A Flexible Evolutionary Algorithm with Dynamic Mutation Rate Archive

Krejca, Martin S.; Witt, Carsten

Published in:
Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'24

Link to article, DOI:
10.1145/3638529.3654076

Publication date:
2024

Document Version
Publisher's PDF, also known as Version of record

A Flexible Evolutionary Algorithm With Dynamic Mutation Rate Archive*

Martin S. Krejca
Laboratoire d'Informatique (LIX), CNRS, École Polytechnique, Institut Polytechnique de Paris
Palaiseau, France
martin.krejca@polytechnique.edu

Carsten Witt
Technical University of Denmark
Lyngby, Denmark
cawi@dtu.dk

ABSTRACT

We propose a new, flexible approach for dynamically maintaining successful mutation rates in evolutionary algorithms using k-bit flip mutations. The algorithm adds successful mutation rates to an archive of promising rates that are favored in subsequent steps. Rates expire when their number of unsuccessful trials has exceeded a threshold, while rates currently not present in the archive can enter it in two ways: (i) via user-defined minimum selection probabilities for rates combined with a successful step or (ii) via a stagnation detection mechanism increasing the value for a promising rate after the current bit-flip neighborhood has been explored with high probability. For the minimum selection probabilities, we suggest different options, including heavy-tailed distributions.

We conduct rigorous runtime analysis of the flexible evolutionary algorithm on the OneMax and Jump functions, on general unimodal functions, on minimum spanning trees, and on a class of hurdle-like functions with varying hurdle width that benefit particularly from the archive of promising mutation rates. In all cases, the runtime bounds are close to or even outperform the best known results for both stagnation detection and heavy-tailed mutations.

CCS CONCEPTS

• **Theory of computation → Theory of randomized search heuristics**; • **Mathematics of computing → Evolutionary algorithms**.

KEYWORDS

Theory, runtime analysis, evolutionary algorithm, parameter adaptation, archive

ACM Reference Format:
Martin S. Krejca and Carsten Witt. 2024. A Flexible Evolutionary Algorithm With Dynamic Mutation Rate Archive. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3638529.3654076

*Due to space constraints, several proofs had to be omitted from this paper. A full technical report is available on arXiv [19].

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0494-9/24/07.
https://doi.org/10.1145/3638529.3654076

1 INTRODUCTION

The success of evolutionary algorithms (EAs) depends crucially on their parametrization [12]. At their core is the careful balance of exploration and exploitation. On the one hand, an EA should find a, potentially locally, optimal solution if it is close by. On the other hand, it should escape local optima and keep finding better solutions within reasonable time.

To achieve this behavior, an EA needs to be flexible enough to consider solutions that are at different distances from its current solutions. As there are many ways to make such a choice, this has led to various approaches that have been studied theoretically [12]. Some approaches consider a static distribution over the search radius (known as *mutation rate*). Others adjust the mutation rate dynamically during the run. Yet other approaches allow to accept worse solutions in order to search more locally for alternative, better solutions. And some approaches do not work with multi-sets of solutions but with distributions over solutions instead. We discuss some prominent ideas in more detail in Section 2.

To the best of our knowledge, almost all of these approaches base their choice for the next mutation rate on ad hoc knowledge, not exploiting the information from previous iterations extensively. This means that recurring structures in the search space are not necessarily well exploited. In contrast, approaches that do attempt to incorporate knowledge from previous iterations are hard to analyze and there are much fewer runtime results.

We propose a flexible EA (the *flex-EA*, Algorithm 1), which aims to satisfy both needs. The flex-EA maintains an archive of mutation rates that were successful in the past, called *active*. In each iteration, it picks a mutation rate based primarily on the active rates, but it still considers all other rates, based on user-defined probabilities. Successful rates are added to the archive while unsuccessful ones are discarded after a certain amount of time, defined by the user. If no rate is active, the flex-EA picks the successor of the last-active rate, reminiscent of stagnation detection [28]. If the archive becomes too full without seeing any progress, it is reset.

The flex-EA allows many of its parameters to be adjusted in order to incorporate problem-specific knowledge. However, in the absence of such knowledge, we recommend to choose the probabilities of inactive rates according to a heavy-tailed distribution – a common, general-purpose choice, first suggested by Doerr et al. [11], also known as a *power-law* – and to choose the times for discarding active rates in the same manner as done in stagnation detection [28]. This results in an essentially parameter-less algorithm.

We show the efficiency of the flex-EA by analyzing it on a diverse set of problems, namely, on unimodal and on jump functions, on the minimum spanning tree problem, and on hurdle-like functions with

varying hurdle width. For all of these problems, the performance of the flex-EA is close to or even better than the best known results for stagnation detection and heavy-tailed mutation.

For unimodal functions (Section 5), the progress of the flex-EA is mainly determined by its probability to improve solutions locally (Theorem 5.1). For OneMax and LeadingOnes of problem size n and with the recommended parametrization, this results in the common bounds of $O(n \log(n))$ and $O(n^2)$, respectively.

For the Jump benchmark of problem size n and gap size k (Section 6), the recommended parametrization results for $k = o(n/\log n)$ in a bound of $(2+o(1))\binom{n}{k}$ (Corollary 6.2), which is optimal for unbiased algorithms with unary mutation, up to a factor of $2+o(1)$ [14]. For many other values of k, the result is a product of $\binom{n}{k}$ and a factor that depends at most quadratically on k.

For the combinatorial problem of finding a minimum spanning tree on a graph with m edges (Section 7), we rely on the flexibility to incorporate problem-specific knowledge, and we adjust the parameters such that the algorithm mainly makes progress by flipping one or two edges into or out of solutions. With this choice, the flex-EA has an expected runtime of $(1+o(1))(m^2 \log(m))$ (Theorem 7.1), which matches the best known bound [26].

Last, to showcase the benefit of an archive, we introduce a problem that features many local optima separated by gaps of alternating widths (Section 8). Once the flex-EA determines the necessary rates, it maintains them successfully until the global optimum is found. It does so in an expected time that is faster by a poly-logarithmic factor than heavy-tailed mutation and faster by a super-polynomial factor than stagnation detection (Theorem 8.1). Both, heavy-tailed mutation and stagnation detection are slowed down by requiring to find the correct mutation rate repeatedly, although heavy-tailed mutation does this far more efficiently than stagnation detection.

Overall, our results show that the flex-EA has a great performance with the recommended parameter setting while effectively not requiring any parameter choices. In settings with repeating local structures, this parametrization even outperforms the operators it is based on. In the case of problem-specific knowledge, the flex-EA can achieve results that match the best known bounds. For all of these settings, the state space of the algorithm remains simple enough that a mathematical analysis is still tractable.

2 RELATED WORK

There exists a plethora of different operators for choosing good mutation rates [12]. We provide an overview on established concepts that are more closely related to the flex-EA. In particular, we only concern unary operators. Furthermore, we focus on single-trajectory algorithms that only maintain a single search point at a time and update it iteratively. Using populations and diversity-maintaining operators opens up for a new level of complexity in the design and analysis of EAs. In addition, this excludes estimation-of-distribution algorithms (EDAs) [18], which maintain a probability distribution over the search space instead of a single search point.

Non-elitism. A non-elitist algorithm is one in which worse solutions can be selected over better solutions. Although this approach does not directly change the mutation rate, it can affect the success of the current rate by creating solutions in a distance that is more favorable [4, 10]. This requires a careful balance of the selection probability of solutions. In comparison, the flex-EA is an elitist algorithm, i. e., it aims at finding mutation rates that are good for the currently best solution without modifying it.

Static rate distributions. A large class of EAs, most prominently the $(1+1)$ EA, use a static distribution for selecting mutation rates. Since this distribution is fixed, it needs to accommodate all situations that may arise during optimization. The $(1+1)$ EA uses a single rate that creates solutions with high probability locally around the current search point. This leads to long waiting times for escaping local optima [17].

Artificial immune systems (AISes) use operators that change solutions more drastically [3]. However, since this makes it harder to find local improvements, AISes often use additional operators that influence how large the changes to a solution actually are [36].

