Fast Mutation in Crossover-based Algorithms*

Denis Antipov†
ITMO University
St. Petersburg, Russia
and
Laboratoire d'Informatique (LIX),
CNRS, École Polytechnique,
Institut Polytechnique de Paris
Palaiseau, France
antipovden@yandex.ru

Maxim Buzdalov
ITMO University
St. Petersburg, Russia
mbuzdalov@gmail.com

Benjamin Doerr
Laboratoire d'Informatique (LIX),
CNRS, École Polytechnique,
Institut Polytechnique de Paris
Palaiseau, France
doerr@mpi-inf.mpg.de

November 18, 2020

*Extended version of the paper [ABD20] in the proceedings of GECCO. This version contains all proofs and other details that had to be omitted in the conference version for reasons of space. Also, we have greatly expanded the experimental section.

The theoretical research was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH, in a joint call with Gaspard Monge Program for optimization, operations research and their interactions with data sciences. The empirical research was supported by RFBR and CNRS, project number 20-51-15009.

†Corresponding author

Abstract

The heavy-tailed mutation operator proposed in Doerr, Le, Makhmara, and Nguyen (GECCO 2017), called *fast mutation* to agree with the previously used language, so far was proven to be advantageous only in mutation-based algorithms. There, it can relieve the algorithm designer from finding the optimal mutation rate and nevertheless obtain a performance close to the one that the optimal mutation rate gives.

In this first runtime analysis of a crossover-based algorithm using a heavy-tailed choice of the mutation rate, we show an even stronger impact. For the $(1+(\lambda,\lambda))$ genetic algorithm optimizing the OneMax benchmark function, we show that with a heavy-tailed mutation rate a linear runtime can be achieved. This is asymptotically faster than what can be obtained with any static mutation rate, and is asymptotically equivalent to the runtime of the self-adjusting version of the choice of the $(1 + (\lambda,\lambda))$ genetic algorithm. This result is complemented by an empirical study which shows the effectiveness of the fast mutation also on random MAX-3SAT instances.

1 Introduction

It is often cited as a strength of evolutionary algorithms (EAs) that by setting the parameters right the algorithm can be adjusted to the particular problem to be solved. However, it is also known that this process of optimizing the parameters is time-consuming and needs a lot of expert knowledge.

The theoretical research in this field (see, e.g., [AD11, DN20, Jan13, NW10]) has contributed to this challenge via mathematical runtime analyses for general parameter values, which allow to understand the influence of the parameter on the performance and allow to derive optimal parameter values. Examples include (i) the works of Jansen, de Jong, and Wegener [JJW05] as well as Doerr and Künnemann [DK15], which determine the runtime of the $(1 + \lambda)$ EA on OneMax for general value of λ and from this conclude that a linear speed-up with regard to the number of iterations exists only for $\lambda = O\big(\frac{\log(n)\log\log(n)}{\log\log\log(n)}\big)$, (ii) Witt's analysis [Wit06] of the runtime of the $(\mu + 1)$ EA for general values of μ on the LeadingOnes benchmark, which in particular shows that for $\mu = O(\frac{n}{\log n})$ a larger parent population does not lead to an asymptotic slow-down of the algorithm, or (iii) the results of Lehre [Leh10, Leh11] and many follow-up works, which for many non-elitist algorithms determine asymptotically precise thresholds for the selection pressure that separate a highly inefficient regime from one with polynomial runtimes.

Concerning the mutation rate p of the standard bit mutation operator for bit strings of length n, which is our main object of interest in this work, a large number of classic results suggests that a value of $p = \frac{1}{n}$ or close by is a good choice. We note that a mutation rate of $p = \frac{1}{n}$ means that on average a single bit is flipped. The recommendation $p = \frac{1}{n}$ can already be found in [Bäc93, Müh92]. Rigorously proven results show, among others, that only $p = \Theta(\frac{1}{n})$ can give an $O(n \log n)$ runtime of the $(1 + 1)$ EA on OneMax [DJW02], that the asymptotically optimal mutation rate for the $(1+1)$ EA on LeadingOnes is approximately $p = \frac{1.59}{n}$, that $p = (1 \pm o(1))\frac{1}{n}$ is the asymptotically best mutation rate of the $(1 + 1)$ EA for all pseudo-Boolean linear functions [Wit13], that only a mutation rate below $\frac{c}{n}$, where c is a specific constant, guarantees a polynomial runtime of the $(1 + 1)$ EA on all monotonic functions [DJS+13, Len18], and that $(1 \pm o(1))\frac{1}{n}$ is the optimal mutation rate for the $(1 + \lambda)$ EA on OneMax when λ is small [GW17].

In the light of this previous state of the art, it came as a surprise when Doerr, Le, Makhmara, and Nguyen [DLMN17] determined the runtime of the $(1 + 1)$ EA on jump functions for general mutation rates and observed that here much higher mutation rates were optimal[1]. The jump function $\textsc{Jump}_{nk}$ (we deviate here from the notation of [DLMN17]) is a function defined on bit-string of length n which is mostly identical to the easy OneMax function, but which has a valley of low fitness of Hamming width $k-1$ around the global optimum. Consequently, elitist algorithms can leave this local optimum only by flipping k specific bits (and [Doe20a] suggests that non-elitist algorithms cannot do better). As shown in [DLMN17], for this multimodal benchmark function the insights gained previously on unimodal functions like OneMax, linear functions, or LeadingOnes do not apply. The optimal mutation rate for $\textsc{Jump}_{nk}$ was found to be $(1 \pm o(1))\frac{k}{n}$. Deviating from this optimal rate by a small constant factor leads to a runtime increase by a factor of $e^{\Omega(k)}$. Consequently, the choice of the mutation rate for this problem is truly delicate.

To overcome this difficulty, the use of a random mutation rate chosen according to a heavy-tailed distribution, more specifically, a power-law distribution with exponent $\beta > 1$, was suggested. This mutation operator, called *fast mutation* in agreement with previous uses of heavy-tailed distributions in continuous evolutionary computation [SH87, YL97, YLL99], samples a

[1] As a reviewer of [ABD20] pointed out, in [Prü04] an upper bound was shown for the runtime of the $(1 + 1)$ EA with general mutation rate on the hurdle problem with hurdle width 2 and 3. This upper bound is minimized by the mutation rates $\frac{2}{n}$ and $\frac{3}{n}$. This could have been seen earlier as a hint that larger mutation rates can be useful. Since the central research question discussed in [Prü04] was whether crossover is beneficial or not, apparently this detail was overlooked by the broader scientific audience.

random number $\alpha \in [1..\lfloor \frac{n}{2} \rfloor]$ with probability proportional to $\alpha^{-\beta}$ and then flips each bit independently with rate $\frac{\alpha}{n}$. Each application of this operator samples the value of α independently.

The main result in [DLMN17] is that the $(1+1)$ EA with this mutation operator optimizes $\textsc{Jump}_{nk}$ in a time that is only by a factor of $O(k^{\beta-0.5})$ larger than the time resulting from standard bit mutation with the optimal rate. Given that missing the optimal rate (which is only accessible when knowing k) by a small constant factor already incurs a runtime increase by a factor of $e^{\Omega(k)}$, the $O(k^{\beta-0.5})$ price for having a one-size-fits-all mutation operator appears to be a good investment. From the asymptotic point of view β should be taken arbitrarily close to 1, but the experiments conducted in [DLMN17] suggested that $\beta = 1.5$ is a good choice. Both theory and experiments showed that the choice of β is not overly critical. For this reason, it is fair to call fast mutation a parameterless operator.

Since the fast mutation operator is nothing else than a random linear combination of standard bit mutation operators with rates $\frac{\alpha}{n}, \alpha = 1, \dots, \lfloor \frac{n}{2} \rfloor$, it is not surprising that the resulting runtime is higher than the one from the best of these individual operators. Rather, it is surprising that by simply averaging over the available options, one comes relatively close to the optimum, and this in a scenario where for static rates a small deviation from the optimum leads to a significantly increased runtime.

In this work, we observe an even more surprising strength of the fast mutation operator. We investigate how the $(1 + (\lambda, \lambda))$ genetic algorithm ($(1+(\lambda,\lambda))$ GA), first proposed in Doerr, Doerr, and Ebel [DDE15], performs with the fast mutation operator. The $(1 + (\lambda, \lambda))$ GA is an evolutionary algorithm that creates λ offspring from a unique parent individual with an unusually high mutation rate (independently, apart from the fact that they all have the same Hamming distance from the parent), selects the best of these, and creates another λ individuals via a biased crossover between this mutation winner and the original parent. The best of these is taken as the new parent individual if it is at least as good as the previous parent (see Section 2 for more details).

This combination of a high mutation rate and crossover with the parent as repair mechanism allows the algorithm to more efficiently explore the search space when the parameters are chosen suitably. Both from informal considerations and from existing runtime results, the right parameterization seems to be that the mutation rate is $p = \frac{\lambda}{n}$ and the crossover bias, that is, the rate with which the crossover offspring takes bits from the mutation winner, is $c = \frac{1}{\lambda}$. The informal argument for this is that a single application of mutation and crossover generates a bit string distributed as if generated via standard bit mutation with rate $\frac{1}{n}$.

With a number of runtime analyses [DDE15, BD17, DD18, ADK19] supporting this choice[2], we fix this relation of the three parameters in the remainder of this work. Since the mutation rate is the starting point of our research, we can alternatively first choose a mutation rate of type $p = \frac{\alpha}{n}$ and then set $\lambda = pn$ and $c = \frac{1}{pn}$.

