Stagnation Detection meets Fast Mutation

Benjamin Doerr, Amirhossein Rajabi

Stagnation Detection Meets Fast Mutation*

Benjamin Doerr[1] and Amirhossein Rajabi[2]

[1] Laboratoire d'Informatique (LIX), CNRS, École Polytechnique, Institute Polytechnique de Paris, Palaiseau, France
[2] Technical University of Denmark, Kgs. Lyngby, Denmark

Abstract. Two mechanisms have recently been proposed that can significantly speed up finding distant improving solutions via mutation, namely using a random mutation rate drawn from a heavy-tailed distribution ("fast mutation", Doerr et al. (2017)) and increasing the mutation strength based on a stagnation detection mechanism (Rajabi and Witt (2020)). Whereas the latter can obtain the asymptotically best probability of finding a single desired solution in a given distance, the former is more robust and performs much better when many improving solutions in some distance exist.
In this work, we propose a mutation strategy that combines ideas of both mechanisms. We show that it can also obtain the best possible probability of finding a single distant solution. However, when several improving solutions exist, it can outperform both the stagnation-detection approach and fast mutation. The new operator is more than an interleaving of the two previous mechanisms and it outperforms any such interleaving.

Keywords: Mutation operator · parameter control · jump functions · theory.

1 Introduction

Leaving local optima is a challenge for evolutionary algorithms. Mutation-based approaches are challenged by the fact that the typical mutation rate of $p = 1/n$ rarely leads to offspring in a larger distance from the parent. When using larger mutation rates, the choice of the mutation rate is critical and small constant-factor deviations from the optimal rate can lead to huge performance losses [11, Cor. 4.2].

Two ways to overcome this problem were proposed recently, namely the use of a random mutation rate sampled from a power-law distribution ("fast mutation") [11] and the successive increase of the mutation rate when a stagnation-detection mechanism indicates that the current rate is unlikely to generate solutions not seen yet [19]. An improved version of this stagnation-detection approach [21], the so-called SD-RLS algorithm based on k-bit mutation instead of standard bit mutation, can find a single improving solution in distance m in expected time $(1 + o(1))\binom{n}{m}$ (without knowing that the distance to the desired

* Author generated version.

solution is m). Apart from lower order terms, this is the same runtime that can be obtained via a repeated use of the best unbiased mutation operator that is aware of m (which is, naturally, flipping m random bits). It is faster than the fast $(1+1)$ EA by a factor of $\Omega(m)$.

While the SD-RLS algorithm thus is very efficient in finding a single desired solution (and thus has very good runtimes on the classic jump functions benchmark (see Section 5 for a definition)), this algorithm has a poor performance when there are several improving solutions in distance m as now the stagnation detection approach leads to too much time spent on too small mutation strengths. Taking as an extreme example the generalized jump function [5] (see again Section 5 for a definition) having a valley of low fitness of width δ, $\delta \geq 2$ a constant, in distance $n/4$ from the optimum, we easily see that the SD-RLS takes an expected time of $\Omega(n^{\delta-1})$ to traverse the fitness valley, whereas the $(1+1)$ EA both with the classic mutation operator and with fast mutation does so in expected constant time.

Our results: Based on the insight that fast mutation and stagnation detection have complementary strengths, we design a mutation-based approach that takes inspiration from both approaches. We follow, in principle, the basic version of the improved stagnation-detection approach of [21], that is, we start with mutation strength $r = 1$ and increase r gradually. More precisely, when strength r has been used for a certain number ℓ_r of iterations without that an improvement was found, we increase r by one since we assume that no improvement in distance r exists (we omit some technical details in this first presentation of our approach, e.g., that we do not increase r beyond $n/2.1$, and refer the reader to Algorithm 1 for the full details). Different from [21], when the current strength is r, we do not always flip r random bits as mutation operation, but we choose a random number X_r of bits to flip. This number is equal to r with probability $1-\gamma$, where γ is an algorithm parameter that is usually small (a small constant or $o(1)$). With probability γ, however, X_r deviates from r by an amount following a power-law distribution with exponent β. The precise definition of this case (see again Algorithm 1) is not too important, so for this first exposition we can assume that we sample D from a power-law distribution (with exponent β) on the positive integers and then, each with probability $1/2$, flip $r+D$ or $r-D$ random bits (where we do nothing if this number is not between 1 and n).

Since with probability $1-\gamma$ we essentially follow the basic approach of [21], it is not surprising that we find a single closest improving solution in distance m in an expected time of $\frac{1}{1-\gamma}(1+o(1))\binom{n}{m}$, again without that the algorithm needs to know m (Theorem 5). If $\gamma = o(1)$, this is again the optimal time of $(1+o(1))\binom{n}{m}$ discussed above. We note, however, that our algorithm is simpler than the solution presented in [21]. The basic SD-RLS algorithm proposed in [21] obtains a runtime of $(1+o(1))\binom{n}{m}$ only with high probability and otherwise fails. To turn this algorithm into one that never fails and has an expected runtime of $(1+o(1))\binom{n}{m}$, a robust version of the SD-RLS was developed in [21] as well. This version repeats previous phases as follows. When the ℓ_r uses of strength r have not led to an improvement, before increasing the rate to $r+1$, first another ℓ_i

iterations are performed with strength i, for $i = r-1, \dots, 1$. In our approach, such an additional effort is not necessary since the fast mutations automatically render the algorithm robust.