Arguably the best of both worlds is achieved by choosing the mutation rate according to a heavy-tailed distribution, as introduced by Doerr et al. [11] in their seminal paper. This approach, known as fast mutation, finds local improvements quickly and also has a good probability to use larger mutation rates to escape local optima. Since its inception, this approach has seen successful applications in a variety of theoretical works [1–3, 5, 9, 13–16, 24, 35, 37]. A drawback of fast mutation is that it never adjusts its distribution. Thus, to find improving solutions at larger distances, it exclusively relies on the decent probability of choosing a sufficiently large rate. Depending on the distance, this can result in long waiting times.

Dynamic rate distributions. Some algorithms change the distribution for choosing the mutation rate during the optimization process [7]. While this technically allows for complex distributions, to the best of our knowledge, most algorithms that fall into this category distribute the probability mass around a single rate.

A common approach is the 1/5 success rule (e. g., [6, 29, 32]), which increases the current rate if more than one fifth of the last tries with the current rate are successful, and it decreases it otherwise. While this approach automatically adjusts the rate to one for finding an improving solution, it forgets the previously used rate and, instead, aims to rediscover it if it becomes useful again later.

A more deterministic approach to a similar idea is stagnation detection [27, 28]. This operator starts with the smallest rate and picks it repeatedly until the algorithm is certain with high probability that it cannot find an improving solution with the current rate. It then increases the rate and repeats the process. This approach has the benefit that it can escape local optima very well. However, similar to the one-fifth success rule, the algorithm focuses on a single rate, meaning that it needs to rediscover previously good rates. This problem was addressed partially by adding a radius memory to the algorithm [26], which saves the last successful rate in order to re-use it. However, if more than a single previous rate are useful, this approach falls back to the problem of the initial algorithm.

Another, very recent approach [14] proposes to combine stagnation detection with fast mutation in the following way: The algorithm follows classic stagnation detection but does not always create a solution with the current mutation rate; it also has a chance of picking a mutation rate from a heavy-tailed distribution around the current rate. This proves very effective in escaping certain local optima where there are many improving solutions at a further distance [14]. The resulting runtime is better than that of classic

stagnation detection and thus also than that of the flex-EA. However, we believe that this variant still has problems of re-using previously good rates, as it does not save them.

Learning-based algorithms. Doerr et al. [8] propose an algorithm that chooses its mutation rate according to a distribution that is adjusted over time. This adjustment follows the idea of reinforcement learning and incorporates information from many past iterations. A related concept are *hyper-heuristics*, e.g., [20, 21], which are algorithms that aim at selecting in each iteration an algorithm that is best suited for the current sate. This choice is based on the previously observed performance of the algorithms to select from.

The flex-EA is similar to these approaches, with the main difference that it updates its mutation rate distribution in a simpler way. This simplification allows a rigorous analysis of the flex-EA on a wide variety of problems while still performing very well.

3 PRELIMINARIES

The natural numbers $\mathbb{N}$ include 0. For $a, b \in \mathbb{R}$, let $[a..b] = [a, b] \cap \mathbb{N}$, and let $[a] = [1..a]$. By $\log(\cdot)$ we denote the binary logarithm.

We consider pseudo-Boolean optimization, that is, for a given $n \in \mathbb{N}_{\geq 1}$, the optimization of functions $f\colon \{0,1\}^n \to \mathbb{R}$. We call f a *fitness function*, and we assume that n is given implicitly. When using big-O notation, we assume asymptotics in this implicit n. Furthermore, we call each $\boldsymbol{x} \in \{0,1\}^n$ an *individual*, and $f(\boldsymbol{x})$ its *fitness*. For each $i \in \mathbb{N}$, let $\boldsymbol{x}_i$ denote the value of $\boldsymbol{x}$ at position i. Last, let $|\boldsymbol{x}|_1$ be the number of 1s of $\boldsymbol{x}$, and $|\boldsymbol{x}|_0$ its number of 0s.

For each $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^n$, let $d_{\mathrm{H}}(\boldsymbol{x}, \boldsymbol{y}) = |\{i \in \mathbb{N} \mid \boldsymbol{x}_i \neq \boldsymbol{y}_i\}|$ denote the Hamming distance of $\boldsymbol{x}$ and $\boldsymbol{y}$. For each $r \in [n]$, we define the *r-bit flip neighborhood of $\boldsymbol{x}$* to be the set $\{\boldsymbol{y} \in \{0,1\}^n \mid d_{\mathrm{H}}(\boldsymbol{x}, \boldsymbol{y}) = r\}$.

Given a fitness function f and an algorithm A optimizing f, we define the *runtime* of A on f as the (random) number of evaluations of f until the first time that an optimum of f is evaluated. To this end, we assume that each individual that A creates is evaluated exactly once. For the flex-EA in Section 4, the runtime is essentially the number of iterations until an optimum is created for the first time (plus one, due to the initialization).

4 THE FLEXIBLE EVOLUTIONARY ALGORITHM

The *flexible evolutionary algorithm* (flex-EA, Algorithm 1) is an elitist $(1+1)$-type evolutionary algorithm choosing its mutation rate each iteration according to a probability distribution given by a vector (the *frequency vector*). The flex-EA updates its frequency vector in each iteration, aiming to give promising rates a high probability. This update is influenced by different factors. In the following, we first discuss the high-level idea of the flex-EA and its different parts. Afterward, in Section 4.1, we propose good default choices for the various parameters of the algorithm. The result is a basically parameter-less algorithm that can be thought of as a superposition of heavy-tailed mutation [11] and stagnation detection [28].

Update of the frequency vector. The update of the vector $\boldsymbol{p} \in [0,1]^n$ aims to give (mutation) rates a high probability if choosing them in the past succeeded in improving the fitness of the current solution. If a rate repeatedly fails to be successful, the probability of choosing it (called its *frequency*) is reduced. To this end, the flex-EA groups all rates into those that receive a high frequency (called *active*) and those that receive a low frequency (called *inactive*).

The active frequencies follow a uniform distribution. Each active frequency has a counter of how often it failed to succeed. If this counter exceeds a user-defined limit, the respective frequency becomes inactive. If this results in no frequencies being active and the frequency at index $r \in [n]$ was the last active one, then the frequency at index $r+1$ is set to active (identifying $n+1$ with 1).

The update process is superimposed by user-defined lower bounds per frequency. If a frequency would be set to a value less than its lower bound, it takes on its lower bound instead.

In addition, the flex-EA keeps a global counter that counts how many times in a row no success was achieved with any mutation rate. If this counter reaches a specific limit, all frequencies become inactive, and the frequency at position 1 becomes active. This effectively resets the frequency vector.

The flex-EA in more detail. Algorithm 1 shows the detailed flex-EA. The frequency vector is denoted by $\boldsymbol{p} \in [0,1]^n$, and the set of active rates (called the *archive*) by $A \subseteq [n]$. The lower bounds for each frequency are provided by the user to the flex-EA via the *lower-bound vector* $\boldsymbol{\ell} \in [0,1]^n$. We assume that $\sum_{i \in [n]} \boldsymbol{\ell}_i \leq 1$.

Each rate $i \in [n]$ has a counter $\boldsymbol{c}_i \in \mathbb{N}$ for counting its number of failed attempts to improve the current solution. If $\boldsymbol{c}_i$ reaches a user-defined limit $\boldsymbol{C}_i \in \mathbb{R}_{\geq 0}$, frequency i becomes inactive. These limits are provided by the user as the *count bound vector* $\boldsymbol{C} \in \mathbb{R}^n_{\geq 0}$.

The global counter is denoted by $g \in \mathbb{N}$. Its limit that, if reached, resets the archive of the flex-EA is denoted by $G \in \mathbb{R}_{\geq 0}$. The value G is chosen as the count bound of the smallest active rate $m \in [n]$ divided by its current frequency. This resembles the normal bound $\boldsymbol{C}_m$ but accounts for rate m being only chosen once in $\boldsymbol{p}_m^{-1}$ iterations in expectation.

