Fast Genetic Algorithms

Benjamin Doerr
École Polytechnique
Laboratoire d'Informatique (LIX)
Palaiseau, France

Huu Phuoc Le
École Polytechnique
Palaiseau, France

Régis Makhmara
École Polytechnique
Laboratoire d'Informatique (LIX)
Palaiseau, France

Ta Duy Nguyen
École Polytechnique
Palaiseau, France

ABSTRACT

For genetic algorithms (GAs) using a bit-string representation of length n, the general recommendation is to take $1/n$ as mutation rate. In this work, we discuss whether this is justified for multimodal functions. Taking jump functions and the (1+1) evolutionary algorithm (EA) as the simplest example, we observe that larger mutation rates give significantly better runtimes. For the $\textsc{Jump}_{m,n}$ function, any mutation rate between $2/n$ and m/n leads to a speed-up at least exponential in m compared to the standard choice.

The asymptotically best runtime, obtained from using the mutation rate m/n and leading to a speed-up super-exponential in m, is very sensitive to small changes of the mutation rate. Any deviation by a small $(1 \pm \varepsilon)$ factor leads to a slow-down exponential in m. Consequently, any fixed mutation rate gives strongly sub-optimal results for most jump functions.

Building on this observation, we propose to use a random mutation rate α/n, where α is chosen from a power-law distribution. We prove that the $(1 + 1)$ EA with this heavy-tailed mutation rate optimizes any $\textsc{Jump}_{m,n}$ function in a time that is only a small polynomial (in m) factor above the one stemming from the optimal rate for this m. Our heavy-tailed mutation operator yields similar speed-ups (over the best known performance guarantees) for the vertex cover problem in bipartite graphs and the matching problem in general graphs.

Following the example of fast simulated annealing, fast evolution strategies, and fast evolutionary programming, we propose to call genetic algorithms using a heavy-tailed mutation operator *fast genetic algorithms*.

CCS CONCEPTS

•Computing methodologies → Genetic algorithms; •Theory of computation → *Evolutionary algorithms;*

KEYWORDS

Evolutionary algorithm; mutation operator; heavy-tailed distribution; power-law distribution; multimodal optimization.

ACM Reference format:
Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. 2017. Fast Genetic Algorithms. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071301

GECCO '17, Berlin, Germany

978-1-4503-4920-8/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3071178.3071301

1 INTRODUCTION

One of the basic variation operators in evolutionary algorithmics is mutation, which is generally understood as a mild modification of a single parent individual. When using a bit-string representation, the most common mutation operator is *standard-bit mutation*, which flips each bit of the parent bit-string $x \in \{0,1\}^n$ independently with some probability p_n. The general recommendation is to use a *mutation rate* of $p_n = 1/n$. The expected number of bits parent and offspring differ in then is one. $p_n = 1/n$ is also the mutation rate which maximizes the probability to create a Hamming neighbor y as offspring of the parent x, that is, y differs from x in exactly one bit. This mutation rate also gives the asymptotically optimal expected optimization times for several simple evolutionary algorithms on classic simple test problems (see Subsection 2 for the details).

In this work, we argue that the $1/n$ recommendation could be the result of an over-fitting to these simple unimodal test problems. As a first indication for this, we determine the optimal mutation rate for optimizing *jump functions*, which were introduced in [14]. The function $\textsc{Jump}_{m,n}$, $m \geq 2$, differs from the simple unimodal OneMax function (counting the number of ones in the bit-string) in that the fitness on the last $m-1$ suboptimal fitness levels is replaced by a very small value. Consequently, an elitist algorithm quickly finds a search point on the thin plateau of local optima, but then needs to flip m bits to jump over the fitness valley to the global optimum.

Denote by $T_p(m,n)$ the expected *optimization time* (number of search points evaluated until the optimum is found) of the $(1 + 1)$ evolutionary algorithm (EA) with mutation rate p on the function $\textsc{Jump}_{m,n}$. Extending the result of [14] to arbitrary mutation rates, we observe that for all $m = o(n)$, the classic choice of the mutation rate gives an expected optimization time of

$$T_{1/n}(m,n) = (1 + o(1))en^m,$$

whereas the choice of $p_n = m/n$ leads to an expected optimization time of

$$T_{m/n}(m,n) = \left((1+o(1))\tfrac{e}{m}\right)^m n^m,$$

an improvement super-exponential in m. This is optimal apart from lower order terms, that is, $T_{\mathrm{opt}}(m,n) := \inf_{p\in[0,1/2]}\{T_p(m,n)\}$ satisfies $T_{\mathrm{opt}} = (1+o(1))T_{m/n}$.

This large runtime improvement by choosing an uncommonly large mutation rate may be surprising, but as our proofs reveal [13], there is a good reason for it. It is true that raising the mutation rate from $1/n$ to m/n decreases the rate of 1-bit flips from roughly $1/e$ to roughly me^{-m}. However, finding a particular Hamming neighbor is much easier than finding the required distance-m search point. Consequently, the factor me^{-m} slow-down of the roughly $n/2$ one-bit improvements occurring in a typical optimization process is significantly outnumbered by the factor $m^m e^{-m}$ speep-up of finding the m-bit jump to the global optimum.

These observations suggest that the traditional choice of the mutation rate, leading to a maximal rate of 1-bit flips, is not ideal. Instead, one should rather optimize the mutation rate with the aim of maximizing the rate of the largest required long-distance jump in the search space.

Continuing with the example of the jump functions, however, we also observe that small deviations from the optimal mutation rate lead to significant performance losses. When optimizing the function $\textsc{Jump}_{m,n}$ with a mutation rate that differs from m/n by a small constant factor in either direction, the expected optimization becomes larger than $T_{m/n}(m,n) \approx T_{\mathrm{opt}}(m,n)$ by a factor exponential in m. Consequently, there is no good one-size-fits-all mutation rate and finding a good mutation rate for an unknown multimodal problem requires a deep understanding of the fitness landscape.

Based on these insights, we propose to use standard-bit mutation not with a fixed rate but with a rate chosen randomly according to a *heavy-tailed* distribution. Such a distribution ensures that the number of bits flipped is not strongly concentrated around its mean, which eases having jumps of all sizes in the search space. More precisely, the heavy-tailed mutation operator we propose first chooses a number $\alpha \in [1..n/2]$ according to a power-law distribution D_n^β with exponent $\beta > 1$ and then creates the offspring via standard-bit mutation with rate α/n.

