quickly (see Figure 3). Unreliable values are indicated with gray font. We add $\text{Bin}_{>0}(3, 1/2)$ for comparison.

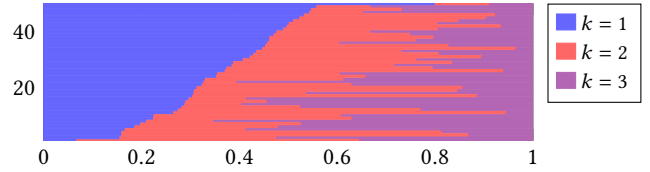| k	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	$\text{Bin}_{>0}(3, \frac{1}{2})$
1	0.558	0.448	0.42937	0.42857	0.42857	0.413	0.371	0.364	0.42857
2	0.331	0.414	0.42797	0.42857	0.42857	0.408	0.364	0.342	0.42857
3	0.110	0.138	0.14266	0.14286	0.14286	0.179	0.265	0.294	0.14286

Figure 7: $P^*(k \mid n = 3, \lambda = 1\,024)$ for each of the 50 independent runs of our optimizer. The expected optimization time is identical for all of them (0.875).
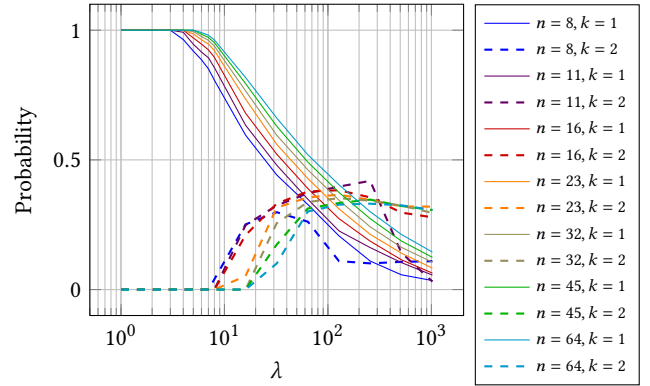

Figure 8: $P^*(1 \mid \lambda, n)$ and $P^*(2 \mid \lambda, n)$ in dependence of λ, for selected problem dimensions n.

decreases as λ grows. We have seen this for $n = 16$ in Fig. 5. This trend generalizes to other problem dimensions, which can be seen in Fig. 8, where we plot the probabilities $P^*(1 \mid \lambda, n)$ and $P^*(2 \mid \lambda, n)$ for $n \in \{8, 11, 16, 23, 32, 45, 64\}$.

$P^*(2 \mid \lambda, n)$ is non-monotonic in λ. We clearly see from Fig. 8 that, for fixed dimension n, the optimal probability to flip two bits is non-monotonic in λ. The λ at which it becomes non-zero appears to be monotonic in n, however, the set of λ we used is not enough to determine the exact threshold. It is $\lambda = 8$ for $n = 8$, $8 < \lambda \le 16$ for $n \in \{11, 16, 23\}$, and it is $16 < \lambda \le 32$ for $n \in \{32, 45, 64, 91, 100\}$. Surprisingly enough, this threshold is always larger than the value at which $P^*(1 \mid \lambda, n)$ becomes less than one. We do not see any pattern in the points at which $\Pr[k = 2]$ starts to decrease again.

Flipping all bits can be optimal. Intuitively, flipping more than $n/2$ bits can be optimal in an elitist algorithm only when

Table 2: Combinations of n and λ for which $P^*(n \mid n, \lambda) > 0$, i.e., the optimal probability of flipping all bits is non-zero.

n	3	4	5	6	7	8	16	32	64	128	256	512	1024
8		+	+	+	+	+			+	+	+	+	+
11						+				+	+	+	+
16			+	+	+	+					+		
23													
32			+	+	+	+							+
45													
64				+	+	+							
91								+					
100				+	+		+						

Table 3: Number of different mutation strengths k for which $P^*(k \mid \lambda, n) > 0$, in dependence of n and λ.