The update of the frequency vector is a bit involved, as frequencies may be capped by their respective lower bound, thus disallowing to distribute the probability mass uniformly among all active frequencies. Note that the flex-EA aims to give each active frequency the same probability. The total probability to distribute among these $|A| =: \varphi$ frequencies is $1 - \sum_{i \in [n] \setminus A} \boldsymbol{\ell}_i =: M$, as all inactive frequencies are at their respective lower bound. Hence, each active frequency should get a probability of $\frac{M}{\varphi}$. In order to account for the lower bounds of the active frequencies, the active frequencies are adjusted sequentially, starting with the largest lower bound, going to the smallest. If the lower bound of the first frequency is less than the intended value $\frac{M}{\varphi}$, then all active frequencies get this value. Otherwise, the first frequency is set to its lower bound, and the remaining probability mass is adjusted. This process is repeated.

Algorithm 2 shows this approach algorithmically. We note that the algorithm is presented in a way that aims to easily understand the logic of the update. It is not optimized for efficiency. A more efficient implementation would make use of a data structure for the archive that quickly allows to add and delete indices while keeping them ordered with respect to their lower bound. This can be achieved, for example, with a balanced search tree.

4.1 Recommended Parameter Choices

The flex-EA requires the user to provide $2n$ parameters, which can be daunting. Hence, we propose a recommendation that works well

Algorithm 1: The flexible evolutionary algorithm (flex-EA) with given lower-bound vector $\boldsymbol{\ell}$ and count bound vector $\boldsymbol{C}$, maximizing a pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$, stopping at a user-defined termination criterion.

$t \leftarrow 0$;
$g^{(t)} \leftarrow 0$;
$A^{(t)} \leftarrow \{1\}$;
for $i \in [n]$ **do** $c_i^{(t)} \leftarrow 0$;
$\boldsymbol{x}^{(t)} \leftarrow$ solution from $\{0,1\}^n$ chosen uniformly at random;
while termination criterion not met **do**
  $\boldsymbol{p}^{(t)} \leftarrow$ distribute probability mass over $A^{(t)}$, respecting $\boldsymbol{\ell}$ (Algorithm 2);
  $m \leftarrow \min A^{(t)}$;
  $G^{(t)} \leftarrow \frac{C_m}{p_m^{(t)}}$;
  choose $r \in [n]$ according to $\boldsymbol{p}^{(t)}$;
  $\boldsymbol{y}^{(t)} \leftarrow$ copy $\boldsymbol{x}^{(t)}$ and flip r bits uniformly at random in this copy;
  if $f(\boldsymbol{y}^{(t)}) > f(\boldsymbol{x}^{(t)})$ **then**
    $\boldsymbol{x}^{(t+1)} \leftarrow \boldsymbol{y}^{(t)}$;
    $A^{(t+1)} \leftarrow A^{(t)} \cup \{r\}$;
    $g^{(t+1)} \leftarrow 0$;
    $c_r^{(t+1)} \leftarrow 0$;
  else
    if $f(\boldsymbol{y}^{(t)}) = f(\boldsymbol{x}^{(t)})$ **then** $\boldsymbol{x}^{(t+1)} \leftarrow \boldsymbol{y}^{(t)}$;
    $g^{(t+1)} \leftarrow g^{(t)} + 1$;
    $c_r^{(t+1)} \leftarrow c_r^{(t)} + 1$;
    if $g^{(t+1)} \geq G^{(t)}$ **then**
      $g^{(t+1)} \leftarrow 0$;
      $A^{(t+1)} \leftarrow \{1\}$;
      $c_1^{(t+1)} \leftarrow 0$;
    else if $c_r^{(t+1)} \geq C_r$ **then**
      $A^{(t+1)} \leftarrow A^{(t)} \setminus \{r\}$;
      if $A^{(t+1)} = \emptyset$ **then**
        $r' \leftarrow$ **if** $r+1 \leq n$ **then** $r+1$ **else** 1;
        $A^{(t+1)} \leftarrow A^{(t+1)} \cup \{r'\}$;
        $c_{r'}^{(t+1)} \leftarrow 0$;
  $t \leftarrow t+1$;

in many cases, as we prove in most of the following sections. For the lower-bounds vector $\boldsymbol{\ell}$, we recommend to choose a heavy-tailed distribution, inspired by the fast-mutation operator [11]. For the count bound vector $\boldsymbol{C}$, we recommend to choose the same values as for stagnation detection [28]. We explain both recommendations in more detail below and then briefly discuss the resulting algorithm.

Heavy-tailed lower bounds. Let $\beta \in (1,2)$ (called the *power-law exponent*), and let $N\colon (1,2) \to \mathbb{R}_{\geq 1}$ denote a normalization with

Algorithm 2: The algorithm that, given an index set $A \subseteq [n]$ and a lower-bound vector $\boldsymbol{\ell}$ of size n, returns a frequency vector $\boldsymbol{p}$ of size n where all possible probability mass is distributed as evenly as possible over the frequencies with indices in A, respecting $\boldsymbol{\ell}$.

$M \leftarrow 1 - \sum_{i \in [n] \setminus A} \ell_i$;
for $i \in [n] \setminus A$ **do** $p_i \leftarrow \ell_i$;
$\varphi \leftarrow |A|$;
$(S_i)_{i \in [\varphi]} \leftarrow A$ sorted in descending order by $\boldsymbol{\ell}$;
for $i \in [\varphi]$ **do**
  if $\ell_{S_i} \leq \frac{M}{\varphi - i + 1}$ **then**
    for $j \in [i..\varphi]$ **do** $p_{S_j} \leftarrow \frac{M}{\varphi - i + 1}$;
    return $\boldsymbol{p}$;
  else
    $p_{S_i} \leftarrow \ell_{S_i}$;
    $M \leftarrow M - \ell_{S_i}$;
return $\boldsymbol{p}$;

$N\colon \alpha \mapsto \sum_{i \in [n]} i^{-\alpha}$. We recommend for all $i \in [n]$ to choose

$$\ell_i = \frac{1}{2N(\beta)} i^{-\beta}.$$

Note that $\sum_{i \in [n]} \ell_i = \frac{1}{2}$. Hence, the flex-EA still has a probability mass of $\frac{1}{2}$ remaining to distribute among the active frequencies.

This recommendation follows the ideas of *fast mutation*, introduced in [11]. This mutation operator first chooses a rate $r \in [n]$ according to a heavy-tailed distribution and performs standard bit mutation afterward. That is, it flips each bit of a given individual independently with probability $\frac{r}{n}$. Doerr et al. [11] name the variant of the $(1+1)$ EA that uses fast mutation the Fast (1+1) EA.

Doerr et al. [11] choose the heavy-tailed distribution as even large rates have a reasonable probability of at least $\frac{1}{n^2}$ of being chosen. The authors show that the Fast (1+1) EA has a super-exponential runtime speed-up in the parameter k compared to the classic $(1+1)$ EA when optimizing $\textsc{Jump}_k$, and the operator has seen great success in other settings (see also Section 2). Hence, we propose to also use it for the flex-EA.

Stagnation detection bounds. Let $R \in \mathbb{R}_{>1}$. We recommend

$$C_i = \binom{n}{i} \ln(R)$$

for all $i \in [n]$ and call this the *standard-SD choice*.

This recommendation follows the ideas of *stagnation detection* (SD), introduced by Rajabi and Witt [28]. In their paper, the authors introduce the algorithm *SD-RLS*^{r}, which is also a $(1+1)$-type elitist algorithm that performs the same mutation as the flex-EA but chooses its mutation rate differently in the following deterministic manner. The SD-RLS^r starts with rate 1 and counts how many times it used this rate. It then uses rate 1 for at most C_1 iterations. If it is successful in finding a strictly improving solution during this time, the algorithm resets its counter. If rate 1 is not successful, the SD-RLS^r chooses rate 2 from now on, for a total of at most C_2 iterations. If it is successful during this time, it resets its counter

and reverts back to rate 1. Otherwise, it picks the next-higher rate and adjusts the iteration limit according to the formula above. We note that the SD-RLS$^{\mathrm{r}}$ only tries out rates up to $\frac{n}{2}$, but the formula for the bounds generalizes to all values in $[n]$.

