Lazy Parameter Tuning and Control

Antipov, Denis; Buzdalov, Maxim; Doerr, Benjamin

Published in:
Algorithmica

DOI:
10.1007/s00453-023-01098-z

Publication date:
2024

Citation for published version (APA):
Antipov, D., Buzdalov, M., & Doerr, B. (2024). Lazy Parameter Tuning and Control: Choosing All Parameters Randomly from a Power-Law Distribution. *Algorithmica*, *86*(2), 442-484. https://doi.org/10.1007/s00453-023-01098-z

Algorithmica
https://doi.org/10.1007/s00453-023-01098-z

Lazy Parameter Tuning and Control: Choosing All Parameters Randomly from a Power-Law Distribution

Denis Antipov[1] · **Maxim Buzdalov**[2,3] · **Benjamin Doerr**[4]

Received: 25 October 2021 / Accepted: 14 January 2023

Abstract
Most evolutionary algorithms have multiple parameters and their values drastically affect the performance. Due to the often complicated interplay of the parameters, setting these values right for a particular problem (parameter tuning) is a challenging task. This task becomes even more complicated when the optimal parameter values change significantly during the run of the algorithm since then a dynamic parameter choice (parameter control) is necessary. In this work, we propose a lazy but effective solution, namely choosing all parameter values (where this makes sense) in each iteration randomly from a suitably scaled power-law distribution. To demonstrate the effectiveness of this approach, we perform runtime analyses of the $(1+(\lambda,\lambda))$ genetic algorithm with all three parameters chosen in this manner. We show that this algorithm on the one hand can imitate simple hill-climbers like the $(1+1)$ EA, giving the same asymptotic runtime on problems like OneMax, LeadingOnes, or Minimum Spanning Tree. On the other hand, this algorithm is also very efficient on jump functions, where the best static parameters are very different from those necessary to optimize simple problems. We prove a performance guarantee that is comparable to the best performance known for static parameters. For the most interesting case that the jump size k is constant, we prove that our performance is asymptotically better than what can be

✉ Denis Antipov
antipovden@yandex.ru

Maxim Buzdalov
mbuzdalov@gmail.com

Benjamin Doerr
lastname@lix.polytechnique.fr

1 School of Computer Science, The University of Adelaide, Adelaide, Australia

2 ITMO University, St. Petersburg, Russia

3 Department of Computer Science, Aberystwyth University, Aberystwyth, United Kingdom

4 Laboratoire d'Informatique (LIX), CNRS, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Published online: 22 February 2023

obtained with any static parameter choice. We complement our theoretical results with a rigorous empirical study confirming what the asymptotic runtime results suggest.

Keywords Runtime analysis · Dynamic parameter choice · Crossover · Theory

1 Introduction

Evolutionary algorithms (EAs) are general-purpose randomized search heuristics. They are adapted to the particular problem to be solved by choosing suitable values for their parameters. This flexibility is a great strength on the one hand, but a true challenge for the algorithm designer on the other. Missing the right parameter values can lead to catastrophic performance losses.

Despite being a core topic of both theoretical and experimental research, general advice on how to set the parameters of an EA are still rare. The difficulty stems from the fact that different problems need different parameters, different instances of the same problem may need different parameters, and even during the optimization process on one instance the most profitable parameter values may change over time.

In an attempt to design a simple one-size-fits-all solution, Doerr et al. [22] proposed to use random parameter values chosen independently in each iteration from a power-law distribution (note that random mutation rates were used before [15, 16], but with different distributions and for different reasons). Mostly via mathematical means, this was shown to be highly effective for the choice of the mutation rate of the $(1 + 1)$ EA when optimizing the jump benchmark, which has the property that the optimal mutation rate depends strongly on the problem instance. More precisely, for a jump function with representation length n and jump size $2 \leq k = o(\sqrt{n})$, the standard mutation rate $p = 1/n$ gives an expected runtime of $(1 + o(1))en^k$, where $e \approx 2.718$ is Euler's number. The asymptotically optimal mutation rate $p = k/n$ leads to a runtime of $(1 + o(1))n^k(e/k)^k$. Deviating from the optimal rate by a small constant factor increases the runtime by a factor exponential in k. When using the mutation rate α/n, where $\alpha \in [1..n/2]$ is sampled independently in each iteration from a power-law distribution with exponent $\beta > 1$, the runtime becomes $\Theta(k^{\beta-0.5}n^k(e/k)^k)$, where the constants hidden by the asymptotic notation are independent of n and k. Consequently, apart from the small polynomial factor $\Theta(k^{\beta-0.5})$, this randomized mutation rate gives the performance of the optimal mutation rate and in particular also achieves the super-exponential runtime improvement by a factor of $(e/k)^{\Theta(k)}$ over the standard rate $1/n$.

The idea of choosing parameter values randomly according to a power-law distribution was quickly taken up by other works. In [32, 39], variants of the heavy-tailed mutation operator were proposed and analyzed on TwoMax, Jump, MaxCut, and several sub-modular problems. In [27, 30, 49], power-law mutation in multi-objective optimization was studied. In [12], the authors compared power-law mutation and artificial immune systems. In [2], heavy-tailed mutation was regarded for the $(1 + (\lambda, \lambda))$ GA), however again only for a single parameter and this parameter being the mutation rate. Very recently, the first analysis of a heavy-tailed choice of a parameter of the selection operator was conducted [17].

While optimizing a single parameter is already non-trivial (and the latest work [2] showed that the heavy-tailed mutation rate can even give results better than any static mutation rate, that is, it can inherit advantages of dynamic parameter choices), the really difficult problem is finding good values for several parameters of an algorithm. Here the often intricate interplay between the different parameters can be a true challenge (see, e.g., [23] for a theory-based determination of the optimal values of three parameters).

The only attempt to choose randomly more than one parameter was made in [3] for the $(1 + (\lambda, \lambda))$ GA having a three-dimensional parameter space spanned by the parameters population size λ, mutation rate p, and crossover bias c. For this algorithm, first proposed in [14], the product $d = pcn$ of mutation rate, crossover bias, and representation length describes the expected distance of an offspring from the parent. It was argued heuristically in [3] that a reasonable parameter setting should have $p = c$, that is, the same mutation rate and crossover bias. With this reduction of the parameter space to two dimensions, the parameter choice in [3] was made as follows. Independently (and independently in each iteration), both λ and d were chosen from a power-law distribution. Mutation rate and crossover bias were both set to $\sqrt{d/n}$ to ensure $p = c$ and $pcn = d$. When using unbounded power-law distributions with exponents $\beta_\lambda = 2 + \varepsilon$ and $\beta_d = 1 + \varepsilon'$ with $\varepsilon, \varepsilon' > 0$ any small constants, this randomized way of setting the parameters gave an expected runtime of $e^{O(k)}(\frac{n}{k})^{(1+\varepsilon)k/2}$ on jump functions with jump size $k \geq 3$. This is very similar (slightly better for $k < \frac{1}{\varepsilon}$, slightly worse for $k > \frac{1}{\varepsilon}$) to the runtime of $(\frac{n}{k})^{(k+1)/2}e^{O(k)}$ obtainable with the optimal static parameters. This is a surprisingly good performance for a parameterless approach, in particular, when compared to the runtime of $\Theta(n^k)$ of many classic evolutionary algorithms. Note that both for the static and dynamic parameters only upper bounds were proven,[1] hence we cannot make a rigorous conclusion on which algorithm performs better on jump. The proofs of these upper bounds however suggest to us that they are tight.

Our Results: While the work [3] showed that in principle it can be profitable to choose more than one parameter randomly from a power-law distribution, it relied on the heuristic assumption that one should take the mutation rate equal to the crossover bias. There is nothing wrong with using such heuristic insight, however, one has to question if an algorithm user (different from the original developers of the $(1 + (\lambda, \lambda))$ GA) would have easily found this relation $p = c$.

In this work, we show that such heuristic preparations are not necessary: One can simply choose all three parameters of the $(1 + (\lambda, \lambda))$ GA from (scaled) power-law distributions and obtain a runtime comparable to the ones seen before. More precisely, when using the power-law exponents $2 + \varepsilon$ for the distribution of the population size and $1+\varepsilon'$ for the distributions of the parameters p and c and scaling the distributions for p and c by dividing by $\sqrt{n}$ (to obtain a constant distance of parent and offspring with constant probability), we obtain the same $e^{O(k)}(\frac{n}{k})^{(1+\varepsilon)k/2}$ runtime guarantee as in [3]. From our theoretical results one can see that the exact choice of ε' does not affect the asymptotical runtime neither on easy functions such as OneMax, nor on hard functions

[1] A lower bound of $(\frac{n}{k})^{k/2}e^{\Theta(k)}$ fitness evaluations on the runtime of the $(1 + (\lambda, \lambda))$ GA with static parameters was shown in [6], but this bound was proven for the initialization in the local optimum of Jump_k and it does not include the runtime until the algorithm gets to the local optimum from a random solution.

such as JUMP$_k$. Hence if an algorithm user would choose all exponents as $2+\varepsilon$, which is a natural choice as it leads to a constant expectation and a super-constant variance as usually desired from a power-law distribution, the resulting runtimes would still be $O(n \log n)$ for ONEMAX and $e^{O(k)}(\frac{n}{k})^{(1+\varepsilon)k/2}$ for jump functions with gap size k.

With this approach, the only remaining design choice is the scaling of the distributions. It is clear that this cannot be completely avoided simply because of the different scales of the parameters (mutation rates are in [0, 1], population sizes are positive integers). However, we argue that here very simple heuristic arguments can be employed. For the population size, being a positive integer, we simply use a power-law distribution on the non-negative integers. For the mutation rate and the crossover bias, we definitely need some scaling as both number have to be in [0, 1]. Recalling that (and this is visible right from the algorithm definition) the expected distance of offspring from their parents in this algorithm is $d = pcn$ and recalling further the general recommendation that EAs should generate offspring with constant Hamming distance from the parent with reasonable probability (this is, for example, implicit both in the general recommendation to use a mutation rate of $1/n$ and in the heavy-tailed mutation operator proposed in [22]), a scaling leading to a constant expected value of d appears to be a good choice. We obtain this by taking both p and c from power-law distributions on the positive integers scaled down by a factor of $\sqrt{n}$. This appears again to be the most natural choice. We note that if an algorithm user would miss this scaling and scale down both p and c by a factor of n (e.g., to obtain an expected constant number of bits flipped in the mutation step), then our runtime estimates would increase by a factor of $n^{\frac{\beta_p+\beta_c}{2}-1}$, which is still not much compared to the roughly $n^{k/2}$ runtimes we have and the $\Theta(n^k)$ runtimes of many simple evolutionary algorithms.

Our precise result is a mathematical runtime analysis of this heavy-tailed algorithm for arbitrary parameters of the three heavy-tailed distributions (power-law exponent and upper bound on the range of positive integers it can take, including the case of no bound) on a set of "easy" problems (ONEMAX, LEADINGONES, the minimum spanning tree and the partition problem) and on JUMP function. We show that on easy problems the heavy-tailed $(1 + (\lambda, \lambda))$ GA asymptotically is not worse than the $(1+1)$ EA, and on JUMP it significantly outperforms the $(1+1)$ EA for a wide range of the parameters of power-law distributions. These results show that the absolutely best performance can be obtained by guessing correctly suitable upper bounds on the ranges. Since guessing these parameters wrong can lead to significant performance losses, whereas the gains from these optimal parameter values are not so high, we would rather advertise our parameter-less "layman recommendation" to use unrestricted ranges and power-law exponents slightly more than two for the population size and slightly more than one for other parameters. These recommendations are supported by the empirical study shown in Sect. 4.

Our work also provides an example where a dynamic (here simply randomized) parameter choice provably gives an asymptotic runtime improvement. This improvement is significantly more pronounced than the $o(\sqrt{\log n})$ factor speed-up observed in [2, 13] for the optimization of ONEMAX via the $(1 + (\lambda, \lambda))$ GA.

We note that our situation is different, e.g., from the optimization of jump functions via the $(1+1)$ EA. Here the mutation rate $\frac{k}{n}$ is asymptotically optimal [22] for JUMP$_k$.

Clearly, for the easy OneMax-type part of the optimization process, the mutation rate $\frac{1}{n}$ would be superior, but the damage from using the larger rate $\frac{k}{n}$ only leads to a lower-order increase of the runtime.

We prove that this is different for the optimization of the jump functions via the $(1 + (\lambda, \lambda))$ GA. Since this effect is already visible for constant values of k, and in fact strongest visible, to ease the presentation, we assume that k is constant. We note that only for constant k the different variants of the $(1 + (\lambda, \lambda))$ GA had a polynomial runtime, so clearly, k constant (and not too large) is the most interesting case.

For constant k, our result is $e^{O(k)}(\frac{n}{k})^{(1+\varepsilon)k/2}$. The best runtime that could be obtained with a static mutation rate was $e^{O(k)}n^{(k+1)/2}k^{-k/2}$. Hence by choosing ε sufficiently small, our upper bound is asymptotically smaller than the best upper bound for static parameters. Unfortunately, no lower bounds were proven in [6] for static parameters. To rigorously support our claim that dynamic parameter choices can asymptotically outperform static ones when optimizing jump functions via the $(1 + (\lambda, \lambda))$ GA, in Sect. 3.3 we prove such a lower bound. Since this is not the main topic of this work, we shall not go as far as proving that the upper bound for static parameters is tight, but we content ourselves with a simpler proof of a weaker lower bound, which however suffices to support our claim of the superiority of dynamic parameter choices.

In summary, our results demonstrate that choosing all parameters of an algorithm randomly according to a simple (scaled) power-law can be a good way to overcome the problem of choosing appropriate fixed or dynamic parameter values. We are optimistic that this approach will lead to a good performance also for other algorithms and other optimization problems.

2 Preliminaries

In this section we collect definitions and tools which we use in the paper. To avoid misreading of our results, we note that we use the following notation. By $\mathbb{N}$ we denote the set of all positive integer numbers and by $\mathbb{N}_0$ we denote the set of all non-negative integer numbers. We write $[a..b]$ to denote an integer interval including its borders and $[a, b]$ to denote a real-valued interval including its borders. For any probability distribution $\mathcal{L}$ and random variable X, we write $X \sim \mathcal{L}$ to indicate that X follows the law $\mathcal{L}$. We denote the binomial law with parameters $n \in \mathbb{N}$ and $p \in [0, 1]$ by $\mathrm{Bin}(n, p)$. We denote the geometric distribution taking values in $\{1, 2, \dots\}$ with success probability $p \in [0, 1]$ by $\mathrm{Geom}(p)$. We denote by T_I and T_F the number of iterations and the number of fitness evaluations performed until some event holds (which is always specified in the text).

2.1 Objective Functions

In this paper we consider five benchmark functions and problems, namely OneMax, LeadingOnes, the minimum spanning tree problem, the partition problem and Jump_k.

All of them are pseudo-Boolean functions, that is, they are defined on the set of bit strings of length n and return a real number.

OneMax returns the number of one-bits in its argument, that is, $\textsc{OneMax}(x) = \textsc{OM}(x) = \sum_{i=1}^{n} x_i$. It is one of the most intensively studied benchmarks in evolutionary computation. Many evolutionary algorithms can find the optimum of OneMax in time $O(n \log n)$ [4, 34, 37, 47]. The $(1 + (\lambda, \lambda))$ GA with a fitness-dependent or self-adjusting choice of the population size [13, 14] or with a heavy-tailed random choice of the population size [1] is capable of solving OneMax in linear time when the other two parameters are chosen suitably depending on the population size.

LeadingOnes returns the number of the leading ones in a bit string. In more formal words we maximize function

$$\textsc{LeadingOnes}(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j.$$

The runtime of the most classic EAs is at least quadratic on LeadingOnes. More precisely, the runtime of the $(1+1)$ EA is $\Theta(n^2)$ [20, 41], the runtime of the $(\mu+1)$ EA is $\Theta(n^2 + \mu n \log(n))$ [47], the runtime of the $(1 + \lambda)$ EA is $\Theta(n^2 + \lambda n)$ [34] and the runtime of the $(\mu + \lambda)$ EA is $\Omega(n^2 + \frac{\lambda n}{\max\{1, \log(\lambda/n)\}})$ [9]. It was shown in [5] that the $(1 + (\lambda, \lambda))$ GA with standard parameters ($\lambda \in [1..\frac{n}{2}]$, $p = \frac{\lambda}{n}$ and $c = \frac{1}{\lambda}$) also has a $\Theta(n^2)$ runtime on LeadingOnes.

In the **minimum spanning tree problem** (MST for brevity) we are given an undirected graph $G = (V, E)$ with positive integer edge weights defined by a weight function $\omega : E \to \mathbb{N}_{\geq 1}$. We assume that this graph does not have parallel edges or loops. The aim is to find a connected subgraph of a minimum weight. By n we denote the number of vertices, by m we denote the number of edges in G.