This mutation operator shares many desirable properties with the classic operator. For example, the probability that a single bit (or any other constant number of bits) is flipped is constant. This implies that many classic runtime results hold for our new mutation operator as well. Also, any search point can be created from any parent with positive probability. This probability, however, in the worst case is much higher than when using the classic mutation operator. Consequently, the (tight) general $O(n^n)$ runtime bound for the (1 + 1) EA optimizing any pseudo-Boolean function with unique optimum [14] improves to $O(n^\beta 2^n)$.

For our main example for a multimodal landscape, the jump functions, we prove that the (1 + 1) EA with our heavy-tailed mutation operator finds the optimum of *any* function $\textsc{Jump}_{m,n}$ with $m > \beta - 1$ in expected time

$$T_{D_n^\beta}(m,n) \le O\left(m^{\beta-0.5}\left((1+o(1))\tfrac{e}{m}\right)^m n^m\right),$$

which is again an improvement super-exponential in m over the classic runtime $T_{1/n}(m,n)$ and only a small polynomial factor of $O(m^{\beta-0.5})$ slower than $T_{\mathrm{opt}}(m,n)$, the expected runtime stemming from the mutation rate which is optimal for this m. Note that in return for this small polynomial factor loss over the best instance specific mutation rate, we obtained a single mutation operator that achieves a near-optimal (apart from this small polynomial factor) performance on all instances. Note further that the restriction $m > \beta - 1$ is automatically fulfilled when using a $\beta < 3$, which is both a good choice from the view-point of heavy-tailed distributions and in the light of the $O(m^{\beta-0.5})$ slow-down factor.

We observe that a small polynomial-factor slow-down cannot be avoided when aiming at a competitive performance on all instances. We prove a lower bound result showing that no randomized choice D of the mutation rate can give a performance of $T_D(m,n) = O(m^{0.5}T_{\mathrm{opt}}(m,n))$ for all m. Consequently, by choosing β close to 1, we get essentially the theoretically best performance on all jump functions.

Some elementary experiments show that the above runtime improvements are visible already from small problem sizes on. For $m = 8$, the (1 + 1) EA using the heavy-tailed operator with $\beta = 1.5$ was faster than the classic choice by a factor of at least 2000 on each instance size $n \in \{20, 30, \ldots, 150\}$.

Our very precise mathematical analyses are made possible by regarding the clean test example of the jump functions. To indicate that heavy-tailed mutation operators can be useful also for combinatorial optimization problems, we regard in Section 5.5 two such problems regarded previously in the evolutionary computation literature. For both, we prove a runtime guarantee significantly superior to the guarantees known for the classic mutation operator.

Overall, these results show that multimodal optimization problems might need mutation operators that move faster through the search space than standard-bit mutation with mutation rate $1/n$. A simple way of achieving this goal that in addition works uniformly well over all required jump sizes are the heavy-tailed mutation operators suggested in this work. To the best of our knowledge, this is the first time that a heavy-tailed mutation operator is proposed for discrete evolutionary algorithms. Heavy-tailed mutation operators have been regarded before in simulated annealing [36], evolutionary programming [42], evolution strategies [41], and other subfields of evolutionary computation, however, always in continuous search spaces. Since these algorithms were called *fast* by their inventors, that is, fast simulated annealing, fast evolutionary programming, and fast evolution strategies, for reasons of consistency, we shall call genetic algorithms employing such operators *fast genetic algorithms*, well aware of the fact that this first scientific work regarding heavy-tailed mutation in discrete search spaces does by far not give a complete picture on this approach. The results obtained in this work, however, indicate that this is a promising direction deserving more research efforts.

2 RELATED WORK

2.1 Static Mutation Rates

For reasons of space, we cannot discuss the whole literature on what is the right way to choose the *mutation rate*, that is, the expected fraction of the bit positions in which parent and mutation offspring

differ. Restricting ourselves to evolutionary algorithms for discrete optimization problems, the long-standing recommendation, based, e.g., on [2, 26] is that a mutation rate of $1/n$, that is, flipping in average one bit, is a good choice. A mutation rate of roughly this order of magnitude is used in many experimental works. Nevertheless, in particular in evolutionary algorithms using crossover, the interplay between mutation and crossover may ask for a different choice of the mutation rate. For example, in algorithms using first crossover and then applying mutation to the crossover offspring, a smaller mutation rate can be used to implicitly reduce the mutation probability, that is, the probability that an individual is subject to mutation at all. The $(1 + (\lambda, \lambda))$ GA [8] works best with a higher mutation rate, because it uses crossover with the parent as repair mechanism after the mutation phase.

For simple elitist mutation-based algorithms, which are the best object to study the working principles of mutation in isolation, the following results have been proven: For the $(1 + 1)$ EA, it was shown that $p = 1/n$ is asymptotically the unique best mutation rate for the class of all pseudo-Boolean linear functions [40]. For the LeadingOnes test function, a slightly higher rate of approximately $1.59/n$ is optimal [4]. For the $(1 + \lambda)$ EA optimizing OneMax, things are less clear. For any constant $r > 0$, the $(1 + \lambda)$ EA with mutation rate r/n takes an expected number of $(1+o(1))\big(\frac{1}{2}\frac{n \ln \ln \lambda}{\ln \lambda} + \frac{e^r}{r}\frac{n \ln n}{\lambda}\big)$ generations to find the optimum of OneMax [19]. This indicates that for small values of λ, $1/n$ is the best mutation rate. However, in [9] it was shown that a mutation rate of $\ln(\lambda)/2n$ gives an expected runtime of $O\big(\frac{n}{\log \lambda} + \frac{n \log n}{\sqrt{\lambda}}\big)$, which is asymptotically faster than the previous bound when, e.g., $\lambda \geq (\ln n)^{2+\varepsilon}$ for any constant $\varepsilon > 0$.

2.2 Dynamic Mutation Rates

Since our heavy-tailed mutation operator can be seen as a dynamic choice of the mutation rate (according to a relatively trivial dynamics), let us quickly review the few results close to ours. There is a general belief that a dynamic choice of the mutation rate can be profitable, typically starting with a higher rate and reducing it during the run of the algorithm. Despite this, dynamic choices of the mutation rate are still not that often seen in today's applied research. On the theory side, the first work [24] analyzing a dynamic choice of the mutation strength proposes to take in iteration t the mutation rate $2^{(t-1) \bmod (\lceil \log_2 n \rceil - 1)}/n$. In other words, the mutation rates $1/n, 2/n, 4/n, \ldots, 2^{\lceil \log_2 n \rceil - 2}/n$ are used in a cyclic manner. The $(1 + 1)$ EA using this dynamic mutation rate has an expected runtime larger by a factor of $\Theta(\log n)$ for several classic test problems. On the other hand, there are problems where this dynamic EA has a polynomial runtime, whereas all static choices of the mutation rate lead to an exponential runtime. We remark without proof that these results would also hold if the mutation rate was chosen in each iteration uniformly at random from the set of these powers of two. We note without formal proof that the arguments used in the proof of Theorem 4.1 together with Corollary 4.2 show that either version of this dynamic EA would have a runtime of $\exp(\Omega(m))T_{\text{opt}}(m, n)$ on $\text{Jump}_{m,n}$ for most values of m (namely all that are a small constant factor away from the nearest power of two) and all values of n.