The right choice of the mutation rate is non-trivial. The good news from [DDE15] is that any rate between $p = \omega(\frac{1}{n})$ and $p = o(\frac{\log n}{n})$ leads to a runtime of $o(n \log n)$ on OneMax, that is, asymptotically faster than the performance of classic evolutionary algorithms. The optimal mutation rate of

$$p = \Theta\left(\frac{1}{n}\sqrt{\frac{\log(n)\log\log(n)}{\log\log\log(n)}}\right),$$

however, is non-trivial to find [DD18]. It yields an expected runtime on OneMax of

$$E[T] = \Theta\left(n\sqrt{\frac{\log(n)\log\log\log(n)}{\log\log(n)}}\right).$$

Our main research goal in this work is understanding how the $(1 + (\lambda, \lambda))$ GA performs when instead of standard bit mutation with a fixed mutation rate p the fast mutation operator is used. With the previously suggested relations between mutation rate, offspring number, and crossover bias, this means that first a number α is sampled from a power-law distribution, then $\lambda = \alpha$ offspring are generated via flipping ℓ bits chosen uniformly at random, where $\ell \sim \mathrm{Bin}(n, \frac{\alpha}{n})$,[3] and finally λ times a biased crossover with bias $c = \frac{1}{\alpha}$ between parent and mutation winner is performed. We call this modified algorithm the *fast* $(1 + (\lambda, \lambda))$ *GA*.

Our main result is that not only the use of the fast mutation operator in the $(1 + (\lambda, \lambda))$ GA relieves us from finding a good mutation rate, but surprisingly we can even obtain a runtime that is faster than the runtime of the $(1 + (\lambda, \lambda))$ GA with any fixed mutation rate: If the power-law exponent β satisfies $2 < \beta < 3$, then the fast $(1 + (\lambda, \lambda))$ GA has an expected runtime of $O(n)$ on OneMax.

We note that a linear runtime of the $(1 + (\lambda, \lambda))$ GA on OneMax was obtained earlier with a self-adjusting choice of the mutation rate based on the one-fifth rule [DD18]. While this worked well on OneMax, experimental [GP14] and theoretical [BD17] studies on MAX-3SAT instances showed

[2] We note that the work [ADK20] conducted in parallel to ours suggests that a different choice is necessary when large fitness valleys need to be crossed.

[3] This mutation can be interpreted as a standard bit mutation with rate $\frac{\alpha}{n}$, but conditional on having the same number of flipped bits for all individuals.

that this approach carries the risk that the population size λ increases rapidly because the problem structure may just not allow a one-fifth success rate, regardless how large λ is. Since this behavior increases the time complexity of each iteration, it leads to a significant performance loss. Such problems, naturally, cannot arise with the static behavior of the fast mutation operator.

Via an empirical study, we show that the fast mutation operator indeed without any modification also solves well the MAX-3SAT instances for which the one-fifth rule variant of the $(1 + (\lambda, \lambda))$ GA did not perform well in [BD17] (unless enriched with a suitable cap on λ). However, our study also shows that on OneMax itself, the self-adjusting $(1 + (\lambda, \lambda))$ GA is by a constant factor faster than the fast $(1 + (\lambda, \lambda))$ GA. Since the runtime loss from a degenerate behavior of the one-fifth rule version of the $(1+(\lambda, \lambda))$ GA can be large (due to the population size of order n), we draw from these results the recommendation to use the more robust fast $(1 + (\lambda, \lambda))$ GA on a novel problem rather than the self-adjusting $(1 + (\lambda, \lambda))$ GA.

2 Notation and Problem Statement

The $(1 + (\lambda, \lambda))$ GA, first presented in [DDE15], has the following working principles. It stores one current individual x, which is initialized with a random bit string. Each iteration of the $(1+(\lambda, \lambda))$ GA consists of two phases, which are the *mutation* phase and the *crossover* phase. In the mutation phase the algorithm first chooses the *mutation strength* ℓ following the binomial distribution with parameters n and p, where p is usually called the *mutation rate*. It then creates λ mutants by copying the current individual x and flipping exactly ℓ bits which are chosen uniformly at random, independently for each mutant. After that the mutant with the best fitness is chosen as the winner of the mutation phase x' (all ties are broken uniformly at random). In the crossover phase the algorithm λ times performs a crossover between x and x' by taking each bit from x' with probability c and from x otherwise. The probability c is called the *crossover bias.* The best crossover offspring y (all ties are again broken uniformly at random) is compared with the current individual x. If y is not worse, then it replaces x. The main hope behind this algorithm is that with a high mutation rate, the mutation winner x' contains some beneficial solution elements, and that the crossover with the parent acts as repair mechanism that removes the destructions caused by the high mutation rate.

The standard parameter setting proposed in [DDE15] uses the mutation rate $p = \frac{\lambda}{n}$ and the crossover bias $c = \frac{1}{\lambda}$. These parameters guarantee that if the mutation winner contains some beneficial bit (and differs from the parent

by $O(\lambda)$ bits, which is very likely), then with constant probability there is a crossover offspring that has all bits repaired apart from the beneficial one.

When we consider a static parameter setting, the best runtime which the $(1+(\lambda,\lambda))$ GA can reach on the OneMax function is $\Theta\left(n\sqrt{\frac{\log(n)\log\log\log(n)}{\log\log(n)}}\right)$, see [DD18]. Using the fitness-dependent parameter choice $\lambda = \sqrt{\frac{n}{n-f(x)}}$, we can achieve a $\Theta(n)$ runtime [DDE15]. For the static and even more the fitness-dependent setting, one has to question if the typical algorithm user would have found good parameter settings. For this reason, approaches that do not require manually finding a good static or fitness-dependent parameter settings appear preferable.

One such approach in which the value of λ is controlled according to a simple one-fifth rule was proposed in [DD18]. It was proven that this modification lets the $(1 + (\lambda, \lambda))$ GA find the optimum of the OneMax function in $\Theta(n)$ fitness evaluations, which is asymptotically the same as when using the optimal fitness-dependent value of λ.

In this paper we propose to choose λ in each iteration from some heavy-tailed distribution. More precisely, the probability that we choose $\lambda = i$ is

$$\Pr[\lambda = i] = \begin{cases} C_{\beta,u} i^{-\beta}, & \text{if } i \in [1..u], \\ 0, & \text{otherwise,} \end{cases}$$

where $\beta \in \mathbb{R}$ is the power-law exponent of the distribution (which is always considered as a constant), $u \in \mathbb{N}$ is an upper bound on the choice of λ (and may depend on n), and $C_{\beta,u} := (\sum_{i=1}^{u} i^{-\beta})^{-1}$ is the normalization coefficient. All our runtime results on OneMax will hold for the classic choice $u = \lfloor n/2 \rfloor$. We introduce this additional parameter because the Max-SAT analyses in [BD17] showed that sometimes a stricter upper bound on λ is necessary. For that reason, it is interesting to see also in the OneMax analyses how small an upper bound on λ can be taken so that a linear runtime is still obtained.

The detailed pseudocode of the fast $(1 + (\lambda, \lambda))$ GA is shown in Algorithm 1. Our main result will be that this simple way of choosing λ gives us a linear runtime for all $\beta \in (2, 3)$ and $u \geq \ln^{\frac{1}{3-\beta}}(n)$.

2.1 Useful Tools

In this section we collect some classic results which are used in our proofs. First, to be able to make the transition between the number of iterations and the number of fitness evaluations, we use Wald's equation [Wal45].

Algorithm 1: The fast $(1+(\lambda,\lambda))$ GA with power-law exponent β and upper limit u maximizing $f : \{0,1\}^n \to \mathbb{R}$

$x \leftarrow$ random bit string of length n;
while *not terminated* **do**
  Choose λ from $[1..u]$ with $\Pr[\lambda = i] \sim i^{-\beta}$;
  Choose $\ell \sim \text{Bin}\left(n, \frac{\lambda}{n}\right)$;
  for $i \in [1..\lambda]$ **do**
    $x^{(i)} \leftarrow$ a copy of x;
    Flip ℓ bits in $x^{(i)}$ chosen uniformly at random;
  end
  $x' \leftarrow \arg\max_{z \in \{x^{(1)},\dots,x^{(\lambda)}\}} f(z)$;
  for $i \in [1..\lambda]$ **do**
    Create $y^{(i)}$ by taking each bit from x' with probability $\frac{1}{\lambda}$ and from x with probability $\frac{\lambda-1}{\lambda}$;
  end
  $y \leftarrow \arg\max_{z \in \{y^{(1)},\dots,y^{(\lambda)}\}} f(z)$;
  if $f(y) \geq f(x)$ **then**
    $x \leftarrow y$;
  end
end

Lemma 1 (Wald's equation)**.** *Let $(X_t)_{t\in\mathbb{N}}$ be a sequence of real-valued random variables and let T be a positive integer random variable. Let also all following conditions be true.*

1. *All X_n have the same finite expectation.*
2. *For all $t \in \mathbb{N}$ we have $E[X_t \mathbb{1}_{\{T\geq t\}}] = E[X_t] \Pr[T \geq t]$.*
3. *$\sum_{t=1}^{+\infty} E[|X_t| \mathbb{1}_{\{T\geq t\}}] < \infty$.*
4. *$E[T]$ is finite.*

Then we have

$$E\left[\sum_{t=1}^{T} X_t\right] = E[T]E[X_1].$$

We use the following inequality to estimate the probability that at least one of λ Bernoulli trials succeeds.