n	3	4	5	6	7	8	16	32	64	128	256	512	1024
8	1	2	2	3	3	5	4	5	7	8	8	8	8
11	1	2	2	2	2	4	5	5	4	7	9	11	11
16	1	1	2	3	3	4	6	4	5	5	9	9	10
23	1	1	2	2	2	4	5	6	6	6	7	9	11
32	1	1	2	2	3	4	5	7	6	5	7	7	9
45	1	1	1	2	2	3	5	7	6	7	8	8	8
64	1	1	1	2	3	4	4	7	6	8	8	7	8
91	1	1	1	2	2	3	6	6	6	7	7	9	9
100	1	1	1	2	3	2	6	6	6	7	8	9	9

Table 4: Values of $P^*(k \mid n = 11, \lambda)$ for selected λ.

k	4	5	6	7	8	16	32	64	$\mathrm{Bin}_{>0}(11, \frac{1}{2})$
1	0.9919	0.9459	0.9156	0.8937	0.8555	0.6337	0.4797	0.3483	0.0054
2						0.2483	0.3241	0.3714	0.0269
3									0.0806
4				0.0530					0.1612
5									0.2257
6									0.2257
7						0.1068	0.1865		0.1612
8						0.0162	0.0677	0.0938	0.0806
9						0.0748	0.0217		0.0269
10	0.0081	0.0541	0.0844	0.1063	0.0875	0.0270			0.0054
11					0.0040				0.0005

Figure 9: $P^*(k \mid n = 64, \lambda)$, in dependence of λ, and for all values $k \in [50, 64]$ for which there is at least one λ such that $P^*(k \mid n, \lambda) > 0$. Note that, for all these k, $P^*(k \mid n, \lambda) = 0$ for $\lambda \in \{128, 256, 512, 1024\}$.

$\mathrm{OM}(x) < n/2$. It may therefore be surprising that even for mutation strength $\ell = n$ (i.e., flipping all bits) the optimal static probability can be non-zero already for comparatively small λ. In Tab. 2 we show for which combinations of n and λ the optimal probability of flipping all bits is non-zero. Note that for some n we have not seen any λ where this probability is nonzero, and this may be related to the parity of n: for $n \in \{23, 45, 91\}$ the small λ feature rather a nonzero $P^*(n - 1 \mid n, \lambda)$ instead. So far we do not have any explanation for the observed patterns.

We take a set $\Lambda = \{6, 7, 8, 16, 32, 64\}$ and show an example in Fig. 9, where we display $P^*(k \mid n = 64, \lambda \in \Lambda)$ for all $k \in [50, 64]$ for which there exists at least one $\lambda \in \Lambda$ with $P^*(k \mid n = 64, \lambda) > 0$. Interestingly, $P^*(k \mid n = 64, \lambda) = 0$ for all tested $\lambda > 64$. We also see that only nine different k appear out of $[50, 64]$, of which at most three have a non-zero optimal probability for any of the tested λ.

The number of mutation strengths k with $P^*(k \mid \lambda, n) > 0$ increases with λ, but not monotonically. We summarize in Tab. 3 the number of different mutation strengths k for which $P^*(k \mid \lambda, n) > 0$. While there is a general trend for an increasing number of such k with increasing λ, these trends are non-monotonic. From the previous insights, however, it is clear that for every n there exists a threshold $\lambda(n)$ such that for all $\lambda > \lambda(n)$ the number of k with $P^*(k \mid \lambda, n) > 0$ is equal to n.