Rajabi and Witt [28] choose the bounds above because this guarantees that if rate $r \in [n]$ is currently used and there is at least one solution in Hamming distance r of the currently best solution, the SD-RLS$^{\mathrm{r}}$ find this improvement with probability at least $1 - R^{-1}$, as the probability of not doing so is at most $\left(1 - \binom{n}{r}^{-1}\right)^{C_r} \leq R^{-1}$. Hence, the value R determines how unlikely it is to fail in such a case. Since the flex-EA is in some aspects similar to the SD-RLS$^{\mathrm{r}}$, we recommend to copy its bounds.

Discussion. When considering the flex-EA with heavy-tailed lower bounds and with the standard-SD choice, it has two parameter values left to choose, namely, the power-law exponent $\beta \in (1, 2)$ as well as the parameter $R \in \mathbb{R}_{>1}$. However, the exact choice of these parameters is typically not very relevant, as we show in this article. Doerr et al. [11] recommend to choose $\beta = \frac{3}{2}$. Since the flex-EA can act similarly to the algorithm discussed by Doerr et al. [11], we recommend this choice too. For the parameter R, Rajabi and Witt [28] show that a choice of $R \geq n^{4+\epsilon}$ for a small constant $\epsilon > 0$, with R being a polynomial, is usually sufficient. We show in our results below that this is also the case for the flex-EA.

Since we provide recommendations for the only two parameter choices left, we consider the flex-EA to be an essentially parameterless algorithm. However, one needs to keep in mind that the flex-EA allows to choose its parameters $\boldsymbol{\ell}$ and C more flexibly, which can be a good choice if given extra information (see also Section 7).

5 RUNTIME ON UNIMODAL FUNCTIONS

We analyze the flex-EA on unimodal pseudo-Boolean functions, that is, functions $f\colon \{0,1\}^n \to \mathbb{R}$ with a unique global optimum $z^* \in \{0,1\}^n$ such that, for all $x \in \{0,1\}^n \setminus \{z^*\}$, there is a $y \in \{0,1\}^n$ with $d_{\mathrm{H}}(x, y) = 1$ such that $f(x) < f(y)$. For each unimodal function f, we call the cardinality of the range of f the number of *fitness levels of* f, and each set of all individuals of the same fitness a *fitness level*. Each individual (besides the global optimum) has at least one neighbor in a fitness level of better fitness.

The flex-EA optimizes unimodal functions efficiently in expectation if its parameters are chosen well. To this end, we rely exclusively on the lower bound of rate 1.

Theorem 5.1. *Let f be a unimodal fitness function with L fitness levels. For all $j \in [L-1]$, let q_j denote a lower bound on the probability that, given an individual z in fitness level j, flipping a single bit in z creates an offspring with strictly better fitness. Furthermore, consider the flex-EA with $\boldsymbol{\ell}_1 > 0$.*

Then the expected runtime of the flex-EA on f is at most

$$1 + \boldsymbol{\ell}_1^{-1} \sum_{j \in [L-1]} q_j^{-1}.$$

Especially, it is at most $1 + n(L-1)\boldsymbol{\ell}_1^{-1}$.

Proof. We start with the first claim. The plus 1 is for initialization. For the remaining bound, we apply the fitness level method [33, Theorem 2]· Let $j \in [L-1]$ be the fitness level of the current solution of the flex-EA. In order to leave level j, it is sufficient to choose rate 1 and then create an offspring with strictly better fitness. The probability of the former is $\boldsymbol{\ell}_1$, and the one of the latter is at least q_j. Since these events are independent, the flex-EA leaves level j with a probability of at least $\boldsymbol{\ell}_1 q_j$. This concludes this case.

For the second claim, note that since f is unimodal, there is always at least one improving solution that only requires a single bit to be flipped. Hence, it holds that $q_j \geq \frac{1}{n}$. Applying the first claim concludes the proof. □

Theorem 5.1 leads to useful bounds on various functions. For example, the coarser bound at the end results already in the common expected runtime of $\mathrm{O}(n^2)$ for the LeadingOnes benchmark [31] if we assume a constant lower bound for rate 1, as is the case with heavy-tailed lower bounds. For functions where the probability of leaving a fitness level depends on the current state, we need to estimate these transition probabilities more carefully. Doing so leads to an expected runtime bound of $\mathrm{O}(n \log n)$ for the OneMax benchmark, as we show below (Corollary 5.2). Since we require bounds for similar functions in the following sections, we introduce a slight generalization of OneMax and analyze it instead.

For all $k \in [0..n]$, let *trimmed OneMax* (OM_k) be defined as

$$x \mapsto \begin{cases} k + \mathrm{OM}(x) & \text{if } |x|_1 \leq n - k, \\ n - \mathrm{OM}(x) & \text{else.} \end{cases}$$

Note that OM_k is a unimodal function with $n+1$ fitness levels. Its global optimum is achieved for all individuals with exactly $n-k$ 1s. The case $k = 0$ results in OneMax. We get the following bound.

Corollary 5.2. *Let $k \in [0..n]$. Consider the flex-EA with $\boldsymbol{\ell}_1 > 0$ optimizing OM_k. Then the expected runtime is $\mathrm{O}\big(\boldsymbol{\ell}_1^{-1} n \log(n)\big)$.*

Proof. We apply Theorem 5.1 and use its notation. For all $j \in [n+1]$, denote the fitness level that represents an OM_k-value of $j-1$. We say that individuals with at most $n-k$ 1s are on the *left branch* of the function, and the other ones on the *right branch*.

Let $j \in [n]$. We bound q_j with respect to whether the current individual is on the left or the right branch. If it is on the left branch, it has $j-1-k$ 1s. If mutation flips any of the $n-j+1+k$ 0s, then the result strictly improves the fitness. Hence, $q_j = \frac{n-j+1+k}{n}$.

If the current individual is on the right branch, it has $n-j+1$ 1s. If mutation flips any of these 1s, then the result strictly improves the fitness. Hence, $q_j = \frac{n-j+1}{n}$.

Combining both cases, by Theorem 5.1 and disregarding the plus 1, we obtain an upper bound on the expected runtime of

$$\boldsymbol{\ell}_1^{-1} \sum_{j \in [n]} \left(\frac{n}{n-j+1+k} \cdot \mathbb{1}\{j \geq k+1\} + \frac{n}{n-j+1} \cdot \mathbb{1}\{j < k+1\} \right)$$
$$\leq \boldsymbol{\ell}_1^{-1} \cdot 2n \sum\nolimits_{j \in [n]} j^{-1} = \mathrm{O}\big(\boldsymbol{\ell}_1^{-1} n \log(n)\big).$$

□

6 RUNTIME ON JUMP FUNCTIONS

We consider the well known Jump benchmark, which, for all $k \in [n]$ is defined as

$$x \mapsto \begin{cases} k + \mathrm{OM}(x) & \text{if } |x|_1 \in [n-k] \cup \{n\}, \\ n - \mathrm{OM}(x) & \text{else.} \end{cases}$$

This function is identical to OM_k except that its unique global optimum is at the all-1s bit string. Consequently, the individuals

with exactly $n-k$ 1s are all local optima. The set of all local optima is often called the *plateau* of the function. For elitist algorithms, in order to leave the plateau, exactly all k 0s of an individual need to be changed into a 1, which requires, in expectation, at least $\binom{n}{k}$ tries for unbiased algorithms with unary mutation [14].

With Corollary 6.2, we show that the flex-EA with the recommended parameter choices optimizes Jump in this time. Beforehand, we prove a runtime bound for a more general parametrization.