Lemma 2. *For all $p \in [0,1]$ and all $\lambda > 0$ we have*

$$1 - (1-p)^\lambda \geq \frac{\lambda p}{1+\lambda p}.$$

Proof. By [RS14, Lemma 8] (or (1.4.19) in [Doe20b]) we have $(1-p)^\lambda \leq \frac{1}{1+\lambda p}$. Hence,

$$1 - (1-p)^\lambda \geq 1 - \frac{1}{1+\lambda p} = \frac{\lambda p}{1+\lambda p}. \qquad \square$$

We frequently use the following bounds on the partial sums of the generalized harmonic series.

Lemma 3. *For all $u \geq 1$ and for all $\alpha \neq 1$ we have $\sum_{i=1}^{\lfloor u \rfloor} i^{-\alpha} \geq \frac{u^{1-\alpha}-1}{1-\alpha}$. For $\alpha = 1$ we have $\sum_{i=1}^{\lfloor u \rfloor} i^{-\alpha} \geq \ln(u)$.*

Proof. We estimate the sum for $\alpha \neq 1$ through the corresponding integral.

$$\sum_{i=1}^{\lfloor u \rfloor} i^{-\alpha} \geq \int_1^u x^{-\alpha} dx = \frac{u^{1-\alpha}-1}{1-\alpha}.$$

The case for $\alpha = 1$ is a well-known bound on the partial sum of the harmonic series. $\square$

Lemma 4. *For all $u \in \mathbb{N}$ we have*

- $\sum_{i=1}^{u} i^{-\alpha} \leq u^{1-\alpha}\frac{2-\alpha}{1-\alpha}$, *if* $\alpha < 0$,
- $\sum_{i=1}^{u} i^{-\alpha} \leq \frac{u^{1-\alpha}}{1-\alpha}$, *if* $\alpha \in [0,1)$,
- $\sum_{i=1}^{u} i^{-\alpha} \leq \frac{\alpha}{\alpha-1}$, *if* $\alpha > 1$,
- $\sum_{i=1}^{u} i^{-\alpha} \leq \ln(u) + 1$, *if* $\alpha = 1$.

Proof of Lemma 4. By analogy with Lemma 3 we estimate the sum through a corresponding integral. If $\alpha < 0$ we have

$$\sum_{i=1}^{u} i^{-\alpha} \leq \int_1^u x^{-\alpha} dx + u^{-\alpha} \leq \frac{u^{1-\alpha}-1}{1-\alpha} + u^{-\alpha} \leq u^{1-\alpha}\frac{2-\alpha}{1-\alpha}.$$

If $\alpha \geq 0$ we have

$$\sum_{i=1}^{u} i^{-\alpha} \leq 1 + \int_{2}^{u+1} (x-1)^{-\alpha} dx \leq 1 + \frac{u^{1-\alpha} - 1}{1-\alpha}$$

If $\alpha \in [0, 1)$, then we have

$$\sum_{i=1}^{u} i^{-\alpha} \leq \frac{u^{1-\alpha} - 1 + 1 - \alpha}{1-\alpha} \leq \frac{u^{1-\alpha}}{1-\alpha}.$$

If $\alpha > 1$, we have

$$\sum_{i=1}^{u} i^{-\alpha} \leq 1 + \frac{1}{\alpha - 1} \leq \frac{\alpha}{\alpha - 1}.$$

The case for $\alpha = 1$ is a well-known bound on the partial sum of the harmonic series. □

3 Runtime Analysis

In this section we prove upper and lower bounds on the runtime of the fast $(1 + (\lambda, \lambda))$ GA on OneMax.

3.1 Upper Bound

Our aim in this subsection is to prove an upper bound on the number of fitness evaluations taken until the fast $(1 + (\lambda, \lambda))$ GA finds the optimum of the OneMax benchmark. Since it is technically easier, we first regard the number of iterations until the optimum is found. For algorithms with fixed population sizes, such a bound would immediately imply a bound on the number of fitness evaluations (namely by multiplying the number of iterations with the fixed number of fitness evaluations per iteration). For the fast $(1 + (\lambda, \lambda))$ GA using a newly sampled value of λ in each iteration, things are not that easy, but Wald's equation (Lemma 1) allows to argue that multiplying with the expected number of fitness evaluations per iteration gives the right result.

Before proceeding with proofs, we now state two theorems that together constitute the main result of this subsection. We start by showing that for reasonable parameter values, the optimum is found in a linear number of iterations.

Theorem 5. *If $\beta \in (1,3)$ and $u \geq \ln^{\frac{1}{3-\beta}}(n)$, then the expected number of iterations until the fast $(1+(\lambda,\lambda))$ GA finds the optimum of* OneMax *function is $O(n)$.*

When $\beta > 2$, the expected number of fitness evaluations per iteration is constant (see Lemma 9). With this observation and Wald's equation, we obtain the following estimate for the runtime.

Theorem 6. *If $\beta \in (2,3)$ and $u \geq \ln^{\frac{1}{3-\beta}}(n)$, then the expected number of fitness evaluations until the fast $(1+(\lambda,\lambda))$ GA finds the optimum of* OneMax *function is $O(n)$.*

We start with the proof of Theorem 5. For the readers' convenience we split the proof into Lemmas 7 and 8. The first lemma is essentially an interpretation of Lemma 7 in [DDE15].

Lemma 7. *If $\lambda \leq \sqrt{\frac{n}{d(x)}}$, where $d(x)$ is the current distance between the current individual x and the optimum, then the probability $p_{d(x)}(\lambda)$ of increasing the fitness in one iteration is at least*

$$C\frac{d(x)\lambda^2}{n},$$

where $C > 0$ is an absolute constant. If $\lambda > \sqrt{\frac{n}{d(x)}}$, then this probability is at least C.

Proof. By [DDE15, Lemma 7], the probability of a true progress $p_{d(x)}(\lambda)$ is at least

$$C'\left(1-\left(1-\frac{d(x)}{n}\right)^{\frac{\lambda^2}{2}}\right),$$

where $C' > 0$ is an absolute constant. By Lemma 2 we have

$$p_{d(x)}(\lambda) \geq C'\left(1-\left(1-\frac{d(x)}{n}\right)^{\frac{\lambda^2}{2}}\right) \geq C'\frac{\frac{d(x)\lambda^2}{2n}}{1+\frac{d(x)\lambda^2}{2n}}.$$

If $\lambda \leq \sqrt{\frac{n}{d(x)}}$, then we have $p_{d(x)}(\lambda) \geq C' \frac{d(x)\lambda^2}{3n}$. Note that $C \coloneqq \frac{C'}{3}$ is an absolute constant as well as C'. If $\lambda > \sqrt{\frac{n}{d(x)}}$, then $p_{d(x)}(\lambda) \geq \frac{C'}{3} = C$.

Since Lemma 7 in [DDE15] is formulated for $\lambda \geq 2$ only, we also note that for $\lambda = 1$ the algorithm essentially performs an iteration of the $(1+1)$ EA. Therefore, the probability for a progress in this case is at least $\frac{d(x)}{en}$. □

Lemma 8. *Let $\beta \in (1,3)$. Then the probability $p_{d(x)}$ of having progress in one iteration given that the current distance to the optimum is $d(x)$ is at least*

$$C(\beta)\frac{d(x)U^{3-\beta}}{n},$$

where $U = \min\{u, \sqrt{\frac{n}{d(x)}}\}$ and $C(\beta)$ is some constant independent of n.

Proof. By Lemma 7 we have

$$p_{d(x)} = \sum_{\lambda=1}^{u} C_{\beta,u}\lambda^{-\beta} p_{d(x)}(\lambda) \geq C_{\beta,u} C \sum_{\lambda=1}^{\lfloor U \rfloor} \frac{d(x)\lambda^{2-\beta}}{n} = C_{\beta,u} C \frac{d(x)}{n} \sum_{\lambda=1}^{\lfloor U \rfloor} \lambda^{2-\beta}$$

If $U \geq 2$, then by Lemma 3 we have

$$\sum_{\lambda=1}^{\lfloor U \rfloor} \lambda^{2-\beta} \geq \frac{U^{3-\beta}-1}{3-\beta} \geq \frac{1-2^{\beta-3}}{3-\beta} U^{3-\beta} \geq \frac{3}{8} U^{3-\beta}.$$

Otherwise, when $U < 2$ we have

$$\sum_{\lambda=1}^{\lfloor U \rfloor} \lambda^{2-\beta} \geq 1 = U^{\beta-3} U^{3-\beta} \geq 2^{\beta-3} U^{3-\beta} \geq \frac{1}{4} U^{3-\beta}.$$

Finally, we estimate

$$p_{d(x)} \geq C_{\beta,u} C \frac{d(x)}{n} \sum_{\lambda=1}^{\lfloor U \rfloor} \lambda^{2-\beta} \geq C_{\beta,u} C \frac{1}{4} \frac{d(x)}{n} U^{3-\beta} = C(\beta)\frac{d(x)U^{3-\beta}}{n}$$

with $C(\beta) \coloneqq \frac{1}{4} C_{\beta,u} C$. □

In order to show a full picture we also computed the values of $p_{d(x)}$ for a wider range of parameters u and β. The results are shown in Table 1. We omit the proofs, since they are similar to the proof of Lemma 8.

We are now ready to prove Theorem 5.

Table 1: The probability $p_{d(x)}$ to increase fitness in one iteration for various values of parameters $\beta \in \mathbb{R}$ and $u \in \mathbb{N}$.