The use of a heavy-tailed mutation rate also helps in situations where the stagnation-detection mechanism takes too long to use larger mutation strengths. Since in phases $r = 1, \dots, 2m$ the probability to flip m bits is at least $\gamma/2$ times the probability of this event in a run of the fast $(1+1)$ EA, it is not surprising that our algorithm finds an improvement in distance m is at most $2/\gamma$ times the time of the fast $(1+1)$ EA, which as discussed above can be significantly faster than the SD-RLS. Such a result could also have been obtained from a simple interleaving of SD-RLS and fast $(1+1)$ EA iterations. Since our heavy-tailed choices of the mutation strength, however, take into account the current strength r, we often obtain better runtimes, often better than both the SD-RLS and the fast $(1+1)$ EA. As the precise statement of these results is technical, we defer the details to Section 4.

As a simple example showing the outperformance of our algorithm, we regard the generalized jump function $\textsc{Jump}_{m,\delta:=m-\Delta}$ for a constant value of $\Delta \geq 2$ and $m = \omega(1)$. This jump function is similar to the classic jump function $\textsc{Jump}_m$, but the valley of low fitness consists not of all search points in positive distance at most $m-1$ from the optimum, but only of those in distance $\Delta+1, \dots, m-1$. Consequently, from the local optimum there is not a single improving solution, but $\Theta(n^\Delta)$. Note that this is still relatively few compared to the fitness valley of size essentially $\binom{n}{m-1}$. On this generalized jump function, the expected runtime of SD-RLS is $O\Big(\binom{n}{\delta-1}\ln(R)\Big)$, the one of the fast $(1+1)$ EA is $O\big(\delta^{\beta-0.5}(en/\delta)^\delta n^{-\Delta}\big)$, and the one of our algorithm is at most $O\big(\binom{n}{\delta}n^{-\Delta}\gamma^{-1}\big)$ (Corollary 11). Since it is also clear that any interleaving of SD-RLS and fast $(1+1)$ EA iterations cannot give a better runtime than the one of the two pure algorithms, this result shows that our algorithm can beat SD-RLS and fast EA (and any simple mix of them) when there are several improving solutions in a given distance.

A short experimental evaluation of the algorithms discussed so far shows that the advantages of our algorithm, proven only via asymptotic runtime results, are also visible for moderate problems sizes.

Structure of this paper: After reviewing the most relevant previous works in Section 2, we introduce our new algorithm in Section 3. In Section 4, we analyze via mathematical means how our algorithm finds an improvement in distance m both when this is typically achieved in phase m (e.g., when there is only one improving solution in distance m) and when this is achieved earlier via the heavy-tailed rates. We use these results in Section 5 to prove several runtime results, among others, for generalized jump functions. We present some experimental results in Section 6. In Section 7, we discuss recommendations on how to set the parameters of our algorithm. We conclude the paper with a short discussion of our results and a pointer to possible future work in Section 8. Due to space restrictions, all proofs had to be omitted from this paper; however, they are available in the preprint [12].

2 Previous Works

This work aims at combining the advantages of stagnation detection and heavy-tailed mutation, so clearly these topics contain the most relevant previous works. Both integrate into the wider questions of how to optimally set the mutation strength of evolutionary algorithms (for this we refer to the recent survey [10]) and how evolutionary algorithms can leave local optima (here we refer to [9, Section 2.1] for a discussion of non-elitist approaches and to the introduction of [8] for a discussion of crossover-based approaches).

For elitist mutation-based approaches, it is clear that when the population has converged to a local optimum the only way to leave this is by mutating a solution from the local optimum into an at least as good solution outside this local optimum. It was observed in [11] (the earlier work [18] contains similar findings for the special case that the nearest improving solution is in Hamming distance two or three) that standard bit mutation with mutation rate $p = \frac{1}{n}$, which is the most recommended way of doing mutation, is not perfectly suitable to perform larger jumps in the search space. In fact, when the nearest improving solution is in Hamming distance m, then a mutation rate of $p = \frac{m}{n}$ is much better, leading to a speed-up by a factor of order $m^{\Theta(m)}$.

Since [11] also observed that missing the optimal rate by a small constant factor leads to performance losses exponential in m, it was proposed to use a mutation rate that is drawn from a (heavy-tailed) power-law distribution. Without the need to know m, this approach led to runtimes that exceed the ones obtained from the optimal rate $p = \frac{m}{n}$ by only a small factor polynomial in m. This price for universality can be made as low as $\Theta(m^{0.5+\varepsilon})$, but not smaller than $\Theta(\sqrt{m})$. Various variants of heavy-tailed mutation operators have been proposed subsequently, also heavy-tailed choices of other parameters have been used with great success [1–4, 6, 7, 13, 15–17, 24].

A different way to cope with local optima was proposed in [19]. When an algorithm is stuck in a local optimum for a sufficiently long time, then with high probability it has explored all search points in a certain radius. Consequently, it is safe to increase the mutation rate, which increases the probability to generate more distant solutions. This is the main idea of a series of works on stagnation detection [19–21]. As shown in [19], this approach can save the polynomial price for universality of the heavy-tailed approach and thus obtain runtimes of the same asymptotic order as when using the optimal (problem-specific) mutation rate. By replacing standard bit mutation with m-bit flips, the time to find a particular solution in Hamming distance m was further reduced to $(1+o(1))\binom{n}{m}$, the same time (apart from lower order terms) one would obtain with the best unbiased mutation operator (which consists of flipping m random bits).