For the classic test functions, the following is known: For LeadingOnes, a fitness-dependent mutation rate gave a small constant-factor improvement over static rates in [4]. For the optimization of OneMax using the $(1 + 1)$ EA, a dynamic mutation rate is known to give runtime improvements only of lower order. Surprisingly, for the $(1 + \lambda)$ EA, a dynamic choice of the mutation rate can lead to an asymptotically better runtime [3], and this mutation rate can be found on the fly in a self-adjusting manner [9]. We note without formal proof that for jump functions, a fitness-dependent mutation rate cannot give a significant improvement over the best static mutation as can be seen from our analysis in Section 4.

2.3 Heavy-Tailed Mutation Operators

The idea to use heavy-tailed mutation operators is not new in evolutionary computation and, more generally, heuristic optimization. However, it was so far restricted to continuous optimization. Szu and Hartley suggested to use a (heavy-tailed) Cauchy distribution instead of Gaussian distributions in simulated annealing and report significant speed-ups [36]. This idea was taken up in evolutionary programming [42], in evolution strategies [41], estimation of distribution algorithms (EDAs) [29], and in natural evolution strategies [33]. However, also some doubts on the general usefulness of heavy-tailed mutations have been raised. Based on mathematical considerations and experiments, it has been suggested that heavy-tailed mutations are useful only if the large variations of these operators take place in a low-dimensional subspace and this space contains the good solutions of the problem [21]. Otherwise, the curse of dimensionality makes it just too improbable that a long-range mutation finds a better solution. Also, [31] has pointed out that spherical Cauchy distributions lead to the same order of local convergences as Gaussian distributions, whereas non-spherical Cauchy distributions even lead to a slower local convergence. A heavy-tailed mutation EDA was shown to be significantly inferior to BIPOP-CMA-ES via the BBOB algorithm comparison tool [30].

3 PRELIMINARIES

Throughout this paper, we use the following elementary notation. For $a, b \in \mathbb{R}$, we write $[a..b] := \{z \in \mathbb{Z} \mid a \leq z \leq b\}$ to denote the set of integers in the real interval $[a, b]$. We denote by $\mathbb{N}$ the set of *positive* integers and by $\mathbb{N}_0$ the set of non-negative integers. For $n, m \in \mathbb{N}_0$ with $m \leq n$, we write $\binom{n}{\leq m} := \sum_{i=0}^{m} \binom{n}{i}$ for the number of subsets of an n-element set that have at most m elements. For two bit-strings $x, y \in \{0,1\}^n$ of length n, we denote by $H(x, y) := |\{i \in [1..n] \mid x_i \neq y_i\}|$ the *Hamming distance* of x and y.

For reasons of space, we have to omit the proofs of the results stated below. They can be found in the arXiv version of this paper [13].

3.1 Jump Functions

In this work, we investigate the influence of the mutation operator on the performance of genetic algorithms optimizing multimodal functions. We restrict ourselves to *pseudo-Boolean* functions, that is, functions $f : \{0,1\}^n \to \mathbb{R}$ defined on bit-strings of a given length n. As much as the OneMax test function $\text{OneMax}_n : x \in \{0,1\}^n \mapsto \sum_{i=1}^{n} x_i \in \mathbb{R}$ is the prime example

to study the optimization on easy unimodal fitness landscapes, the most popular test problem for multimodal landscapes are jump functions. For $n \in \mathbb{N}$ and $m \in [1..n]$, Droste, Jansen, and Wegener [14] define the n-dimensional jump function $\textsc{Jump}_{m,n} : \{0,1\}^n \to \mathbb{R}$ by

$$\textsc{Jump}_{m,n}(x) = \begin{cases} m + \textsc{OneMax}_n(x) & \text{if } \textsc{OneMax}_n(x) \leq n-m \\ & \text{or } \textsc{OneMax}_n(x) = n \\ n - \textsc{OneMax}_n(x) & \text{otherwise} \end{cases}$$

for all $x \in \{0,1\}^n$. In this paper, we are only interested in the non-degenerate case $m \in [2..n/2]$.

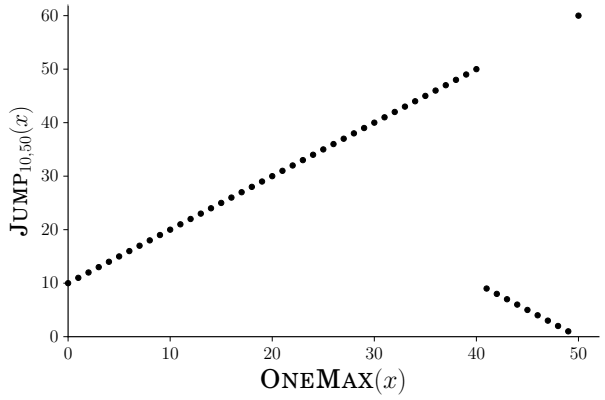

Figure 1: The function $\textsc{Jump}_{m,n}$ for $n = 50$ and $m = 10$.

Jump functions are a useful object to study how well evolutionary algorithms can leave local optima. With the whole radius-m Hamming sphere around the global optimum forming an inferior local optimum of $\textsc{Jump}_{m,n}$, it is very hard for an evolutionary algorithm to not get trapped in this local optimum for a while. Due to the symmetry of the landscape, the only way to leave the local optimum to a better solution is to flip exactly the right m bits. This symmetric and well-understood structure with exactly one fitness valley to be crossed in a typical optimization process makes the jump functions a popular object to study how evolutionary algorithms can cope with local optima.

Droste et al. [14] show that the (1 + 1) EA (made precise in the following subsection) with mutation rate $p = 1/n$ takes an expected number of $T_{1/n}(m,n) = \Theta(n^m + n \log n)$ fitness evaluations to find the maximum of $\textsc{Jump}_{m,n}$. Here and in the following, all asymptotic notation is to be understood that the implicit constants can be chosen independent of n and m. For a broad class of non-elitist algorithms using a mutation rate of c/n, an upper bound of $O(n\lambda \log \lambda + (n/c)^m)$ was shown in [7] for the optimization time on $\textsc{Jump}_{m,n}$. Here c is supposed to be a constant. We are not aware of other runtime analyses for mutation-based algorithms optimizing jump functions.