β	$u \leq \sqrt{\frac{n}{d(x)}}$	$u > \sqrt{\frac{n}{d(x)}}$
< 1	$\Omega\left(\frac{d(x)u^2}{n}\right)$	$\Omega(1)$
$= 1$	$\Omega\left(\frac{d(x)u^2}{n\log(u)}\right)$	$\Omega\left(\frac{1}{\ln(u)} + \left(1 - \frac{\ln(n/d(x))}{2\ln(u)}\right)\right)$
$(1,3)$	$\Omega\left(\frac{d(x)u^{3-\beta}}{n}\right)$	$\Omega\left(\sqrt{\frac{n}{d(x)}}^{1-\beta}\right)$
$= 3$	$\Omega\left(\frac{d(x)\log(u)}{n}\right)$	$\Omega\left(\frac{\log(n/d(x))}{n/d(x)}\right)$
> 3	$\Omega\left(\frac{d(x)}{n}\right)$	

Proof of Theorem 5. We estimate the upper bound on the expectation of the runtime T_I (in terms of iterations) as the sum of expected times until the algorithm leaves each fitness level. By Lemma 8 we have

$$E[T_I] \leq \sum_{d(x)=1}^{n} \frac{1}{p_{d(x)}} \leq \frac{1}{C(\beta)} \left(\sum_{d(x)=1}^{\lfloor n/u^2 \rfloor} \frac{n}{d(x)u^{3-\beta}} + \sum_{d(x)=\lfloor n/u^2 \rfloor+1}^{n} \sqrt{\frac{n}{d(x)}}^{\beta-1} \right).$$

By Lemma 4 we estimate the first sum

$$\sum_{d(x)=1}^{\lfloor n/u^2 \rfloor} \frac{n}{d(x)u^{3-\beta}} \leq \frac{n\left(\ln\left(\frac{n}{u^2}\right)+1\right)}{u^{3-\beta}} \leq \frac{n(\ln(n)+1)}{\ln(n)} = n(1+o(1)),$$

where in the last inequality we used the assumption $u \geq \ln^{\frac{1}{3-\beta}}(n)$. By Lemma 4 we estimate the second sum as follows.

$$\begin{aligned} \sum_{d(x)=\lfloor n/u^2 \rfloor+1}^{n} \sqrt{\frac{n}{d(x)}}^{\beta-1} &\leq \sum_{d(x)=1}^{n} \sqrt{\frac{n}{d(x)}}^{\beta-1} \\ &\leq n^{\frac{\beta-1}{2}} \sum_{d(x)=1}^{n} d(x)^{-\frac{\beta-1}{2}} \leq n^{\frac{\beta-1}{2}} \frac{n^{\frac{3-\beta}{2}}}{(3-\beta)/2} = O(n). \end{aligned}$$

Therefore, we have

$$E[T_I] \leq \frac{1}{C(\beta)} \left(O(n) + O(n)\right) = O(n). \qquad \square$$

Before we prove Theorem 6 we first estimate $E[\lambda]$, which is half the expected cost of one iteration.

Lemma 9. *If λ is sampled from the heavy-tailed distribution with parameter β and upper limit u, then its expected value is*

- $E[\lambda] = \Theta(1)$, *if* $\beta > 2$,
- $E[\lambda] = \Theta(\log(u))$, *if* $\beta = 2$,
- $E[\lambda] = \Theta(u^{2-\beta})$, *if* $\beta \in (1, 2)$,
- $E[\lambda] = \Theta(\frac{u}{\log(u)})$, *if* $\beta = 1$, *and*
- $E[\lambda] = \Theta(u)$, *if* $\beta < 1$,

where the asymptotic notation is for $u \to +\infty$.

Proof. First recall that $C_{\beta,u} = \sum_{i=1}^{u} i^{-\beta}$. By Lemmas 3 and 4 we have

- if $\beta < 1$, then $C_{\beta,u} = \Theta(u^{1-\beta})$,
- if $\beta = 1$, then $C_{\beta,u} = \Theta(\ln(u))$, and
- if $\beta > 1$, then $C_{\beta,u} = \Theta(1)$.

We compute

$$E[\lambda] = \sum_{i=1}^{u} i \Pr[\lambda = i] = C_{\beta,u} \sum_{i=1}^{u} i^{1-\beta}.$$

If $\beta > 2$, then by Lemma 4 we have

$$C_{\beta,u} \leq E[\lambda] \leq C_{\beta,u} \frac{\beta - 1}{\beta - 2}.$$

Hence, $E[\lambda] = \Theta(1)$.

If $\beta = 2$, then $\sum_{i=1}^{u} i^{1-\beta}$ is a partial sum of the harmonic series, thus it is $\Theta(\log(u))$. If $\beta < 2$, then by Lemmas 3 and 4 we have

$$C_{\beta,u} \frac{u^{2-\beta} - 1}{2 - \beta} \leq E[\lambda] \leq C_{\beta,u} \frac{u^{2-\beta}}{2 - \beta}.$$

Therefore, $E[\lambda] = C_{u,\beta}\Theta(u^{2-\beta})$. Together with the estimates of $C_{\beta,u}$ this proves the lemma for $\beta < 2$. □

We are now in the position to prove Theorem 6

Proof of Theorem 6. Let $\{\lambda_t\}_{t\in\mathbb{N}}$ be a sequence of random variables, each following the power-law distribution with parameters β and u. We can assume that for all $t \in \mathbb{N}$ the fast $(1 + (\lambda, \lambda))$ GA chooses $\lambda := \lambda_t$ in iteration t. Since the cost of one iteration is 2λ fitness evaluations (λ for the mutation phase and λ for the crossover phase), the total number of fitness evaluations T_F has the same distribution as $\sum_{t=1}^{T_I} 2\lambda_t$. We aim at proving that the sequence $(\lambda_t)_{t\in\mathbb{N}}$ and T_I allow to use Wald's equation (Lemma 1). We show that conditions (1)–(4) of this lemma are satisfied.

1. All λ_t have the same expectation, which is finite by Lemma 9.

2. The event $T_I \geq t$ is independent of the outcome of λ_t, which implies that for all $i \in [1..u]$ we have $\Pr[T_I \geq t \mid \lambda_t = i] = \Pr[T_I \geq t]$. Therefore, we have

$$\begin{aligned} E[\lambda_t \mathbb{1}_{\{T_I \geq t\}}] &= \sum_{i=1}^{u} i \Pr[\lambda_t = i] \Pr[T_I \geq t \mid \lambda_t = i] \\ &= \Pr[T_I \geq t] \sum_{i=1}^{u} i \Pr[\lambda_t = i] = \Pr[T_I \geq t] E[\lambda_t]. \end{aligned}$$

3. By the previous condition we have

$$\sum_{t=1}^{+\infty} E[|\lambda_t| \cdot \mathbb{1}_{\{T_I \geq t\}}] = \sum_{t=1}^{+\infty} \Pr[T_I \geq t] E[\lambda_t] = E[\lambda] E[T_I],$$

   since for all $t \in \mathbb{N}$ we have $E[\lambda_t] = E[\lambda]$. By Theorem 5 and Lemma 9, both $E[\lambda]$ and $E[T_I]$ are finite, hence their product is finite as well.

4. By Theorem 5 $E[T_I]$ is finite.

Thus, by Wald's inequality we have

$$E[T_F] = E[T_I] E[2\lambda_t].$$

By Theorem 5 and Lemma 9 we conclude

$$E[T_F] = O(n) \cdot \Theta(1) = O(n). \qquad \square$$

Although we are mostly interested in $\beta \in (2, 3)$ and reasonably high upper limit u, a reader might find it interesting to see the upper bounds for the runtimes yielded by different parameters values.

Table 2: Upper bounds on the expected number of iterations and expected number of fitness evaluations for different values of β and u. The last column is calculated by Wald's equation in the same manner as in Theorem 6.

β	$E[T_I]$	$E[T_F] = 2E[T_I]E[\lambda]$
< 1	$O(n)$ if $u \geq \sqrt{\ln(n)}$ $O\left(\frac{n}{u^2}\log\frac{n}{u^2}\right)$ if $u \leq \sqrt{\ln(n)}$	$O(nu^{2-\beta})$ if $u \geq \sqrt{\ln(n)}$ $O\left(u^{-\beta}n\log\frac{n}{u^2}\right)$ if $u \leq \sqrt{\ln(n)}$
$= 1$	$O(n\log(u))$ if $u \geq \sqrt{\ln(n)}$ $O\left(\frac{n}{u^2}\log\left(\frac{n}{u^2}\right)\log(u)\right)$ if $u \leq \sqrt{\ln(n)}$	$O(nu\log(u))$ if $u \geq \sqrt{\ln(n)}$ $O\left(\frac{n}{u}\log\left(\frac{n}{u^2}\right)\log(u)\right)$ if $u \leq \sqrt{\ln(n)}$
$(1,2)$	$O(n)$ if $u \geq \ln^{\frac{1}{3-\beta}}(n)$ $O\left(\frac{n}{u^{3-\beta}}\log\left(\frac{n}{u^2}\right)\right)$ if $u < \ln^{\frac{1}{3-\beta}}(n)$	$O(nu^{2-\beta})$ if $u \geq \ln^{\frac{1}{3-\beta}}(n)$ $O\left(\frac{n}{u}\log\left(\frac{n}{u^2}\right)\right)$ if $u < \ln^{\frac{1}{3-\beta}}(n)$
$= 2$		$O(n\log(u))$ if $u \geq \ln^{\frac{1}{3-\beta}}(n)$ $O\left(\frac{n\log(u)}{u^{3-\beta}}\log\left(\frac{n}{u^2}\right)\right)$ if $u < \ln^{\frac{1}{3-\beta}}(n)$
$(2,3)$		$O(n)$ if $u \geq \ln^{\frac{1}{3-\beta}}(n)$ $O\left(\frac{n}{u^{3-\beta}}\log\left(\frac{n}{u^2}\right)\right)$ if $u < \ln^{\frac{1}{3-\beta}}(n)$
$= 3$	$O(n\log\log(u))$ if $u \geq n^{\frac{1}{\ln\ln(n)}}$ $O\left(\frac{n}{\log(u)}\log\left(\frac{n}{u^2}\right)\right)$ if $u < n^{\frac{1}{\ln\ln(n)}}$	$O(n\log\log(u))$ if $u \geq n^{\frac{1}{\ln\ln(n)}}$ $O\left(\frac{n}{\log(u)}\log\left(\frac{n}{u^2}\right)\right)$ if $u < n^{\frac{1}{\ln\ln(n)}}$
> 3	$O(n\log(n))$	$O(n\log(n))$

For this reason we show the estimates for $E[T_I]$ and $E[T_F]$ for a wider range of parameters values in Table 2. We omit the proofs, since they generally imitate the proofs of Theorems 5 and 6.