To be precise, two approaches are discussed in [21]. The simple one, obtained from just replacing standard bit mutation in [19] by r-bit mutation, obtains the desired runtimes with high probability, but fails completely with some very small probability. For this reason, also a robust version of the algorithm was proposed in [21], which by cyclically reverting to smaller mutation strengths overcomes the problem that, with small probability, a given solution in distance m is not

Algorithm 1: The SD-FEA$_{\beta,\gamma,R}$ for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$. Its parameters are the power-law exponent $\beta > 1$, the probability γ to deviate from rate r in phase r, and the parameter R which defines the maximum length of the r-th phase at $\ell_r = (1-\gamma)^{-1}\binom{n}{r}\ln(R)$.

```
Select x uniformly at random from {0,1}^n and set r_1 ← 1;
u ← 0;
for t ← 1, 2, ... do
    Set s = r_t                                with probability 1−γ or
        s = r_t + pow(β, n − r_t)              with probability γ/2 or
        s = r_t − pow(β, max{1, r_t − 1})      with probability γ/2;
    Create y by flipping s bits in a copy of x uniformly at random;
    u ← u + 1;
    if f(y) > f(x) then
        x ← y;
        r_{t+1} ← 1;
        u ← 0;
    else if f(y) = f(x) and r_t = 1 then
        x ← y;
    if u ≥ ℓ_{r_t} then
        r_{t+1} ← min{r_t + 1, ⌊n/2.1⌋};
        u ← 0;
    else
        r_{t+1} ← r_t;
```

found in the phase which uses m-bit flips. In [20], a variation of SD-RLS was proposed that keeps the successful strength after leaving local optima with the help of the radius memory mechanism, which is beneficial on highly multimodal fitness landscapes. The idea of stagnation detection has also been successfully used in multi-objective evolutionary computation [13].

3 Combining Fast Mutation and Stagnation Detection: The Algorithm SD-FEA$_{\beta,\gamma,R}$

We propose the algorithm SD-FEA$_{\beta,\gamma,R}$ for the maximization of pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$ as defined in Algorithm 1. The function $\mathrm{pow}(\beta, u)$ samples from a power-law distribution with exponent β and range $[1..u]$ as defined in Equation (1) below.

The general idea of this algorithm is that it increases the mutation strength r to $r+1$ when the improvement is not in Hamming distance r with at least a constant probability (with probability $1/R$ roughly) using the stagnation detection mechanism. While the strength is r, called in phase r, the algorithm looks at larger or smaller Hamming distances (with probability γ) besides using the current strength r. The distribution of the distance of the search radius from

to the current strength r follows a power-law distribution. An integer random variable X follows a power-law distribution with parameters β and u if

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

where $C_{\beta,u} := (\sum_{j=1}^{u} j^{-\beta})^{-1}$ is the normalization coefficient. The function $\mathrm{pow}(\beta, u)$ used in Algorithm 1 returns a sample from this distribution.

The algorithm starts with a search point selected uniformly at random from the search space $\{0,1\}^n$ and with the initial strength $r = 1$. There is a counter u for counting the number of unsuccessful steps in finding a strict improvement with the current strength. When the counter exceeds the maximum phase length ℓ_r, the strength r increases by one but not exceeding $n/2.1$. When the algorithm makes progress, the counter and strength are reset to their initial values.

The mutation, which we call s-flip in the following, flips exactly s bits randomly chosen as follows. With probability $1-\gamma$, the algorithm flips exactly r bits in phase r. However, with probability γ, the algorithm deviates from this choice and instead flips a number of bits which differs from r, in either direction, by a value following a power-law distribution. The distribution over s is analyzed in Lemma 1 below.

In this paper, we use maximum phase lengths of

$$\ell_r = \binom{n}{r} / (1-\gamma)\ln(R). \tag{2}$$

This choice is designed for pseudo-Boolean fitness functions. For other search spaces, the maximum phase length should be $\ell_r = |S_r|/(1-\gamma)\ln(R)$, where $|S_r|$ is the number of search points in distance r from the current search point or an upper bound for this. The maximum phase length defined in Equation (2) has a parameter R controlling the probability of failing to find an improvement at the "right" strength. To prove our theoretical results, R should be selected at least $e^{1/\gamma}$. In Section 7, we give some recommendations for choosing the parameters of the SD-FEA$_{\beta,\gamma,R}$.

As *runtime* of a heuristic algorithm on a fitness function f, we define the first point of time t where a search point of maximal fitness has been evaluated.

4 Analysis of the SD-FEA$_{\beta,\gamma,R}$

In this paper, let us define by the *individual gap* of $x \in \{0,1\}^n$ the minimum Hamming distance of x from points with strictly larger fitness function value, that is,

$$\mathrm{IndividualGap}(x) := \min\{H(x,y) : f(y) > f(x), y \in \{0,1\}^n\}.$$

By the *fitness level* of x, we mean all the search points with fitness value $f(x)$. We call the *fitness level gap* of a point $x \in \{0,1\}^n$ the maximum of all individual

gap sizes in the fitness level of x, i. e.,

$$\text{FitnessLevelGap}(x) \coloneqq \max\{\text{IndividualGap}(y) : f(y) = f(x), y \in \{0,1\}^n\}.$$

If the algorithm creates a point at the Hamming distance IndividualGap(x) from the current search point x, with positive probability an improvement can be found. Note that FitnessLevelGap(x) = 1 is allowed, so the definition also covers search points that are not local optima. As long as a strict improvement is not made, the FitnessLevelGap remains the same, although the current search point might be replaced with another search point in the fitness level in phase 1, that is, when the strength is 1.

We now analyze how the SD-FEA$_{\beta,\gamma,R}$ finds better selections. Let the current search point be x. We define by phase r all points of time where radius r is used for search points with fitness value $f(x)$, i. e., while in the fitness level of x. Let E_r be the event of **not** finding the optimum within phase r. For $j \geq i$, let E_i^j denote the event of not finding a strict improvement within phases i to j. Formally, $E_i^j = E_i \cap \cdots \cap E_j$.