The Jump function family has also been an example to study in a rigorous manner the effectiveness of crossover. The first work in this direction [23] shows that a simple $(\mu + 1)$ genetic algorithm with appropriate parameter settings can obtain a better runtime than mutation-based algorithms. Very roughly speaking, for $m = O(\log n)$, this GA has a runtime of $O(4^m \text{ poly}(n))$, reducing the runtime dependence on m from $\Theta(n^m)$ to single-exponential. While this result was the first mathematically supported indication that crossover can be useful in discrete evolutionary optimization, it has, as the authors point out, the limitation that it applies only to a GA that uses crossover very sparingly, namely with probability at most $O(1/(mn))$, which is very different from the typical application of crossover. This dependence was mildly improved to $O(m/n)$ along with allowing wider ranges for other parameters in [25]. Interestingly, the research activity on the problem of rigorously proving the usefulness of crossover shifted away from jump functions to real royal road functions [23, 34] (which are still similar to jump functions), simplified Ising models [15, 35], and the all-pairs shortest path problem [10, 11]. Only last year, Dang et al. [5, 6] by carefully analyzing the population dynamics could show that a simple GA employing crossover and using natural parameter settings can obtain an expected optimization time of $O(n^{m-1} \log n)$ on $\textsc{Jump}_{m,n}$ for constant $m \geq 3$, which is an $O(n/\log n)$ factor speed-up over comparably simple mutation-based EAs.

3.2 The (1 + 1) EA

To study the working principles of different mutation operators, we regard the most simple evolutionary algorithm: the (1 + 1) evolutionary algorithm (EA). This is a common approach in the theory of evolutionary algorithms, which is based on the experience that results for this simple EA often are valid in a similar manner for more complicated EAs. Without proof, remark that most of our findings in an analogous manner hold for many elitist mutation-based $(\mu + \lambda)$ EAs.

The (1 + 1) EA, given as Algorithm 1, starts with a random search point $x \in \{0,1\}^n$. In the main optimization loop, it creates a mutation offspring from the parent x, which replaces the parent unless it has an inferior fitness. Since our focus is on how long this EA takes to create an optimal solution, we do not specify a termination criterion.

Algorithm 1: The (1+1) EA with static mutation rate p for maximizing $f : \{0,1\}^n \to \mathbb{R}$.

1 **Initialization:** Sample $x \in \{0,1\}^n$ uniformly at random;
2 **Optimization: for** $t = 1, 2, 3, \ldots$ **do**
3 Sample $y \in \{0,1\}^n$ by flipping each bit in x with probability p ; //mutation step
4 **if** $f(y) \geq f(x)$ **then** $x \leftarrow y$; //selection step;

As usual in theoretically oriented works in evolutionary computation, we regard the number of fitness evaluations it took to achieve the desired goal as performance measure of an evolutionary algorithm. For this reason, we define $T_p(m,n)$ to be the expected number of fitness evaluations the (1 + 1) EA performs when optimizing the $\textsc{Jump}_{m,n}$ function until it first evaluates the optimal solution. We shall always assume that the mutation rate p is in $[0, 1/2]$. When p depends on the bit-string length n, as, e.g., in the recommended choice $p = 1/n$, we shall for the ease of reading usually make this functional dependence not explicit (e.g., by writing $p(n)$), but simply continue to write p.

4 STATIC MUTATION RATES

In this section, we analyze the performance of the (1 + 1) EA on jump functions when employing the standard-bit mutation operator that flips each bit independently with fixed probability $p \in (0, 1/2]$. Our main result is that the mutation rate giving the asymptotically best runtime on $\textsc{Jump}_{m,n}$ functions is $p = m/n$, which is far from the standard choice of $1/n$ when m is large. Moreover, we observe that a small constant factor deviation from the m/n mutation rate immediately incurs a runtime increase by a factor exponential in $\Omega(m)$. To obtain these results, we first determine (with sufficient precision) the optimization time of the (1 + 1) EA on $\textsc{Jump}_{m,n}$ functions.

Theorem 4.1. *For all $n \in \mathbb{N}$, $m \in [2..n/2]$ and $p \in (0, 1/2]$, the expected optimization time $T_p(m, n)$ of the (1 + 1) EA with mutation rate p on the n-dimensional test problem $\textsc{Jump}_{m,n}$ satisfies*

$$\left(1-\tbinom{n}{\leq m-1}2^{-n}\right)\frac{1}{p^m(1-p)^{n-m}} \leq T_p(m,n) \leq \frac{1}{p^m(1-p)^{n-m}} + \frac{2\ln\frac{n}{m}}{p(1-p)^{n-1}}.$$

In particular, if $p \leq \frac{m}{n}$, then

$$\tfrac{1}{2}\frac{1}{p^m(1-p)^{n-m}} \leq T_p(m,n) \leq 3\frac{1}{p^m(1-p)^{n-m}}.$$

Consequently, for any $p \in \left[\frac{2}{n}, \frac{m}{n}\right]$, $T_p(m, n) \leq O(2^{-m}T_{1/n}(m, n))$.

Corollary 4.2. *The best possible optimization time $T_{\text{opt}}(m, n) := \inf\{T_p(m, n) \mid p \in [0, 1/2]\}$ for a static mutation rate satisfies*

$$\tfrac{1}{2}\frac{n^m}{m^m}\left(\tfrac{n}{n-m}\right)^{n-m} \leq T_{\text{opt}}(m,n) \leq 3\frac{n^m}{m^m}\left(\tfrac{n}{n-m}\right)^{n-m}.$$

These bounds also hold for $T_{m/n}(m, n)$, whereas for all $0 < \varepsilon < 1$, any mutation rate $p \in [0, 1/2] \setminus [(1-\varepsilon)m/n, (1+\varepsilon)m/n]$ gives a runtime slower than $T_{\text{opt}}(m, n)$ by a factor of at least $\frac{1}{6}\exp(m\varepsilon^2/5)$.

5 DESIGN AND ANALYSIS OF HEAVY-TAILED MUTATION OPERATORS

In the previous section, we observed that an asymptotically optimal mutation rate for the $\textsc{Jump}_{m,n}$ function is m/n rather than the general suggestion of $1/n$. However, due to the strong concentration of the number of bits that are flipped, we also observed that a small constant factor deviation from this optimal mutation rate incurs a significant increase in the runtime (exponential in m). From the view-point of algorithms design, this suggests that to get a good performance when optimizing multimodal functions, the algorithm designer needs to know beforehand which multi-bit flips will be needed to escape local optima. This is, clearly, an unrealistic assumption for any real-world optimization problem. To overcome this difficulty, we now design a mutation operator such that the number of bits flipped is not strongly concentrated, but instead follows a heavy-tailed distribution, more precisely, a power-law distribution.