In the proofs of Theorems 5 and 6 we aimed at delivering only asymptotical upper bounds disregarding the leading constant in order not to reduce the readability of the paper. However, for the complete picture, without proof we estimate the leading constant delivered by our arguments.

Recall that $C(\beta) = \frac{1}{12}C_{\beta,u}C'$. From the proof of Lemma 7 in [DDE15] we can show that C' which is used in Lemma 7 is at least $\frac{1}{e}(1 - \exp(-\exp(-\frac{3}{2}))) \approx 0.0735$. For any upper bound $u = \omega(1)$ we also have $C_{\beta,u} \approx \beta - 1$. Hence, we estimate the upper bound on the leading constant.

$$\frac{1}{C(\beta)}\left(1 + \frac{2}{3-\beta}\right) \approx \frac{12(5-\beta)}{(3-\beta)(\beta-1)C'} \approx 164\frac{(5-\beta)}{(3-\beta)(\beta-1)}.$$

Taking into account the leading constant hidden in Lemma 9, which is $\frac{\beta-1}{\beta-2}$ if $\beta > 2$, we estimate the upper bound on the leading constant for $E[T_F]$ delivered by Theorem 6 as

$$328 \cdot \frac{(5-\beta)}{(3-\beta)(\beta-2)}. \tag{1}$$

3.2 Lower Bound

In this section we prove the tightness of our upper bounds by showing a lower bound of $\Omega(n)$ fitness evaluations for the runtime of the fast $(1+(\lambda,\lambda))$ GA on OneMax. This is a special case of a deeper result [TG06], which showed the same lower bound for all comparison-based algorithms (which the $(1+(\lambda,\lambda))$ GA is). For the readers' convenience, we give an elementary proof as well.

Theorem 10. *The expected runtime of the fast $(1+(\lambda,\lambda))$ GA with parameter $\beta \in \mathbb{R}$ and any upper limit $u \in \mathbb{N}$ on the* OneMax *function is at least $\Omega(\frac{n}{E[\lambda]})$ iterations, where $E[\lambda]$ is estimated as in Lemma 9, and $\Omega(n)$ fitness evaluations.*

Proof. The progress in one iteration cannot be greater than the number ℓ of bits which we flip in each mutant, since we cannot obtain more than ℓ new one-bits in the winner x' of the mutation phase. Therefore, after we have sampled λ, the expected progress is

$$E[f(y) - f(x) \mid \lambda] \leq E[\ell \mid \lambda] = \lambda.$$

The expected progress in one iteration thus is

$$E[f(y) - f(x)] = \sum_{i=1}^{u} \Pr[\lambda = i] E[f(y) - f(x) \mid \lambda = i] \leq E[\lambda].$$

Let x_0 be the initial individual. Since it is chosen uniformly at random, its expected fitness is $E[f(x_0)] = \frac{n}{2}$. Hence, by the additive drift theorem [HY01] the expectation of the number of iterations T_I before the algorithm finds the optimum is at least

$$E[T_I] \geq \frac{n - E[f(x_0)]}{E[\lambda]} = \frac{n}{2E[\lambda]}.$$

Now we can use Wald's equation as we did in the proof of Theorem 6. We obtain

$$E[T_F] = E[T_I]E[2\lambda] \geq \frac{n}{2E[\lambda]} \cdot 2E[\lambda] = n. \qquad \square$$

4 Experiments

Our theoretical findings show that the fast $(1+(\lambda,\lambda))$ GA with the natural choice $\beta \in (2,3)$ has a linear runtime on OneMax, which matches the

performance of the self-adjusting $(1 + (\lambda, \lambda))$ GA. Due to their asymptotic nature, our results cannot indicate which of the two linear-time algorithms is faster, how the fast $(1 + (\lambda, \lambda))$ GA compares with other algorithms on reasonable problem sizes, and how its performance depends on $\beta \in (2, 3)$. For the latter, the only estimate we have from theory, eq. (1), provides a very large upper bound on the constant factor, which could suggest that $\beta = 2.5 + \varepsilon$ may be better than $\beta = 2.5 - \varepsilon$ for $0 < \varepsilon < 0.5$, but without a matching lower bound this is speculative. To answer these questions, but also to investigate the performance on a slightly less artificial problem, we performed a series of experiments.

As algorithms, we regarded randomized local search (RLS) and the $(1 + 1)$ EA with a standard bit mutation as well as the self-adjusting $(1 + (\lambda, \lambda))$ GA, which controls λ (and thus $p = \lambda/n$ and $c = 1/\lambda$) via the one-fifth success rule [DD18].

We have also considered the version of the $(1 + (\lambda, \lambda))$ GA with the one-fifth rule with an upper limit of $2 \ln(n + 1)$ on the value of λ, introduced in [BD17], since it showed a much better performance on the MAX-3SAT problem than without this upper limit. For the same reason, we also consider the fast $(1 + (\lambda, \lambda))$ GA with the same upper limit of $2 \ln(n + 1)$ on the value of λ, which is imposed by setting the distribution parameter u to $u = 2 \ln(n + 1))$. To investigate the effect of varying u further, we also conduct a series of experiments with a fixed problem size n and different values of u.

For the fast $(1 + (\lambda, \lambda))$ GA, we used the values of $\beta \in \{2.1, 2.3, 2.5, 2.7, 2.9\}$ unless noted otherwise. In all the adaptive versions of the $(1 + (\lambda, \lambda))$ GA, the initial value of λ is set to 1.

The source code used to perform these experiments is a part of a larger project dedicated to the $(1 + (\lambda, \lambda))$ GA available on GitHub[4].

4.1 Implementation Details and Their Discussion

In all runs we used slightly modified versions of the algorithms to avoid counting obviously unnecessary fitness evaluations. The particular changes are as follows.

- In the (1+1) EA, if standard bit mutation flips zero bits, then we resample the offspring until it is different from the parent. This is equivalent to not counting the fitness evaluation of the offspring identical to the parent.

[4] `https://github.com/mbuzdalov/generic-onell`

- In all versions of the $(1+(\lambda,\lambda))$ GA, we resample ℓ until $\ell \neq 0$. This is equivalent to not counting the fitness evaluations in iterations with $\ell = 0$ because here all offspring are identical to the parent. In the crossover phase, samples taking all bits from the parent x are repeated (without evaluating the fitness of the copy of the parent) and samples taking all bits from the mutation winner x' are not evaluated (that is, do not count towards the number of fitness evaluations). Additionally, x' also participates in the selection of the best among x and the crossover results $y^{(i)}$. When there is a tie, then the crossover winner has a higher priority than x'.

We consider these natural modifications instead of the original algorithms in this section, since we are sure that anyone implementing these algorithms for solving practical problems would do the same. For a practitioner it does not make sense to waste fitness evaluations on individuals which are identical to their parents, while in theoretical works these are often counted since constant factors are often ignored. We note that similar modifications of algorithms were called *practice-aware* in [PD18]. We note that there are much more ways to tune the runtime of the $(1+(\lambda,\lambda))$ GA in a practical application, see, e.g., [GP14]. In contrast to the modifications described above, for these it is not clear to what extent they are useful in general or only for particular problems. For this reason, we did not consider them in this work.

Clearly our theoretical results from Section 3 apply to these mildly modified algorithms. For the upper bounds it is enough to note that by resampling identical individuals and by having x' participate in the selection, the probability to have a progress in one iteration only increases. Thus, repeating the arguments from Theorem 5 we obtain the same upper bound on the expected number of iterations. Since our implementation does not affect the choice of λ, its expected value $E[\lambda]$ stays the same. The cost of one iteration is at most 2λ (but can be smaller). Thus, by Wald's equation we obtain the same upper bound on the expected number of fitness evaluations as in Theorem 6. For the lower bound we use the same arguments as in Theorem 10, with the only change that since we cannot choose $\ell = 0$, we have

$$E[\ell \mid \lambda] = \frac{\lambda}{1-\left(1-\frac{1}{\lambda}\right)^{\lambda}} \leq \frac{\lambda}{1-\frac{1}{e}},$$

which still gives us a lower bound of $\Omega(n)$ fitness evaluations.

4.2 Experimental Setup

The experiments were performed on the OneMax function and on random satisfiable instances of the MAX-3SAT problem, that is, the problem of maximizing the number of satisfied clauses in a Boolean formula represented in conjunctive normal form. The second problem was chosen for two reasons. First, it is a more practical problem than OneMax, second, there are already theoretical and empirical results for the $(1 + (\lambda, \lambda))$ GA on this function (see [BD17]). For this problem on n variables, the number of clauses was chosen to be $4n \ln n$. An all-ones bit string is assumed to be a planted optimal solution; this is without loss of generality, as all considered algorithms are unbiased. For each clause, three participating variables and their signs (i.e., whether it is negated or not) are sampled uniformly and independently until this clause is satisfied by the planted solution (that is, not all three variables are negated). Note that these are easy instances of the MAX-3SAT problem, so the presented results on this problem should not be considered as if the proposed algorithms are competitive in solving this problem in general. However, these instances have a lower fitness-distance correlation, which makes them harder in particular for the $(1 + (\lambda, \lambda))$ GA.