A closer look for small λ. It is well known that small values of λ are preferable for the optimization of simple function such as OneMax [21]. We therefore take a more detailed look at small values in Tab. 4, where we list for $n = 11$, $\lambda \in [4..8] \cup \{16, 32, 64\}$, and for all possible mutation strengths $k \in [11]$ the optimal probability $P^*(k \mid n, \lambda)$. We recall from above that for $\lambda \leq 3$ the one-point distribution assigning all weight to $k = 1$ is optimal. For $\lambda = 4$ and $\lambda = 7$, the optimal probability $P^*(1 \mid n = 11, \lambda)$ is slightly less than 1, and the remaining probability mass is assigned to $k = 10$, i.e., to the operator flipping all bits but one. For $\lambda = 8$ two more values, $k = 4$ and $k = 11$, are also chosen with positive probability. As already discussed in the context of Tab. 3 and Fig. 6, the number of mutation strengths k with $P^*(k \mid n = 11, \lambda) > 0$ increases for increasing λ, and the distribution converges to the conditional binomial distribution $\mathrm{Bin}_{>0}(11, 1/2)$, which we include in the table for reference.

5 RUNTIME COMPARISON

After having focused on the distributions in the previous section, we now study the runtime of the optimal $(1 + \lambda)$ UUSD-EA in comparison to other common $(1 + \lambda)$ UUSD-EAs. We include in our comparison the $(1 + \lambda)$ EA variants with SBM, $\mathrm{SBM}_{>0}$, and $\mathrm{SBM}_{0 \to 1}$ standard bit mutation operators, the $(1 + \lambda)$ RLS, and the $(1 + \lambda)$ fastGA with different $\beta \in \{1.3, 1.5, 1.7\}$. We also considered the variant of the latter which directly samples mutation strengths ℓ proportional to $\ell^{-\beta}$ for the same parameter values.

It is not difficult to see that, for any fixed n, the expected runtime of the optimal $(1 + \lambda)$ UUSD-EAs converge to $1 - 1/2^n$ as $\lambda \to \infty$. This is also the case for all $(1 + \lambda)$ UUSD-EA variants that assign a

positive probability to each positive mutation strength $k \in [n]$, and this for all problems $f : \{0, 1\}^n \to \mathbb{R}$. Since SBM and fast mutation satisfy these requirements, the expected runtime of all $(1 + \lambda)$ EA variants as well as that of the fastGA variants also converges to $1 - 1/2^n$, but at a possibly much different speed. The expected runtime of the generalized $(1 + \lambda)$ RLS, in contrast, converges to $n/2$ on ONEMAX, since, hand-waivingly, this is the expected distance to the optimum after initialization and the probability to make a progress of 1 in each iteration converged to 1 as $\lambda \to \mathbb{R}$.

In Table 5 we present the expected runtimes of the optimal $(1 + \lambda)$ UUSD-EA(s) on ONEMAX for all different combinations of n and λ we have. Note that these numbers are the lower bounds for all $(1 + \lambda)$ UUSD-EAs, including the algorithms mentioned above. Note also that algorithms obtaining a better expected runtime require adaptive parameter choices.

In Figures 10 we illustrate the regret in the expected runtime of common $(1 + \lambda)$ UUSD-EAs compared to the optimal one, for $n \in \{3, 100\}$, respectively. Corresponding to our discussion above, we observe that for $n = 3$ all algorithms, with the exception of RLS, converge to the same optimal expected runtime of $0.875 = 1 - 1/8$ when $\lambda \to \infty$, whereas the generalized $(1 + \lambda)$ RLS converges to 1.5, which corresponds to a factor $12/7 \approx 1.71...$ relative disadvantage.

Note that the regrets are not monotone for those algorithms which never flip zero bits, except for the $(1 + \lambda$ RLS. As λ grows from the small values, their regret decreases, most probably as flipping more than one bit becomes a better choice. However, with further increase of λ, the fact that these algorithms flip many bits with a small probability turns to a disadvantage. Note how the heavy-tailed algorithm that samples the mutation strength directly from a power-law distribution with the smallest tested $\beta = 1.3$ becomes a clear winner at $\lambda \geq 7$. The similar behavior is seen also for $n = 100$ with the exception that the values of λ are not large enough to observer the complete convergence picture.