Theorem 6.1. *Let $R \in \mathbb{R}_{>1}$ and $k \in [2..n]$. Consider the flex-EA with the standard-SD choice $\boldsymbol{C}$ with parameter R and with lower bounds $\boldsymbol{\ell}$ such that $\boldsymbol{\ell}_1, \boldsymbol{\ell}_k > 0$ and $1 - \sum_{i \in [n] \setminus \{k\}} \boldsymbol{\ell}_i = \Theta(1)$. Furthermore, consider that the flex-EA optimizes $\textsc{Jump}_k$. Let $L = (1 - \sum_{i \in [n] \setminus \{k\}} \boldsymbol{\ell}_i)^{-1}$. Then the expected runtime is*

$$\mathrm{O}\Big(n\boldsymbol{\ell}_1^{-1} \log(nR)\Big) + \min\left\{ \binom{n}{k} \cdot \boldsymbol{\ell}_k^{-1}, \binom{n}{k} \cdot \left(L + \mathrm{O}\left(\frac{k}{n-2k+3} \log(R)\right) + \boldsymbol{\ell}_k^{-1} R^{-1/L}\right) + 2^n \cdot \mathbb{1}\left\{k \geq \frac{n}{2}\right\} \ln(R) \right\}.$$

Proof. Let T denote the runtime of the algorithm on $\textsc{Jump}_k$. We split T into the following two phases: Phase 1 considers the time until the current solution of the algorithm has a fitness of at least n, that is, the current solution is on the plateau or optimal. Phase 2 starts in the iteration that phase 1 stops and considers the time until the current solution is optimal. Let T_1 and T_2 denote the runtimes of the respective phases. By the linearity of expectation and by $T = T_1 + T_2$, it holds that $\mathrm{E}[T] = \mathrm{E}[T_1] + \mathrm{E}[T_2]$. We consider the expectations of both phases separately.

Phase 1. Note that the time until the algorithm finds an optimum of OM_k is an upper bound for T_1. Hence, the same is true for their expectations. By Corollary 5.2, we obtain $\mathrm{E}[T_1] = \mathrm{O}(\boldsymbol{\ell}_1^{-1} n \log(n))$.

Phase 2. Note that before the global optimum is found, no rate enters the archive due to being successful, since the global optimum is the only remaining strictly better solution. We bound the expected time of this phase in two different ways. The first way relies exclusively on the lower bound $\boldsymbol{\ell}_k$ for sampling the global optimum, and the second way considers a course of events similar to stagnation detection. It assumes that rate 1 is active in iteration T_1, that is, at the beginning of phase 2. Calling the respective bounds B_1 and B_2, we then conclude that $\mathrm{E}[T_2] \leq \min\{B_1, B_2\}$, as either bound is valid.

1. Relying only on the lower bound. In each iteration, the probability for choosing rate k is at least $\boldsymbol{\ell}_k$. Then, since the current individual has exactly k 0s, the probability to mutate it into the global optimum is $\binom{n}{k}^{-1}$. Hence, in each iteration, the probability to create the global optimum is at least $\boldsymbol{\ell}_k \binom{n}{k}^{-1} =: q$. Since each iteration has an independent chance to create the global optimum, T_2 follows a geometric distribution with a success probability of at least q. Hence, $\mathrm{E}[T_2] \leq q^{-1} = \boldsymbol{\ell}_k^{-1} \binom{n}{k}$.

2. Rate 1 is active at the start. Let D denote the event that $1 \in A^{(T_1)}$, and assume that D occurs. Then $G^{(T_1)} = \boldsymbol{C}_1 / \boldsymbol{p}_1^{(T_1)} \leq \boldsymbol{C}_1 / \boldsymbol{\ell}_1$. Thus, if no strict improvement is found within at most $\boldsymbol{C}_1 / \boldsymbol{\ell}_1$ iterations, the algorithm reverts to classic stagnation detection. Since the current individual is on the plateau, finding a strict improvement means to find the global optimum (and thus ending the phase).

Let $T_{2,1}$ denote the first time in phase 2 such that $g^{(T_{2,1}+T_1)} \geq \boldsymbol{C}_1 / \boldsymbol{\ell}_1$, and let $T_{2,2} = T_2 - T_{2,1}$. Since no mutation rate but k can lead to creating the global optimum, the global counter increases at most $\boldsymbol{C}_1 / \boldsymbol{\ell}_1$ times. Hence, $T_{2,1} \leq \boldsymbol{C}_1 / \boldsymbol{\ell}_1$, and thus $\mathrm{E}[T_{2,1} \mid D] \leq \boldsymbol{C}_1 / \boldsymbol{\ell}_1$.

After $T_{2,1}$, if the algorithm did not find the optimum yet, it performs classic stagnation detection by maintaining a single active rate, based on $\boldsymbol{C}$. Since all rates in $[k-1]$ cannot lead to an improvement, a total of $\sum_{i \in [k-1]} \binom{n}{i} \ln(R)$ iterations is spent until rate k becomes active. For $k \leq \frac{n}{2}$, we obtain by [26, Lemma 1 (a)]

$$\begin{aligned}\sum_{i \in [k-1]} \binom{n}{i} \ln(R) &\leq \frac{n-(k-2)}{n-(2k-3)} \binom{n}{k-1} \ln(R) \\ &= \frac{n-(k-2)}{n-(2k-3)} \frac{k}{n-k+1} \binom{n}{k} \ln(R) \\ &= \mathrm{O}\left(\frac{k}{n-2k+3}\right) \binom{n}{k} \ln(R).\end{aligned}$$

For $k > \frac{n}{2}$, we obtain $\sum_{i \in [k-1]} \binom{n}{i} \ln(R) \leq 2^n \ln(R)$.

Overall, for $\gamma := \mathrm{O}\left(\frac{k}{n-2k+3}\right)\binom{n}{k} + 2^n \cdot \mathbb{1}\left\{k > \frac{n}{2}\right\}$, it takes at most $\gamma \ln(R)$ iterations until rate k is active or the optimum is found.

Assume that the optimum is not found until rate k is active. Let S denote the event that the algorithm samples the optimum within the next $\boldsymbol{C}_k \ln(R)$ iterations. The probability (unconditional on S) to create the optimum in a single iteration is $(1 - \sum_{i \in [n] \setminus \{k\}} \boldsymbol{\ell}_i) \cdot \binom{n}{k}^{-1} = L^{-1} \binom{n}{k}^{-1}$. The process of creating the optimum, conditional on S, follows a truncated geometric distribution [28] and is dominated by a geometric distribution with success probability $L^{-1}\binom{n}{k}^{-1}$, since the actual process stops after at most $\boldsymbol{C}_k \ln(R)$ attempts. Hence, $\mathrm{E}[T_{2,2} \mid D, S] \leq \gamma \ln(R) + L\binom{n}{k}$.

In the case that S does not hold, we use the same arguments as when relying only on $\boldsymbol{\ell}_k$ and obtain $\mathrm{E}[T_{2,2} \mid D, \overline{S}] \leq \gamma \ln(R) + \boldsymbol{\ell}_k^{-1} \binom{n}{k}$.

Next, we trivially bound $\Pr[D, S] \leq 1$ as well as $\Pr[D, \overline{S}] \leq \Pr[\overline{S} \mid D]$. By the definition of S, the estimates above of finding the global optimum in a single iteration, as well as that $\binom{n}{k} = \boldsymbol{C}_k$ due to the standard-SD choice, $\Pr[\overline{S} \mid D] \leq \left(1 - L^{-1} \boldsymbol{C}_k^{-1}\right)^{\boldsymbol{C}_k \ln(R)} \leq R^{-1/L}$. Thus, we obtain

$$\mathrm{E}[T_2 \mid D] \leq \frac{\boldsymbol{C}_1}{\boldsymbol{\ell}_1} + \gamma \ln(R) + L\binom{n}{k} + \boldsymbol{\ell}_k^{-1} \binom{n}{k} R^{-1/L}.$$

We conclude by trivially bounding $\Pr[D] \leq 1$ and by adding the term $\boldsymbol{C}_1 / \boldsymbol{\ell}_1 = n\boldsymbol{\ell}_1^{-1} \ln(R)$ also to our first bound for phase 2. □

Theorem 6.1 shows the intricate interplay of the parameters of the flex-EA. For $k \geq \frac{n}{2}$, the value of the lower bound for rate k has a huge impact on the expected runtime, as it is better to choose rate k due to its lower bound than to wait for it to become the solely active rate. The latter requires to wait in the order of at least the central binomial coefficient, which is exponential in n.

In contrast, if $k < \frac{n}{2}$ is sufficiently small, the value L of all probability mass put on rate k, except for the lower bounds, plays an important role, as it can speed up the process of choosing rate k. If the optimum is not found when rate k is active, the algorithm uses the lower bound of rate k again. However, choosing the parameter R sufficiently large makes it unlikely for this case to occur.