This problem can be solved by minimizing the following fitness function defined on all subgraphs $G' = (V, E')$ of the given graph G.

$$f(G') = (W_{\text{total}} + 1)^2 cc(G') + (W_{\text{total}} + 1)|E'| + \sum_{e \in E'} \omega(e),$$

where $cc(G')$ is the number of connected components in G' and W_{total} is the total weight of the graph G, that is, the sum of all edge weights. This definition of the fitness guarantees that any connected graph has a better (in this case, smaller) fitness than any unconnected graph and any tree has a better fitness than any graph with cycles.

The natural representation for subgraphs used in [38] is via bit-strings of length m, where each bit corresponds to some particular edge in graph G. An edge is present in subgraph G' if and only if its corresponding bit is equal to one. In [38] it was shown that the $(1 + 1)$ EA solves the MST problem with the mentioned representation and fitness function in expected number of $O(m^2 \log(W_{\text{total}}))$ iterations.

In the **partition problem** we have a set of n objects with positive integer weights $w_1, w_2, \ldots, w_n$ and our aim is to split the objects into two sets (usually called *bins*) such that the total weight of the heavier bin is minimal. Without loss of generality

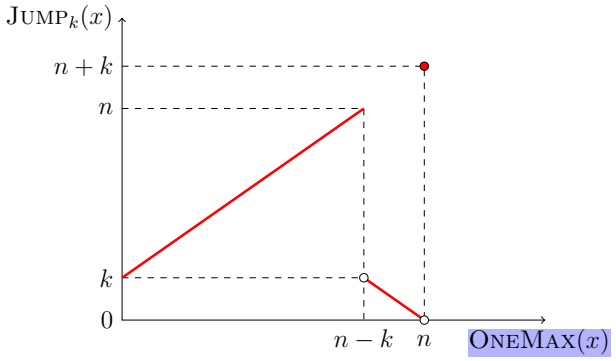

Fig. 1 Plot of the JUMP$_k$ function. As a function of unitation, the function value of a search point x depends only on the number of one-bits in x

we assume that the weights are sorted in a non-increasing order, that is, $w_1 \geq w_2 \geq \cdots \geq w_n$. By w we denote the total weight of all objects, that is, $w = \sum_{i=1}^{n} w_i$. By a $(1+\delta)$ approximation (for any $\delta > 0$) we mean a solution in which the weight of the heavier bin is at most by a factor of $(1+\delta)$ greater than in an optimal solution.

Each partition into two bins can be represented by a bit string of length n, where each bit corresponds to some particular object. The object is put into the first bin if and only if the corresponding bit is equal to one. As fitness $f(x)$ of an individual x we consider the total weight of the objects in the heavier bin in the partition which corresponds to x.

In [46] it was shown that the (1+1) EA finds a $(\frac{4}{3}+\varepsilon)$ approximation for any constant $\varepsilon > 0$ of any partition problem in linear time and that it finds a $\frac{4}{3}$ approximation in time $O(n^2)$.

The function JUMP$_k$ (where k is a positive integer parameter) is defined via ONEMAX as follows.

$$\text{JUMP}_k(x) = \begin{cases} \text{OM}(x) + k, & \text{if OM}(x) \in [0..n-k] \cup \{n\}, \\ n - \text{OM}(x), & \text{if OM}(x) \in [n-k+1..n-1]. \end{cases}$$

A plot of JUMP$_k$ is shown in Fig. 1. The main feature of JUMP$_k$ is a set of local optima at distance k from the global optimum and a valley of extremely low fitness in between. Most EAs optimizing JUMP$_k$ first reach the local optima and then have to perform a jump to the global one, which turns out to be a challenging task for most classic algorithms. In particular, for all values of μ and λ it was shown that $(\mu+\lambda)$ EA and (μ, λ) EA have a runtime of $\Omega(n^k)$ fitness evaluations when they optimize JUMP$_k$ [20, 26]. Using a mutation rate of $\frac{k}{n}$ [22], choosing it from a power-law distribution [22], or setting it dynamically with a stagnation detection mechanism [28, 42–44] reduces the runtime of the $(1+1)$ EA by a $k^{\Theta(k)}$ factor, however, for constant k the runtime of the $(1+1)$ EA remains $\Theta(n^k)$. Many crossover-based algorithms have a better runtime on JUMP$_k$, see [18, 19, 31, 35, 40, 50] for results on algorithms different from the $(1+(\lambda,\lambda))$ GA. Those beating the $\tilde{O}(n^{k-1})$ runtime shown in [19] may appear somewhat artificial and overfitted to the precise definition of the jump function, see

[48]. Outside the world of genetic algorithms, the estimation-of-distribution algorithm cGA and the ant-colony optimizer 2-MMAS$_{ib}$ were shown to optimize jump functions with small $k = O(\log n)$ in time $O(n \log n)$ [7, 25, 33]. Runtime analyses for artificial immune systems, hyperheuristics, and the Metropolis algorithm exist [10, 11, 36], but their runtime guarantees are asymptotically weaker than $O(n^k)$ for constant k.

2.2 Power-Law Distributions

We say that an integer random variable X follows a power-law distribution with parameters β and u if

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

Here $C_{\beta,u} = (\sum_{j=1}^{u} j^{-\beta})^{-1}$ denotes the normalization coefficient. We write $X \sim \text{pow}(\beta, u)$ and call u the bounding of X and β the power-law exponent.

The main feature of this distribution is that while having a decent probability to sample $X = \Theta(1)$ (where the asymptotic notation is used for $u \to +\infty$), we also have a good (inverse-polynomial instead of negative-exponential) probability to sample a super-constant value. The following lemmas show the well-known properties of the power-law distributions. Their proofs can be found, for example, in [3].

Lemma 1 (Lemma 1 in [3]) *For all positive integers a and b such that $b \geq a$ and for all $\beta > 0$, the sum $\sum_{i=a}^{b} i^{-\beta}$ is*

- $\Theta((b+1)^{1-\beta} - a^{1-\beta})$, *if* $\beta < 1$,
- $\Theta(\log(\frac{b+1}{a}))$, *if* $\beta = 1$, *and*
- $\Theta(a^{1-\beta} - (b+1)^{1-\beta})$, *if* $\beta > 1$,

where Θ notation is used for $b \to +\infty$.

Lemma 2 (Lemma 2 in [3]) *The normalization coefficient $C_{\beta,u} = (\sum_{j=1}^{u} j^{-\beta})^{-1}$ of the power-law distribution with parameters β and u is*

- $\Theta(u^{\beta-1})$, *if* $\beta < 1$,
- $\Theta(\frac{1}{\log(u)+1})$, *if* $\beta = 1$, *and*
- $\Theta(1)$, *if* $\beta > 1$,

where Θ notation is used for $u \to +\infty$.

Lemma 3 (Lemma 3 in [3]) *The expected value of the random variable $X \sim \text{pow}(\beta, u)$ is*

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

where Θ *notation is used for* $u \to +\infty$.

Lemma 4 *If* $X \sim \text{pow}(\beta, u)$, *then* $E[X^2]$ *is*

- $\Theta(u^2)$, *if* $\beta < 1$,
- $\Theta(\frac{u^2}{\log(u)+1})$, *if* $\beta = 1$,
- $\Theta(u^{3-\beta})$, *if* $\beta \in (1, 3)$,
- $\Theta(\log(u) + 1)$, *if* $\beta = 3$, *and*
- $\Theta(1)$, *if* $\beta > 3$,

where Θ *notation is used for* $u \to +\infty$.

Lemma 4 simply follows from Lemma 1.

2.3 The Heavy-Tailed $(1 + (\lambda, \lambda))$ GA

We now define the heavy-tailed $(1 + (\lambda, \lambda))$ GA. The main difference from the standard $(1 + (\lambda, \lambda))$ GA is that at the start of each iteration the mutation rate p, the crossover bias c, and the population size λ are randomly chosen as follows. We sample $p \sim n^{-1/2} \text{pow}(\beta_p, u_p)$ and $c \sim n^{-1/2} \text{pow}(\beta_c, u_c)$. The population size is chosen via $\lambda \sim \text{pow}(\beta_\lambda, u_\lambda)$. Here the upper limits u_λ, u_p and u_c can be any positive integers, except we require u_p and u_c to be at most $\sqrt{n}$ (so that we choose both p and c from interval $(0, 1]$). The power-law exponents β_λ, β_p and β_c can be any non-negative real numbers. We call these parameters of the power-law distribution the *hyperparameters of the heavy-tailed* $(1 + (\lambda, \lambda))$ GA and we give recommendations on how to choose these hyperparameters in Sect. 3.3. The pseudocode of this algorithm is shown in Algorithm 1. We note that it is not necessary to store the whole offspring population, since only the best individual has a chance to be selected as a mutation or crossover winner. Hence also large values for λ are algorithmically feasible.

Concerning the scalings of the power-law distributions, we find it natural to choose the integer parameter λ from a power-law distribution without any normalization. For the scalings of the power-law determining the parameters p and c, we argued already in the introduction that the scaling factor of $n^{-1/2}$ is natural as it ensures that the Hamming distance between parent and offspring, which is pcn for this algorithm, is one with constant probability. We see that there is some risk that an algorithm user misses this argument and, for example, chooses a scaling factor of n^{-1} for the mutation rate, which leads to the Hamming distance between parent and mutation offspring being one with constant probability. A completely different alternative would be to choose $c \sim \frac{1}{\text{pow}(\beta_m, u_m)}$, inspired by the recommendation "$c := 1/(pn)$" made for static parameters in [14]. Without proof, we note that these and many similar strategies increase the runtime by at most a factor of $\Theta(n^c)$, c a constant independent of n and k, thus not changing the general $n^{(0.5+\varepsilon)k}$ runtime guarantee proven in this work.

The following theoretical results exist for the $(1 + (\lambda, \lambda))$ GA. With optimal static parameters the algorithm solves OneMax in approximately $O(n\sqrt{\log(n)})$ fitness evaluations [14]. The runtime becomes slightly worse on the random satisfiability instances due to a weaker fitness-distance correlation [8]. In [5] it was shown that the runtime of the $(1 + (\lambda, \lambda))$ GA on LeadingOnes is the same as the runtime of the most classic

Algorithm 1: The heavy-tailed $(1 + (\lambda, \lambda))$ GA maximizing a pseudo-Boolean function $f : \{0, 1\}^n \to \mathbb{R}$.

$x \leftarrow$ random bit string of length n;
while *not terminated* **do**
  Choose $p \sim n^{-1/2}\,\mathrm{pow}(\beta_p, u_p)$;
  Choose $c \sim n^{-1/2}\,\mathrm{pow}(\beta_c, u_c)$;
  Choose $\lambda \sim \mathrm{pow}(\beta_\lambda, u_\lambda)$;
  Mutation phase:
  Choose $\ell \sim \mathrm{Bin}(n, p)$;
  for $i \in [1..\lambda]$ **do**
    $x^{(i)} \leftarrow$ a copy of x;
    Flip ℓ bits in $x^{(i)}$ chosen uniformly at random;
  end
  $x' \leftarrow \arg\max_{z \in \{x^{(1)}, \ldots, x^{(\lambda)}\}} f(z)$;
  Crossover phase:
  for $i \in [1..\lambda]$ **do**
    Create $y^{(i)}$ by taking each bit from x' with probability c and from x with probability $(1 - c)$;
  end
  $y \leftarrow \arg\max_{z \in \{y^{(1)}, \ldots, y^{(\lambda)}\}} f(z)$;
  if $f(y) \geq f(x)$ **then**
    $x \leftarrow y$;
  end
end

algorithms, that is, $\Theta(n^2)$, which means that it is not slower than most other EAs despite the absence of a strong fitness-distance correlation. The analysis of the $(1 + (\lambda, \lambda))$ GA with static parameters on $\textsc{Jump}_k$ in [6] showed that the $(1 + (\lambda, \lambda))$ GA (with uncommon parameters) can find the optimum in $e^{O(k)}(\frac{n}{k})^{(k+1)/2}$ fitness evaluations, which is roughly a square root of the $\Theta(n^k)$ runtime of many classic algorithms on this function.

Concerning dynamic parameter choices, a fitness-dependent parameter choice was shown to give linear runtime on OneMax [14], which is the best known runtime for crossover-based algorithms on OneMax. In [13], it was shown that also the self-adjusting approach of controlling the parameters with a simple one-fifth rule can lead to this linear runtime. The adapted one-fifth rule with a logarithmic cap lets the $(1 + (\lambda, \lambda))$ GA outperform the $(1 + 1)$ EA on random satisfiability instances [8].

Choosing λ from a power-law distribution and taking $p = \frac{\lambda}{n}$ and $c = \frac{1}{\lambda}$ lets the $(1 + (\lambda, \lambda))$ GA optimize OneMax in linear time [2]. Also, as it was mentioned in the introduction, with randomly chosen parameters (but with some dependencies between several of them) the $(1 + (\lambda, \lambda))$ GA can optimize $\textsc{Jump}_k$ in time of $e^{O(k)}(\frac{n}{k})^{(1+\varepsilon)k/2}$ [3]. For the LeadingOnes it was shown in [5] that the runtime of the $(1 + (\lambda, \lambda))$ GA is $\Theta(n^2)$ and that any dynamic choice of λ does not change this asymptotical runtime.

In our proofs we use the following language (also for the $(1 + (\lambda, \lambda))$ GA with static parameters). When we analyse the $(1 + (\lambda, \lambda))$ GA on Jump and the algorithm has already reached the local optimum, then we call the mutation phase *successful* if all k zero-bits of x are flipped to ones in the mutation winner x'. We call the crossover phase *successful* if the crossover winner has a greater fitness than x.

2.4 Useful Tools

An important tool in our analysis is Wald's equation [45] as it allows us to express the expected number of fitness evaluations through the expected number of iterations and the expected cost of one iteration.

Lemma 5 *(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_t 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

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

In our analysis of the heavy-tailed $(1 + (\lambda, \lambda))$ GA we use the following multiplicative drift theorem.

Theorem 6 (Multiplicative Drift [21]) *Let $S \subset \mathbb{R}$ be a finite set of positive numbers with minimum $s_{\min}$. Let $\{X_t\}_{t\in\mathbb{N}_0}$ be a sequence of random variables over $S \cup \{0\}$. Let T be the first point in time t when $X_t = 0$, that is,*

$$T = \min\{t \in \mathbb{N} : X_t = 0\},$$

which is a random variable. Suppose that there exists a constant $\delta > 0$ such that for all $t \in \mathbb{N}_0$ and all $s \in S$ such that $\Pr[X_t = s] > 0$ we have

$$E[X_t - X_{t+1} \mid X_t = s] \geq \delta s.$$

Then for all $s_0 \in S$ such that $\Pr[X_0 = s_0] > 0$ we have

$$E[T \mid X_0 = s_0] \leq \frac{1 + \ln(s_0/s_{\min})}{\delta}.$$

We use the following well-known relation between the arithmetic and geometric means.

Lemma 7 *For all positive a and b it holds that $a + b \geq 2\sqrt{ab}$.*

3 Runtime Analysis

In this section we perform a runtime analysis of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on the easy problems OneMax, LeadingOnes, and Minimum Spanning Tree as well as the more difficult Jump problem. We show that this algorithm can efficiently escape local optima and that it is capable of solving Jump functions much faster than the known mutation-based algorithms and most of the crossover-based EAs. At the same time it does not fail on easy functions like OneMax, unlike the $(1 + (\lambda, \lambda))$ GA with those static parameters which are optimal for Jump [6].

From the results of this section we distill the recommendations to use β_p and β_c slightly greater than one and to use β_λ slightly greater than two. We also suggest to use almost unbounded power-law distributions, taking $u_c = u_p = \sqrt{n}$ and $u_\lambda = 2^n$. These recommendations are justified in Corollary 16.

3.1 Easy Problems

In this subsection we show that the heavy-tailed $(1 + (\lambda, \lambda))$ GA has a reasonably good performance on the easy problems OneMax, LeadingOnes, minimum spanning tree, and partition.

3.1.1 OneMax

The following result shows that the heavy-tailed $(1 + (\lambda, \lambda))$ GA just like the simple $(1 + 1)$ EA solves the OneMax problem in $O(n \log(n))$ iterations.

Theorem 8 *If $\beta_\lambda > 1$, $\beta_p > 1$, and $\beta_c > 1$, then the heavy-tailed $(1 + (\lambda, \lambda))$ GA finds the optimum of* OneMax *in $O(n \log(n))$ iterations. The expected number of fitness evaluations is*

- $O(n \log(n))$, *if* $\beta_\lambda > 2$,
- $O(n \log(n)(\log(u_\lambda) + 1))$, *if* $\beta_\lambda = 2$, *and*
- $O(n u_\lambda^{2-\beta_\lambda} \log(n))$, *if* $\beta_\lambda \in (1, 2)$.

The central argument in the proof of Theorem 8 is the observation that the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs an iteration equivalent to one of the $(1 + 1)$ EA with a constant probability, which is shown in the following lemma.