6 CONCLUSION

We have analyzed in this paper the dependence of $P^*(k \mid n, \lambda)$, the optimal probability of flipping k bits in the $(1 + \lambda)$ EA-UUSD, in dependence of n and λ. Among other insights, we have shown that the $(1 + \lambda)$ RLS is optimal when λ is small, and that the value for which it ceases to be optimal increases with increasing n. We have also seen that, for fixed n, the distribution $P^*(k \mid n, \lambda)$ converges to the conditional binomial distribution $\text{Bin}_{>0}(n, 1/2)$ when $\lambda \to \infty$.

For future work, we consider the following particularly exciting.

1) Formalizing the observations into rigorous results: We believe that some of the observations made in this work could be formalized with reasonable effort.

2) Analyzing benefits of generalized mutation for more complex problems: For practitioners, our work is perhaps most interesting in that it invites to consider mutation operators through the lens of probability distributions over the set of possible radii. This idea should show its full potential on problems that are more complex than ONEMAX. The fastGA proposed in [14] and fast AIS from [10] are compelling examples that shows that such generalization can indeed be very beneficial [25, 26].

3) Transferring the generalizations to variation operators of higher arity: The quest for analyzing more general variation
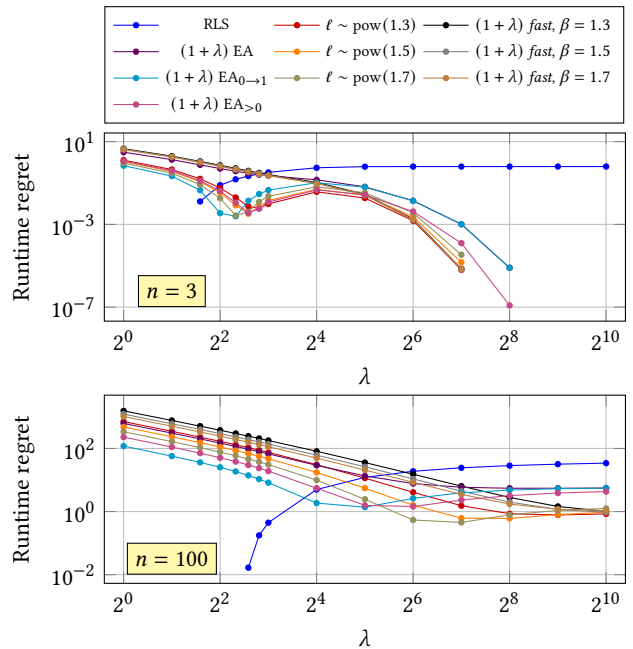

Figure 10: Regrets of expected runtime of different $(1 + \lambda)$ **UUSD-EAs on** ONEMAX, **in dependency of** λ, **compared to the runtime of the optimal** $(1 + \lambda)$ **UUSD-EA computed in Sec. 4, for** $n = 3$ **(top) and** $n = 100$ **(bottom). The algorithms that sample the mutation strength** ℓ **directly from the power-law distribution with parameter** β **are denoted as** $\ell \sim \text{pow}(\beta)$. **The notation for other algorithms is standard.**

operators is not restricted to mutation alone, but also generalizes to variation operators of higher arity, called "crossover" or "recombination" operators in evolutionary computation. In a simplest extension, one could study effects of changing the binomial distribution associated with the number of bits that is taken from each parent in uniform crossover. We did not yet investigate this idea further, but we hope that a de-coupling of mean and variance, similarly as proposed for variation in [28], may be beneficial.

4) Interplay of generalized mutation with other variation operators: The benefits of generalized mutation operators are very likely not restricted to mutation-only algorithms, but could also improve algorithms that use variation operators of different arities. First examples demonstrating clear advantages of heavy-tailed mutation in the $(1 + (\lambda, \lambda))$ GA [12] were recently shown in [2, 3].