We prove that the (1 + 1) EA with this operator optimizes all $\textsc{Jump}_{m,n}$ functions with a runtime larger than the optimal runtime $T_{\text{opt}}(m, n)$ only by a small factor polynomially bounded in m, which is much better than the exponential (in m) performance loss incurred from missing the optimal static mutation rate by a few percent. We also show that such a small polynomial loss is unavoidable when aiming for an algorithm that shows a good performance on all jump functions. Finally, we show that similar performance gains from using a heavy-tailed mutation operator can also be observed with two combinatorial optimization problems, namely the problem of computing small vertex covers in complete bipartite graphs and the problem of computing large matchings in arbitrary graphs.

5.1 The Heavy-tailed Mutation Operator $\texttt{fmut}_\beta$

The main reason why only a very carefully chosen mutation rate gave near-optimal results in Corollary 4.2 is the strong concentration behavior of the binomial distribution. If we flip bits independently with probability m/n, then with high probability the actual number of bits flipped is strongly concentrated around m. The probability that we flip $(1-\varepsilon)m/n$ bits and less, or $(1+\varepsilon)m/n$ bits and more, at most $2\exp(-\varepsilon^2 m/3)$, that is, is exponentially small in m (this follows directly from classic Chernoff bounds, e.g., [1, Corollary 1.10 (a) and (c)]). Hence to obtain a good performance on a wider set of jump functions (that is, with parameter m varying at least by small constant factors), we cannot employ standard-bit mutation with a static mutation rate.

To overcome the negative effect of strong concentration and at the same time be structurally close to the established way to performing mutation, we propose to use standard-bit mutation with a mutation rate that is chosen randomly in each iteration according to a power-law distribution with exponent β greater than 1. This keeps the property of standard-bit mutation with probability $1/n$ that with constant probability that a single bit is flipped. This property is important to have a good performance in easy unimodal parts of the search space, and in particular, to easily approach the global optimum once one has entered its basin of attraction. At the same time, the heavy-tailed choice of the mutation rate ensures that with probability $\Theta(k^{-\beta})$, exactly k bits are flipped. Hence this event, necessary to leave a local optimum with the $(k-1)$ Hamming ball around it being part of its basin of attraction, is much more likely than when using the classic mutation operator, which flips k bits only with probability $k^{-\Theta(k)}$.

To keep the operator and its analysis simple, we only use mutation rates of type α/n with $\alpha \in [1..n/2]$. We show at the end of this section that no random choice of the mutation rate (including continuous ones) can give a performance on jump functions significantly better than the one stemming from our mutation operator, which justifies this restriction to integer values for α. To ease reading, we shall always write $n/2$ even in cases where an integer is required, e.g., as boundary of the range of a sum. Of course, in all such cases $n/2$ is to be understood as $\lfloor n/2 \rfloor$.

The discrete power-law distribution $D^\beta_{n/2}$: Let $\beta > 1$ be a constant. Then the discrete power-law distribution $D^\beta_{n/2}$ on $[1..n/2]$ is defined as follows. If a random variable X follows the distribution $D^\beta_{n/2}$, then

$$\Pr[X = \alpha] = \left(C^\beta_{n/2}\right)^{-1}\alpha^{-\beta}$$

for all $\alpha \in [1..n/2]$, where the normalization constant is $C^\beta_{n/2} := \sum_{i=1}^{n/2} i^{-\beta}$. Note that $C^\beta_{n/2}$ is asymptotically equal to $\zeta(\beta)$, the Riemann zeta function ζ evaluated at β. We have

$$\zeta(\beta) - \frac{\beta}{\beta-1}\left(\frac{n}{2}\right)^{-\beta+1} \leq C^\beta_{n/2} \leq \zeta(\beta)$$

for all $\beta > 1$. As orientation, e.g., $\zeta(1.5) \approx 2.612$, $\zeta(2) \approx 1.645$, and $\zeta(3) = 1.202$ are some known values of the ζ function.

The heavy-tailed mutation operator $\texttt{fmut}_\beta$: We define the mutation operator $\texttt{fmut}_\beta$ (with the f again referring to the word *fast* usually employed when heavy-tailed distributions are used) as follows: when the parent individual is some bit-string $x \in \{0,1\}^n$, the mutation operator $\texttt{fmut}_\beta$ first chooses a random mutation rate α/n with $\alpha \in [1..n/2]$ chosen according to the power-law distribution $D^\beta_{n/2}$ and then creates an offspring by flipping each bit of the parent independently with probability α/n (that is, via standard-bit mutation with rate α/n).

We collect some important properties of the heavy-tailed mutation operator. Again, we have to skip the proofs.

Lemma 5.1. *Let $n \in \mathbb{N}$ and $\beta > 1$. Let $x \in \{0,1\}^n$ and $y = \texttt{fmut}_\beta(x)$.*

1. *The probability that x and y differ in exactly one bit is $P^\beta_1 := \Pr[H(x,y) = 1] = \left(C^\beta_{n/2}\right)^{-1} \Theta(1)$ with the constants implicit in the $\Theta(\cdot)$ independent of n and β.*
2. *For any $k \in [2..n/2]$ with $k > \beta - 1$, the probability that x and y differ in exactly k bits is $P^\beta_k := \Pr[H(x,y) = k] \geq \left(C^\beta_{n/2}\right)^{-1} \Omega(k^{-\beta})$ with the constants implicit in the $\Theta(\cdot)$ independent of n, k, and β.*
3. *Let $z \in \{0,1\}^n$. We have $\Pr[\texttt{fmut}_\beta(x) = z] \geq \left(C^\beta_{n/2}\right)^{-1} \Omega(2^{-n} n^{-\beta})$. If $\beta - 1 < H(x,z) \leq n/2$ or $H(x,z) = 1$, then $\Pr[\texttt{fmut}_\beta(x) = z] = \left(C^\beta_{n/2}\right)^{-1} \Omega(H(x,z)^{-\beta}) \binom{n}{H(x,z)}^{-1}$. In both cases, the implicit constants can be chosen independent of z, β, and n.*
4. *The expected number of bits that x and y differ in is*

$$\mathbb{E}(H(x,y)) = \begin{cases} \Theta(1) & \text{if } \beta > 2, \\ \Theta(\ln(n)) & \text{if } \beta = 2, \\ \Theta(n^{2-\beta}) & \text{if } 1 < \beta < 2, \end{cases}$$

where the implicit constants may depend on β.