Before computing the probabilities of these events, we need to know the distribution of the offspring in an iteration. The following lemma will be used throughout this paper, showing the distribution of the number of flipping bits (i. e., the variable s in Algorithm 1) in each iteration. We recall that in phase r, with a relatively large probability $1-\gamma$, the algorithm flips r bits. However, with probability γ, it uses power-law distributions to flip less or more than r bits.

Lemma 1. *Let r be the current strength in an iteration of the algorithm SD-FEA$_{\beta,\gamma,R}$. Let X be the integer random variable corresponding to the number of bits that are flipped, that is, the variable s in Algorithm 1. Then*

$$\Pr[X = \alpha] = \begin{cases} (\gamma/2) \cdot C_{\beta,r-1} \cdot (r-\alpha)^{-\beta} & 1 \leq \alpha < r, \\ 1-\gamma & \alpha = r, \\ (\gamma/2) \cdot C_{\beta,n-r} \cdot (\alpha-r)^{-\beta} & r < \alpha \leq n, \end{cases}$$

and for $r = 1$, $\Pr[X = 0] = \gamma/2$.

The following lemma estimates the probability of reaching a phase that is greater than the fitness gap size. In the statement of the lemma, recall that the parameter R controls the length of the phase.

Lemma 2. *Let $\beta > 1$, $0 < \gamma < 1$ and $R > 1$. Consider the SD-FEA$_{\beta,\gamma,R}$ maximizing a pseudo-Boolean fitness function $f\colon \{0,1\}^n \to \mathbb{R}$. Let $x \in \{0,1\}^n$ be the current search point immediately following a strict improvement or the initial search point. Let $m = \text{IndividualGap}(x)$. Let E_1^{r-1} denote the probability of not finding an improvement in phases 1 to $r-1$. Then for $m < r \leq \lfloor \frac{n}{2.1} \rfloor$, we have*

$$\Pr[E_1^{r-1}] \leq R^{-1-(\gamma/2)\cdot\left(\frac{\ln(1.1)}{\beta}\right)^{\beta} C_{\beta,n}(r-m-1)}.$$

The next lemma is used to estimate the number of iterations in phases larger than the fitness level gap. With a good choice of the parameters γ and R, the following result becomes $o(1/s_m)$, that is, the number of steps at larger strengths is negligible compared to the number of steps at the phase m.

Lemma 3. *Let $\beta > 1$, $0 < \gamma < 1$ and $R \geq e^{1/\gamma}$. Consider the SD-FEA$_{\beta,\gamma,R}$ maximizing a pseudo-Boolean fitness function $f\colon \{0,1\}^n \to \mathbb{R}$. Let $x \in \{0,1\}^n$ be the current search point immediately following a strict improvement or the initial search point. Assume $m = \text{FitnessLevelGap}(x)$ and $m \leq \lfloor n/2.1 \rfloor$. Let s_m be a lower bound on the probability that an improvement is found from search points in the fitness level of x conditional on flipping m bits. Then the expected number of iterations spent with strengths larger than m is at most*

$$O\left(R^{-1}\gamma^{-1}\frac{1}{s_m}\right).$$

The following lemma, a combinatorial inequality taken from [21], will be used to count the number of iterations spent with strengths smaller than the fitness level gap.

Lemma 4 (Lemma 1 in [21]). *For any integer $m \leq n/2$, we have*

$$\sum_{i=1}^{m}\binom{n}{i} \leq \frac{n-(m-1)}{n-(2m-1)}\binom{n}{m}.$$

We now present the first main result. In the following theorem, we provide two rigorous upper bounds on the escaping time from a local optimum.

Theorem 5. *Let $\beta > 1$, $0 < \gamma < 1$ and $R \geq e^{1/\gamma}$. Consider the SD-FEA$_{\beta,\gamma,R}$ maximizing a pseudo-Boolean fitness function $f\colon \{0,1\}^n \to \mathbb{R}$. Let $x \in \{0,1\}^n$ be the current search point immediately following a strict improvement or the initial search point. Let $m = \text{FitnessLevelGap}(x)$. Define T as the time SD-FEA$_{\beta,\gamma,R}$ takes to create a strict improvement. If $m \leq n/2.1$, then*

$$E[T] \leq \binom{n}{m}\left(\frac{1}{1-\gamma} + O\left(\frac{m\ln(R)}{(1-\gamma)n} + R^{-1}\gamma^{-1}\right)\right).$$

Moreover, for all $m \leq n$, we have

$$E[T] = O\left(2^n\frac{\ln(R)}{1-\gamma} + \gamma^{-1}\binom{n}{m}|\lfloor\tfrac{n}{2.1}\rfloor - m|^{\beta}\right).$$

Theorem 5 provides a good upper bound on the escaping time from a local optimum when there are only few ways to leave it. However, it is not as good when there are many ways to leave the local optimum. The following theorem considers such scenarios. The constant r' defined in the theorem basically represents the first phase that the probability of finding one of the improvements is at least constant, and its value is an integer between 1 and m.

Theorem 6. *Let $\beta > 1$, $0 < \gamma < 1$ and $R \geq e^{1/\gamma}$. Consider the SD-FEA$_{\beta,\gamma,R}$ maximizing a pseudo-Boolean fitness function $f\colon \{0,1\}^n \to \mathbb{R}$. Let $x \in \{0,1\}^n$ be the current search point immediately following a strict improvement or the initial search point. Let $m = \text{FitnessLevelGap}(x)$ and s_m be a lower bound on the probability that a strict improvement is found from search points in the fitness level of x conditional on flipping m bits. Define T as the time SD-FEA$_{\beta,\gamma,R}$ takes to create a strict improvement. If $m \leq n/2.1$, then*

$$E[T] \leq \frac{1}{s_m} \cdot \frac{1}{\gamma}(m - r')^\beta \cdot O\left(1 + \frac{r' \ln(R)}{(1-\gamma)n}\right),$$

where $r' = \min\left\{m, \arg\max_r \left\{\binom{n}{r} \leq \frac{1}{s_m}\frac{1}{\gamma}(m-r)^\beta\right\}\right\}$.