To speed-up the experiments, we used the incremental fitness evaluation technique, which is more commonly seen in gray-box optimization and in problem-aware solvers. We note that this led only to a faster implementation of the algorithm, not to a different algorithm behavior. In particular, the number of iterations or fitness evaluations performed are not affected. We modified the implementation as follows.

During mutation we do not copy the parent individual, but instead directly generate the bit indices which are different in the parent and the offspring (the "patch"). Following that, we evaluate the fitness of the offspring based on the fitness of the parent and the patch. For RLS and the (1+1) EA, if the new fitness is at least as good as the one of the parent, we apply the patch to the parent, turning it into the offspring. For the $(1 + (\lambda, \lambda))$ GA, we select the best patch out of all the mutants' patches (based on their fitness values). The subsequent applications of crossover translate to subsamplings of that patch, so that fitness evaluation is again based on the parent's fitness.

For OneMax, evaluation of the offspring's fitness based on the parent's fitness and the patch is rather straightforward: only the bits at the affected indices are checked. This results in an expected $O(1)$ amount of work per each iteration of both RLS and the $(1 + 1)$ EA, and in the $\Theta(\lambda^2)$ amount of work for the $(1 + (\lambda, \lambda))$ GA, which still helps much because λ is typically much smaller than n.

For MAX-3SAT, the incremental evaluation is more difficult as it involves some preprocessing on the side of the fitness function. It amounts to constructing lists of clauses affected by the changed bits and to evaluating the satisfaction status of these clauses before and after the change. For the logarithmic density of clauses employed in this paper, this amounts to $\Theta(\log n)$ expected work per iteration of RLS and the $(1+1)$ EA, and to $\Theta(\lambda^2 \log n)$ expected work for the $(1+(\lambda,\lambda))$ GA, which is still faster than direct evaluation, but less efficient than what is possible for OneMax.

We also note that the particular structure of all the considered algorithms also allows to optimize the memory requirements: the memory used by RLS and the $(1+1)$ EA is $\Theta(n)$ words resulting from storing a single bit vector, whereas the $(1+(\lambda,\lambda))$ GA uses $\Theta(n+\lambda)$ words in expectation, as only the best patches for each of the phases need to be stored.

In our experiments we chose the problem sizes n to be powers of two, so that the asymptotic behavior of the algorithms is easier to investigate visually. For OneMax, we limit the problem size to 2^{22}, and for MAX-3SAT, the upper limit is 2^{16}. These sizes were derived from the affordable computational times. We did not reach the size of 2^{20} on MAX-3SAT as in [BD17], because the incremental fitness evaluations have a weaker impact with fast mutation. Indeed, whenever λ is sampled from a heavy-tailed distribution, the distribution of λ^2, and hence of the wall-clock running time, has an even heavier tail, so occasional high values of λ result in very expensive iterations. For each algorithm, each problem setting, and each problem size, 100 independent runs were performed. For the MAX-3SAT problem, a new random instance was created for each run.

Our runtime results are shown in Figures 1-4. In Figures 1-3 the x-axis indicates the problem size in a logarithmic scale, and the y-axis indicates the ratio of the runtime to the problem size. In this visualization a linear runtime results in a horizontal plot and any runtime in $\Theta(n \log n)$ gives a linearly increasing plot.

4.3 Runtimes on OneMax

In Figure 1 we show the results of the runs on the OneMax function. If we do not consider $\beta = 2.1$, which turns out to be too small (and therefore gives a too large expected value of λ), then all versions of the fast $(1+(\lambda,\lambda))$ GA start outperforming the $(1+1)$ EA already at population size $n = 2^{10}$ and then outperform RLS at $n = 2^{20}$ or earlier. Recalling the discussion after the proof of Theorem 5 we note that our estimate of the leading constant in the runtime was overly pessimistic, otherwise we would have no chance to outperform RLS on these problem sizes.

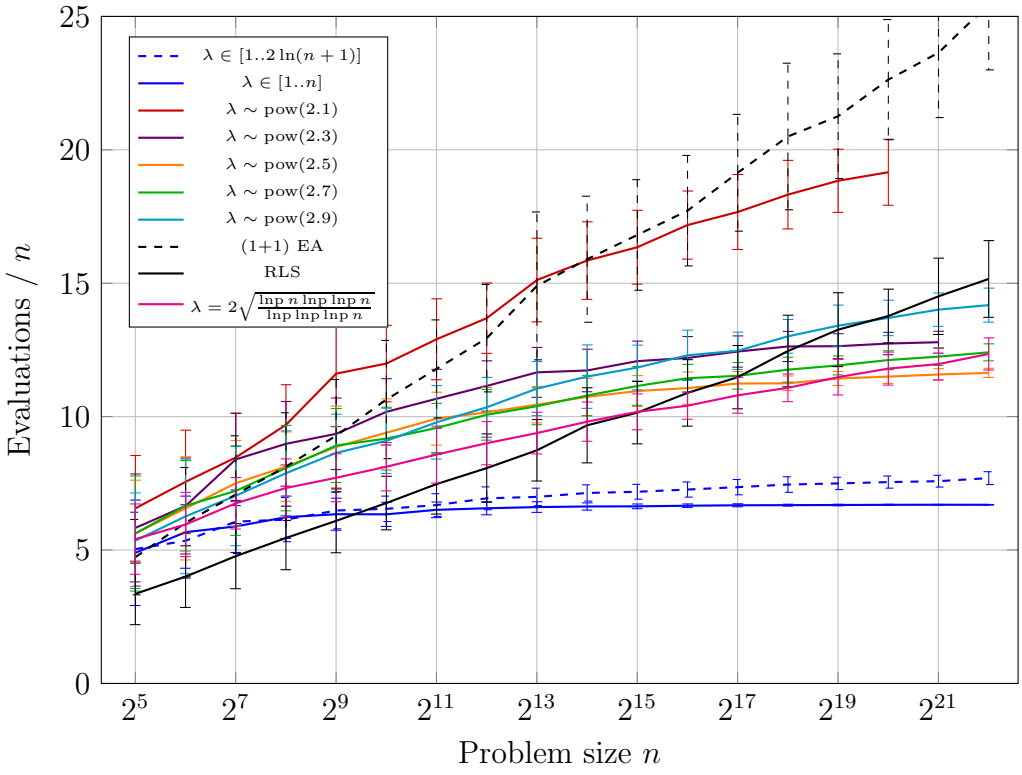

Figure 1: Mean runtimes and their standard deviation of different algorithms on OneMax benchmark problem. By $\lambda \in [1..u]$ we denote the self-adjusting parameter choice via the one-fifth rule in the interval $[1..u]$. The indicated confidence interval for each value X is $[E[X] - \sigma(X), E[x] + \sigma(X)]$, where $\sigma(X)$ is the standard deviation of X. We write $\operatorname{lnp} x := \ln(x+1)$.

The one-fifth rule shows a much better performance and yields a runtime of the $(1 + (\lambda, \lambda))$ GA which is very close to linear already from $n = 2^{10}$ on for both linear and logarithmic upper bounds on λ. The plots for the heavy-tailed choice of λ do not look horizontal, but they show a strongly marked tendency that they will do so at larger population sizes. The runtimes for all β except $\beta = 2.1$ are quite well concentrated, as well as the runtimes of the $(1 + (\lambda, \lambda))$ GA with the one-fifth rule, in contrast to the runtimes of the $(1 + 1)$ EA and RLS. We have no results for $\beta = 2.1$ for population sizes $n \geq 2^{21}$ and for $\beta = 2.3$ for $n \geq 2^{22}$, since they were too expensive (in terms of computational resources) and most likely not too insightful.

Figure 1 also features the runtime plot of an asymptotically optimal static choice for λ. It has been proven in [DD18] that the theoretically asymptotically optimal static choice is $\lambda = \Theta(\sqrt{\frac{\ln(n)\ln\ln(n)}{\ln\ln\ln(n)}})$. By using

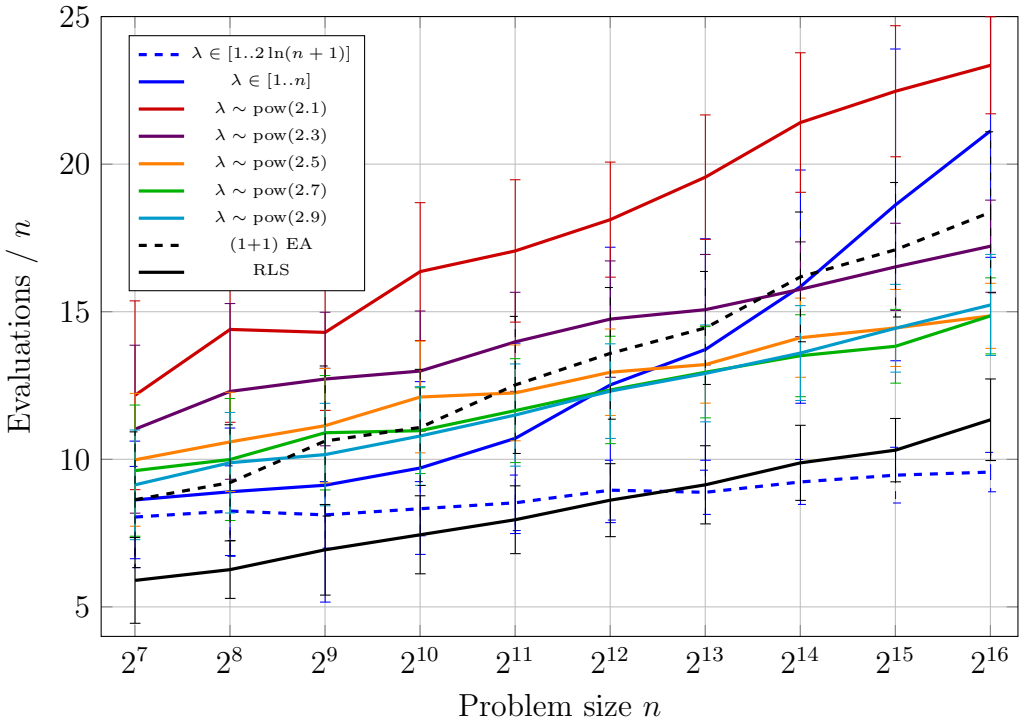

Figure 2: Mean runtimes and their standard deviation of different algorithms on MAX-3SAT instances with $4n\ln(n)$ clauses. By $\lambda \in [1..u]$ we denote the self-adjusting parameter choice via the one-fifth rule in the interval $[1..u]$. The indicated confidence interval for each value X is $[E[X]-\sigma(X), E[x]+\sigma(X)]$, where $\sigma(X)$ is the standard deviation of X.