The following corollary summarizes how the expected runtime improves with the recommended parameter choices. Up to a factor of $2 + o(1)$, the result is asymptotically optimal if k is sufficiently far away from $\frac{n}{2}$. The factor of 2 is a result of the lower bounds, which take up a total probability mass of $\frac{1}{2}$ in the recommended setting. Hence, even if rate k is the only active rate, the probability of choosing it is only about $\frac{1}{2}$.

Corollary 6.2. *Let $\beta \in (1,2)$ and $\varepsilon \in \mathbb{R}_{>0}$ be constants, let $R = n^{2(\beta+\varepsilon)}$, and let $k \in [2..n]$. Consider the flex-EA with the standard-SD choice $\boldsymbol{C}$ with parameter R and with heavy-tailed lower bounds $\boldsymbol{\ell}$. Furthermore, consider that the flex-EA optimizes $\textsc{Jump}_k$. Then the expected runtime is*

$$\big(1 + o(1)\big) \min \left\{ \binom{n}{k} \cdot 2N(\beta) k^\beta, \right.$$

$$\left. \binom{n}{k} \cdot \left(2 + O\left(\frac{k}{n-2k+3} \log(n) + n^{-\varepsilon}\right)\right) + 2^n \cdot \mathbb{1}\left\{k \geq \frac{n}{2}\right\} \ln(R) \right\}.$$

Especially, for $k = o\left(\frac{n}{\log(n)}\right)$, this simplifies to $\big(2 + o(1)\big)\binom{n}{k}$.

Proof. The first bound follows from Theorem 6.1. Due to the assumptions on $\boldsymbol{\ell}$, it holds that $\boldsymbol{\ell}_1^{-1} = \Theta(1)$, that $\boldsymbol{\ell}_k = \frac{1}{2N(\beta)} k^{-\beta}$, and that $1 - \sum_{i \in [n] \setminus \{k\}} \boldsymbol{\ell}_i \geq \frac{1}{2}$. Furthermore, due to the choice of R, it follows that $O\big(n\boldsymbol{\ell}_1^{-1} \log(nR)\big) = O(n \log(n))$. Since $k \geq 2$, the other summand is $\Omega\big(n^2\big)$, so the summand $O\big(n\boldsymbol{\ell}_1^{-1} \log(nR)\big)$ is of lower order. Last, since $L^{-1} \geq \frac{1}{2}$, it follows that $R^{-1/L} \leq R^{-1/2} \leq n^{-(\beta+\varepsilon)}$. Substituting all of the values yields the desired result.

For the case $k = o\left(\frac{n}{\log(n)}\right)$, note that $\frac{k}{n-2k+3} \log(n) = o(1)$. Hence, the second term in the minimum is $\big(2 + o(1)\big)\binom{n}{k}$. This term is smaller than the first term in the minimum, since $N(\beta) \geq 1$ as well as $k^\beta \geq 2$, concluding the proof. □

7 BENEFITS OF THE RATE ARCHIVE ON MINIMUM SPANNING TREES

The prior works on SD-RLS$^{\mathrm{r}}$ (i. e., RLS with a classical stagnation detection mechanism) in [28] and on SD-RLS$^{\mathrm{m}}$, a variant with a radius memory, in [26] include analyses of the algorithms on the classical minimum spanning tree problem. Interestingly, both stagnation detection algorithms find optimal solutions to the problem in efficient polynomial expected time (see detailed expressions below), which stands in contrast to the bound $O\big(m^2(\log n + \log w_{\max})\big)$, where n is the number of vertices, m the number of edges and $w_{\max}$ is the largest edge weight, from the seminal work [22] analyzing the (1+1) EA on the MST problem. It is well known [30] that the dependency of the bound on $w_{\max}$ can be removed and that the runtime bound $O\big(m^3 \ln n\big)$ is obtained by studying the (1+1) EA on a rank-equivalent fitness function, however, even that bound is probably far from tight. Before that, polynomial bounds were only known for randomized local search algorithms with 1- and 2-bit flip neighborhoods, but not for globally searching algorithms.

In this section, we will analyze the flex-EA on the MST problem and prove results matching the best classical stagnation detection algorithm. The setting is as follows. Given is an undirected, weighted graph $G = (V, E)$, where $n = |V|$, $m = |E|$ and the weight of edge e_i, where $i \in \{1, \dots, m\}$, is a positive integer w_i. Let $c(x)$ denote the number of connected components in the subgraph described by the search point $x \in \{0,1\}^m$. The fitness function $f\colon \{0,1\}^m \to \mathbb{R}$ considered in [22], to be minimized, is defined by

$$f(x) := M^2(c(x) - 1) + M\Big(\sum\nolimits_{i \in [m]} x_i - (n-1)\Big) + \sum\nolimits_{i \in [m]} w_i x_i$$

for an integer $M \geq n^2 w_{\max}$, where $w_{\max}$ denotes the largest edge weight. Hence, f returns the total weight of a given spanning tree and penalizes unconnected graphs as well as graphs containing cycles so that such graphs are always inferior than spanning trees. The authors of [28] showed that SD-RLS$^{\mathrm{r}}$ starting with an arbitrary spanning tree finds an MST in $(1 + o(1))\big(m^2 \ln m + (4m \ln m)\mathrm{E}[S]\big)$ fitness calls where $\mathrm{E}[S]$ is the expected number of strict improvements. The variant SD-RLS$^{\mathrm{m}}$ with radius memory analyzed in [26] has an expected runtime of at most

$$(1 + o(1))\big((m^2/2)(1 + \ln(\textstyle\sum_{i \in [m]} r_i))\big) \leq (1 + o(1))\big(m^2 \ln m\big),$$

where r_i is the rank of the ith edge in the sequence sorted by increasing edge weights. This holds since SD-RLS$^{\mathrm{m}}$ for a sufficiently long time only flips two bits uniformly at random; then its stochastic search trajectory on the given MST instance is indistinguishable from an instance, where the i-largest weight from the original set $w_1, \dots, w_m$ is replaced by the rank number r_i [25]. Essentially, the bound for SD-RLS$^{\mathrm{m}}$ is better by a factor of 2 than for the classical RLS^{1,2} algorithm choosing uniformly at random a rate $r \in \{1,2\}$ and flipping r bits uniformly. In fact, RLS^{1,2} wastes every second step since 1-bit flips are never accepted on spanning trees.

We obtain Theorem 7.1 below. Choosing $\boldsymbol{\ell}_1 = o(1)$, it matches the bound from [26]. While it does not improve the bound, which seems to be the best obtainable with the present methods, it shows the robustness of the flex-EA and features a simpler analysis than in [26]. Finally, it assumes a uniform starting point instead of an initialization with a spanning tree as in [26].

We note that the parameter settings for the $\boldsymbol{C}_i$ and particularly the $\boldsymbol{\ell}_i$ in Theorem 7.1 deviate from the recommended settings discussed in Section 4.1 and used earlier in this paper. In fact, we take a gray-box perspective here and use a minor amount of problem-specific knowledge to favor the small rates 1 and 2. If the expected value was to hold conditioned on an event of probability $1 - o(1)$ only, then the conditions on $\boldsymbol{\ell}_i$ for $i \geq 3$ could be relaxed.

Theorem 7.1. *Consider an instance to the MST problem, modeled with the classical fitness function from [22]. Let flex-EA with the following parameter settings run on this function:*

- *$\boldsymbol{C}_i = m^i \ln(m^c)$ for a sufficiently large constant $c > 0$ and $i = 1, 2$ (the other $\boldsymbol{C}_i$ may be chosen arbitrarily),*
- *$\boldsymbol{\ell}_1 = \omega(1/\ln m)$ and $\boldsymbol{\ell}_1 \leq 1/2$,*
- *$\boldsymbol{\ell}_2 = \Omega\big(1/(m^2 \ln m)\big)$, and $\boldsymbol{\ell}_i = o\big(1/(m^5 \ln^2 m)\big)$ for $i = 3..n$.*

Then with probability $1 - o(1)$, the runtime is at most

$$\begin{aligned}&((1 - \boldsymbol{\ell}_1)^{-1} + o(1))\big((m^2/2)(1 + \ln(\textstyle\sum_{i \in [m]} r_i))\big)\\ &\quad \leq ((1 - \boldsymbol{\ell}_1)^{-1} + o(1))\big(m^2 \ln m\big),\end{aligned}$$

where r_i is the rank of the ith edge in the sequence sorted by increasing edge weights. The expected runtime is bounded in the same way.