After having established some tools for obtaining upper bounds on the time required to escape from local optima, we now analyze the performance of SD-FEA$_{\beta,\gamma,R}$ on the sub-problems without local optima. On *unimodal functions* the gap of all search points in the search space (except for the global optima) is 1, so the algorithm can always make progress in phase 1.

In the following theorem, we state how SD-FEA$_{\beta,\gamma,R}$ behaves on unimodal functions compared to RLS using an upper bound based on the fitness-level method [22]. The theorem and its proof are similar to the second part of Lemma 4 in [21], and with a good choice of parameters γ and R, the same asymptotic result can be achieved (see the following corollary).

Theorem 7. *Let $\beta > 1$, $0 < \gamma < 1$ and $R \geq e^{1/\gamma}$. Let $f\colon \{0,1\}^n \to \mathbb{R}$ be a unimodal function and $|\text{Im}(f)|$ be the number of its fitness values. Let f_i be the i-th fitness value in an increasing order of the fitness values of f. We consider all fitness levels $A_1, \ldots, A_{|\text{Im}(f)|}$ such that A_i contains search points with fitness value f_i. Let s_i be a lower bound on the probability that RLS finds an improvement from any search point in A_i. Denote by T the runtime of SD-FEA$_{\beta,\gamma,R}$ on f. Then*

$$E[T] \leq \left(\frac{1}{1-\gamma} + O\big(R^{-1}\gamma^{-1}\big)\right) \sum_{i=1}^{|\text{Im}(f)|-1} \frac{1}{s_i}.$$

The following unimodal benchmark functions OneMax and LeadingOnes have been extensively studied in the literature. They are defined by

$$\textsc{OneMax}(x) \coloneqq \|x\|_1,$$
$$\textsc{LeadingOnes}(x) \coloneqq \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$$

for all $x = (x_1, \ldots, x_n) \in \{0,1\}^n$, where $\|x\|_1$ is the number of one-bits in the bit string.

The corollary below is a result of Theorem 7 applied on the unimodal functions OneMax with $s_i = (n - (i-1))/n$ and LeadingOnes with $s_i = 1/n$.

Corollary 8. *The expected runtime of the SD-FEA$_{\beta,\gamma,R}$ with $\beta > 1$, $\gamma = o(1)$ and $R \geq e^{1/\gamma}$ on* OneMax *is at most $(1+o(1))n \ln n$ and on* LeadingOnes *is at most $(1+o(1))n^2$.*

5 Analysis on Jump$_{k,\delta}$

In this section, we use the results in the previous section to prove a bound on a generalization of Jump$_\delta$ called Jump$_{k,\delta}$ with two parameters k and δ, see Figure 1 for a depiction.

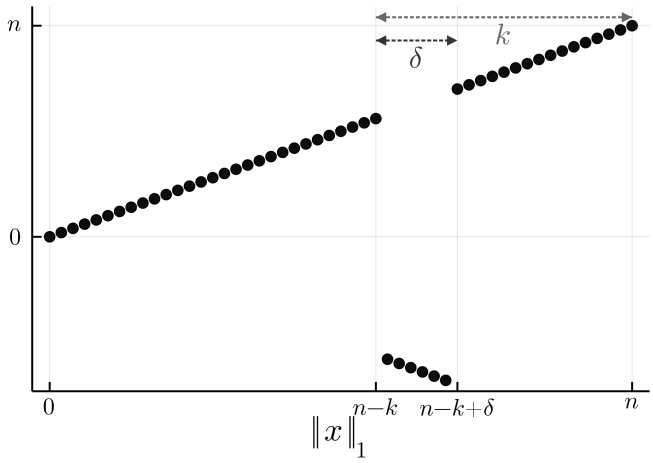

Fig. 1. The function Jump$_{k,\delta}$.

This function is based on the well-known Jump benchmark [14], in which the place of the jump with size δ starts at the Hamming distance k from the global optimum. In other words, after the jump, there is a unimodal sub-problem of length $k-\delta$. The classical Jump function is a special case of Jump$_{k,\delta}$ with $k = \delta$, i. e., Jump$_\delta$ = Jump$_{\delta,\delta}$. Formally, for all $x \in \{0,1\}^n$, we have

$$\textsc{Jump}_{k,\delta}(x) = \begin{cases} \|x\|_1 & \text{if } \|x\|_1 \in [0..n-k] \cup [n-k+\delta..n], \\ -\|x\|_1 & \text{otherwise.} \end{cases}$$

We refer the interested reader to see [5] for more information about Jump$_{k,\delta}$, where the performance of the $(1+1)$ EA, the (1+1) FEA$_\beta$, and the robust version of SD-RLS (SD-RLS$^{\mathrm{r}}$) are carefully analyzed. Also, Rajabi and Witt [20] independently define the jump function with an offset to analyze the recovery time for the strength in the algorithm SD-RLS with radius memory (SD-RLS$^{\mathrm{m}}$) after leaving the local optimum. Recently, Witt in [23] analyzes the performance

of some other algorithms on the function $\textsc{Jump}_{k,\delta}$ (which is called JumpOffset in the paper).

We want to show that the algorithm SD-FEA$_{\beta,\gamma,R}$ performs relatively efficiently on $\textsc{Jump}_{k,\delta}$ in both cases when $k = \delta$ (i. e., $\textsc{Jump}_\delta$) and $k > \delta$. In the first case, when there is only one improving solution, SD-FEA$_{\beta,\gamma,R}$ with $\gamma = o(1)$ optimizes $\textsc{Jump}_\delta$ as efficient as SD-RLS$^{\text{r}}$ thanks to Theorem 5. The result is formally proven in Theorem 9.