$\mathrm{lnp}(n) := \ln(n+1)$ instead to avoid issues with logarithms of too small values, and by fitting the outer constant factor using auxiliary experiments with fixed $\lambda \in [2..12]$, we have found that $\lambda = 2\sqrt{\frac{\mathrm{lnp}(n)\,\mathrm{lnp}\,\mathrm{lnp}(n)}{\mathrm{lnp}\,\mathrm{lnp}\,\mathrm{lnp}(n)}}$ approximates the optimal choices quite well, so we have used the version of the $(1+(\lambda,\lambda))$ GA with this choice in our plots. We also see that with the choice of $\beta = 2.5$ the fast $(1+(\lambda,\lambda))$ GA outperforms the statically optimal parameter choice at problem sizes $n \geq 2^{20}$.

4.4 Runtimes on MAX-3SAT

Figure 2 shows the results of the experiments on the MAX-3SAT problem. As previously shown in [BD17], large values of λ can be harmful. For this reason, the $(1+(\lambda,\lambda))$ GA with the unbounded one-fifth rule is outperformed already by the simple (1+1) EA. The authors of [BD17] proposed to limit the

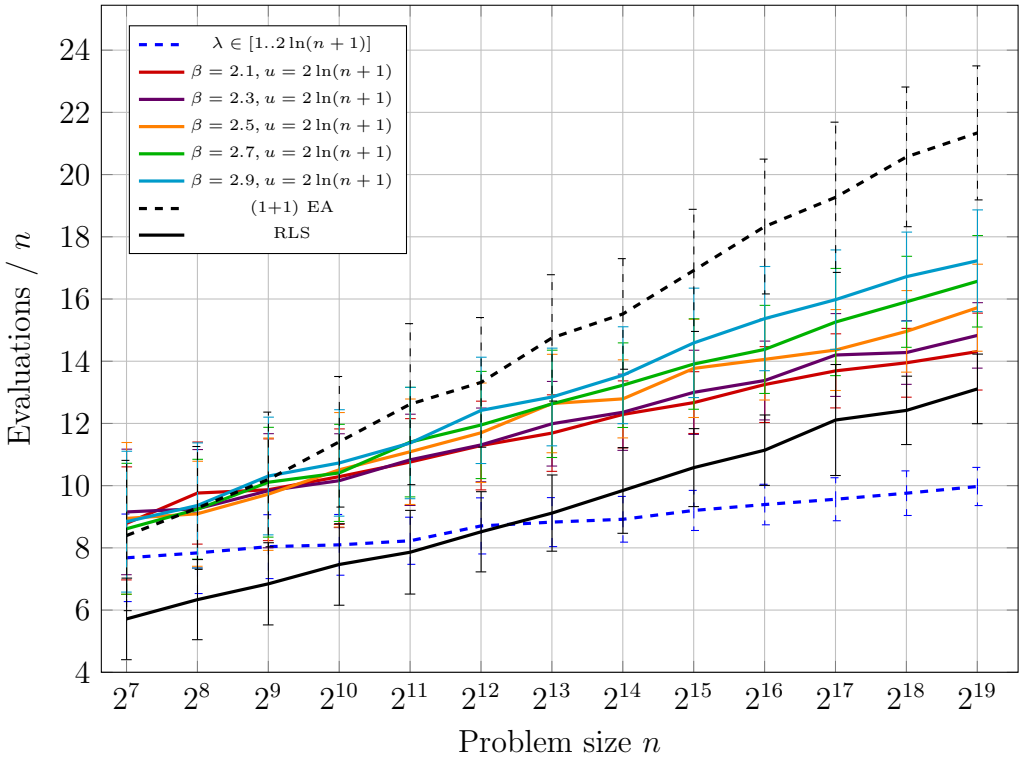

Figure 3: Mean runtimes and their standard deviation of different algorithms on MAX-3SAT instances with $4n \ln(n)$ clauses with logarithmically capped population sizes. By $\lambda \in [1..u]$ we denote the self-adjusting parameter choice via the one-fifth rule in the interval $[1..u]$. The indicated confidence interval for each value X is $[E[X] - \sigma(X), E[x] + \sigma(X)]$, where $\sigma(X)$ is the standard deviation of X.

value which λ can take by $2\ln(n+1)$, which greatly improved the performance up to the point that RLS was outperformed on this problem.

As we see in Figure 2, the fast $(1 + (\lambda, \lambda))$ GA is quite efficient even without an upper limit on λ. Except for the case $\beta = 2.1$, we managed to outperform the $(1+1)$ EA and the self-adjusting $(1+(\lambda,\lambda))$ GA without an upper limit on λ. Nevertheless, RLS and the self-adjusting $(1 + (\lambda, \lambda))$ GA with a logarithmic cap on λ remained faster.

The runtimes of all algorithms appear super-linear in the plots, which agrees with the impression given from [BD17].

4.5 Effects of Capping for MAX-3SAT

Since apparently large values of λ are not helpful when optimizing MAX-3SAT instances (due to the weaker fitness-distance correlation), we conducted some experiments with the fast $(1 + (\lambda, \lambda))$ GA choosing λ from a power-law distribution on a smaller range $[1..u]$ of values. Based on the previous experience, we started with an upper limit of $u = 2\ln(n + 1)$. These results are presented in Figure 3.

Using this upper limit reduced the computational burden associated with heavy-tailed distributions and allowed us to regard problem sizes up to 2^{19}. The upper limit also led a better performance in terms of fitness evaluations. When comparing Figure 2 and Figure 3 around the problem size $n = 2^{16}$, we see that for $\beta \in \{2.1, 2.3\}$ a significant speed-up was obtained, whereas for $2.5 \leq \beta \leq 2.9$ the differences of the corresponding mean running times are negligible. This is not surprising given that for smaller values of β, the inefficient high values of λ are sampled more often. Interestingly, in combination with the upper limit small values of β gave the best performance. This suggests that it is important to use moderately large values of λ often and that only too large values lead to efficiency losses.

To investigate the effect of the particular choice of the upper limit u on the running time for various values of β, we performed additional experiments where the problem size was fixed to $n = 2^{16}$, but the upper limits were varying. Figure 4 presents these results, where u was taken from the set $u \in \{2^2, 2^3, \ldots, 2^{13}\}$. Note that high values of u again prevented us from choosing a higher problem size. We also plot for reference the performance of the $(1 + 1)$ EA on the same problem size.

The plots in Figure 4 indicate that for $2.1 \leq \beta \leq 2.5$ the dependency on the upper limit has a clear optimal value: Too small values of u prevent the $(1 + (\lambda, \lambda))$ GA from choosing the more efficient mid-size values of λ, too high values of u lead to sampling too large values of λ too often, which have little chance of making progress and at the same time are very costly. It can be seen, however, that already for $\beta = 2.5$ the subsequent increase of the running time is not too pronounced. Higher values of β tend to a monotonic behavior, up to the deviations from the mean running time. This basically indicates that the sensitivity to the upper limit of the distribution is not large even in practice.

4.6 Summary of Experimental Results

Summing up, from the results of the experiments we conclude the following three points.