Lemma 9 *If β_p, β_c and β_λ are all strictly greater than one, then with probability $\rho = \Theta(1)$ the heavy-tailed $(1 + (\lambda, \lambda))$ GA chooses $p = c = \frac{1}{\sqrt{n}}$ and $\lambda = 1$ and performs an iteration of the $(1 + 1)$ EA with mutation rate $\frac{1}{n}$.*

Proof Since we choose p, c and λ independently, then by the definition of the power-law distribution and by Lemma 2 we have

$$\begin{aligned}\rho &= \Pr\left[p = \frac{1}{\sqrt{n}}\right] \Pr\left[c = \frac{1}{\sqrt{n}}\right] \Pr[\lambda = 1] \\ &= C_{\beta_p, u_p} 1^{-\beta_p} \cdot C_{\beta_c, u_c} 1^{-\beta_c} \cdot C_{\beta_\lambda, u_\lambda} 1^{-\beta_\lambda} \\ &= \Theta(1) \cdot \Theta(1) \cdot \Theta(1) = \Theta(1).\end{aligned}$$

If we have $\lambda = 1$, then we have only one mutation offspring which is automatically chosen as the mutation winner x'. Note that although we first choose $\ell \sim \mathrm{Bin}(n, p)$ and then flip ℓ random bits in x, the distribution of x' in the search space is the same as if we flipped each bit independently with probability p (see Section 2.1 in [14] for more details).

In the crossover phase we create only one offspring y by applying the biased crossover to x and x'. Each bit of this offspring is different from the bit in the same position in x if and only if it was flipped in x' (with probability p) and then taken from x' in the crossover phase (with probability c). Therefore, y is distributed in the search space as if we generated it by applying the standard bit mutation with mutation rate pc to x. Hence, we can consider such iteration of the heavy-tailed $(1 + (\lambda, \lambda))$ GA as an iteration of the $(1 + 1)$ EA which uses a standard bit mutation with mutation rate $pc = \frac{1}{n}$. □

We are now in position to prove Theorem 8.

Proof of Theorem 8 By Lemma 9 with probability at least ρ, which is at least some constant independent of the problem size n, the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs an iteration of the $(1 + 1)$ EA. Hence, the probability $P(i)$ to increase fitness in one iteration if we have already reached fitness i is

$$P(i) \geq \rho \cdot \frac{n - i}{en}.$$

Hence, we estimate the total runtime in terms of iterations as a sum of the expected runtimes until we leave each fitness level.

$$E[T_I] \leq \sum_{i=0}^{n-1} \frac{1}{P(i)} \leq \frac{1}{\rho} \sum_{i=0}^{n-1} \frac{ne}{n - i} \leq \frac{en \ln(n)}{\rho} = O(n \log(n)).$$

To compute the expected number of fitness evaluations until we find the optimum we use Wald's equation (Lemma 5). Since in each iteration of the heavy-tailed $(1 + (\lambda, \lambda))$ GA we make 2λ fitness evaluations, we have

$$E[T_F] = E[T_I] \cdot E[2\lambda].$$

By Lemma 3 we have

$$E[\lambda] = \begin{cases} \Theta(1), & \text{if } \beta_\lambda > 2, \\ \Theta(\log(u_\lambda) + 1), & \text{if } \beta_\lambda = 2, \\ \Theta(u_\lambda^{2-\beta_\lambda}), & \text{if } \beta_\lambda \in (1, 2). \end{cases}$$

Therefore,

$$E[T_F] = \begin{cases} O(n \log(n)), & \text{if } \beta_\lambda > 2, \\ O(n \log(n)(\log(u_\lambda) + 1)), & \text{if } \beta_\lambda = 2, \\ O(n u_\lambda^{2-\beta_\lambda} \log(n)), & \text{if } \beta_\lambda \in (1, 2). \end{cases}$$

□

Theorem 8 shows that the heavy-tailed $(1 + (\lambda, \lambda))$ GA can fall back to a $(1 + 1)$ EA behavior and turn into a simple hill climber. Since we do not have a matching lower bound, our analysis leaves open the question to what extent the heavy-tailed $(1 + (\lambda, \lambda))$ GA benefits from iterations in which it samples parameter values different from the ones used in the lemma above. On the one hand, in [2] it was shown that if we choose only one parameter λ from the power-law distribution and set the other parameters to their optimal values in the $(1 + (\lambda, \lambda))$ GA (namely, $p = \frac{\lambda}{n}$ and $c = \frac{1}{\lambda}$ [14]), then we have a linear runtime on OneMax. This indicates that there is a chance that the heavy-tailed $(1 + (\lambda, \lambda))$ GA with an independent choice of three parameters can also have a $o(n \log(n))$ runtime on this problem. On the other hand, the probability that we choose p and c close to their optimal values is not high, hence we have to rely on making good progress when using non-optimal parameters values. Our experiments presented in Sect. 4.1 suggest that such parameters do not yield the desired progress speed and that the heavy-tailed $(1 + (\lambda, \lambda))$ GA has an $\Omega(n \log(n))$ runtime (see Fig. 3). For this reason, we rather believe that the heavy-tailed $(1 + (\lambda, \lambda))$ GA proposed in this work has an inferior performance on OneMax than the one proposed in [2]. Since our new algorithm has a massively better performance on jump functions, we feel that losing a logarithmic factor in the runtime on OneMax is not too critical.

Lemma 9 also allows us to transform any upper bound on the runtime of the $(1 + 1)$ EA which was obtained via the fitness level argument or via drift with the fitness into the same asymptotical runtime for the heavy-tailed $(1 + (\lambda, \lambda))$ GA. We give three examples in the following subsections.

3.1.2 LeadingOnes

For the LeadingOnes problem, we now show that arguments analogous to the ones in [41] can be used to prove an $O(n^2)$ runtime guarantee also for the heavy-tailed $(1 + (\lambda, \lambda))$ GA.

Theorem 10 *If $\beta_\lambda > 1$, $\beta_p > 1$, and $\beta_c > 1$, then the expected runtime of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on* LeadingOnes *is $O(n^2)$ iterations. In terms of fitness evaluations the expected runtime is*

$$E[T_F] = \begin{cases} O(n^2), & \text{if } \beta_\lambda > 2, \\ O(n^2(\log(u_\lambda) + 1)), & \text{if } \beta_\lambda = 2, \\ O(n^2 u_\lambda^{2-\beta_\lambda}), & \text{if } \beta_\lambda \in (1, 2). \end{cases}$$

Proof The probability that the heavy-tailed $(1 + (\lambda, \lambda))$ GA improves the fitness in one iteration is at least the probability that it performs an iteration of the $(1+1)$ EA that improves the fitness. By Lemma 9 the probability that the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs an iteration of the $(1 + 1)$ EA is $\Theta(1)$. The probability that the $(1 + 1)$ EA increases the fitness in one iteration is at least the probability that it flips the first zero-bit in the string and does not flip any other bit, which is $\frac{1}{n}(1 - \frac{1}{n})^{n-1} \geq \frac{1}{en}$. Hence, the probability that the heavy-tailed $(1 + (\lambda, \lambda))$ GA increases the fitness in one iteration is $\Omega(\frac{1}{n})$.

Therefore, the expected number of iterations before the heavy-tailed $(1 + (\lambda, \lambda))$ GA improves the fitness is $O(n)$ iterations. Since there will be no more than n improvements in fitness before we reach the optimum, the expected total runtime of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on LeadingOnes is at most $O(n^2)$ iterations. Since by Lemma 3 with $\beta_\lambda > 1$ the expected cost of one iteration is

$$E[2\lambda] = \begin{cases} \Theta(1), & \text{if } \beta_\lambda > 2, \\ \Theta(\log(u_\lambda) + 1), & \text{if } \beta_\lambda = 2, \\ \Theta(u_\lambda^{2-\beta_\lambda}), & \text{if } \beta_\lambda \in (1, 2), \end{cases}$$

by Wald's equation (Lemma 5) the expected total runtime in terms of fitness evaluations is

$$E[T_F] = E[2\lambda]E[T_I] = \begin{cases} O(n^2), & \text{if } \beta_\lambda > 2, \\ O(n^2(\log(u_\lambda) + 1)), & \text{if } \beta_\lambda = 2, \\ O(n^2 u_\lambda^{2-\beta_\lambda}), & \text{if } \beta_\lambda \in (1, 2). \end{cases}$$

□

3.1.3 Minimum Spanning Tree Problem

We proceed with the runtime on the minimum spanning tree problem. Reusing some of the arguments from [38] and some more from the later work [21], we show that the expected runtime of the heavy-tailed $(1 + (\lambda, \lambda))$ GA admits the same upper bound $O(m^2 \log(W_{\text{total}}))$ as the $(1 + 1)$ EA.

Theorem 11 *If $\beta_\lambda > 1$, $\beta_p > 1$, and $\beta_c > 1$, then the expected runtime of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on minimum spanning tree problem is $O(m^2 \log(W_{\text{total}}))$ iterations. In terms of fitness evaluations it is*

$$E[T_F] = \begin{cases} O(m^2 \log(W_{\text{total}})), & \text{if } \beta_\lambda > 2, \\ O(m^2 \log(W_{\text{total}})(\log(u_\lambda) + 1)), & \text{if } \beta_\lambda = 2, \\ O(m^2 u_\lambda^{2-\beta_\lambda} \log(W_{\text{total}})), & \text{if } \beta_\lambda \in (1, 2). \end{cases}$$

Proof In [38] it was shown that starting with a random subgraph of G, the $(1 + 1)$ EA finds a spanning tree graph in $O(m \log(n))$ iterations. We now briefly adjust these arguments to the heavy-tailed $(1 + (\lambda, \lambda))$ GA. If G' is disconnected, then the

probability to reduce the number of connected components is at most the probability that the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs an iteration of the $(1+1)$ EA multiplied by the probability that an iteration of the $(1 + 1)$ EA adds an edge which connects two connected components (and does not add or remove other edges from the subgraph G'). The latter probability is at least $\frac{cc(G')-1}{m}(1-\frac{1}{m})^{m-1} \geq \frac{cc(G')-1}{em}$, since there are at least $cc(G') - 1$ edges which we can add to connect a pair of connected components. Therefore, by the fitness level argument we have that the expected number of iterations before the heavy-tailed $(1 + (\lambda, \lambda))$ GA finds a connected graph is $O(m \log(n))$.

If the algorithm has found a connected graph, with probability $\Omega(\frac{|E'|-(n-1)}{m})$ the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs an iteration of the $(1 + 1)$ EA that removes an edge participating in a cycle (since there are at least $(|E'| - (n - 1))$ such edges). Therefore, in $O(m \log(m))$ iterations the heavy-tailed $(1 + (\lambda, \lambda))$ GA finds a spanning tree (probably not the minimum one). Note that $O(m \log(m)) = O(m \log(n))$, since we do not have loops and parallel edges and thus $m \leq \frac{n(n-1)}{2}$.

Once the heavy-tailed $(1 + (\lambda, \lambda))$ GA has obtained a spanning tree, it cannot accept any subgraph that is not a spanning tree. Therefore, we can use the multiplicative drift argument from [21]. Namely, we define a potential function $\Phi(G')$ that is equal to the weight of the current tree minus the weight of the minimum spanning tree. In [21] it was shown that for every iteration t of $(1 + 1)$ EA, we have

$$E[\Phi(G'_t) - \Phi(G'_{t+1}) \mid \Phi(G'_t) = W] \geq \frac{W}{em^2},$$

where G'_t denotes the current graph in the start of iteration t. By Lemma 9 and since the weight of the current graph cannot decrease in one iteration, for the heavy-tailed $(1 + (\lambda, \lambda))$ GA we have

$$E[\Phi(G'_t) - \Phi(G'_{t+1}) \mid \Phi(G'_t) = W] \geq \frac{\rho W}{em^2}$$

for some ρ, which is a constant independent of m and W. Since the edge weights are integers, we have $\Phi(G'_t) \geq 1 =: s_{\min}$ for all t such that G'_t is not an optimal solution. We also have $\Phi(G'_0) \leq W_{\text{total}}$ by the definition of W_{total}. Therefore, by the multiplicative drift theorem (Theorem 6) we have that the expected runtime until we find the optimum starting from a spanning tree is at most

$$\frac{1 + \ln(W_{\text{total}})}{\rho/(em^2)} = O(m^2 \log(W_{\text{total}})).$$

Together with the runtime to find a spanning tree, we obtain a total expected runtime of

$$E[T_I] = O(m \log(n)) + O(m \log(n)) + O(m^2 \log(W_{\text{total}})) = O(m^2 \log(W_{\text{total}}))$$

iterations. By Lemma 3 and by Wald's equation (Lemma 5) the expected number of fitness evaluations is therefore

$$E[T_F] = E[2\lambda]E[T_I] = \begin{cases} O(m^2 \log(W_{\text{total}})), & \text{if } \beta_\lambda > 2, \\ O(m^2 \log(W_{\text{total}})(\log(u_\lambda) + 1)), & \text{if } \beta_\lambda = 2, \\ O(m^2 u_\lambda^{2-\beta_\lambda} \log(W_{\text{total}})), & \text{if } \beta_\lambda \in (1, 2). \end{cases}$$

□

3.1.4 Approximations for the Partition Problem

We finally regard the partition problem. We use similar arguments as in [46] (slightly modified to exploit multiplicative drift analysis) to show that the heavy-tailed $(1 + (\lambda, \lambda))$ GA also finds a $(\frac{4}{3} + \varepsilon)$ approximation in linear time. For $\frac{4}{3}$ approximations we improve the $O(n^2)$ runtime result of [46] and show that both the $(1 + 1)$ EA and the heavy-tailed $(1 + (\lambda, \lambda))$ GA succeed in $O(n \log(w))$ fitness evaluations.

Theorem 12 *If $\beta_\lambda > 2$, $\beta_p > 1$, and $\beta_c > 1$, then the heavy-tailed $(1 + (\lambda, \lambda))$ GA finds a $(\frac{4}{3} + \varepsilon)$ approximation to the partition problem in an expected number of $O(n)$ iterations. The expected number of fitness evaluations is*

$$E[T_F] = \begin{cases} O(n), & \text{if } \beta_\lambda > 2, \\ O(n(\log(u_\lambda) + 1)), & \text{if } \beta_\lambda = 2, \\ O(n u_\lambda^{2-\beta_\lambda}), & \text{if } \beta_\lambda \in (1, 2). \end{cases}$$

The heavy-tailed $(1 + (\lambda, \lambda))$ GA and the $(1 + 1)$ EA also find a $\frac{4}{3}$ approximation in an expected number of $O(n \log(w))$ iterations. The expected number of fitness evaluations for the heavy-tailed $(1 + (\lambda, \lambda))$ GA is

$$E[T_F] = \begin{cases} O(n \log(w)), & \text{if } \beta_\lambda > 2, \\ O(n \log(w)(\log(u_\lambda) + 1)), & \text{if } \beta_\lambda = 2, \\ O(n u_\lambda^{2-\beta_\lambda} \log(w)), & \text{if } \beta_\lambda \in (1, 2). \end{cases}$$

Proof We first recall the definition of a *critical object* from [46]. Let $\ell \geq \frac{w}{2}$ be the fitness of the optimal solution. Let $i_1 < i_2 < \cdots < i_k$ be the indices of the objects in the heavier bin. Then we call the object r in the heavier bin the critical one if it is the object with the smallest index such that

$$\sum_{j : i_j \leq r} w_{i_j} > \ell.$$

In other words, the critical object is the object in the heavier bin such that the total weight of all previous (non-lighter) objects in that bin is not greater than ℓ, but the total weight of all previous objects together with the weight of this object is greater

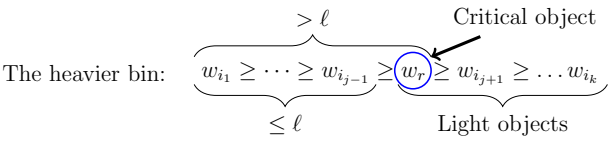

Fig. 2 Illustration of the definition of the critical object

than ℓ. We call the weight of the critical object the *critical weight*. We also call the objects in the heavier bin which have index at least r the *light objects*. This notation is illustrated in Fig. 2.

We now show that at some moment the critical weight becomes at most $\frac{w}{3}$ and does not exceed this value in the future. For this we consider two cases.

Case 1: $w_2 > \frac{w}{3}$. Note that in this case we also have $w_1 > \frac{w}{3}$, since $w_1 \geq w_2$ and the weight of all other objects is $w - w_1 - w_2 < \frac{w}{3}$. If the two heaviest objects are in the same bin, then the weight of this (heavier) bin is at least $\frac{2w}{3}$. In any partition in which these two objects are separated the weight of the heavier bin is at most $\max\{w - w_1, w - w_2\} < \frac{2w}{3}$, therefore if the algorithm generates such a partition it would replace a partition in which the two heaviest objects are in the same bin. For the same reason, once we have a partition with the two heaviest objects in different bins, we cannot accept a partition in which they are in the same bin.

The probability of separating the two heaviest objects into two different bins is at least the probability that the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs an iteration of the $(1 + 1)$ EA (which by Lemma 9 is $\Theta(1)$) multiplied by the probability that in this iteration we move one of these two objects into a different bin and do not move the second object. This is at least

$$\Theta(1) \cdot \frac{2}{n}\left(1 - \frac{1}{n}\right) = \Theta\left(\frac{1}{n}\right).$$

Consequently, in an expected number of $O(n)$ iterations the two heaviest objects will be separated into different bins.