Theorem 9. *The expected runtime $E[T]$ of SD-FEA$_{\beta,\gamma,R}$ with $\beta > 1$, $\gamma = o(1)$ and $R \geq e^{1/\gamma}$ on $\textsc{Jump}_\delta$ with $2 \leq \delta = o(n/\ln(R))$ satisfies*

$$E[T] \leq \binom{n}{\delta}(1 + o(1)).$$

For $\gamma = \Theta(1)$, by closely following the analysis of Theorem 9, it is easy to see that the expected runtime of SD-FEA$_{\beta,\gamma,R}$ on $\textsc{Jump}_\delta$ is

$$\binom{n}{\delta}\left(\frac{1}{1-\gamma} + o(1)\right),$$

which is still very efficient.

We now present an upper bound on the runtime of the proposed algorithm on $\textsc{Jump}_{k,\delta}$.

Theorem 10. *The expected runtime $E[T]$ of SD-FEA$_{\beta,\gamma,R}$ with $\beta > 1$, $0 < \gamma < 1$ and $R \geq e^{1/\gamma}$ on $\textsc{Jump}_{k,\delta}$ with $\delta = o(n/\ln(R))$ satisfies*

$$E[T] = O\left(\binom{n}{\delta}\binom{k}{\delta}^{-1}(\delta - r')^\beta \cdot \gamma^{-1} + n \ln n\right),$$

where $r' = \min\left\{\delta, \arg\max_r\left\{\binom{n}{r} \leq \binom{n}{\delta}\binom{k}{\delta}^{-1}\frac{1}{\gamma}(\delta - r)^\beta\right\}\right\}$.

In the following corollary, we see a scenario where we have $r' \geq \delta - c$ for some constant c, resulting in that the term $(\delta - r')^\beta$ disappears from the asymptotic upper bound. This is also an example where the SD-FEA$_{\beta,\gamma,R}$ can asymptotically outperform the (1+1) FEA$_\beta$.

Corollary 11. *Let $\Delta \geq 2$ be a constant. The expected runtime $E[T]$ of SD-FEA$_{\beta,\gamma,R}$ with $\beta > 1$, $0 < \gamma < 1$ and $R \geq e^{1/\gamma}$ on $\textsc{Jump}_{k,\delta}$ with $k = \omega(1) \cap O(\ln n)$ and $\delta = k - \Delta$ satisfies*

$$E[T] = O\left(\binom{n}{\delta}\binom{k}{\delta}^{-1}\gamma^{-1}\right).$$

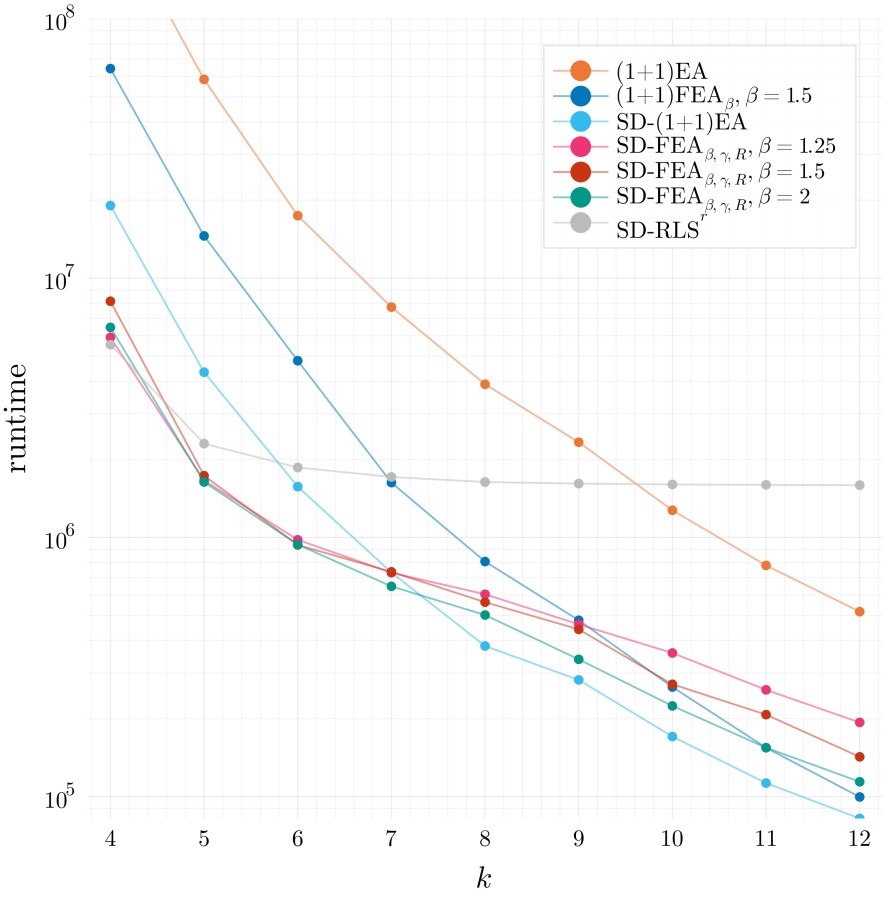

Fig. 2. Average number (over 200 runs) of fitness calls the mentioned algorithms spent to optimize $\textsc{Jump}_{k,4}$ ($n = 100$) with different values for k.

6 Experiments

In this section, we present the results of the experiments carried out to measure the performance of the proposed algorithm and several related ones on concrete problem sizes.