The proof had to be omitted from this paper. Its main idea is to consider a phase of length $\Theta\big((1 - \boldsymbol{\ell}_1)^{-1} m^2 \ln m\big)$ after finding the

first spanning tree and to show that within the phase, the archive only contains rates 1 and 2 w. h. p. Then the above-mentioned stochastic equivalence to RLS with a 2-bit flip neighborhood is applied.

8 A FUNCTION WHERE THE ARCHIVE STORES SEVERAL RATES SIMULTANEOUSLY

Most of the above results do not exploit the full power of the archive A; most of the time, it was sufficient that the archive contained only one rate and larger archive sizes were even detrimental for the analysis. A major difference of the flex-EA to classical stagnation detection is that a successful rate is kept for the next iteration; however, to some extent this idea was already present in stagnation detection with radius memory [26], which we will compare against.

The purpose of this section is to present an example where flex-EA needs to focus on two rates simultaneously. This can be beneficial on a multimodal function where fitness gaps of different sizes occur frequently. Our example resembles a Hurdle function [23] and features two different gap sizes, so that the archive should include the two corresponding rates. It seems possible to extend the example to a larger number of gap sizes.

Assume the numbers $s := (3/4)n - \sqrt{n}$, $(3/4)n$, $g := \log n$ and $\sqrt{n}/g$ are integers. We define

$$\textsc{TwoRates}(x) := \begin{cases} \mathrm{OM}(x) & \text{if } |x| < s \text{ or } |x| \geq (3/4)n \\ & \text{or } |x| = s + ig \text{ for } i \in \{0, 2, 3, 5, 6, \dots\}, \\ -1 & \text{otherwise,} \end{cases}$$

where the values of i alternatingly increase by 2 and 1.

The hurdles of width g, which are located in the interval $[s..3n/4]$ for the number of one-bits, are overcome at the corresponding rate with probability at least

$$\frac{\binom{n/4}{g}}{\binom{n}{g}} \geq \left(\frac{n/4-g}{n}\right)^g = \left(1 - \frac{\mathrm{O}(\log n)}{n}\right)^{\log n} 4^{-\log n} = \frac{(1-\mathrm{o}(1))}{n^2}$$

(using the inequalities $\frac{(n-k)^k}{k!} \leq \binom{n}{k} \leq \frac{n^k}{k!}$) and at most

$$\frac{\binom{n/4+\sqrt{n}}{g}}{\binom{n}{g}} \leq \left(\frac{n/4+\sqrt{n}}{n-g}\right)^g = 2^{\log(1/4+\mathrm{O}(1/\sqrt{n}))(\log n)} = \frac{(1+\mathrm{o}(1))}{n^2} \quad (1)$$

and accordingly those of width $2g$ with the probability bounds $(1-\mathrm{o}(1))/n^4$ and $(1+\mathrm{o}(1))/n^4$, respectively.

We now prove that the flex-EA optimizes TwoRates efficiently, while classical stagnation detection algorithms with and without radius memory need superpolynomial time. Moreover, the Fast (1+1) EA from [11] is by a at least a polylogarithmic factor of $\Omega\big((\log n)^{\beta-1/2}\big) = \omega\big(\sqrt{\log n}\big)$ slower than our approach.

Theorem 8.1. *The expected runtime of the flex-EA with the recommended parameter set on TwoRates is bounded by $\mathrm{O}(n^{4.5})$. For the stagnation detection algorithms of the type SD-RLS in [26, 28] with typical parameter choice $R = n^d$ for a constant $d > 0$, the expected runtime is $n^{\Omega(\log n)}$. Moreover, the expected runtime of the Fast (1+1) EA is by a factor $\Omega\big((\log n)^{\beta-1/2}\big)$ larger than for the flex-EA.*

In the proof, we shall need the following useful helper result. It relates to the fact that flipping exactly g zero-bits and no one-bits may not be optimal to cross a hurdle of size g, and it bounds the increase in probability when flipping both zero- and one-bits.

Lemma 8.2. *Let $x \in \{0,1\}^n$ be a search point with $a \geq 0$ one-bits and let $p(n, a, d, i)$, where $d, i \geq 0$, be the probability of mutating x into a point containing exactly $a+d$ one-bits when flipping $d+2i$ bits uniformly at random. If $d+2i \leq n^{1/3}$, then $p(n,a,d,i)/p(n,a,d,0) \leq (1+\mathrm{o}(1))(\mathrm{e}(2+d/i)(1-a/n))^i \leq (1+\mathrm{o}(1))\mathrm{e}^{d+2i}(1-a/n)^i$.*

The main idea for the upper bound for the flex-EA is as follows: the expected time to cross a hurdle is $\mathrm{O}(n^4)$ and there are $\Theta(\sqrt{n}/\log n)$ hurdles. The probability of choosing a beneficial rate of g or $2g$ is $\Omega(1/\log n)$ since the cardinality of the archive is bounded by $\mathrm{O}(\log n)$ with high probability. The lower bound for the Fast (1+1) EA stems from the fact that rates of $\Omega(\log n)$ must be chosen with high probability to cross a hurdle, which has probability $\mathrm{O}((\log n)^{-\beta})$ in the heavy-tailed mutation. Another factor of $\Theta(\sqrt{\log n})$ is lost since the Fast (1+1) EA uses standard-bit mutation instead of a flipping a deterministic number of bits.

We conjecture that the runtime of the SD-FEA from [14] on TwoRates is asymptotically at least as large as the one of the Fast (1+1) EA. Essentially, rates in the order $\Omega(\log n)$ must be used to obtain a sufficient probability of crossing a hurdle. Then in the same way as described above, the selection probability $\mathrm{O}((\log n)^{-\beta})$ in the heavy-tailed component impacts the runtime bound.

9 CONCLUSION

We introduced the flex-EA – a flexible evolutionary algorithm that maintains an archive of previously successful mutation rates. The algorithm is highly configurable and thus allows to exploit problem-specific knowledge. However, we also recommended a default configuration, based on heavy-tailed distributions and stagnation detection, which results in an essentially parameter-less algorithm. Using this recommended configuration, we proved that the flex-EA achieves standard runtimes on OneMax and on LeadingOnes, and that it achieves a near best-possible runtime on Jump for unbiased mutation-only algorithms. Furthermore, for a hurdle-like problem that features repetitive sub-structures, this configuration outperforms the Fast (1+1) EA (which uses heavy-tailed mutations) and stagnation detection. When provided with some mild problem-specific knowledge, the flex-EA matches the best known runtime result for the minimum spanning tree problem.

Although the performance of the flex-EA is very good on the problems we considered, the algorithm still has some shortcomings that can be improved. For one, on shifted jump benchmarks [34], the combination of heavy-tailed mutation and stagnation detection by Doerr and Rajabi [14] should prove superior, as the flex-EA basically defaults to classic stagnation detection. However, this shortcoming can be overcome by adjusting the mutation of the flex-EA in a similar way as done by Doerr and Rajabi [14] for their algorithm. Another potential shortcoming is that the archive of the flex-EA can become very crowded when many different mutation rates prove useful. If the mutation rates are removed in an incorrect order, this can result in long waiting times. It is unclear under what circumstances this effect actually occurs.

Overall, the flex-EA proposes an arguably new paradigm for evolutionary algorithms that performs very well and can be studied in multiple domains that exceed the content of this paper.