5) Extensions to the dynamic case: We studied in this work the case of *static* distributions $P^*(k \mid n, \lambda)$, whereas it is well known that a dynamic choice of the mutation rates, or algorithms' parameters in general, can lead to significant performance gains [11, 22]. Combining the analyses made in [7] for the optimal dynamic 1pt and the optimal dynamic SBM operators with the approach taken in this work (the generalization to arbitrary distributions) would provide an exact quantification of the disadvantage of static against dynamic mutation operator choices.

Table 5: Expected runtimes (in generations) of the optimal $(1 + \lambda)$ UUSD-EA(s) on OneMax for different combinations of n (rows) and λ (columns), rounded to two digits. For every fixed n, the optimal expected runtime converges to $1 - 1/2^n$ as $\lambda \to \infty$.

n	λ														
	1	2	3	4	5	6	7	8	16	32	64	128	256	512	1024
3	3.50	2.26	1.87	1.64	1.48	1.37	1.28	1.21	0.96	0.88	0.88	0.88	0.87	0.87	0.87
5	7.97	4.78	3.78	3.26	2.92	2.67	2.49	2.35	1.78	1.39	1.10	0.98	0.97	0.97	0.97
8	16.20	9.32	7.12	6.04	5.36	4.89	4.54	4.27	3.15	2.43	1.98	1.67	1.39	1.14	1.01
11	25.59	14.45	10.84	9.11	8.04	7.30	6.76	6.34	4.63	3.53	2.82	2.33	2.01	1.83	1.62
16	43.00	23.87	17.64	14.62	12.83	11.60	10.71	10.02	7.26	5.46	4.31	3.54	3.01	2.61	2.26
23	69.95	38.35	28.02	22.99	20.04	18.04	16.60	15.50	11.14	8.32	6.51	5.30	4.46	3.83	3.34
32	107.69	58.52	42.40	34.52	29.91	26.84	24.62	22.94	16.34	12.14	9.42	7.62	6.38	5.47	4.79
45	166.58	89.83	64.63	52.27	45.03	40.25	36.82	34.23	24.17	17.84	13.74	11.05	9.21	7.87	6.86
64	259.25	138.90	99.31	79.86	68.44	60.95	55.59	51.56	36.09	26.45	20.23	16.17	13.39	11.41	9.92
91	400.44	213.38	151.76	121.44	103.60	91.93	83.62	77.39	53.70	39.07	29.70	23.59	19.45	16.50	14.31
100	449.42	239.17	169.88	135.79	115.70	102.57	93.24	86.24	59.70	43.36	32.90	26.09	21.48	18.22	15.79

REFERENCES

[1] Authors Anonymized. 2021. Code and data repository for this paper. Blinded for GECCO reviews.

[2] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. 2020. Fast mutation in crossover-based algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'20)*. ACM, 1268–1276. https://doi.org/10.1145/3377930.3390172

[3] Denis Antipov and Benjamin Doerr. 2020. Runtime Analysis of a Heavy-Tailed $(1 + (\lambda, \lambda))$ Genetic Algorithm on Jump Functions. In *Proc. of Parallel Problem Solving from Nature (PPSN'20) (LNCS)*, Vol. 12270. Springer, 545–559. https://doi.org/10.1007/978-3-030-58115-2_38

[4] 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) (LNCS)*, Vol. 8672. Springer, 892–901.

[5] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. 2015. Black-box Complexity of Parallel Search with Distributed Populations. In *Proc. of Foundations of Genetic Algorithms (FOGA'15)*. ACM, 3–15.

[6] Nathan Buskulic and Carola Doerr. 2019. Maximizing drift is not optimal for solving OneMax. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'19, Companion Material)*. ACM, 425–426. https://doi.org/10.1145/3319619.3321952 An extension of this work is to appear in the Evolutionary Computation journal.