We ran an implementation of SD-FEA$_{\beta,\gamma,R}$ with $\beta \in \{1.25, 1.5, 2\}$, $\gamma = 1/4$ and $R = 25$ on the fitness function $\textsc{Jump}_{k,\delta}$ of size $n = 100$ with the jump size $\delta = 4$ and k varying from 4 to 13. We recall that we have the classical $\textsc{Jump}$ function for $k = 4$. We compared our algorithm with the classical $(1 + 1)$ EA with standard mutation rate $1/n$, the (1+1) FEA$_\beta$ from [11] with $\beta = 1.5$, the SD-(1+1) EA presented in [19] with $R = n^2$, and SD-RLS$^{\mathrm{r}}$ from [21] with $R = n^2$. The parameter settings for these algorithms were all recommended in the

corresponding papers. The parameter values for our algorithm were chosen in an ad-hoc fashion, slightly inspired by our theoretical results. All data presented is the average number of fitness calls over 200 runs.

As can be seen in Figure 2, SD-RLS$^{\mathrm{r}}$ outperforms the rest of the algorithms for $k = 4$, i. e., when there is only one improving solution for local optima. Our SD-FEA$_{\beta,\gamma,R}$ needs roughly $(1-\gamma)^{-1}$ times more fitness function calls than that since it "wastes" a fraction of γ of the iterations on wrong mutation strengths in phase 4. Not all these iterations are wasted as the small differences for different values of β show. The higher β is, the smaller values the power-law distribution typically takes, meaning that the mutation rate in these iterations stays closer to the ideal one. All three variants of the SD-FEA$_{\beta,\gamma,R}$ significantly outperform the (1+1) FEA$_\beta$, SD-(1+1) EA and $(1 + 1)$ EA. As k is increasing, the average running time of SD-RLS$^{\mathrm{r}}$ improves only little and remains almost without change after $k = 5$; consequently, this algorithm becomes less and less competitive for growing k. This is natural since this algorithm necessarily has to reach phase 4 to be able to flip 4 bits. All other algorithms, especially the (1+1) FEA$_\beta$, perform increasingly better with larger k.

In a middle regime of $k \in \{5, 6, 7\}$, the SD-FEA$_{\beta,\gamma,R}$ has the best average running time among the algorithms regarded. Although both with $k = 4$ and for $k \geq 8$, the SD-FEA$_{\beta,\gamma,R}$ is not the absolutely best algorithm, but its performance loss over the most efficient algorithm (SD-RLS$^{\mathrm{r}}$ for $k = 4$ and SD-(1+1) EA for $k \geq 8$) is small. This finding supports our claim that our algorithm is a good approach to leaving local optima of various kinds.

For a large k, such as 10 or 11, the good performance of the SD-(1+1) EA and (1+1) FEA$_\beta$ might appear surprising. The reason for the slightly weaker performance of our algorithm is the relatively small width of the valley of low fitness ($\delta = 4$), where our algorithm cannot fully show its advantages, but pays the price of sampling from the right heavy-tailed distribution only with probability $\gamma/2$.

7 Recommended Parameters

In this section, we use our theoretical and experimental results to derive some recommendations for choosing the parameters β, γ, and R of our algorithm. We note that having three parameters for a simple $(1 + 1)$-type optimizer might look frightening at first, but a closer look reveals that setting these parameters is actually not too critical.

For the power-law exponent β, as in [11], there is little indication that the precise value is important. The value $\beta = 1.5$ suggested in [11] gives good results even though in our experiments, $\beta = 2$ gave slightly better results. We do not have an explanation for this, but in the light of the small differences we do not think that a bigger effort to optimize β is justified.

Different from the previous approaches building on stagnation detection, our algorithm also does not need specific values for the parameter R, which governs the maximum phase length $\ell_r = \frac{1}{1-\gamma}\binom{n}{r}\ln(R)$ and in particular leads to the property that a single improving solution in distance m is found in phase m with

probability $1 - \frac{1}{R}$ (as follows from the proof of Lemma 2). Since we have the heavy-tailed mutations available, it is less critical if an improvement in distance m is missed in phase m. At the same time, since our heavy-tailed mutations also allow to flip more than r bits in phase r, longer phases obtained by taking a larger value of R usually do not have a negative effect on the runtime. For these reasons, the times computed in Theorem 5 depend very little on R. Since the phase length depends only logarithmically on R, we feel that it is safe to choose R as some mildly large constant, say $R = 25$.

The most interesting choice is the value for γ, which sets the balance between the SD-RLS mode of the algorithm and the heavy-tailed mutations. A large rate $1 - \gamma$ of SD-RLS iterations is good to find a single improvement, but can lead to drastic performance losses when there are more improving solutions. Such trade-offs are often to be made in evolutionary computation. For example, the simple RLS heuristic using only 1-bit flips is very efficient on unimodal problems (e.g., has a runtime of $(1 + o(1))n \ln n$ on OneMax), but fails on multimodal problems. In contrast, the $(1 + 1)$ EA flips a single bit only with probability approximately $\frac{1}{e}$, and thus optimizes OneMax only in time $(1 + o(1))en \ln n$, but can deal with local optima. In a similar vein, a larger value for γ in our algorithm gives some robustness to situations where in phase r other mutations than r-bit flips are profitable – at the price of a slowdown on problems like classic jump functions, where a single improving solution has to be found. It has to be left to the algorithm user to set this trade-off suitably. Taking the example of RLS and the $(1 + 1)$ EA as example, we would generally recommend a constant factor performance loss to buy robustness, that is, a constant value of γ like, e.g., $\gamma = 0.25$.