Note that the weight of the heaviest object cannot be greater than the weight of the heavier bin (even in the optimal solution), hence we have $w_2 \leq w_1 \leq \ell$. Therefore, when the two heaviest objects are separated into different bins neither of them can be the critical one. Hence, the critical weight is now at most $w_3 < \frac{w}{3}$.

Case 2: $w_2 \leq \frac{w}{3}$. Since the heaviest object can never be the critical one, the critical weight is at most $w_2 \leq \frac{w}{3}$.

Once the critical weight is at most $\frac{w}{3}$, we define a potential function

$$\Phi(x_t) = \max\left\{f(x_t) - \ell - \tfrac{w}{6}, 0\right\},$$

where x_t is the current individual of the heavy-tailed $(1 + (\lambda, \lambda))$ GA at the beginning of iteration t. Note that this potential function does not increase due to the elitist selection of the $(1 + (\lambda, \lambda))$ GA.

We now show that as long as $\Phi(x_t) > 0$, any iteration which moves any light object to the lighter bin and does not move other objects reduces the fitness (and the potential). Recall that the weight of each light object is at most $\frac{w}{3}$. Then the weight of the bin which was heavier before the move is reduced by the weight of the moved object. The weight of the other bin becomes at most

$$w - f(x_t) + \frac{w}{3} < \ell - \frac{w}{6} + \frac{w}{3} \leq \ell + \frac{w}{6}.$$

Therefore, the weight of both bins becomes smaller than the weight of the bin which was heavier before the move, hence such a partition is accepted by the algorithm.

Now we estimate the expected decrease of the potential in one iteration. Recall that by Lemma 9 the probability that the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs an iteration of the $(1 + 1)$ EA is at least some $\rho = \Theta(1)$. The probability that in such an iteration we move only one particular object is $\frac{1}{n}(1 - \frac{1}{n})^{n-1} \geq \frac{1}{en}$. Hence we have two options.

- If there is at least one light object with weight at least $\Phi(x_t)$, then moving it we decrease the potential to zero, since the wight of the heavier bin becomes not greater than $\ell + \frac{w}{6}$ and the weight of the lighter bin also cannot become greater than $\ell + \frac{w}{6}$ as it was shown earlier. Hence, we have

$$E\left[\Phi(x_t) - \Phi(x_{t+1}) \mid \Phi(x_t) = s\right] \geq \frac{s\rho}{en}.$$

- Otherwise, the move of any light object decreases the potential by the weight of the moved object, since the heavy bin will remain the heavier one after such a move. The total weight of the light objects is at least $f(x_t) - \ell \geq \Phi(x_t)$. Let L be the set of indices of the light objects. Then we have

$$E\left[\Phi(x_t) - \Phi(x_{t+1}) \mid \Phi(x_t) = s\right] \geq \rho \sum_{i \in L} \frac{w_i}{en} \geq \frac{s\rho}{en}.$$

Now we are in position to use the multiplicative drift theorem (Theorem 6). Note that the maximum value of potential function is $\frac{w}{2}$ and its minimum positive value is $\frac{1}{6}$ (since $f(x_t)$ and ℓ are integer values and $\frac{w}{6}$ is divided by $\frac{1}{6}$). Therefore, denoting T_I as the smallest t such that $\Phi(x_t) = 0$, we have

$$E[T_I] \leq \frac{1 + \ln(3w)}{\rho/(en)} = \Theta(n \log(w)).$$

When $\Phi(x_t) = 0$, we have

$$f(x_t) \leq \ell + \tfrac{w}{6} \leq \ell + \tfrac{\ell}{3} = \tfrac{4}{3}\ell,$$

which means that x_t is a $\frac{4}{3}$ approximation of the optimal solution.

To show that we obtain a $(\frac{4}{3}+\varepsilon)$ approximation in expected linear time for all constants $\varepsilon > 0$, we use a modified potential function Φ_ε, which is defined by

$$\Phi_\varepsilon(x_t) = \begin{cases} 0, & \text{if } \Phi(x_t) \leq \frac{\varepsilon w}{2}, \\ \Phi(x_t), & \text{otherwise.} \end{cases}$$

For this potential function the drift is at least as large as for Φ, but its smallest non-zero value is $\frac{\varepsilon w}{2}$. Hence, by the multiplicative drift theorem (Theorem 6) the expectation of the first time $T_I(\varepsilon)$ when Φ_ε turns to zero is at most

$$E[T_I(\varepsilon)] \leq \frac{1+\ln\left(\frac{w}{6}/\frac{\varepsilon w}{2}\right)}{\rho/(en)} = O(n).$$

When $\Phi(x_t) = 0$, we have

$$f(x_t) \leq \ell + \frac{w}{6} + \frac{\varepsilon w}{2} \leq \ell + \frac{\ell}{3} + \varepsilon\ell = \left(\frac{4}{3}+\varepsilon\right)\ell,$$

therefore x_t is a $(\frac{4}{3}+\varepsilon)$ approximation.

By Lemma 3 and by Wald's equation (Lemma 5) we also have the following estimates on the runtimes T_F and $T_F(\varepsilon)$ in terms of fitness evaluations.

$$E[T_F] = E[2\lambda]\cdot E[T_I] = \begin{cases} O(n\log(w)), & \text{if } \beta_\lambda > 2, \\ O(n\log(w)(\log(u_\lambda)+1)), & \text{if } \beta_\lambda = 2, \\ O(nu_\lambda^{2-\beta_\lambda}\log(w)), & \text{if } \beta_\lambda \in (1,2), \end{cases}$$

$$E[T_F(\varepsilon)] = E[2\lambda]\cdot E[T_I(\varepsilon)] = \begin{cases} O(n), & \text{if } \beta_\lambda > 2, \\ O(n(\log(u_\lambda)+1)), & \text{if } \beta_\lambda = 2, \\ O(nu_\lambda^{2-\beta_\lambda}), & \text{if } \beta_\lambda \in (1,2). \end{cases}$$

□

3.2 JUMP Functions

In this subsection we show that the heavy-tailed $(1+(\lambda,\lambda))$ GA performs well on jump functions, hence there is no need for the informal argumentation [3] to choose mutation rate p and crossover bias c identical. The main result is the following theorem, which estimates the expected runtime until we leave the local optimum of JUMP_k.

Theorem 13 *Let $k \in [2..\frac{n}{4}]$, $u_p \geq \sqrt{2k}$, and $u_c \geq \sqrt{2k}$. Assume that we use the heavy-tailed $(1+(\lambda,\lambda))$ GA (Algorithm 1) to optimize JUMP_k, starting already in the local optimum. Then the expected number of fitness evaluations until the optimum is found is shown in Table 1, where p_{pc} denotes the probability that both p and c are in $[\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}]$. Table 2 shows estimates for p_{pc}.*

Table 1 Influence of the hyperparameters β_λ, u_λ on the expected number $E[T_F]$ of fitness evaluations the heavy-tailed $(1+(\lambda,\lambda))$ GA starting in the local optimum takes to optimize $\textsc{Jump}_k$

	$E[T_F]p_{pc}$	
	if $u_\lambda < \left(\frac{n}{k}\right)^{k/2}$	if $u_\lambda \geq \left(\frac{n}{k}\right)^{k/2}$
$\beta_\lambda \in [0,1)$	$e^{O(k)} \frac{1}{u_\lambda} \left(\frac{n}{k}\right)^k$	$u_\lambda e^{O(k)}$
$\beta_\lambda = 1$		$\frac{u_\lambda e^{O(k)}}{1+\ln\left(u_\lambda \left(\frac{n}{k}\right)^{k/2}\right)}$
$\beta_\lambda \in (1,2)$		$e^{O(k)} u_\lambda^{2-\beta_\lambda} \left(\frac{n}{k}\right)^{(\beta_\lambda-1)k/2}$
$\beta_\lambda = 2$	$e^{O(k)} \frac{\ln(u_\lambda)+1}{u_\lambda} \left(\frac{n}{k}\right)^k$	$e^{O(k)} \ln(u_\lambda) \left(\frac{n}{k}\right)^{k/2}$
$\beta_\lambda \in (2,3)$	$e^{O(k)} \frac{1}{u_\lambda^{3-\beta_\lambda}} \left(\frac{n}{k}\right)^k$	$\mathbf{e^{O(k)} \left(\frac{n}{k}\right)^{(\beta_\lambda-1)k/2}}$
$\beta_\lambda = 3$	$e^{O(k)} \frac{1}{\ln(u_\lambda+1)} \left(\frac{n}{k}\right)^k$	$e^{O(k)} \left(\frac{n}{k}\right)^k / \ln\left(\left(\frac{n}{k}\right)^k\right)$
$\beta_\lambda > 3$	$e^{O(k)} \left(\frac{n}{k}\right)^k$	

Since all runtime bounds are of type $E[T_F] = F(\beta_\lambda, u_\lambda)/p_{pc}$, where $p_{pc} = \Pr[p \in [\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}] \wedge c \in [\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}]]$ and $F(\beta_\lambda, u_\lambda)$ is some function of β_λ and u_λ, to ease reading we only state $F(\beta_\lambda, u_\lambda) = E[T_F]p_{pc}$ and show the influence of the hyperparameters on p_{pc} in Table 2. Asymptotical notation is used for $n \to +\infty$. The bold cell shows the result for the hyperparameters suggested in Corollary 16

Table 2 Influence of the hyperparameters β_p and β_c on $p_{pc} = \Pr[p \in [\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}] \wedge c \in [\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}]]$ when both u_p and u_c are at least $\sqrt{2k}$

	$0 \leq \beta_p < 1$	$\beta_p = 1$	$\beta_p > 1$
$\beta_c < 1$	$\Theta\left(\frac{k^{(1-(\beta_p+\beta_c)/2)}}{u_p^{1-\beta_p} u_c^{1-\beta_c}}\right)$	$\Theta\left(\frac{k^{(1-\beta_c)/2}}{u_c^{1-\beta_c} \log(u_p)}\right)$	$\Theta\left(\frac{k^{(1-(\beta_p+\beta_c)/2)}}{u_c^{1-\beta_c}}\right)$
$\beta_c = 1$	$\Theta\left(\frac{k^{(1-\beta_p)/2}}{u_p^{1-\beta_p} \log(u_c)}\right)$	$\Theta\left(\frac{1}{\log(u_p)\log(u_c)}\right)$	$\Theta\left(\frac{k^{(1-\beta_p)/2}}{\log(u_c)}\right)$
$\beta_c > 1$	$\Theta\left(\frac{k^{(1-(\beta_p+\beta_c)/2)}}{u_p^{1-\beta_p}}\right)$	$\Theta\left(\frac{k^{(1-\beta_c)/2}}{\log(u_p)}\right)$	$\mathbf{\Theta\left(k^{(1-(\beta_p+\beta_c)/2)}\right)}$

Asymptotical notation is used for $n \to +\infty$. The bold cell shows the result for the hyperparameters suggested in Corollary 16

The proof of Theorem 13 follows from similar arguments as in [3, Theorem 6], the main differences being highlighted in the following two lemmas.

Lemma 14 *Let $k \leq \frac{n}{4}$. If $u_p \geq \sqrt{2k}$ and $u_c \geq \sqrt{2k}$, then the probability $p_{pc} = \Pr[p \in [\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}] \wedge c \in [\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}]]$ is as shown in Table 2.*

Proof Since we choose p and c independently, we have

$$p_{pc} = \Pr\left[p \in \left[\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}\right]\right] \cdot \Pr\left[c \in \left[\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}\right]\right].$$

By the definition of the power-law distribution and by Lemmas 1 and 2, we have

$$\Pr\left[p \in \left[\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}\right]\right] = C_{\beta_p, u_p} \sum_{i=\lceil\sqrt{k}\rceil}^{\lfloor\sqrt{2k}\rfloor} i^{-\beta_p}$$
$$= \begin{cases} \Theta\left(\left(\frac{\sqrt{k}}{u_p}\right)^{1-\beta_p}\right), & \text{if } 0 \le \beta_p < 1 \\ \Theta\left(\frac{1}{\log(u_p)}\right), & \text{if } \beta_p = 1 \\ \Theta\left(k^{\frac{1-\beta_p}{2}}\right), & \text{if } \beta_p > 1. \end{cases}$$

We can estimate $\Pr[c \in [\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}]]$ in the same manner, which gives us the final estimate of p_{pc} shown in Table 2. □

Now we proceed with an estimate of the probability to find the optimum in one iteration after choosing p and c.

Lemma 15 *Let $k \in [2..\frac{n}{4}]$. Let λ, p and c be already chosen in an iteration of the heavy-tailed $(1+(\lambda,\lambda))$ GA and let $p, c \in [\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}]$. If the current individual x of the heavy-tailed $(1+(\lambda,\lambda))$ GA is in the local optimum of $\textsc{Jump}_k$, then the probability that the algorithm generates the global optimum in one iteration is at least $e^{-\Theta(k)} \min\{1, (\frac{k}{n})^k \lambda^2\}$.*

Proof The probability $P_{pc}(\lambda)$ that we find the optimum in one iteration is the probability that we have a successful mutation phase and a successful crossover phase in the same iteration. If we denote the probability of a successful mutation phase by p_M and the probability of a successful crossover phase by p_C, then we have $P_{pc}(\lambda) = p_M p_C$. Then with q_ℓ being some constant which denotes the probability that the number ℓ of bits we flip in the mutation phase is in $[pn, 2pn]$, by Lemmas 3.1 and 3.2 in [6] we have

$$P_{pc}(\lambda) = p_M p_C = \frac{q_\ell}{2} \min\left\{1, \lambda\left(\frac{p}{2}\right)^k\right\} \cdot \frac{1}{2} \min\left\{1, \lambda c^k (1-c)^{2pn-k}\right\}$$
$$\ge \frac{q_\ell}{4} \min\left\{1, \lambda\left(\frac{1}{2}\sqrt{\frac{k}{n}}\right)^k\right\} \min\left\{1, \lambda\sqrt{\frac{k}{n}}^k \left(1-\sqrt{\frac{2k}{n}}\right)^{2\sqrt{2kn}}\right\}$$
$$= \frac{q_\ell}{4} \min\left\{1, \lambda 2^{-k}\sqrt{\frac{k}{n}}^k\right\} \min\left\{1, \lambda e^{-\Theta(k)}\sqrt{\frac{k}{n}}^k\right\}$$

If $\lambda \ge \sqrt{\frac{n}{k}}^k$, then we have

$$P_{pc}(\lambda) \ge \frac{q_\ell}{4} 2^{-k} e^{-\Theta(k)} = e^{-\Theta(k)}.$$

Otherwise, if $\lambda < \sqrt{\frac{n}{k}}^k$, then both minima are equal to their second argument. Thus, we have

$$P_{pc}(\lambda) \geq \frac{q_\ell}{4}\lambda^2 2^{-k} e^{-\Theta(k)} \left(\frac{k}{n}\right)^k = e^{-\Theta(k)} \left(\frac{k}{n}\right)^k \lambda^2.$$

Bringing the two cases together, we finally obtain

$$P_{pc}(\lambda) \geq e^{-\Theta(k)} \min\left\{1, \left(\frac{k}{n}\right)^k \lambda^2\right\}.$$

□

Now we are in position to prove Theorem 13.

Proof of Theorem 13 Let the current individual x of the heavy-tailed $(1 + (\lambda, \lambda))$ GA be already in the local optimum. Let P be the probability of event F when the algorithm finds optimum in one iteration. By the law of total probability this probability is at least

$$P \geq p_{(F|pc)} \cdot p_{pc},$$

where $p_{(F|pc)}$ denotes $\Pr[F \mid p, c \in [\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}]]$ and p_{pc} denotes $\Pr[p, c \in [\sqrt{\frac{k}{n}}, \sqrt{\frac{2k}{n}}]]$.

The number T_I of iterations until we jump to the optimum follows a geometric distribution $\mathrm{Geom}(P)$ with success probability P. Therefore,

$$E[T_I] = \frac{1}{P} \leq \frac{1}{p_{(F|pc)} p_{pc}}.$$

Since in each iteration the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs 2λ fitness evaluations (with λ chosen from the power-law distribution), by Wald's equation (Lemma 5) the expected number $E[T_F]$ of fitness evaluations the algorithm makes before it finds the optimum is

$$E[T_F] = E[T_I]E[2\lambda] \leq \frac{2E[\lambda]}{p_{(F|pc)} \cdot p_{pc}}.$$

In the remainder we show how $E[\lambda]$, $p_{(F|pc)}$ and p_{pc} depend on the hyperparameters of the algorithm.