[7] Maxim Buzdalov and Carola Doerr. 2020. Optimal Mutation Rates for the $(1 + \lambda)$ EA on OneMax. In *Proc. of Parallel Problem Solving from Nature (PPSN'20) (LNCS)*, Vol. 12270. Springer, 574–587. https://doi.org/10.1007/978-3-030-58115-2_40

[8] 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) (LNCS)*, Vol. 11102. Springer, 29–41. https://doi.org/10.1007/978-3-319-99259-4_3

[9] 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

[10] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. 2018. Fast Artificial Immune Systems. In *Parallel Problem Solving from Nature*. Lecture Notes in Computer Science, Vol. 11102. 67–78.

[11] Benjamin Doerr and Carola Doerr. 2020. 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*. Springer, 271–321.

[12] Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science* 567 (2015), 87 – 104.

[13] Benjamin Doerr, Carola Doerr, and Jing Yang. 2020. Optimal parameter choices via precise black-box analysis. *Theoretical Computer Science* 801 (2020), 1–34. https://doi.org/10.1016/j.tcs.2019.06.014

[14] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. 2017. Fast genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*. ACM, 777–784. https://doi.org/10.1145/3071178.3071301

[15] Benjamin Doerr and Frank Neumann. 2020. *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Springer. https://doi.org/10.1007/978-3-030-29414-4

[16] Carola Doerr and Johannes Lengler. 2017. Introducing Elitist Black-Box Models: When Does Elitist Behavior Weaken the Performance of Evolutionary Algorithms? *Evolutionary Computation* 25 (2017). https://doi.org/10.1162/evco_a_00195

[17] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer M. Shir, and Thomas Bäck. 2020. Benchmarking discrete optimization heuristics with IOHprofiler. *Applied Soft Computing* 88 (2020), 106027. https://doi.org/10.1016/j.asoc.2019.106027

[18] Christian Gießen and Carsten Witt. 2017. The Interplay of Population Size and Mutation Probability in the $(1 + \lambda)$ EA on OneMax. *Algorithmica* 78, 2 (2017), 587–609.

[19] Christian Gießen and Carsten Witt. 2018. Optimal Mutation Rates for the $(1+\lambda)$ EA on OneMax Through Asymptotically Tight Drift Analysis. *Algorithmica* 80, 5 (2018), 1710–1731. https://doi.org/10.1007/s00453-017-0360-y

[20] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (2001), 159–195. https://doi.org/10.1162/106365601750190398

[21] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. 2005. On the Choice of the Offspring Population Size in Evolutionary Algorithms. *Evol. Comput.* 13, 4 (2005), 413–440. https://doi.org/10.1162/106365605774666921

[22] 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.

[23] Per Kristian Lehre and Dirk Sudholt. 2019. Parallel Black-Box Complexity with Tail Bounds. *CoRR* abs/1902.00107 (2019). arXiv:1902.00107 http://arxiv.org/abs/1902.00107

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

[25] Laurent Meunier, Herilalaina Rakotoarison, Pak-Kan Wong, Baptiste Rozière, Jérémy Rapin, Olivier Teytaud, Antoine Moreau, and Carola Doerr. 2020. Black-Box Optimization Revisited: Improving Algorithm Selection Wizards through Massive Benchmarking. *CoRR* abs/2010.04542 (2020). arXiv:2010.04542 https://arxiv.org/abs/2010.04542

[26] Vladimir Mironovich and Maxim Buzdalov. 2017. Evaluation of heavy-tailed mutation operator on maximum flow test generation problem. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17, Companion Material)*. ACM, 1423–1426. https://doi.org/10.1145/3067695.3082507

[27] Raymond Ros and Nikolaus Hansen. 2008. A simple modification in CMA-ES achieving linear time and space complexity. In *Parallel Problem Solving from Nature – PPSN X*. Number 5199 in Lecture Notes in Computer Science. 296–305.

[28] Furong Ye, Carola Doerr, and Thomas Bäck. 2019. Interpolating Local and Global Search by Controlling the Variance of Standard Bit Mutation. In *Proc. of IEEE Congress on Evolutionary Computation (CEC'19)*. IEEE, 2292–2299.