8 Conclusion

In this work, we proposed a way to combine stagnation detection with heavy-tailed mutation. Our theoretical and experimental results indicate that our new algorithm inherits the good properties of the previous stagnation detection approaches, but is superior in the following respects.

- The additional use of heavy-tailed mutation greatly speeds up leaving a local optimum if there is more than one improving solution in a certain distance m. This is because to leave the local optimum, it is not necessary anymore to complete phase $m - 1$.
- Compared to the robust SD-RLS, which is the fairest point of comparison, our algorithm is significantly simpler, as it avoids the two nested loops (implemented via the parameters r and s in [21]) that organize the reversion to smaller rates. Compared to the SD-$(1 + 1)$ EA, our approach can obtain the better runtimes of the SD-RLS approaches in the case that few improving solutions are available, and compared to the simple SD-RLS of [21], our approach surely converges.
- Again comparing our approach to the robust SD-RLS, our approach gives runtimes with exponential tails. Let m be constant. If the robust SD-RLS

misses an improvement in distance m in the m-th phase and thus in time $O(n^m)$ – which happens with probability $n^{-\Theta(1)}$ for typical parameter settings –, then strength m is used again only after the $(m+1)$-st phase, that is, after $\Omega(n^{m+1})$ iterations. If our algorithm misses such an improvement in phase m, then in each of the subsequent $\ell_{m+1} = \Omega(n^{m+1})$ iterations, it still has a chance of $\Omega(n^{-m}\gamma)$ to find this particular improvement. Hence the probability that finding this improvement takes $\Omega(n^{m+1})$ time, is only $(1-\Omega(n^{-m}\gamma))^{\Omega(n^{m+1})} \leq \exp(-\Omega(n\gamma))$.

As discussed in Section 7, the three parameters of our approach are not too critical to set. For these reasons, we believe that our combination of stagnation detection and heavy-tailed mutation is a very promising approach.

As the previous works on stagnation detection, we have only analyzed stagnation detection in the context of a simple hillclimber. This has the advantage that it is clear that the effects revealed in our analysis are truly caused by our stagnation detection approach. Given that there is now quite some work studying stagnation detection in isolation, for future work it would be interesting to see how well stagnation detection (ideally in the combination with heavy-tailed mutation as proposed in this work) can be integrated into more complex evolutionary algorithms.

Acknowledgement

This work was supported by a public grant as part of the Investissements d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH and a research grant by the Danish Council for Independent Research (DFF-FNU 8021-00260B) as well as a travel grant from the Otto Mønsted foundation.

References

1. Antipov, D., Buzdalov, M., Doerr, B.: Fast mutation in crossover-based algorithms. In: Genetic and Evolutionary Computation Conference, GECCO 2020. pp. 1268–1276. ACM (2020)
2. Antipov, D., Buzdalov, M., Doerr, B.: First steps towards a runtime analysis when starting with a good solution. In: Parallel Problem Solving From Nature, PPSN 2020, Part II. pp. 560–573. Springer (2020)
3. 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)
4. 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)
5. Bambury, H., Bultel, A., Doerr, B.: Generalized jump functions. In: Genetic and Evolutionary Computation Conference, GECCO 2021. pp. 1124–1132. ACM (2021)
6. 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)

7. Corus, D., Oliveto, P.S., Yazdani, D.: Fast immune system-inspired hypermutation operators for combinatorial optimization. IEEE Transactions on Evolutionary Computation **25**, 956–970 (2021)
8. Dang, D., 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 Transactions on Evolutionary Computation **22**, 484–497 (2018)
9. Doerr, B.: Does comma selection help to cope with local optima? In: Genetic and Evolutionary Computation Conference, GECCO 2020. pp. 1304–1313. ACM (2020)
10. Doerr, B., Doerr, C.: Theory of parameter control for discrete black-box optimization: provable performance gains through dynamic parameter choices. In: Doerr, B., Neumann, F. (eds.) Theory of Evolutionary Computation: Recent Developments in Discrete Optimization, pp. 271–321. Springer (2020), also available at https://arxiv.org/abs/1804.05650
11. 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)
12. Doerr, B., Rajabi, A.: Stagnation detection meets fast mutation. CoRR **abs/2201.12158** (2022), https://arxiv.org/abs/2201.12158
13. 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)
14. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science **276**, 51–81 (2002)
15. Friedrich, T., Göbel, A., Quinzan, F., Wagner, M.: Evolutionary algorithms and submodular functions: Benefits of heavy-tailed mutations. CoRR **abs/1805.10902** (2018)
16. Friedrich, T., Göbel, A., Quinzan, F., Wagner, M.: Heavy-tailed mutation operators in single-objective combinatorial optimization. In: Parallel Problem Solving from Nature, PPSN 2018, Part I. pp. 134–145. Springer (2018)
17. 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)
18. Prügel-Bennett, A.: When a genetic algorithm outperforms hill-climbing. Theoretical Computer Science **320**, 135–153 (2004)
19. Rajabi, A., Witt, C.: Self-adjusting evolutionary algorithms for multimodal optimization. In: Genetic and Evolutionary Computation Conference, GECCO 2020. pp. 1314–1322. ACM (2020)
20. Rajabi, A., Witt, C.: Stagnation detection in highly multimodal fitness landscapes. In: Genetic and Evolutionary Computation Conference, GECCO 2021. pp. 1178–1186. ACM (2021)
21. Rajabi, A., Witt, C.: Stagnation detection with randomized local search. In: Evolutionary Computation in Combinatorial Optimization, EvoCOP 2021. pp. 152–168. Springer (2021)
22. Wegener, I.: Theoretical aspects of evolutionary algorithms. In: Automata, Languages and Programming, ICALP 2001. pp. 64–78. Springer (2001)
23. 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)
24. 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)