First we note that p_{pc} was estimated in Lemma 14. Also, by Lemma 3 the expected value of λ is

$$E[\lambda] = \begin{cases} \Theta(u_\lambda), & \text{if } \beta_\lambda < 1, \\ \Theta(\frac{u_\lambda}{\log(u_\lambda)+1}), & \text{if } \beta_\lambda = 1, \\ \Theta(u_\lambda^{2-\beta_\lambda}), & \text{if } \beta_\lambda \in (1,2), \\ \Theta(\log(u_\lambda)+1), & \text{if } \beta_\lambda = 2, \\ \Theta(1), & \text{if } \beta_\lambda > 2. \end{cases}$$

Finally, we compute the conditional probability of F via the law of total probability.

$$p_{(F|pc)} = \sum_{i=1}^{u_\lambda} \Pr[\lambda = i] P_{pc}(i),$$

where $P_{pc}(i)$ is as defined in Lemma 15, in which it was shown that $P_{pc}(i) \geq e^{-\Theta(k)} \min\{1, (\frac{k}{n})^k i^2\}$. We consider two cases depending on the value of u_λ.

Case 1: when $u_\lambda \leq (\frac{n}{k})^{k/2}$. In this case we have $P_{pc}(i) \geq e^{-\Theta(k)}(\frac{k}{n})^k i^2$, hence

$$\begin{aligned} p_{(F|pc)} &\geq \sum_{i=1}^{u_\lambda} C_{\beta_\lambda, u_\lambda} i^{-\beta_\lambda} e^{-\Theta(k)} \left(\frac{k}{n}\right)^k i^2 \\ &= e^{-\Theta(k)} \left(\frac{k}{n}\right)^k C_{\beta_\lambda, u_\lambda} \sum_{i=1}^{u_\lambda} i^{2-\beta_\lambda} \\ &= e^{-\Theta(k)} \left(\frac{k}{n}\right)^k E[\lambda^2]. \end{aligned}$$

By Lemma 4 we estimate $E[\lambda^2]$ and obtain

$$p_{(F|pc)} \geq \begin{cases} e^{-\Theta(k)} \left(\frac{k}{n}\right)^k u_\lambda^2, & \text{if } \beta_\lambda < 1, \\ e^{-\Theta(k)} \left(\frac{k}{n}\right)^k \frac{u_\lambda^2}{\ln(u_\lambda)+1}, & \text{if } \beta_\lambda = 1, \\ e^{-\Theta(k)} \left(\frac{k}{n}\right)^k u_\lambda^{3-\beta_\lambda}, & \text{if } \beta_\lambda \in (1,3), \\ e^{-\Theta(k)} \left(\frac{k}{n}\right)^k (\ln(u_\lambda)+1), & \text{if } \beta_\lambda = 3, \\ e^{-\Theta(k)} \left(\frac{k}{n}\right)^k, & \text{if } \beta_\lambda > 3. \end{cases}$$

Case 2: when $u_\lambda > (\frac{n}{k})^{k/2}$. In this case we have $P_{pc}(i) \geq e^{-\Theta(k)}(\frac{k}{n})^k i^2$, when $i \leq (\frac{n}{k})^{k/2}$ and we have $P_{pc}(i) \geq e^{-\Theta(k)}$, when $i > (\frac{n}{k})^{k/2}$. Therefore, we have

$$p_{(F|s)} \geq \sum_{i=1}^{\lfloor (\frac{n}{k})^{k/2} \rfloor} C_{\beta_\lambda, u_\lambda} i^{-\beta_\lambda} \left(\frac{k}{n}\right)^k i^2 e^{-\Theta(k)} + \sum_{i=\lfloor (\frac{n}{k})^{k/2} \rfloor + 1}^{u_\lambda} C_{\beta_\lambda, u_\lambda} i^{-\beta_\lambda} e^{-\Theta(k)}$$
$$= C_{\beta_\lambda, u_\lambda} e^{-\Theta(k)} \left(\left(\frac{k}{n}\right)^k \sum_{i=1}^{\lfloor (\frac{n}{k})^{k/2} \rfloor} i^{2-\beta_\lambda} + \sum_{i=\lfloor (\frac{n}{k})^{k/2} \rfloor + 1}^{u_\lambda} i^{-\beta_\lambda} \right).$$

Estimating the sums via Lemma 1, we obtain

$$p_{(F|pc)} \geq \begin{cases} e^{-\Theta(k)}, & \text{if } \beta_\lambda < 1, \\ e^{-\Theta(k)} \left(1 + \ln\left(u_\lambda \left(\frac{k}{n}\right)^{k/2}\right)\right) \frac{1}{\ln(u)+1}, & \text{if } \beta_\lambda = 1, \\ e^{-\Theta(k)} \left(\frac{k}{n}\right)^{(\beta-1)k/2}, & \text{if } \beta_\lambda \in (1,3), \\ e^{-\Theta(k)} \left(\frac{k}{n}\right)^k \ln\left(\left(\frac{n}{k}\right)^k\right), & \text{if } \beta_\lambda = 3, \\ e^{-\Theta(k)} \left(\frac{k}{n}\right)^k, & \text{if } \beta_\lambda > 3, \end{cases}$$

Gathering the estimates for the two cases and the estimates of $E[\lambda]$ and p_{pc} together, we obtain the runtimes listed in Table 1. □

3.3 Recommended Hyperparameters

In this subsection we subsume the results of our runtime analysis to show most preferable parameters of the power-law distributions for the practical use. We point out the runtime with such parameters on OneMax and Jump_k in Corollary 16. We then also prove a lower bound on the runtime of the $(1 + (\lambda, \lambda))$ GA with static parameters to show that when k is constant (that is, the most interesting case, since only then we have a polynomial runtime), then the performance of the heavy-tailed $(1 + (\lambda, \lambda))$ GA is asymptotically better than the best performance we can obtain with the static parameters.

Corollary 16 *Let $\beta_\lambda = 2 + \varepsilon_\lambda$ and $\beta_p = 1 + \varepsilon_p$ and $\beta_c = 1 + \varepsilon_c$, where $\varepsilon_\lambda, \varepsilon_p, \varepsilon_c > 0$ are some constants. Let also u_λ be at least 2^n and $u_p = u_c = \sqrt{n}$. Then the expected runtime of the heavy-tailed $(1 + (\lambda, \lambda))$ GA is $O(n \log(n))$ fitness evaluations on* OneMax *and $e^{O(k)} (\frac{n}{k})^{(1+\varepsilon_\lambda)k/2}$ fitness evaluations on Jump_k, $k \in [2..\frac{n}{4}]$.*

This corollary follows from Theorems 8 and 13. We only note that for the runtime on Jump_k the same arguments as in Theorem 8 show us that the runtime until we reach the local optimum is at most $O(n \log(n))$, which is small compared to the runtime until we reach the global optimum. Also we note that when β_p and β_c are both greater than one and $u_p = u_c = \sqrt{n} \geq \sqrt{2k}$, by Lemma 14 we have $p_{pc} = \Theta(k^{-\frac{\varepsilon_p + \varepsilon_c}{2}})$,

which is implicitly hidden in the $e^{O(k)}$ factor of the runtime on $\textsc{Jump}_k$. We also note that $u_\lambda = 2^n$ guarantees that $u_\lambda > (\frac{n}{k})^{k/2}$, which yields the runtimes shown in the right column of Table 1.

Corollary 16 shows that when we have (almost) unbounded distributions and use power-law exponents slightly greater than one for all parameters except the population size, for which we use a power-law exponent slightly greater than two, we have a good performance both on easy monotone functions, which give us a clear signal towards the optimum, and on the much harder jump functions, without any knowledge of the jump size.

We now also show that the proposed choice of the hyper-parameters gives us a better performance than any static parameters choice on $\textsc{Jump}_k$ for constant k. As we have already noted in the introduction, only for such values of k different variants of the $(1 + (\lambda, \lambda))$ GA and many other classic EAs have a polynomial runtime, hence this case is the most interesting to consider. We prove the following theorem which holds for any static parameters choice of the $(1 + (\lambda, \lambda))$ GA, even when we use different population sizes λ_M and λ_C in the mutation and in the crossover phases respectively.

Theorem 17 *Let n be sufficiently large. Then the expected runtime of the $(1 + (\lambda, \lambda))$ GA with any static parameters p, c, λ_M and λ_C on $\textsc{Jump}_k$ with $k \leq \frac{n}{512}$ is at least $B := \frac{1}{91\sqrt{\ln(n/k)}}(\frac{2n}{k})^{(k+1)/2}$.*

Before we prove Theorem 17, we give a short sketch of the proof to ease the further reading. First we show that with high probability the $(1 + (\lambda, \lambda))$ GA with static parameters starts at a point with approximately $\frac{n}{2}$ one-bits. In the second step we handle a wide range of parameter settings and show that for them we cannot obtain a runtime better than B by showing that the probability to find the optimum in one iteration is at most $1/B$. For the remaining settings we then show that we are not likely to observe an $\Omega(n)$ progress in one iteration, hence with high probability there is an iteration when we have a fitness which is $\frac{n}{2} + \Omega(n)$ and at the same time which is $n - k - \Omega(n)$. From that point on the probability that we have a progress which is $\Omega(k \log(\frac{n}{k}))$ is very unlikely to happen hence with high probability the $(1 + (\lambda, \lambda))$ GA does not reach the local optima of $\textsc{Jump}_k$ (nor the global one) in $\Omega(\frac{n}{k \log(\frac{n}{k})})$ iterations which is equal to $\Omega(\frac{(\lambda_M+\lambda_C)n}{k \log(\frac{n}{k})})$ fitness evaluations by the definition of the algorithm. For the narrowed range of parameters this yields the lower bound.

To transform these informal arguments into a rigorous proof we use several auxiliary tools. The first of them is Lemma 14 from [29], which we formulate as follows.[2]

Lemma 18 (Lemma 14 in [29]) *Let x be a bit string of length n with exactly m one-bits in it. Let y be an offspring of x obtained by flipping each bit independently with probability $\frac{r}{n}$, where $r \leq \frac{n}{2}$. Let also m' be a random variable denoting the number of one-bits in y. Then for any $\Delta \geq 0$ we have*

$$\Pr\left[m' - m \geq (n - 2m)\frac{r}{n} + \Delta\right] \leq \exp\left(\frac{-\Delta^2}{2(1 - r/n)(r + \Delta/3)}\right).$$

[2] Note that in [29] the authors prove upper bounds on both getting a too high and a too low number of one-bits after applying a standard bit mutation. Since we only use the first one, we do not mention the second bound here.

We also use the following lemma, which bounds the probability to make a jump to a certain point.

Lemma 19 *If we are in distance $d \leq \frac{n}{2}$ from the unique optimum of any function, then the probability P that the $(1 + (\lambda, \lambda))$ GA with mutation rate p, crossover bias c and population sizes for the mutation and crossover phases λ_M and λ_C respectively finds the optimum in one iteration is at most*

$$\begin{aligned} P &\leq \min\left\{1, \lambda_M p^d (1-p)^{n-d} + \lambda_M \lambda_C (pc)^d (1-pc)^{n-d}\right\} \\ &\leq \min\left\{1, 2\lambda_M \lambda_C \left(\frac{d}{2n}\right)^d\right\}. \end{aligned}$$

A very similar, but less general result has been proven in [6] (Theorem 16).

Proof. Without loss of generality we assume that the unique optimum is the all-ones bit string. Hence, the current individual has exactly d zero-bits. Let p_ℓ be the probability that we choose ℓ as the number of bits to flip at the start of the iteration of the $(1 + (\lambda, \lambda))$ GA. Let also $p_m(\ell)$ be the probability (conditional on the chosen ℓ) that the mutation winner has all zero-bits flipped to ones. Note that this is necessary for crossover to be able to create the global optimum. Let $p_c(\ell)$ be the probability that conditional on the chosen ℓ and on that we flip all d zero-bits in the mutation winner, we then flip $\ell - d$ zeros in the mutation winner in at least one crossover offspring. Then by the law of total probability we have

$$P = \sum_{\ell=0}^{n} p_\ell p_m(\ell) p_c(\ell). \tag{1}$$

For $\ell < d$ the probability that we flip all d zero-bits in the mutation winner is zero. For $\ell = d$ we flip all d zero-bits in one particular mutation offspring with probability $q_m(\ell) = \binom{n}{d}^{-1}$. Since we create all λ_M offspring independently, the probability that we flip all d zero-bits in at least one offspring is

$$p_m(\ell) = 1 - (1 - q_m(\ell))^{\lambda_M} \leq \lambda_M q_m(\ell) = \lambda_M \binom{n}{d}^{-1},$$

where we used Bernoulli inequality. Since when we create such an offspring in the mutation phase, we already find the optimum, we assume that we do not need to perform the crossover and therefore, $p_c(\ell) = 1$ in this case.

When $\ell > d$, the probability to flip all d zero-bits in one offspring is

$$q_m(\ell) = \binom{n-d}{\ell-d}\binom{n}{\ell}^{-1}.$$

The probability to do so in one of λ_M independently created offspring is thus

$$p_m = 1 - (1 - q_m(\ell))^{\lambda_M} \leq \lambda_M q_m(\ell) = \lambda_M \binom{n-d}{\ell-d}\binom{n}{\ell}^{-1}.$$

The probability that in one crossover offspring we take from the current individual all $\ell - d$ bits which are zeros in the mutation winner and take from the mutation winner all d bits which are zeros in the current individual is $q_c(\ell) = c^d(1-c)^{\ell-d}$. Consequently, the probability that we do this in at least one of λ_C independently created individuals is

$$p_c(\ell) = 1 - (1 - q_c(\ell))^{\lambda_C} \leq \lambda_C q_c(\ell) = \lambda_C c^d (1-c)^{\ell-d}.$$

Recall that ℓ is chosen from the binomial distribution $\mathrm{Bin}(n, p)$, thus we have $p_\ell = \binom{n}{\ell} p^\ell (1-p)^{n-\ell}$. Putting all the estimates above into (1) we obtain

$$\begin{aligned}
P &= p_d p_m(d) + \sum_{\ell=d+1}^{n} p_\ell p_m(\ell) p_c(\ell) \\
&\leq \binom{n}{d} p^d (1-p)^{n-d} \lambda_M \binom{n}{d}^{-1} \\
&\quad + \sum_{\ell=d+1}^{n} \binom{n}{\ell} p^\ell (1-p)^{n-\ell} \lambda_M \binom{n-d}{\ell-d}\binom{n}{\ell}^{-1} \lambda_C c^d (1-c)^{\ell-d} \\
&= \lambda_M p^d (1-p)^{n-d} + \lambda_M \lambda_C (1-p)^{n-d} (pc)^d \sum_{\ell=d+1}^{n} \binom{n-d}{\ell-d} \left(\frac{p(1-c)}{1-p}\right)^{\ell-d} \\
&= \lambda_M p^d (1-p)^{n-d} + \lambda_M \lambda_C (1-p)^{n-d} (pc)^d \sum_{i=1}^{n-d} \binom{n-d}{i} \left(\frac{p(1-c)}{1-p}\right)^{i} \\
&\leq \lambda_M p^d (1-p)^{n-d} + \lambda_M \lambda_C (1-p)^{n-d} (pc)^d \left(\frac{p(1-c)}{1-p} + 1\right)^{n-d} \\
&= \lambda_M p^d (1-p)^{n-d} + \lambda_M \lambda_C (1-p)^{n-d} (pc)^d \left(\frac{1-pc}{1-p}\right)^{n-d} \\
&= \lambda_M p^d (1-p)^{n-d} + \lambda_M \lambda_C (1-pc)^{n-d} (pc)^d.
\end{aligned}$$

We now consider function $f_d(x) = x^d(1-x)^{n-d}$ on interval $x \in [0, 1]$. To find its maximum, we consider its value in the ends of the interval (which is zero in both ends) and in the roots of its derivative, which is

$$\begin{aligned}
f_d'(x) &= d x^{d-1}(1-x)^{n-d} - (n-d) x^d (1-x)^{n-d-1} \\
&= x^{d-1}(1-x)^{n-d-1}(d - nx).
\end{aligned}$$

Hence, the only root of the derivative is in $x = \frac{d}{n}$. Since $f_d(x)$ is a smooth function, it reaches its maximum there, which is,

$$f_d\left(\frac{d}{n}\right) = \left(\frac{d}{n}\right)^d \left(1 - \frac{d}{n}\right)^{n-d}.$$

Since we assume that $d \leq \frac{n}{2}$, we conclude that for all $x \in [0, 1]$ we have

$$f_d(x) \leq \left(\frac{d}{n}\right)^d \left(1 - \frac{d}{n}\right)^{n-d} = \left(\frac{d}{n}\right)^d \left(\left(1 - \frac{d}{n}\right)^{\frac{n}{d}-1}\right)^d \leq \left(\frac{d}{2n}\right)^d.$$

Hence we have both $p^d(1-p)^{n-d} \leq (\frac{d}{2n})^d$ and $(1-pc)^{n-d}(pc)^d \leq (\frac{d}{2n})^d$, from which we conclude

$$\begin{aligned} P &\leq \lambda_M p^d (1-p)^{n-d} + \lambda_M \lambda_C (1-pc)^{n-d}(pc)^d \\ &\leq (\lambda_M + \lambda_M \lambda_C)\left(\frac{d}{2n}\right)^d \leq 2\lambda_M \lambda_C \left(\frac{d}{2n}\right)^d. \end{aligned}$$

Since P cannot exceed one, we also have

$$\begin{aligned} P &\leq \min\left\{1, \lambda_M p^d (1-p)^{n-d} + \lambda_M \lambda_C (1-pc)^{n-d}(pc)^d\right\} \\ &\leq \min\left\{1, 2\lambda_M \lambda_C \left(\frac{d}{2n}\right)^d\right\}. \end{aligned}$$

□

An important corollary from Lemma 19 is the following lower bound for the case when we use too small population sizes.