5.2 Evolutionary Algorithms Using the Heavy-tailed Mutation Operator $\texttt{fmut}_\beta$

Since we decided to call algorithms using the heavy-tailed mutation operator $\texttt{fmut}_\beta$ *fast evolutionary algorithms*, we denote the (1 + 1) EA using the operator $\texttt{fmut}_\beta$ from now by (1 + 1) FEA$_\beta$. We do likewise for any other $(\mu + \lambda)$ EA, which we call $(\mu + \lambda)$ FEA$_\beta$ when it employs the mutation operator $\texttt{fmut}_\beta$.

In this first section analyzing the performance of fast EAs, we show that many runtime analyses remain valid for the corresponding fast EA (apart from changes in the leading constant, which in many classic results is not made explicit anyway). An elementary observation is that fast EAs use the mutation rate $1/n$ with constant probability (Lemma 5.1 1). Consequently, all previous runtime analysis which are robust to interleaving with other mutation steps remain valid for the FEAs (apart from constant factor changes of the runtime). These are, in particular, all analyses of elitist EAs based on the fitness level method [37] and on drift arguments using the fitness as potential function. We list some such results in the following theorem. The references point to the original work for the non-fast EA.

Theorem 5.2. *Let $\beta > 1$. The expected runtimes of the (1+1) FEA$_\beta$ on the OneMax and LeadingOnes test functions and on the minimum spanning tree problem are respectively: $O(n \log n)$ [26], $O(n^2)$ [31], and $O(m^2 \log(n w_{\max}))$ [27]. For all $\lambda \leq n^{1-\varepsilon}$, the expected runtimes of the (1+λ) FEA$_\beta$ on the same problems are: $O(\frac{n\lambda \log\log(\lambda)}{\log(\lambda)} + n \log n)$, $O(n^2/\lambda)$ [12, 22], and $O(m^2 \log(n w_{\max})/\lambda)$ [27]. Finally, for all $\mu \leq n^{O(1)}$, the expected runtime of the $(\mu + 1)$ FEA$_\beta$ is $O(\mu n + n \log(n))$ on OneMax and $O(\mu n \log(n) + n^2)$ on LeadingOnes [39].*

For the classic (1 + 1) EA with mutation rate $1/n$, it is known that it finds the optimum of any pseudo-Boolean fitness function in an expected number of at most n^n iterations. This bound is tight in the sense that there are concrete fitness functions for which an expected optimization time of $\Omega(n^n)$ could be proven. These are classic results from [14, Theorems 6 to 8].

Moreover, also problems that generally are perceived as easy may have instances where the classic (1 + 1) EA needs $n^{\Theta(n)}$ time to find the optimum. This was demonstrated for the minimum makespan scheduling problem [38], see also [28, Theorem 7.5 and Lemma 7.8]. While in average (n jobs having random lengths in $[0,1]$) the classic (1 + 1) EA approximates the optimum up to an additive error of 1 in time $O(n^2)$, there are instances of n jobs with processing times in $[0,1]$ such that the (1 + 1) EA needs time $n^{\Omega(n)}$ to only find a solution that is better than $\frac{4}{3}$ times the optimum.

The following result shows that fast EAs only have an exponential worst-case runtime as opposed to the super-exponential times just discussed.

Theorem 5.3. *Let $n \in \mathbb{N}$ and $\beta > 1$. Consider any fast EA creating at least a constant ratio of its offspring via the $\texttt{fmut}_\beta$ mutation operator. Then its expected optimization time on any fitness function $f : \{0,1\}^n \to \mathbb{R}$ is at most $O(C^\beta_{n/2} 2^n n^\beta)$.*

5.3 Runtime Analysis for the (1 + 1) FEA$_\beta$ Optimizing Jump Functions

We now proceed with analyzing the performance of the (1+1) FEA$_\beta$ on jump functions. We show that the (1 + 1) FEA$_\beta$ for all $m \in [2..n/2]$ with $m > \beta - 1$ has an expected optimization time of $O(C^\beta_{n/2} m^{\beta - 0.5} T_{\text{opt}}(m,n))$ for the function $\textsc{Jump}_{m,n}$. By a mild abuse of notation, we denote by $T_\beta(m,n)$ the expected optimization time of the (1 + 1) FEA$_\beta$ on the $\textsc{Jump}_{m,n}$ function.

Theorem 5.4. *Let $n \in \mathbb{N}$ and $\beta > 1$. For all $m \in [2..n/2]$ with $m > \beta - 1$, the expected optimization time $T_\beta(m,n)$ of the (1 + 1) EA with mutation operator $\texttt{fmut}_\beta$ is $T_\beta(m,n) = O\left(C^\beta_{n/2} m^{\beta - 0.5} T_{\text{opt}}(m,n)\right)$, where the constants implicit in the big-O notation can be chosen independent from β, m, and n.*

We do not discuss the case $m \leq \beta - 1$ as for $\beta < 3$, this case does not exist, and we do not have any indication that larger values of β are useful.

5.4 A Lower Bound for a Uniformly Good Performance on all Jump Function

The runtime analysis of the previous subsection showed that the (1 + 1) EA with the heavy-tailed mutation operator optimizes any $\textsc{Jump}_{m,n}$ function in an expected time that is only a small polynomial (in m) factor larger than the runtime stemming from the (for this m) optimal mutation rate. The question remains if this relatively small polynomial factor increase is necessary. In this section, we answer this affirmatively. While choosing $\beta = 1 + \varepsilon$ can reduce the loss factor to $\Theta(m^{0.5+\varepsilon})$ for any $\varepsilon > 0$, no randomized choice of the mutation probability can uniformly obtain a loss factor of $\sqrt{m}$ (or lower). To this aim, let us extend the definition of $T_\beta(m,n)$ and denote by $T_D(m,n)$ the expected number of iterations it takes the (1 + 1) EA to find the optimum of $\textsc{Jump}_{m,n}$ when in each iteration the mutation rate is chosen randomly according to the distribution D.

Theorem 5.5. *Let $c > 0$ and let n be sufficiently large. Then for every distribution D on $[0, 1/2]$, there exists an $m \in [2..n/2]$ such that $T_D(m,n) \geq c\sqrt{m}T_{\text{opt}}(m,n)$.*

5.5 Combinatorial Optimization Problems

In this subsection, we show how our heavy-tailed mutation operator improves two existing runtime results for combinatorial optimization problems. As for jump functions, the reason for these improvements is that multi-bit flips occur with much higher rate when using the heavy-tailed mutation operator.