REFERENCES

[1] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. 2022. Fast mutation in crossover-based algorithms. *Algorithmica* 84 (2022), 1724–1761. https://doi.org/10.1007/s00453-022-00957-5

[2] Denis Antipov and Benjamin Doerr. 2020. Runtime analysis of a heavy-tailed $(1 + (\lambda, \lambda))$ genetic algorithm on jump functions. In *Prof. of PPSN '20*. Springer, 545–559. https://doi.org/10.1007/978-3-030-58115-2_38

[3] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. 2021. Fast immune system-inspired hypermutation operators for combinatorial optimization. *IEEE Transactions on Evolutionary Computation* 25, 5 (2021), 956–970. https://doi.org/10.1109/TEVC.2021.3068574

[4] Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. 2021. Escaping local optima with non-elitist evolutionary algorithms. In *Proc. of AAAI '21*. AAAI Press, 12275–12283. https://doi.org/10.1609/AAAI.V35I14.17457

[5] Duc-Cuong Dang, Anton V. Eremeev, Per Kristian Lehre, and Xiaoyu Qin. 2022. Fast non-elitist evolutionary algorithms with power-law ranking selection. In *Proc. of GECCO '22*. ACM, 1372–1380. https://doi.org/10.1145/3512290.3528873

[6] Benjamin Doerr and Carola Doerr. 2015. Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings. In *Proc. of GECCO '15*. ACM Press, 1335–1342. https://doi.org/10.1145/2739480.2754684

[7] Benjamin Doerr and Carola Doerr. 2020. Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. In *Theory of Evolutionary Computation – Recent Developments in Discrete Optimization*, Benjamin Doerr and Frank Neumann (Eds.). Springer, 271–321. https://doi.org/10.1007/978-3-030-29414-4_1

[8] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. k-bit mutation with self-adjusting k outperforms standard bit mutation. In *Proc. of PPSN '16*. Springer, 824–834. https://doi.org/10.1007/978-3-319-45823-6_77

[9] Benjamin Doerr, Yassine Ghannane, and Marouane Ibn Brahim. 2022. Towards a stronger theory for permutation-based evolutionary algorithms. In *Proc. of GECCO '22*. ACM, 1390–1398. https://doi.org/10.1145/3512290.3528720

[10] Benjamin Doerr, Taha El Ghazi El Houssaini, Amirhossein Rajabi, and Carsten Witt. 2023. How well does the metropolis algorithm cope with local optima?. In *Proc. of GECCO '23*. ACM, 1000–1008. https://doi.org/10.1145/3583131.3590390

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

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

[13] Benjamin Doerr and Zhongdi Qu. 2023. A first runtime analysis of the NSGA-II on a multimodal problem. *IEEE Transactions on Evolutionary Computation* 27, 5 (2023), 1288–1297. https://doi.org/10.1109/TEVC.2023.3250552

[14] Benjamin Doerr and Amirhossein Rajabi. 2023. Stagnation detection meets fast mutation. *Theoretical Computer Science* 946 (2023), 113670. https://doi.org/10.1016/j.tcs.2022.12.020

[15] Tobias Friedrich, Andreas Göbel, Francesco Quinzan, and Markus Wagner. 2018. Heavy-tailed mutation operators in single-objective combinatorial optimization. In *Proc. of PPSN '18*. Springer, 134–145. https://doi.org/10.1007/978-3-319-99253-2_11

[16] Tobias Friedrich, Francesco Quinzan, and Markus Wagner. 2018. Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In *Proc. of GECCO '18*. ACM, 293–300. https://doi.org/10.1145/3205455.3205515

[17] Thomas Jansen and Ingo Wegener. 1999. On the analysis of evolutionary algorithms – A proof that crossover really can help. In *Proc. of ESA '99*. Springer, 184–193. https://doi.org/10.1007/3-540-48481-7_17

[18] Martin Krejca and Carsten Witt. 2020. Theory of estimation-of-distribution algorithms. In *Theory of Evolutionary Computation – Recent Developments in Discrete Optimization*, Benjamin Doerr and Frank Neumann (Eds.). Springer, 405–442. https://doi.org/10.1007/978-3-030-29414-4

[19] Martin S. Krejca and Carsten Witt. 2024. A Flexible Evolutionary Algorithm With Dynamic Mutation Rate Archive. *CoRR* abs/2404.04 (2024). arXiv:2404.04015

[20] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2020. Simple Hyper-Heuristics Control the Neighbourhood Size of Randomised Local Search Optimally for LeadingOnes. *Evolutionary Computation* 28, 3 (2020), 437–461. https://doi.org/10.1162/EVCO_A_00258

[21] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2023. When move acceptance selection hyper-heuristics outperform Metropolis and elitist evolutionary algorithms and when not. *Artificial Intelligence* 314 (2023), 103804:1–103804:23. https://doi.org/10.1016/J.ARTINT.2022.103804

[22] Frank Neumann and Ingo Wegener. 2007. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science* 378 (2007), 32–40. https://doi.org/10.1016/j.tcs.2006.11.002

[23] Phan Trung Hai Nguyen and Dirk Sudholt. 2018. Memetic algorithms beat evolutionary algorithms on the class of hurdle problems. In *Proc. of GECCO '18*. ACM, 1071–1078. https://doi.org/10.1145/3205455.3205456

[24] Francesco Quinzan, Andreas Göbel, Markus Wagner, and Tobias Friedrich. 2021. Evolutionary algorithms and submodular functions: Benefits of heavy-tailed mutations. *Natural Computing* 20, 3 (2021), 561–575. https://doi.org/10.1007/s11047-021-09841-7

[25] Günther R. Raidl, Gabriele Koller, and Bryant A. Julstrom. 2006. Biased mutation operators for subgraph-selection problems. *IEEE Transaction on Evolutionary Computation* 10, 2 (2006), 145–156. https://doi.org/10.1109/TEVC.2006.871251

[26] Amirhossein Rajabi and Carsten Witt. 2021. Stagnation detection in highly multimodal fitness landscapes. In *Proc. of GECCO '21*. ACM Press, 1178–1186. https://doi.org/10.1145/3449639.3459336

[27] Amirhossein Rajabi and Carsten Witt. 2022. Self-adjusting evolutionary algorithms for multimodal optimization. *Algorithmica* 84, 6 (2022), 1694–1723. https://doi.org/10.1007/s00453-022-00933-z

[28] Amirhossein Rajabi and Carsten Witt. 2023. Stagnation detection with randomized local search. *Evolutionary Computation* 31, 1 (2023), 1–29. https://doi.org/10.1162/evco_a_00313

[29] I. Rechenberg. 1973. *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag.

[30] Joachim Reichel and Martin Skutella. 2009. On the size of weights in randomized search heuristics. In *Proc. of FOGA 2009*. 21–28. https://doi.org/10.1145/1527125.1527130

[31] Günter Rudolph. 1997. *Convergence properties of evolutionary algorithms*. Verlag Dr. Kovač.

[32] M. Schumer and Kenneth Steiglitz. 1968. Adaptive step size random search. *IEEE Transactions on Automatic Control* 13, 3 (1968), 270–276. https://doi.org/10.1109/TAC.1968.1098903

[33] Dirk Sudholt. 2013. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 17, 3 (2013), 418–435. https://doi.org/10.1109/TEVC.2012.2202241

[34] Carsten Witt. 2023. How majority-vote crossover and estimation-of-distribution algorithms cope with fitness valleys. *Theoretical Computer Science* 940 (2023), 18–42. https://doi.org/10.1016/J.TCS.2022.08.014

[35] Mengxi Wu, Chao Qian, and Ke Tang. 2018. Dynamic mutation based Pareto optimization for subset selection. In *Proc. of ICIC '18*. Springer, 25–35. https://doi.org/10.1007/978-3-319-95957-3_4

[36] Christine Zarges. 2020. Theoretical foundations of immune-inspired randomized search heuristics for optimization. In *Theory of Evolutionary Computation – Recent Developments in Discrete Optimization*, Benjamin Doerr and Frank Neumann (Eds.). Springer, 443–474. https://doi.org/10.1007/978-3-030-29414-4_1

[37] Weijie Zheng and Benjamin Doerr. 2023. Theoretical analyses of multiobjective evolutionary algorithms on multimodal objectives. *Evolutionary Computation* 31, 4 (2023), 337–373. https://doi.org/10.1162/EVCO_A_00328