Corollary 20 *Consider the run of the* $(1 + (\lambda, \lambda))$ *GA with static parameters on* $\textsc{Jump}_k$ *with* $k < \frac{n}{2}$. *Let the population sizes which are used for the mutation and crossover phases be* λ_M *and* λ_C *respectively. Let also the current individual* x *be a point outside the fitness valley, but with at least* $\frac{n}{2}$ *one-bits. Then if* $\lambda_M \lambda_C < \ln(\frac{n}{k})(\frac{2n}{k})^{k-1}$, *then the expected runtime until we find the global optimum is at least* $\frac{1}{2\sqrt{\ln(n/k)}}(\frac{2n}{k})^{\frac{k+1}{2}}$.

Proof Since the algorithm has already found the point outside the fitness valley, it will never accept a point inside it as the current individual x. Hence, unless we find the optimum, the distance to it from the current individual is at least k and at most $\frac{n}{2}$.

We now consider the term $(\frac{d}{2n})^d$, which is used in the bound given in Lemma 19, as a function of d and maximize it for $d \in [k, \frac{n}{2}]$. For this purpose we consider its values in the ends of the interval and in the zeros of its derivative, which is,

$$\left(\left(\frac{d}{2n}\right)^d\right)' = \left(\frac{d}{2n}\right)^d \left(d \ln\left(\frac{d}{2n}\right)\right)' = \left(\frac{d}{2n}\right)^d \left(\ln\left(\frac{d}{2n}\right) + 1\right).$$

Hence, the derivative is equal to zero only when $\frac{d}{2n} = e^{-1}$, that is, when $d = \frac{2n}{e}$. Since we only consider d which are at most $\frac{n}{2}$, the derivative does not have roots in this range. We also note that for $d < \frac{2n}{e}$ the derivative is negative, hence the maximal value of $(\frac{d}{2n})^d$ is reached when $d = k$. Therefore, by Lemma 19 we have

$$\begin{aligned} P &\leq \min\left\{1, 2\lambda_M\lambda_C\left(\frac{d}{2n}\right)^d\right\} \\ &\leq \min\left\{1, 2\lambda_M\lambda_C\left(\frac{k}{2n}\right)^k\right\}. \end{aligned} \tag{2}$$

Since $\lambda_M\lambda_C \leq \ln(\frac{n}{k})(\frac{2n}{k})^{k-1}$ and since for all $x \geq 2$ we have $\ln(x) < \frac{x}{2}$, we compute

$$2\lambda_M\lambda_C\left(\frac{k}{2n}\right)^k \leq 2\ln\left(\frac{n}{k}\right)\left(\frac{2n}{k}\right)^{k-1}\left(\frac{k}{2n}\right)^k = \ln\left(\frac{n}{k}\right)\cdot\frac{k}{n} \leq \frac{1}{2}.$$

Therefore, the minimum in (2) is equal to the second argument.

Thus, the runtime T_I (in terms of iterations) is dominated by the geometric distribution with parameter $2\lambda_M\lambda_C\left(\frac{k}{2n}\right)^k \leq \frac{1}{2}$. This implies that the expected number of unsuccessful iterations is $E[T_I] - 1 \geq \frac{E[T_I]}{2}$. Since in each unsuccessful iteration we have exactly $\lambda_M + \lambda_C$ fitness evaluation, we have

$$E[T_F] = (\lambda_M + \lambda_C)\frac{E[T_I]}{2} \geq \frac{\lambda_M + \lambda_C}{4\lambda_M\lambda_C\left(\frac{k}{2n}\right)^k} = \left(\frac{1}{\lambda_M} + \frac{1}{\lambda_C}\right)\cdot\frac{1}{4}\left(\frac{2n}{k}\right)^k.$$

By Lemma 7 we obtain

$$\begin{aligned} E[T_F] &\geq \left(\frac{1}{\lambda_M} + \frac{1}{\lambda_C}\right)\cdot\frac{1}{4}\left(\frac{2n}{k}\right)^k \geq \frac{1}{2}\sqrt{\frac{1}{\lambda_M\lambda_C}}\left(\frac{2n}{k}\right)^k \\ &\geq \frac{1}{2}\sqrt{\frac{1}{\ln\left(\frac{n}{k}\right)}\left(\frac{k}{2n}\right)^{k-1}}\cdot\left(\frac{2n}{k}\right)^k = \frac{1}{2\sqrt{\ln\left(\frac{n}{k}\right)}}\left(\frac{2n}{k}\right)^{\frac{k+1}{2}}. \end{aligned}$$

□

In the following lemma we also show that too large population sizes also yield a too large expected runtime.

Lemma 21 *Consider the run of the* $(1 + (\lambda, \lambda))$ *GA with static parameters on* $\textsc{Jump}_k$ *with* $k < \frac{n}{2}$. *Let the population sizes which are used for the mutation and crossover phases be* λ_M *and* λ_C *respectively. Let also the current individual* x *be a point outside the fitness valley, but with at least* $\frac{n}{2}$ *one-bits. Then if* $\lambda_M\lambda_C > \frac{1}{\ln(\frac{n}{k})}(\frac{2n}{k})^{k+1}$, *then the expected runtime until we find the global optimum is at least* $\frac{1}{16\sqrt{\ln(n/k)}}(\frac{2n}{k})^{\frac{k+1}{2}}$.

Proof By Lemma 7 we have that the cost of one iteration is

$$\lambda_M + \lambda_C \geq 2\sqrt{\lambda_M \lambda_C} > \frac{2}{\sqrt{\ln\left(\frac{n}{k}\right)}} \left(\frac{2n}{k}\right)^{\frac{k+1}{2}},$$

hence to prove this lemma it is enough to consider only the first iteration of the algorithm, which already takes at least $\frac{2}{\sqrt{\ln(n/k)}}(\frac{2n}{k})^{\frac{k+1}{2}}$ fitness evaluations plus one evaluation for the initial individual.

We first show that if $\lambda_M \geq 2^{\frac{n}{2}} - 2$, then we are not likely to sample the optimum before making $2^{\frac{n}{2}} - 1$ fitness evaluations. For this we note that the initial individual and all mutation offspring in the first iteration are sampled independently of the fitness function, thus they are random points in the search space. Therefore, for each of these individuals the probability to be the optimum is 2^{-n}. Consequently, by the union bound, when we create the initial individual and $2^{\frac{n}{2}} - 2$ mutation offspring, the probability that at least one of them is the optimum is at most

$$\frac{2^{\frac{n}{2}} - 1}{2^n} = 2^{-\frac{n}{2}}(1 - 2^{-n}) \leq 2^{-(\frac{n}{2}+1)}.$$

Hence, with probability at least $(1 - 2^{-(\frac{n}{2}+1)})$ we have to make $2^{\frac{n}{2}} - 1$ or more fitness evaluations, which implies that

$$E[T_F] \geq \left(1 - 2^{-(\frac{n}{2}+1)}\right)\left(2^{\frac{n}{2}} - 1\right) = 2^{\frac{n}{2}} - \frac{3}{2} + 2^{-(\frac{n}{2}+1)} \geq 2^{\frac{n}{2}-1},$$

if $n \geq 3$. Without proof we note that $\frac{1}{16\sqrt{\ln(n/k)}}(\frac{2n}{k})^{\frac{k+1}{2}}$ is increasing in k for $k \leq \frac{n}{2}$. Hence, for $k \leq \frac{n}{2}$ we have that

$$\frac{1}{16\sqrt{\ln(n/k)}} \left(\frac{2n}{k}\right)^{\frac{k+1}{2}} \leq \frac{1}{16\sqrt{\ln(2)}} \cdot 4^{\left(\frac{n}{4}+\frac{1}{2}\right)} \leq 2^{\frac{n}{2}-2} \leq E[T_F].$$

In the rest of the proof we assume that $\lambda_M < 2^{\frac{n}{2}} - 2$. Since the mutation winner is chosen based on the fitness, we cannot use the same argument with random points in the search space for the crossover phase. However, we can consider an artificial process, which in parallel runs the crossover phase for each mutation offspring seen as winner. If none of these parallel processes has generated the optimum within m crossover offspring samples, then also the true process has not done so within a total of $1 + \lambda_M + m$ fitness evaluations. We note that in the parallel crossover phases, since no selection has been made, again all offspring are uniformly distributed in $\{0, 1\}^n$.

Let us fix $m = 2^{\frac{n}{2}-1}$. By the union bound, the probability that one of $1+\lambda_M+m\lambda_M$ individuals generated by the artificial process is the optimum is at most

$$\frac{1 + \lambda_M + m\lambda_M}{2^n} < \frac{1 + \left(2^{\frac{n}{2}} - 2\right)(1 + m)}{2^n} = \frac{1 + \left(2^{\frac{n}{2}} - 2\right)\left(2^{\frac{n}{2}-1} + 1\right)}{2^n}$$

$$= \frac{1 + \frac{1}{2}(2^n - 4)}{2^n} \leq \frac{1}{2}.$$

At the same time, if the original $(1 + (\lambda, \lambda))$ GA creates $1 + \lambda_M + m \geq m$ individuals, it also performs at least m fitness evaluations. Hence, the expected number of fitness evaluations is at least

$$E[T_F] \geq \frac{m}{2} = 2^{\frac{n}{2}-2} \geq \frac{1}{16\sqrt{\ln(n/k)}} \left(\frac{2n}{k}\right)^{\frac{k+1}{2}}.$$

□

We are now in position to prove Theorem 17.

Proof of Theorem 17 **Initialization.** Recall that the initial individual is sampled uniformly at random, hence the number of one-bits in it follows a binomial distribution $\mathrm{Bin}(n, \frac{1}{2})$. By the symmetry argument we have that the number of one-bits X in the initial individual is at least $\frac{n}{2}$ with probability at least $\frac{1}{2}$. By Chernoff bounds (see, e.g., Theorem 1.10.1 in [24]) we also have that the probability that X is greater than $\frac{n}{2} + \frac{n}{8}$ is at most

$$\Pr\left[X \geq \left(1 + \frac{1}{4}\right)\frac{n}{2}\right] \leq \exp\left(-\frac{n/2}{48}\right) = e^{-\Theta(n)}.$$

Hence, with probability at least $\frac{1}{2} - e^{-\Theta(n)}$ the initial individual has a number of one-bits (and hence, the fitness) in $[\frac{n}{2}, \frac{5n}{8}]$. We now condition on this event[3].

Narrowing the reasonable population sizes. Since we condition on starting in distance $d \leq \frac{n}{2}$ from the optimum of $\textsc{Jump}_k$, by Corollary 20 and Lemma 21 we have that if we choose λ_M and λ_C such that $\lambda_M\lambda_C \geq \frac{1}{\ln(n/k)}(\frac{2n}{k})^{k+1}$ or $\lambda_M\lambda_C \leq \ln(\frac{n}{k})(\frac{2n}{k})^{k-1}$, then the expected runtime is at least $\frac{1}{16\sqrt{\ln(n/k)}}(\frac{2n}{k})^{\frac{k+1}{2}} = \frac{91B}{16}$. Hence, in the rest of the proof we assume that $\ln(\frac{n}{k})(\frac{2n}{k})^{k-1} < \lambda_M\lambda_C < \frac{1}{\ln(n/k)}(\frac{2n}{k})^{k+1}$.

We note that by Lemma 7 this assumption also implies that the cost of one iteration is

$$\lambda_M + \lambda_C \geq 2\sqrt{\lambda_M\lambda_C} \geq 2\sqrt{\ln\left(\frac{n}{k}\right)}\left(\frac{2n}{k}\right)^{\frac{k-1}{2}}.$$

Narrowing the reasonable mutation rate and crossover bias. We now show that using a too large mutation rate or crossover bias also yields a runtime which is greater than $(\frac{2n}{k})^{\frac{k+1}{2}}$ and therefore greater than B. Conditional on the current individual x

[3] Without proof we note that if the initial individual has less than $\frac{n}{2}$ one-bits, our lower bound would also hold. However, the proof of this fact would require more complicated arguments, hence in order to increase the readability of the paper we avoid considering that case.

being in distance $d \leq \frac{n}{2}$ from the optimum, by Lemma 19 we have that if $pc \geq \frac{1}{2}$ (and therefore, $p \geq \frac{1}{2}$), then we have

$$\begin{aligned} P &\leq \lambda_M (1-p)^{n/2} + \lambda_M \lambda_C (1-pc)^{n/2} \\ &\leq \lambda_M \left(\frac{1}{2}\right)^{\frac{n}{2}} + \lambda_M \lambda_C \left(\frac{1}{2}\right)^{\frac{n}{2}} \leq 2\lambda_M \lambda_C \left(\frac{1}{2}\right)^{\frac{n}{2}}. \end{aligned}$$

Therefore, the expected number of fitness evaluations until we find the optimum is at least

$$E[T_F] \geq \frac{\lambda_M + \lambda_C}{P} \geq \left(\frac{1}{\lambda_M} + \frac{1}{\lambda_C}\right) \cdot 2^{\frac{n}{2}-1}.$$

Since we already assume that $\lambda_M \lambda_C \leq \frac{1}{\ln(n/k)} (\frac{2n}{k})^{k+1}$, by Lemma 7 we have

$$\frac{1}{\lambda_M} + \frac{1}{\lambda_C} \geq 2\sqrt{\frac{1}{\lambda_M \lambda_C}} \geq 2\sqrt{\ln\left(\frac{n}{k}\right)} \left(\frac{k}{2n}\right)^{\frac{k+1}{2}}.$$

Therefore, we have

$$E[T_F] \geq 2^{\frac{n}{2}} \ln\left(\frac{n}{k}\right) \left(\frac{k}{2n}\right)^{\frac{k+1}{2}} = 2^{\frac{n}{2}} \ln\left(\frac{n}{k}\right) \left(\frac{k}{2n}\right)^{k+1} \left(\frac{2n}{k}\right)^{\frac{k+1}{2}}.$$

We note that for $k \in [1, \frac{n}{512}]$ the term $\left(\frac{k}{2n}\right)^{k+1}$ is decreasing in k (we avoid the proof of this fact, but note that it trivially follows from considering the derivative). Consequently, if we assume that $k \leq \frac{n}{512}$, then we have

$$\begin{aligned} E[T_F] &\geq 2^{\frac{n}{2}} \ln(512) \left(\frac{1}{1024}\right)^{\frac{n}{512}+1} \left(\frac{2n}{k}\right)^{\frac{k+1}{2}} = 2^{\frac{n}{2} - 10(\frac{n}{512}+1)} \cdot \ln(512) \left(\frac{2n}{k}\right)^{\frac{k+1}{2}} \\ &= \frac{\ln(512)}{1024} \cdot 2^{\frac{123n}{256}} \left(\frac{2n}{k}\right)^{\frac{k+1}{2}} \geq \frac{\ln(512)}{1024} \cdot 2^{\frac{246}{256}} \left(\frac{2n}{k}\right)^{\frac{k+1}{2}} \geq \frac{1}{91} \left(\frac{2n}{k}\right)^{\frac{k+1}{2}}. \end{aligned}$$

Hence, using $pc \geq \frac{1}{2}$ gives us the expected runtime which is not less than B.

Making a linear progress. For the rest of the proof we assume that we have population sizes such that $\lambda_M \lambda_C \in [\ln(\frac{n}{k})(\frac{2n}{k})^{k-1}, \frac{1}{\ln(n/k)}(\frac{2n}{k})^{k+1}]$ and p and c such that $pc \leq \frac{1}{2}$. We now show that at some iteration before we have already made at least $(\frac{2n}{k})^{\frac{k+1}{2}}$ fitness evaluations we get a current individual x with fitness in $[\frac{n}{2} + \frac{n}{8}, \frac{n}{2} + \frac{n}{4}]$. For this we show that conditional on $f(x) \geq \frac{n}{2}$ we are not likely to increase fitness in one iteration by at least $\frac{n}{8}$ in a very long time.

For this purpose we consider a modified iteration of the $(1 + (\lambda, \lambda))$ GA, where in the crossover phase we create not only λ_C offspring by crossing the current individual x with the mutation winner x', but we create $\lambda_M \cdot \lambda_C$ offspring by performing crossover

between x and each mutation offspring λ_C times. The best offspring in this modified iteration cannot be worse than the best offspring in a non-modified iteration. Hence the probability that we increase the fitness by a least $\frac{n}{8}$ is at most the probability that the best offspring of this modified iteration is better than the current individual x by at least $\frac{n}{8}$.