Let $G = (V, E)$ be an undirected graph. Let $n := |E|$. The *maximum matching problem* consists in finding a largest subset M^* of E such that no two edges in M^* have a vertex in common. In [17, 18], see also [28, Section 6.2], for any constant $\varepsilon > 0$ it is proven that the standard (1 + 1) EA finds a near-maximal matching of size at least $OPT/(1+\varepsilon)$ in an expected time of at most some $T_{\varepsilon,n}$, which is $O(n^{m+1})$ for $m = 2\lceil\frac{1}{\varepsilon}\rceil - 1$. The key argument is that the (1 + 1) EA, similar to the problem-specific algorithms, is able to change the matching status of edges along an augmenting path. Since this is the dominant contribution to $T_{\varepsilon,n}$ and since the multi-bit flips corresponding to such augmentation operations occur more frequently when running the (1 + 1) FEA_β, for this algorithm we obtain an improved runtime guarantee of $(1 + o(1))eC^\beta_{n/2}(e/m)^m m^{\beta-0.5}T_{\varepsilon,n}$, an improvement by roughly a factor of $m^{\Theta(m)}$.

Let now $G = (V, E)$ be a finite graph with $n := |V|$. Then the *vertex cover problem* consists of finding a subset $V' \subseteq V$ of minimal size such that each edge of the graph has at least one of its vertices in V'. Friedrich, Hebbinghaus, Neumann, He, and Witt [16] (see also [28, Chapter 12]) analyze how evolutionary algorithms compute minimum vertex covers. They observe that already on complete bipartite graphs with partition classes of size $m \leq n/2$ and $n-m$, the standard (1+1) EA has an expected optimization time of $\Omega(mn^{m-1} + n\log n)$, where to obtain this precise bound an inspection of their proofs is necessary. For the (1 + 1) FEA_β, we obtain the following superior runtime guarantees. If $m \leq n/3$, then the (1 + 1) FEA_β find the global optimum in only $O(C^\beta_{n/2}n^\beta 2^m + n\log n)$ iterations. For $m \geq n/3$, our general bound of $O(C^\beta_{n/2}n^\beta 2^n)$ gives again a significant improvement over the classic (1 + 1) EA.

6 EXPERIMENTS

We ran an implementation of the (1 + 1) FEA_β against the jump function, with n varying between 20 and 150. For $m = 8$, Figure 2 shows the average number of iterations (on 1000 runs) of the algorithm before reaching the optimal value of $(1, \ldots, 1) \in \mathbb{R}^n$, with different values of the parameter β and along with the classical (1 + 1) EA with mutation rates $1/n$ and m/n. To allow this large number of independent experiments, we stopped the actual run when a local optimum was reached. The remaining runtime from that point on follows a simple geometric distribution, which we sample directly instead of actually running the EA. By this, we obtain a significant speed-up, but we still sample from the precise runtime distribution.

We observe that small values of β give better results, although no significant performance increase can be seen below $\beta = 1.5$ (which is why we depicted only cases with $\beta \geq 1.5$). The runtimes for $\beta = 1.5$ uniformly are better than the one of the classic (1 + 1) EA by a factor of 2000, and worse than the (1 + 1) EA with mutation rate m/n by a factor of 10.

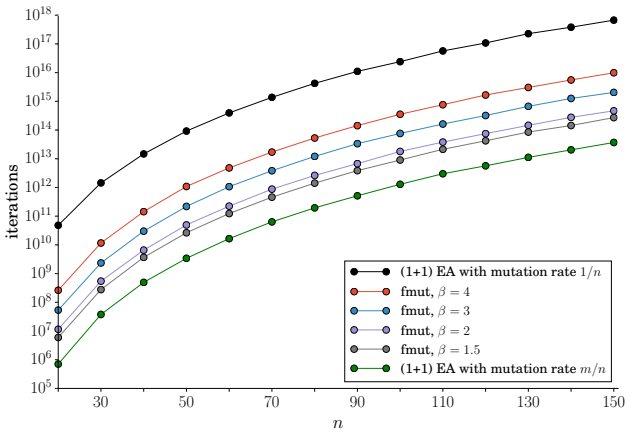

Figure 2: Average number (over 1000 runs) of iterations the (1 + 1) EA took with different mutation operators to find the optimum of $\textsc{Jump}_{8,n}$.

7 CONCLUSIONS AND OUTLOOK

In this work, we took a critical look at the performance of simple mutation-based evolutionary algorithms on multimodal fitness landscapes. Guided by the classic example of jump functions, we observed that mutation rates significantly above the usual recommendation of $1/n$ led to much better results. The proofs of our results suggest that when a multi-bit flip is necessary to leave a local optimum, then so much time is needed to find the right bits to be flipped that it is justified to use a mutation probability high enough that such numbers of bits are sufficiently often touched. The speed-up here greatly outnumbers the slow-down incurred in easy parts of the fitness landscape.

Since we also observe that the optimal performance can only be obtained for a very small interval of mutation probabilities, we suggest to choose the mutation probability randomly according

to a power-law distribution. We prove that this results in a one-size-fits-all mutation operator, giving a nearly optimal performance on all jump functions. We observe that this mutation operator gives an asymptotically equal or better (including massively better) performance on many problems that were analyzed rigorously before.

Let us remark that heavy-tailed mutation is not restricted to bit-string representations. For combinatorial problems for which a bit-string representation is inconvenient, e.g., permutations, one way of imitating standard-bit mutation is to sample a number k according to a Poisson distribution with expectation 1 and then perform k elementary mutation steps, where an elementary mutation step is some simple modification of a search point, e.g., a random swap of two elements in the case of permutations [32]. Obviously, to obtain a heavy-tailed mutation operator one just needs to replace the Poisson distribution with a heavy-tailed distribution, e.g., a power-law distribution as used in this work. We are optimistic that such approaches can lead to similar improvements as observed in this work, but we have not regarded this in detail.

Another important step towards understanding heavy-tailed mutation operators would be to gain experience on its performance on real applications. To make it easiest for other researchers to try our new methods, we have put the (simple) code we used in the repository [20].

Acknowledgements

This research was supported by Labex DigiCosme (project ANR11LABEX0045DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex ParisSaclay (ANR11IDEX000302) as well as by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH.