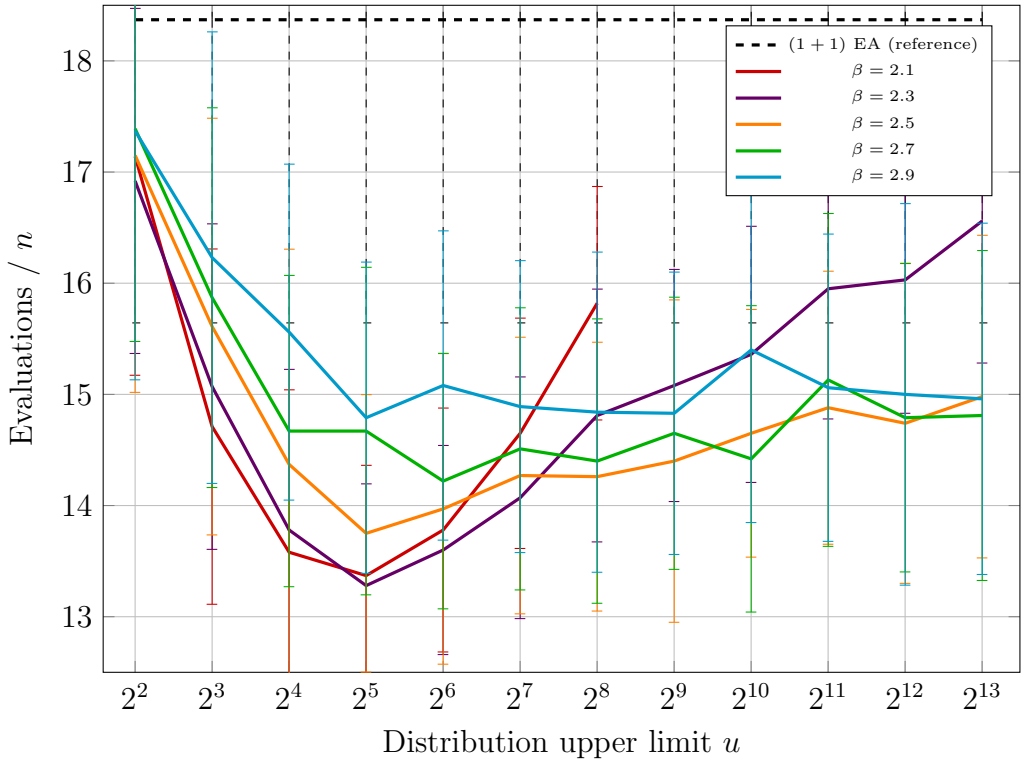

Figure 4: Mean runtimes and their standard deviation of different algorithms on MAX-3SAT instances with $4n\ln(n)$ clauses for different capping values. Problem size is $n = 2^{16}$. The indicated confidence interval for each value X is $[E[X] - \sigma(X), E[x] + \sigma(X)]$, where $\sigma(X)$ is the standard deviation of X.

- The fast $(1 + (\lambda, \lambda))$ GA performs generally well, often beating the classic mutation based algorithms. On OneMax, the self-adjusting $(1+(\lambda, \lambda))$ GA both without and with an upper limit of $u = 2\ln(n+1)$ are superior, on MAX-3SAT only the version with upper limit and RLS are superior.

- The fast $(1+(\lambda, \lambda))$ GA can easily be used as a parameterless algorithm and this is what we suggest. We note that the $(1+(\lambda, \lambda))$ GA with the asymptotically optimal static parameter setting could not beat the fast $(1+(\lambda, \lambda))$ GA on OneMax. The self-adjusting $(1+(\lambda, \lambda))$ GA without an upper limit was superior on OneMax, but significantly inferior on MAX-3SAT. The version with upper limit $u = 2\ln(n + 1)$ was superior on both OneMax and MAX-3SAT. We still do not want to advertise this approach as clearly such limits are problem-specific and non-trivial to find. The logarithmic limit for MAX-3SAT is based on

a substantial mathematical analysis [BD17] of these particular MAX-3SAT instances. For other problems, such a limit may be detrimental, e.g., it may be hard to leave a local optimum with a large basin of attraction.

- The choice of β does not play a big role as long as it is not too close to the borders of the interval $(2, 3)$. Taking β between 2.5 and 2.7 might be a good general recommendation.

5 Conclusion

In this first runtime analysis of a crossover-based algorithm using the fast mutation operator, we observed that the fast mutation operator not only can relieve the algorithm designer from the task of choosing a suitable mutation rate, but it can also lead to runtimes asymptotically better than any static choice of the mutation rate.

Different from previous works, where any power-law exponent greater than one could be used, our work requires that β is between 2 and 3. We note, however, that the power-law distributions are often used with exponents in the open interval $(2, 3)$ and this for good reason. In this regime, we have a heavy tail (as opposed for $\beta > 3$), but we still have a constant expectation (as opposed to $\beta < 2$). Since the complexity of a single iteration is $\Theta(\lambda)$, having a constant expectation $E[\lambda]$ is very natural.

On the technical side, our work shows that algorithms with a heavy-tailed number of offspring can be much easier to analyze than those with a self-adjusting number of offspring (such as the self-adjusting $(1 + (\lambda, \lambda))$ GA [DD18]), since Wald's equation allows to estimate the expected runtime as the product of the expected number of iterations and the expected number of offspring generated in one iteration.

The natural question arising from this work is for which other algorithms and problems such a speed-up can be obtained. Natural candidates are other crossover-based algorithms or algorithms in which dynamic parameter choices could obtain a speed-up over static choices. We note that after this research was conducted, it was found that the $(1+(\lambda, \lambda))$ GA with two of its parameters chosen independently from heavy-tailed distributions has a good performance on jump functions [AD20]. The performance is slightly inferior to the one with optimal static parameters [ADK20], however these were non-trivial to find as they deviated significantly from the previous recommendations.

References

[ABD20] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. Fast mutation in crossover-based algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1268–1276. ACM, 2020.

[AD11] Anne Auger and Benjamin Doerr, editors. *Theory of Randomized Search Heuristics*. World Scientific Publishing, 2011.

[AD20] Denis Antipov and Benjamin Doerr. Runtime analysis of a heavy-tailed $(1+(\lambda,\lambda))$ genetic algorithm on jump functions. In *Parallel Problem Solving From Nature, PPSN 2020, Part II*, pages 545–559. Springer, 2020.

[ADK19] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. A tight runtime analysis for the $(1 + (\lambda, \lambda))$ GA on LeadingOnes. In *Foundations of Genetic Algorithms, FOGA 2019*, pages 169–182. ACM, 2019.

[ADK20] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. The $(1 + (\lambda, \lambda))$ GA is even faster on multimodal problems. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1259–1267. ACM, 2020.

[Bäc93] Thomas Bäck. Optimal mutation rates in genetic search. In *International Conference on Genetic Algorithms, ICGA 1993*, pages 2–8. Morgan Kaufmann, 1993.

[BD17] Maxim Buzdalov and Benjamin Doerr. Runtime analysis of the $(1 + (\lambda, \lambda))$ genetic algorithm on random satisfiable 3-CNF formulas. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 1343–1350. ACM, 2017.

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

[DDE15] Benjamin Doerr, Carola Doerr, and Franziska Ebel. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87–104, 2015.

[DJS+13] Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. Mutation rate matters even when optimizing monotone functions. *Evolutionary Computation*, 21:1–21, 2013.

[DJW02] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[DK15] Benjamin Doerr and Marvin Künnemann. Optimizing linear functions with the $(1 + \lambda)$ evolutionary algorithm—different asymptotic runtimes for different instances. *Theoretical Computer Science*, 561:3–23, 2015.

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

[DN20] Benjamin Doerr and Frank Neumann, editors. *Theory of Evolutionary Computation—Recent Developments in Discrete Optimization*. Springer, 2020. Also available at `https://cs.adelaide.edu.au/~frank/papers/TheoryBook2019-selfarchived.p`

[Doe20a] Benjamin Doerr. Does comma selection help to cope with local optima? In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1304–1313. ACM, 2020.

[Doe20b] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 1–87. Springer, 2020. Also available at `https://arxiv.org/abs/1801.06733`.

[GP14] Brian W. Goldman and William F. Punch. Parameter-less population pyramid. In *Genetic and Evolutionary Computation Conference, GECCO 2014*, pages 785–792. ACM, 2014.

[GW17] Christian Gießen and Carsten Witt. The interplay of population size and mutation probability in the $(1 + \lambda)$ EA on OneMax. *Algorithmica*, 78:587–609, 2017.

[HY01] Jun He and Xin Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127:51–81, 2001.

[Jan13] Thomas Jansen. *Analyzing Evolutionary Algorithms – The Computer Science Perspective*. Springer, 2013.

[JJW05] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13:413–440, 2005.

[Leh10] Per Kristian Lehre. Negative drift in populations. In *Parallel Problem Solving from Nature, PPSN 2010*, pages 244–253. Springer, 2010.

[Leh11] Per Kristian Lehre. Fitness-levels for non-elitist populations. In *Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 2075–2082. ACM, 2011.

[Len18] Johannes Lengler. A general dichotomy of evolutionary algorithms on monotone functions. In *Parallel Problem Solving from Nature, PPSN 2018, Part II*, pages 3–15. Springer, 2018.

[Müh92] Heinz Mühlenbein. How genetic algorithms really work: mutation and hillclimbing. In *Parallel Problem Solving from Nature, PPSN 1992*, pages 15–26. Elsevier, 1992.

[NW10] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer, 2010.

[PD18] Eduardo Carvalho Pinto and Carola Doerr. Towards a more practice-aware runtime analysis of evolutionary algorithms. *CoRR*, abs/1812.00493, 2018.

[Prü04] Adam Prügel-Bennett. When a genetic algorithm outperforms hill-climbing. *Theoretical Computer Science*, 320:135–153, 2004.

[RS14] Jonathan E. Rowe and Dirk Sudholt. The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm. *Theoretical Computer Science*, 545:20–38, 2014.

[SH87] Harold H. Szu and Ralph L. Hartley. Fast simulated annealing. *Physics Letters A*, 122:157–162, 1987.

[TG06] Olivier Teytaud and Sylvain Gelly. General lower bounds for evolutionary algorithms. In *Parallel Problem Solving from Nature, PPSN 2006*, pages 21–31. Springer, 2006.

[Wal45] Abraham Wald. Some generalizations of the theory of cumulative sums of random variables. *The Annals of Mathematical Statistics*, 16:287–293, 1945.

[Wit06] Carsten Witt. Runtime analysis of the (μ + 1) EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14:65–86, 2006.

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

[YL97] Xin Yao and Yong Liu. Fast evolution strategies. In *Evolutionary Programming*, volume 1213 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 1997.

[YLL99] Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3:82–102, 1999.