Consider one particular offspring y' created in this modified iteration. Recall that its parent was created by first choosing a number ℓ from the binomial distribution $\mathrm{Bin}(n, p)$ and then flipping ℓ bits, therefore it is distributed as if we created it by flipping each bit independently with probability p. Then when we create y' we take each flipped bit from its parent with probability c, hence in the resulting offspring each bit is flipped with probability pc, independently of other bits. Consequently the distribution of y' is the same as if we created it via the standard bit mutation with probability of flipping each bit equal to pc. Note that this argument works only when we consider one particular individual, since the mutation offspring are dependent on each other (since they have the same number ℓ of bits flipped) and therefore, their offspring are all also dependent.

To estimate the probability that y' has a fitness by $\frac{n}{8}$ greater than x, we use Lemma 18 with $r = pcn$ (note that since $pc < \frac{1}{2}$, we have $r \leq \frac{n}{2}$, thus we satisfy the conditions of Lemma 18). Since we are conditioning on $m = f(x) \geq \frac{n}{2}$, we have $(n-2m)\frac{r}{n} \leq 0$. Hence, with $\Delta = \frac{n}{8}$ we obtain

$$\begin{aligned}
\Pr\left[f(y') - f(x) \geq \frac{n}{8}\right] &\leq \Pr\left[f(y') - f(x) \geq (n - 2f(x))\frac{r}{n} + \Delta\right] \\
&\leq \exp\left(\frac{-\Delta^2}{2(1-r/n)(r+\Delta/3)}\right) \\
&\leq \exp\left(\frac{-\left(\frac{n}{8}\right)^2}{2\left(pcn + \frac{n}{24}\right)}\right) \leq \exp\left(\frac{-\left(\frac{n}{8}\right)^2}{2\left(\frac{n}{2} + \frac{n}{24}\right)}\right) \\
&\leq \exp\left(-\frac{n^2}{64} \cdot \frac{12}{13n}\right) = e^{-\frac{3n}{208}} \leq e^{-\frac{n}{70}}.
\end{aligned}$$

After $\frac{n}{k}$ modified iterations we create $\frac{\lambda_M \lambda_C n}{k}$ offspring, therefore by the union bound the probability that at least one of them has a fitness by at least $\frac{n}{8}$ greater than the fitness of its parent is at most $\frac{\lambda_M \lambda_C n}{k} e^{-\frac{n}{70}}$. Since we also have $\lambda_M \lambda_C \leq \frac{1}{\ln(n/k)}(\frac{2n}{k})^{k+1}$, this probability is at most

$$\frac{1}{\ln\left(\frac{n}{k}\right)}\left(\frac{2n}{k}\right)^{k+1} \cdot \frac{n}{k} \cdot e^{-\frac{3n}{16}} = \frac{1}{2\ln\left(\frac{n}{k}\right)}\left(\frac{2n}{k}\right)^{k+2} e^{-\frac{n}{70}} \leq \frac{1}{2}\left(\frac{2n}{k}\right)^{k+2} e^{-\frac{n}{70}}.$$

We also note that the term $\left(\frac{2n}{k}\right)^{k+2}$ is increasing in k for $k \in [1, \frac{n}{512}]$ (we omit the proof, since it trivially follows from considering its derivative). Therefore, for such k this probability is at most

$$\frac{1}{2} \cdot 1024^{\frac{n}{512}+2} \cdot e^{-\frac{n}{70}} = 2^{19} \exp\left(\frac{\ln(1024)n}{512} - \frac{n}{70}\right) \leq 2^{19} e^{-0.0007n} = e^{-\Theta(n)}.$$

Let $T_{5n/8}$ be the first iteration when we have x with at least $\frac{5n}{8}$ one-bits. If $T_{5n/8} \leq \frac{n}{k}$, then with probability $1 - e^{-\Theta(n)}$ none of the offspring created up to this moment improved the fitness by more than $\frac{n}{8}$. Hence, we have that x has at most $\frac{5n}{8} + \frac{n}{8} = \frac{3n}{4}$ one-bits. Otherwise, if $T_{5n/8} > \frac{n}{k}$, by this iteration we already make at least

$$(\lambda_M + \lambda_C)\frac{n}{k} \geq 2\sqrt{\ln\left(\frac{n}{k}\right)}\left(\frac{2n}{k}\right)^{\frac{k-1}{2}}\frac{n}{k} \geq \left(\frac{2n}{k}\right)^{\frac{k+1}{2}}.$$

fitness evaluations and thus the runtime exceeds $(\frac{2n}{k})^{\frac{k+1}{2}}$. Hence, in the rest of the proof we assume that at some point we have a current individual x with fitness in $[\frac{5n}{8}, \frac{3n}{4}]$.

Slow progress towards the local optimum. We now show that after reaching fitness at least $\frac{5n}{8}$, the $(1 + (\lambda, \lambda))$ GA makes a progress not greater than $\delta := \frac{26}{3}(k+2)\ln(\frac{n}{k})$ per iteration.

For this purpose we again consider an iteration of the modified algorithm, which generates $\lambda_M\lambda_C$ offspring in each iteration. Recall that each offspring created here can be considered as one created by standard bit mutation with mutation rate pc. We apply Lemma 18 to one particular offspring y' with $r = pcn$ and $\Delta = \frac{r}{4} + \delta$ and obtain

$$\begin{aligned}
\Pr\left[f(y') - f(x) \geq \delta\right] &= \Pr\left[f(y') - f(x) \geq (n - 2f(x))\frac{r}{n}\right. \\
&\quad \left. +(2f(x) - n)\frac{r}{n} + \delta\right] \\
&\leq \Pr\left[f(y') - f(x) \geq (n - 2f(x))\frac{r}{n} + \frac{r}{4} + \delta\right] \\
&\leq \exp\left(-\frac{\left(\frac{r}{4} + \delta\right)^2}{2\left(1 - \frac{r}{n}\right)\left(r + \frac{r}{12} + \frac{\delta}{3}\right)}\right) \\
&\leq \exp\left(-\frac{\left(\frac{r}{4} + \delta\right)^2}{2\left(\frac{13r}{12} + \frac{\delta}{3}\right)}\right).
\end{aligned}$$

For all $\delta > 0$ and $r > 0$ we bound the argument of the exponent as follows.

$$\begin{aligned}
\frac{\left(\frac{r}{4} + \delta\right)^2}{2\left(\frac{13r}{12} + \frac{\delta}{3}\right)} &= \frac{3}{2} \cdot \frac{\left(\frac{13r}{4} + \delta - 3r\right)\left(\frac{r}{4} + \delta\right)}{\left(\frac{13r}{4} + \delta\right)} = \frac{3}{2}\left(1 - \frac{3r}{\frac{13r}{4} + \delta}\right)\left(\frac{r}{4} + \delta\right) \\
&\geq \frac{3}{2}\left(1 - \frac{12}{13}\right)\delta = \frac{3}{26}\delta.
\end{aligned}$$

Recall that $\delta = \frac{26}{3}(k+2)\ln(\frac{n}{k})$. Hence, we have

$$\Pr\left[f(y') - f(x) \geq \delta\right] \leq \exp\left(-\frac{\left(\frac{r}{4}+\delta\right)^2}{2\left(\frac{13r}{12}+\frac{\delta}{3}\right)}\right) \leq e^{-\frac{3\delta}{26}}$$
$$= \exp\left(-(k+2)\ln\left(\frac{n}{k}\right)\right) = \left(\frac{n}{k}\right)^{-(k+2)}.$$

By the union bound the probability that we create such offspring in $\frac{n/4-k}{\delta}$ iterations is at most

$$\frac{\frac{n}{4}-k}{\delta}\lambda_M\lambda_C\left(\frac{n}{k}\right)^{-(k+2)} \leq \frac{n}{4\delta}\cdot\frac{1}{\ln\left(\frac{n}{k}\right)}\left(\frac{n}{k}\right)^{k+1}\left(\frac{n}{k}\right)^{-(k+2)}$$
$$\leq \frac{k}{4\delta} = \frac{3k}{26(k+2)\ln\left(\frac{n}{k}\right)} \leq \frac{3}{26}.$$

If we start at some point x with fitness $f(x) \leq \frac{3n}{4}$ and we do not improve fitness by at least δ for $\frac{n/4-k}{\delta}$ iterations, then we do not reach the local optima or the global optimum in this number of iterations (note that for the considered values of $k \leq \frac{n}{32}$ and δ the value of $\frac{n/4-k}{\delta}$ is at least one). During these iterations we do at least $(\lambda_M+\lambda_C)\frac{n/4-k}{\delta}$ fitness evaluations. Since we have already shown that $\lambda_M + \lambda_C \geq 2\sqrt{\ln(\frac{n}{k})}(\frac{n}{k})^{\frac{k-1}{2}}$, this is at least

$$2\sqrt{\ln\left(\frac{n}{k}\right)}\left(\frac{n}{k}\right)^{\frac{k-1}{2}}\cdot\frac{\frac{n}{4}-k}{\delta} = \left(\frac{n}{k}\right)^{\frac{k+1}{2}}\cdot\frac{(n-4k)k}{2\delta n}\sqrt{\ln\left(\frac{n}{k}\right)}$$

fitness evaluations. We now estimate the factor $\frac{(n-4k)k}{2\delta}\sqrt{\ln(\frac{n}{k})}$. Since by the theorem conditions we have $k \leq \frac{n}{32}$, for n large enough we have

$$\frac{(n-4k)k}{2\delta n}\sqrt{\ln\left(\frac{n}{k}\right)} \geq \frac{7nk}{16\delta n}\sqrt{\ln\left(\frac{n}{k}\right)}$$
$$= \frac{21k\sqrt{\ln\left(\frac{n}{k}\right)}}{416(k+2)\ln\left(\frac{n}{k}\right)} \geq \frac{k}{20\cdot 2k\sqrt{\ln\left(\frac{n}{k}\right)}} \geq \frac{1}{40\sqrt{\ln\left(\frac{n}{k}\right)}}.$$

Summary of the proof. We now bring our arguments together. For the narrowed range of parameters we have shown that (i) with probability $\frac{1}{2} - e^{-\Theta(n)}$ the initial individual has between $\frac{n}{2}$ and $\frac{5n}{8}$ one-bits, (ii) then with probability $1-e^{-\Theta(n)}$ we reach a point which has between $\frac{5n}{8}$ and $\frac{3n}{4}$ one-bits or exceed $\left(\frac{n}{k}\right)^{\frac{k+1}{2}}$ fitness evaluations, (iii) then with probability at least $1 - \frac{3}{26} = \frac{23}{26}$ we do not reach the local optima or the global optimum in $\frac{1}{40\sqrt{\ln(n/k)}}\left(\frac{n}{k}\right)^{\frac{k+1}{2}}$ fitness evaluations. Hence, with probability

at least

$$\left(\frac{1}{2} - e^{-\Theta(n)}\right)\left(1 - e^{-\Theta(n)}\right)\frac{23}{26} = \frac{23}{52} - e^{-\Theta(n)}$$

we do not find the optimum before making $\frac{1}{40\sqrt{\ln(n/k)}}\left(\frac{n}{k}\right)^{\frac{k+1}{2}}$ fitness evaluations. Therefore, the expected runtime T_F (in terms of fitness evaluations) is at least

$$E[T_F] \geq \left(\frac{23}{52} - e^{-\Theta(n)}\right)\frac{1}{40\sqrt{\ln\left(\frac{n}{k}\right)}}\left(\frac{n}{k}\right)^{\frac{k+1}{2}} \geq \frac{1}{91\sqrt{\ln\left(\frac{n}{k}\right)}}\left(\frac{n}{k}\right)^{\frac{k+1}{2}}.$$

□

4 Experiments

As our theoretical analysis gives upper bounds that are precise only up to constant factors, we now use experiments to obtain a better understanding of how the heavy-tailed $(1 + (\lambda, \lambda))$ GA performs on concrete problem sizes. We conducted a series of experiments on OneMax and Jump functions with jump sizes $k \in [2..6]$.

Since our theory-based recommendations for β_p and β_c are very similar, the analysis on the jump functions treats the corresponding distributions very symmetrically, and our preliminary experimentation did not find any significant advantages from using different values for β_p and β_c, we decided to keep them equal in our experiments and denote them together as β_{pc}, such that $\beta_p = \beta_c = \beta_{pc}$.

In all the presented plots we display average values of 100 independent runs of the considered algorithm, together with the standard deviation.

4.1 Results for OneMax

For OneMax we considered the problem sizes $n \in \{2^i \mid 3 \leq i \leq 14\}$ and all combinations of choices of $\beta_{pc} \in \{1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2\}$ and $\beta_\lambda \in \{2.0, 2.2, 2.4, 2.6, 2.8, 3.0, 3.2\}$. For reasons of space, we only present a selection of these results.

Figure 3 presents the running times of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on OneMax against the problem size n for all considered values of β_{pc}, whereas a fixed value of $\beta_\lambda = 2.8$ is used. The running times of the (1+1) EA are also presented for comparison. Since the runtime is normalized by $n \ln(n)$, the plot of the latter tends to a horizontal line, and so do the plots of the heavy-tailed $(1 + (\lambda, \lambda))$ GA with $\beta_{pc} \geq 1.8$. Other plots, after discounting for the noise in the measurements, appear to be convex upwards and, similarly to [2], they will likely become horizontal as n grows. For OneMax, bigger values of β_{pc} appear to be better. Since greater β_{pc} increases the chances of behaving similar to the (1+1) EA during an iteration, this fits to the situation discussed in Lemma 9. The plots look similar also for β_λ different from 2.8, so we do not present them here.

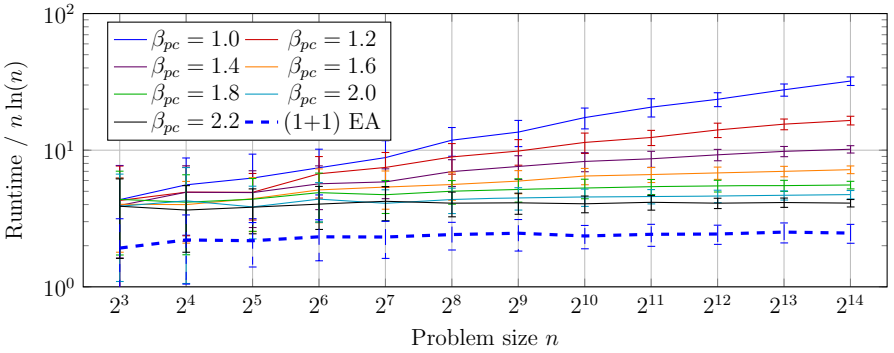

Fig. 3 Running times of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on OneMax starting from a random point, normalized by $n \ln(n)$, for different $\beta_{pc} = \beta_p = \beta_c$ and $\beta_\lambda = 2.8$ in relation to the problem size n. The expected running times of $(1 + 1)$ EA, also starting from a random point, are given for comparison

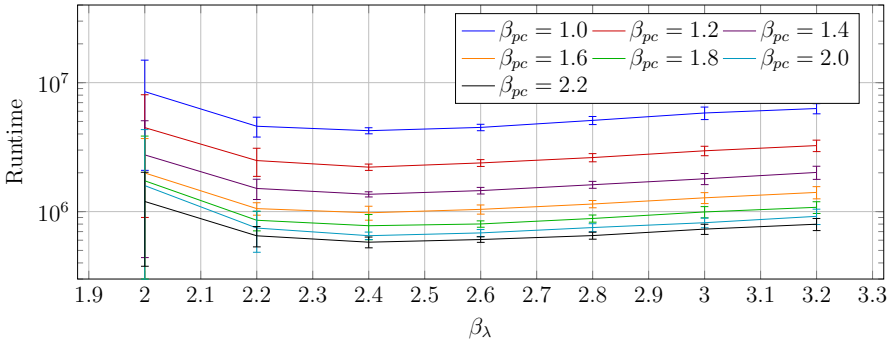

Fig. 4 Running times of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on OneMax starting from a random point, for $n = 2^{14}$ and different $\beta_{pc} = \beta_p = \beta_c$ depending on β_λ

To investigate the dependencies on β_λ and β_{pc} more thoroughly, we consider the largest available problem size $n = 2^{14}$ and plot the runtimes for all parameters configurations in Fig. 4. The general trend of an improving performance with growing β_{pc} can be clearly seen here as well. For β_λ, the picture is less clear. It appears that very small β_λ also result in larger running times, medium values of roughly $\beta_\lambda = 2.4$ yield the best available runtimes, and a further increase of β_λ increases the runtime again, but only slightly. As very large β_λ, such as $\beta_\lambda = 3.2$, correspond to regimes similar to the $(1 + 1)$ EA, this might be a sign that some of the working principles of the $(1 + (\lambda, \lambda))$ GA are still beneficial on an easy problem like OneMax.

4.2 Results for Jump Functions

For Jump functions we used the problem sizes $n \in \{2^i \mid 3 \leq i \leq 7\}$, subject to the condition $k \leq \frac{n}{4}$ and hence $n \geq 4k$, as assumed in the theoretical results of this paper. As running times are higher in this setting, we consider a smaller set of parameter combinations, $\beta_{pc} \in \{1.0, 1.2, 1.4\}$ and $\beta_\lambda \in \{2.0, 2.2, 2.4\}$.