REFERENCES

[1] Anne Auger and Benjamin Doerr. 2011. *Theory of Randomized Search Heuristics: Foundations and Recent Developments.* World Scientific.
[2] Thomas Bäck. 1993. Optimal mutation rates in genetic search. In *ICGA*. Morgan Kaufmann, 2–8.
[3] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. 2014. Unbiased black-box complexity of parallel search. In *Parallel Problem Solving from Nature XIII*. Springer, 892–901.
[4] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. 2010. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *PPSN (1) (Lecture Notes in Computer Science)*, Vol. 6238. Springer, 1–10.
[5] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro Simone Oliveto, Dirk Sudholt, and Andrew M. Sutton. 2016. Emergence of diversity and its benefits for crossover in genetic algorithms. In *PPSN (Lecture Notes in Computer Science)*, Vol. 9921. Springer, 890–900.
[6] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro Simone Oliveto, Dirk Sudholt, and Andrew M. Sutton. 2016. Escaping local optima with diversity mechanisms and crossover. In *GECCO*. ACM, 645–652.
[7] Duc-Cuong Dang and Per Kristian Lehre. 2016. Runtime analysis of non-elitist populations: From classical optimisation to partial information. *Algorithmica* 75 (2016), 428–461.
[8] Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science* 567 (2015), 87–104.
[9] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. 2017. The (1+λ) evolutionary algorithm with self-adjusting mutation rate. In *GECCO*. ACM.
[10] Benjamin Doerr, Edda Happ, and Christian Klein. 2012. Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science* 425 (2012), 17–33.
[11] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Frank Neumann, and Madeleine Theile. 2013. More effective crossover operators for the all-pairs shortest path problem. *Theoretical Computer Science* 471 (2013), 12–26.
[12] Benjamin Doerr and Marvin Künnemann. 2015. Optimizing linear functions with the (1+λ) evolutionary algorithm - Different asymptotic runtimes for different instances. *Theoretical Computer Science* 561 (2015), 3–23.
[13] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. 2017. Fast genetic algorithms. *ArXiv e-prints* (March 2017). arXiv:1703.03334
[14] Stefan Droste, Thomas Jansen, and Ingo Wegener. 2002. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science* 276 (2002), 51–81.
[15] Simon Fischer and Ingo Wegener. 2005. The one-dimensional Ising model: Mutation versus recombination. *Theoretical Computer Science* 344 (2005), 208–225.
[16] Tobias Friedrich, Jun He, Nils Hebbinghaus, Frank Neumann, and Carsten Witt. 2010. Approximating covering problems by randomized search heuristics using multi-objective models. *Evolutionary Computation* 18 (2010), 617–633.
[17] Oliver Giel and Ingo Wegener. 2003. Evolutionary algorithms and the maximum matching problem. In *STACS (Lecture Notes in Computer Science)*, Vol. 2607. Springer, 415–426.
[18] Oliver Giel and Ingo Wegener. 2004. Searching randomly for maximum matchings. *Electronic Colloquium on Computational Complexity (ECCC)* 076 (2004).
[19] Christian Gießen and Carsten Witt. 2015. Population size vs. mutation strength for the (1+λ) EA on OneMax. In *GECCO*. ACM, 1439–1446.
[20] GitHub. 2017. Fast genetic algorithms. (2017). https://github.com/FastGA/fast-genetic-algorithms.
[21] Nikolaus Hansen, Fabian Gemperle, Anne Auger, and Petros Koumoutsakos. 2006. When do heavy-tail distributions help?. In *PPSN (Lecture Notes in Computer Science)*, Vol. 4193. Springer, 62–71.
[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 Ingo Wegener. 2005. Real royal road functions–where crossover provably is essential. *Discrete Applied Mathematics* 149 (2005), 111–125.
[24] Thomas Jansen and Ingo Wegener. 2006. On the analysis of a dynamic evolutionary algorithm. *J. Discrete Algorithms* 4 (2006), 181–199.
[25] Timo Kötzing, Dirk Sudholt, and Madeleine Theile. 2011. How crossover helps in pseudo-Boolean optimization. In *GECCO*. ACM, 989–996.
[26] Heinz Mühlenbein. 1992. How genetic algorithms really work: Mutation and hillclimbing. In *PPSN*. Elsevier, 15–26.
[27] Frank Neumann and Ingo Wegener. 2007. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science* 378 (2007), 32–40.
[28] Frank Neumann and Carsten Witt. 2010. *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity.* Springer Berlin Heidelberg.
[29] Petr Posik. 2009. BBOB-benchmarking a simple estimation of distribution algorithm with Cauchy distribution. In *GECCO (Companion)*. ACM, 2309–2314.
[30] Petr Posík. 2010. Comparison of Cauchy EDA and BIPOP-CMA-ES algorithms on the BBOB noiseless testbed. In *GECCO (Companion)*. ACM, 1697–1702.
[31] Günter Rudolph. 1997. *Convergence Properties of Evolutionary Algorithms.* Kovac.
[32] Jens Scharnow, Karsten Tinnefeld, and Ingo Wegener. 2004. The analysis of evolutionary algorithms on sorting and shortest paths problems. *J. Math. Model. Algorithms* 3 (2004), 349–366.
[33] Tom Schaul, Tobias Glasmachers, and Jürgen Schmidhuber. 2011. High dimensions and heavy tails for natural evolution strategies. In *GECCO*. ACM, 845–852.
[34] Tobias Storch and Ingo Wegener. 2004. Real royal road functions for constant population size. *Theoretical Computer Science* 320 (2004), 123–134.
[35] Dirk Sudholt. 2005. Crossover is provably essential for the ising model on trees. In *GECCO*. ACM, 1161–1167.
[36] Harold H. Szu and Ralph L. Hartley. 1987. Fast simulated annealing. *Physics Letters A* 122 (June 1987), 157–162.
[37] Ingo Wegener. 2001. Theoretical aspects of evolutionary algorithms. In *ICALP (Lecture Notes in Computer Science)*, Vol. 2076. Springer, 64–78.
[38] Carsten Witt. 2005. Worst-case and average-case approximations by simple randomized search heuristics. In *STACS (Lecture Notes in Computer Science)*, Vol. 3404. Springer, 44–56.
[39] Carsten Witt. 2006. Runtime analysis of the (μ + 1) EA on simple pseudo-Boolean functions. *Evolutionary Computation* 14 (2006), 65–86.
[40] Carsten Witt. 2013. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing* 22 (2013), 294–318.
[41] Xin Yao and Yong Liu. 1997. Fast evolution strategies. In *Evolutionary Programming (Lecture Notes in Computer Science)*, Vol. 1213. Springer, 151–162.
[42] Xin Yao, Yong Liu, and Guangming Lin. 1999. Evolutionary programming made faster. *IEEE Trans. Evolutionary Computation* 3 (1999), 82–102.