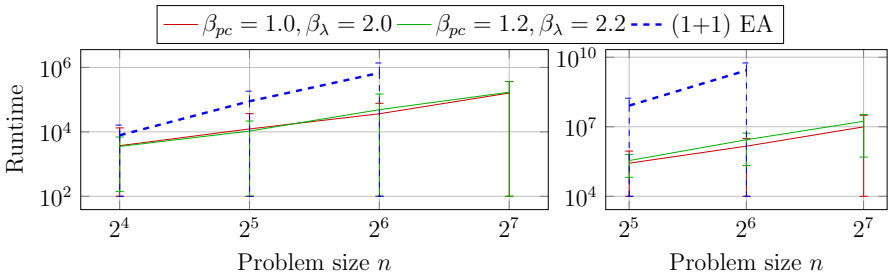

Fig. 5 Running times of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on JUMP, depending on the problem size n, in comparison to the $(1 + 1)$ EA. Jump sizes are $k = 3$ on the left and $k = 5$ on the right

Table 3 The p values for experimental results presented in Fig. 5

k	n	Student's t test	Wilcoxon rank sum test
3	16	1.46×10^{-3}	4.14×10^{-6}
3	32	1.88×10^{-12}	2.87×10^{-20}
3	64	2.59×10^{-14}	5.29×10^{-26}
5	32	9.32×10^{-15}	1.58×10^{-34}
5	64	8.01×10^{-15}	1.58×10^{-34}

Figure 5 presents the results of a comparison of the heavy-tailed $(1 + (\lambda, \lambda))$ GA with the $(1 + 1)$ EA on JUMP with jump parameter $k \in \{3, 5\}$. We chose two most distant distribution parameters for the heavy-tailed $(1 + (\lambda, \lambda))$ GA for presenting in this figure. However, the difference between these is negligible compared to the difference to the $(1 + 1)$ EA. Such a difference aligns well with the theory, as the running time of the $(1 + 1)$ EA is $\Theta(n^k)$, whereas Theorem 13 predicts much smaller running times for the heavy-tailed $(1 + (\lambda, \lambda))$ GA.

Due to large standard deviation, we performed statistical tests on the results presented in Fig. 5 using two statistical tests: the Student's t test as the one which checks mean values which are the subject of our theorems, and the Wilcoxon rank sum test as a non-parametric test. The results are presented in Table 3, where for each row and each test the maximum p value is shown out of two between the $(1 + 1)$ EA and either of the parameterizations of the heavy-tailed $(1 + (\lambda, \lambda))$ GA. The p values in all the cases are very small: except for the case $k = 3$, $n = 16$, they are all well below 10^{-10}, which indicates a vast difference between the algorithms and hence a clear superiority of the heavy-tailed $(1 + (\lambda, \lambda))$ GA.

The parameter study, presented in Fig. 6, suggests that for JUMP the particular values of β_{pc} are not very important, although larger values result in the marginally better performance. However, larger β_λ tend to make the performance worse, which is more pronounced for larger jump sizes k. This finding agrees with upper bound proven in Corollary 16, in which (n/k) is raised to a power that is proportional to $\varepsilon_\lambda = \beta_\lambda - 1$. Each difference is statistically significant with $p < 0.008$ using the Wilcoxon rank sum test.

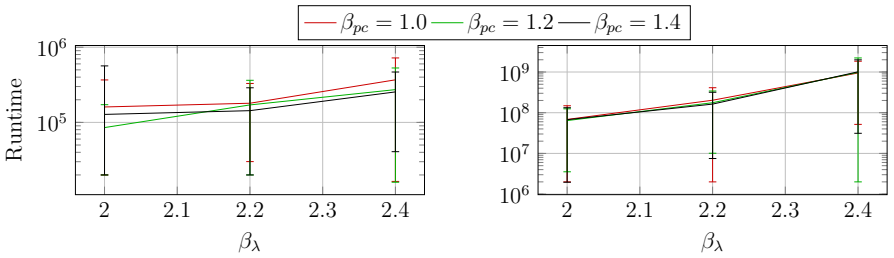

Fig. 6 Dependency of running times of the heavy-tailed $(1 + (\lambda, \lambda))$ GA on JUMP_k on β_λ and β_{pc} for $k = 3$ (on the left) and $k = 6$ (on the right). Problem size $n = 2^7$ is used

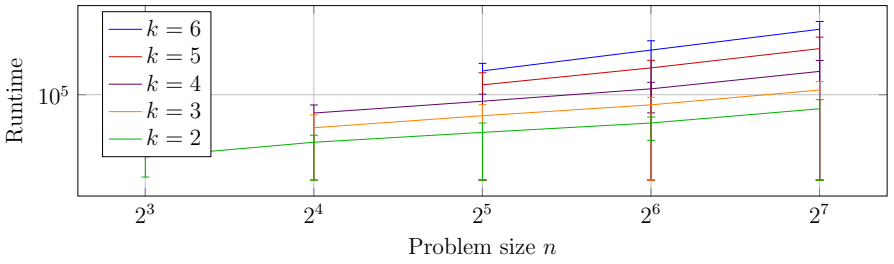

Fig. 7 Running times of the heavy-tailed $(1 + (\lambda, \lambda))$ GA with $\beta_{pc} = 1.0$ and $\beta_\lambda = 2.0$ on JUMP_k for $k \in [2..6]$ depending on the problem size n

Finally, Fig. 7 shows the running times of the heavy-tailed $(1 + (\lambda, \lambda))$ GA for a fixed parameterization $\beta_{pc} = 1.0$ and $\beta_\lambda = 2.0$ for all available values of n and k, to give an impression of the typical running times of this algorithm on the JUMP problem.

5 Conclusion

Using mathematical and experimental methods, we showed that choosing all parameters of an algorithm randomly from a power-law distribution can lead to a very good performance both on easy unimodal and on multimodal problems. This lazy approach to the parameter tuning and control problem requires very little understanding how the parameters influence the algorithm behavior. The only design choice left to the algorithm user is deciding the scaling of the parameters, but we observed that the natural choices worked out very well. Our empirical and theoretical studies show that the precise choice of the (other) parameters of the power-law distributions does not play a significant role and they both suggest to use unbounded power-law distributions and to take a power-law exponent $2 + \varepsilon$ for the population size (so that the expected cost of one iteration is constant) and $1 + \varepsilon$ for other parameters (to maximize the positive effect of a heavy-tailed distribution). With these considerations, one may call our approach essentially *parameter-less*.

Surprisingly, our randomized parameter choice even yields a runtime which is better than the best proven runtime for optimal static parameters on JUMP functions of some jump sizes. An interesting question (which we leave open for the further research) is

whether a random parameter choice can outperform the algorithms with known optimal static parameters also on other problems. The experiments on OneMax when starting with a good solution (in distance $\sqrt{n}$ from the optimum) indicate that it is possible since there we observed a benefit from the key mechanisms of the $(1 + (\lambda, \lambda))$ GA.

Acknowledgements This work was supported by RFBR and CNRS, Project Number 20-51-15009 and by a public grant as part of the Investissements d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Antipov, D., Buzdalov, M., Doerr, B.: Lazy parameter tuning and control: choosing all parameters randomly from a power-law distribution. In: Genetic and Evolutionary Computation Conference, GECCO 2021, pp. 1115–1123. ACM (2021)
2. Antipov, D., Buzdalov, M., Doerr, B.: Fast mutation in crossover-based algorithms. Algorithmica **84**, 1724–1761 (2022)
3. Antipov, D., Doerr, B.: Runtime analysis of a heavy-tailed $(1 + (\lambda, \lambda))$ genetic algorithm on jump functions. In: Parallel Problem Solving From Nature, PPSN 2020, Part II, pp. 545–559. Springer (2020)
4. Antipov, D., Doerr, B.: A tight runtime analysis for the $(\mu + \lambda)$ EA. Algorithmica **83**, 1054–1095 (2021)
5. Antipov, D., Doerr, B., Karavaev, V.: A tight runtime analysis for the $(1 + (\lambda, \lambda))$ GA on LeadingOnes. In: Foundations of Genetic Algorithms, FOGA 2019, pp. 169–182. ACM (2019)
6. Antipov, D., Doerr, B., Karavaev, V.: A rigorous runtime analysis of the $(1 + (\lambda, \lambda))$ GA on jump functions. Algorithmica **84**, 1573–1602 (2022)
7. Benbaki, R., Benomar, Z., Doerr, B.: A rigorous runtime analysis of the 2-MMAS_{ib} on jump functions: ant colony optimizers can cope well with local optima. In: Genetic and Evolutionary Computation Conference, GECCO 2021, pp. 4–13. ACM (2021)
8. Buzdalov, M., Doerr, B.: Runtime analysis of the $(1 + (\lambda, \lambda))$ genetic algorithm on random satisfiable 3-CNF formulas. In: Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 1343–1350. ACM (2017)
9. Badkobeh, G., Lehre, P.K., Sudholt, D.: Unbiased black-box complexity of parallel search. In: Parallel Problem Solving from Nature, PPSN 2014, pp. 892–901. Springer (2014)
10. Corus, D., Oliveto, P.S., Yazdani, D.: On the runtime analysis of the Opt-IA artificial immune system. In: Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 83–90. ACM (2017)
11. Corus, D., Oliveto, P.S., Yazdani, D.: Artificial immune systems can find arbitrarily good approximations for the NP-hard number partitioning problem. Artif. Intell. **274**, 180–196 (2019)
12. Corus, D., Oliveto, P.S., Yazdani, D.: Automatic adaptation of hypermutation rates for multimodal optimisation. In: Foundations of Genetic Algorithms, FOGA 2021, pp. 4:1–4:12. ACM (2021)

13. Doerr, B., Doerr, C.: Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm. Algorithmica **80**, 1658–1709 (2018)
14. Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing new genetic algorithms. Theor. Comput. Sci. **567**, 87–104 (2015)
15. Doerr, B., Doerr, C., Kötzing, T.: Static and self-adjusting mutation strengths for multi-valued decision variables. Algorithmica **80**, 1732–1768 (2018)
16. Doerr, B., Doerr, C., Kötzing, T.: Solving problems with unknown solution length at almost no extra cost. Algorithmica **81**, 703–748 (2019)
17. Dang, D.-C., Eremeev, A.V., Lehre, P.K., Qin, X.: Fast non-elitist evolutionary algorithms with power-law ranking selection. In: Genetic and Evolutionary Computation Conference, GECCO 2022, pp. 1372–1380. ACM (2022)
18. Dang, D.-C., Friedrich, T., Kötzing, T., Krejca, M.S., Lehre, P.K., Oliveto, P.S., Sudholt, D., Sutton, A.M.: Escaping local optima with diversity mechanisms and crossover. In: Genetic and Evolutionary Computation Conference, GECCO 2016, pp. 645–652. ACM (2016)
19. Dang, D.-C., Friedrich, T., Kötzing, T., Krejca, M.S., Lehre, P.K., Oliveto, P.S., Sudholt, D., Sutton, A.M.: Escaping local optima using crossover with emergent diversity. IEEE Trans. Evol. Comput. **22**, 484–497 (2018)
20. Droste, S., Jansen, T., Wegener, I.: On the analysis of the $(1 + 1)$ evolutionary algorithm. Theor. Comput. Sci. **276**, 51–81 (2002)
21. Doerr, B., Johannsen, D., Winzen, C.: Multiplicative drift analysis. Algorithmica **64**, 673–697 (2012)
22. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 777–784. ACM (2017)
23. Doerr, B.: Optimal parameter settings for the $(1 + (\lambda, \lambda))$ genetic algorithm. In: Genetic and Evolutionary Computation Conference, GECCO 2016, pp. 1107–1114. ACM (2016)
24. Doerr, B.: Probabilistic tools for the analysis of randomized optimization heuristics. In: Doerr, B., Neumann, F. (eds.) Theory of Evolutionary Computation: Recent Developments in Discrete Optimization, pp. 1–87. Springer, Cham (2020) . arXiv:1801.06733
25. Doerr, B.: The runtime of the compact genetic algorithm on Jump functions. Algorithmica **83**, 3059–3107 (2021)
26. Doerr, B.: Does comma selection help to cope with local optima? Algorithmica **84**, 1659–1693 (2022)
27. Doerr, B., Qu, Z.: A first runtime analysis of the NSGA-II on a multimodal problem. In: Parallel Problem Solving From Nature, PPSN 2022. Springer (2022). arXiv:2204.13750
28. Doerr, B., Rajabi, A.: Stagnation detection meets fast mutation. In: Evolutionary Computation in Combinatorial Optimization, EvoCOP 2022, pp. 191–207. Springer (2022)
29. Doerr, B., Witt, C., Yang, J.: Runtime analysis for self-adaptive mutation rates. Algorithmica **83**, 1012–1053 (2021)
30. Doerr, B., Zheng, W.: Theoretical analyses of multi-objective evolutionary algorithms on multi-modal objectives. In: Conference on Artificial Intelligence, AAAI 2021, pp. 12293–12301. AAAI Press (2021)
31. Friedrich, T., Kötzing, T., Krejca, M.S., Nallaperuma, S., Neumann, F., Schirneck, M.: Fast building block assembly by majority vote crossover. In: Genetic and Evolutionary Computation Conference, GECCO 2016, pp. 661–668. ACM (2016)
32. Friedrich, T., Quinzan, F., Wagner, M.: Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In: Genetic and Evolutionary Computation Conference, GECCO 2018, pp. 293–300. ACM (2018)
33. Hasenöhrl, V., Sutton, A.M.: On the runtime dynamics of the compact genetic algorithm on jump functions. In: Genetic and Evolutionary Computation Conference, GECCO 2018, pp. 967–974. ACM (2018)
34. Jansen, T., De Jong, K.A., Wegener, I.: On the choice of the offspring population size in evolutionary algorithms. Evol. Comput. **13**, 413–440 (2005)
35. Jansen, T., Wegener, I.: The analysis of evolutionary algorithms—a proof that crossover really can help. Algorithmica **34**, 47–66 (2002)
36. Lissovoi, A., Oliveto, P.S., Warwicker, J.A.: On the time complexity of algorithm selection hyper-heuristics for multimodal optimisation. In: Conference on Artificial Intelligence, AAAI 2019, pp. 2322–2329. AAAI Press (2019)
37. Mühlenbein, H.: How genetic algorithms really work: mutation and hillclimbing. In: Parallel Problem Solving from Nature, PPSN 1992, pp. 15–26. Elsevier (1992)

38. Neumann, F., Wegener, I.: Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Theor. Comput. Sci. **378**, 32–40 (2007)
39. Quinzan, F., Göbel, A., Wagner, M., Friedrich, T.: Evolutionary algorithms and submodular functions: benefits of heavy-tailed mutations. Nat. Comput. **20**, 561–575 (2021)
40. Rowe, J.E., Aishwaryaprajna: The benefits and limitations of voting mechanisms in evolutionary optimisation. In: Foundations of Genetic Algorithms, FOGA 2019, pp. 34–42. ACM (2019)
41. Rudolph, G.: Convergence Properties of Evolutionary Algorithms. Verlag Dr. Kovăc (1997)
42. Rajabi, A., Witt, C.: Self-adjusting evolutionary algorithms for multimodal optimization. In: Genetic and Evolutionary Computation Conference, GECCO 2020, pp. 1314–1322. ACM (2020)
43. Rajabi, A., Witt, C.: Stagnation detection in highly multimodal fitness landscapes. In: Genetic and Evolutionary Computation Conference, GECCO 2021, pp. 1178–1186. ACM (2021)
44. Rajabi, A., Witt, C.: Stagnation detection with randomized local search. In: Evolutionary Computation in Combinatorial Optimization, EvoCOP 2021, pp. 152–168. Springer (2021)
45. Wald, A.: Some generalizations of the theory of cumulative sums of random variables. Ann. Math. Stat. **16**, 287–293 (1945)
46. Witt, C.: Worst-case and average-case approximations by simple randomized search heuristics. In: Symposium on Theoretical Aspects of Computer Science, STACS 2005, pp. 44–56. Springer (2005)
47. Witt, C.: Runtime analysis of the $(\mu + 1)$ EA on simple pseudo-Boolean functions. Evol. Comput. **14**, 65–86 (2006)
48. Witt, C.: On crossing fitness valleys with majority-vote crossover and estimation-of-distribution algorithms. In: Foundations of Genetic Algorithms, FOGA 2021, pp. 2:1–2:15. ACM (2021)
49. Wu, M., Qian, C., Tang, K.: Dynamic mutation based Pareto optimization for subset selection. In: Intelligent Computing Methodologies, ICIC 2018, Part III, pp. 25–35. Springer (2018)
50. Whitley, D., Varadarajan, S., Hirsch, R., Mukhopadhyay, A.: Exploration and exploitation without mutation: solving the jump function in $\Theta(n)$ time. In: Parallel Problem Solving from Nature, PPSN 2018, Part II, pp. 55–66. Springer (2018)