Single-trajectory Search Heuristics on Discrete Multimodal Optimization Problems

Rajabi, Amirhossein

Publication date:
2022

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

Single-trajectory Search Heuristics on Discrete Multimodal Optimization Problems

Amirhossein Rajabi

Kongens Lyngby 2022

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

Nature-inspired optimization algorithms are defined as a class of algorithms that are inspired by natural phenomena, e. g., evolutionary algorithms and simulated annealing. These methods have gained popularity in practice since they are simple to use and quickly provide good results. Single-trajectory search heuristics are nature-inspired optimization algorithms that iteratively develop a trajectory of solutions to a problem. They have a straightforward structure which can be seen in many powerful nature-inspired algorithms, such as randomized local search, the $(1+1)$-evolutionary algorithm, and simulated annealing. They typically have some parameters (e. g., mutation rate) to be set and need a selection strategy for developing the sequence of solutions. Runtime analysis of nature-inspired algorithms is a line of research that offers suggestions for parameter tuning in nature-inspired algorithms. Also, several studies have been conducted to determine how to pick the selection mechanisms in such algorithms.

For a single-trajectory search heuristic, getting out of a local optimum, where all nearby solutions are of lower quality, is difficult, and its mutation and selection mechanism significantly impact the escaping time. This thesis discusses three main strategies used in the literature to overcome local optima in single-trajectory search heuristics. (1) **Global mutations** with a proper probability distribution over solutions can always find a strict improvement with positive probability and eventually leave a local optimum. (2) *Stagnation detection* approaches, as **self-adjusting mechanisms**, gradually increase the mutation rate when getting stuck in a local optimum. (3) Accepting inferior solutions in **non-elitist algorithms** has been used in practice to leave local optima. In this context, we study two well-known non-elitist algorithms, the Metropolis algorithm and simulated annealing, in specific optimization scenarios.

Summary (Danish)

Naturinspirerede optimeringsalgoritmer er defineret som en klasse af algoritmer der er inspireret af naturlige fænomener, for eksempel evolutionære algoritmer og simuleret udglødning. Disse metoder er blevet populære i praksis da de er nemme at bruge og hurtigt finder frem til løsninger af god kvalitet. Enkelttrajektorie-søgeheuristikker er naturinspirerede algoritmer der iterativt udvikler en trajektorie af løsninger til et problem. Disse algoritmer har en simpel struktur der kan findes i mange stærke naturinspirerede algoritmer, f. eks. randomiseret lokal søgning, (1+1)-evolutionær algoritme og simuleret udglødning. Algoritmerne har almindeligvis nogle parametre (f. eks. mutationsraten) der skal indstilles, og har brug for en strategi der udvikler sekvensen af løsninger. Køretidsanalysen af naturinspirerede algoritmer er en forskningsretning der giver anbefalinger til parametervalget i naturinspirerede algoritmer. Derudover der er flere studier der fastslår hvordan man vælger udvælgelsesmekanismerne i sådanne algoritmer.

Det kan være svært for en enkelttrajektorie-søgeheuristik at undvige et lokalt optimum hvor alle tilstødende løsninger er af ringere kvalitet, og heuristikkens mutations- og udvælgelsesmekanismer påvirker undvigelsestiden væsentligt. Denne afhandling diskuterer tre hovedstrategier for at undvige lokale optima i enkelttrajektorie-søgeheuristikker: (1) **Globale mutationer** med en passende sandsynlighedsfordeling over løsningerne har altid en positiv sandsynlighed for en ægte forbedring og kan før eller siden undvige et lokalt optimum. (2) **Stagneringsdetektion** som f. eks. i selvjusterende algoritmer øger mutationsraten gradvist når algoritmer sidder fast i et lokalt optimum. (3) Accept af løsninger af dårligere kvalitet i **ikke-elitære** algoritmer er blevet brugt i praksis til at undvige lokale optima. I denne sammenhæng undersøger vi to velkendte ikke-elitære algoritmer, Metropolis-algoritmen og simuleret udglødning, i specifikke optimeringsscenarier.

Preface

This dissertation was prepared at the Department of Applied Mathematics and Computer Science (DTU Compute) in the section for Algorithms, Logic, and Graphs (AlgoLoG) to meet the criteria for acquiring a Ph.D. in Computer Science.

In this thesis, I propose and analyze several strategies for single-trajectory search heuristics for leaving local optima in discrete optimization problems. The thesis consists of two introductory chapters and six papers.

The work has been supervised by Professor Carsten Witt and co-supervised by Associate Professor Paul Fischer.

Kongens Lyngby, 31-May-2022
Amirhossein Rajabi

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. Carsten Witt, whose sincerity and encouragement I will never forget. Prof. Carsten Witt has been an inspiration as I hurdled through the path of this PhD degree. He also gave me the opportunity and encouragement to strengthen my profile as a future high-skilled researcher. I am deeply thankful for his constant, broad, professional, and constructive feedback, which helped me fill the gaps in my training.

I want to thank Prof. Benjamin Doerr for his hospitality and advice during my external stay at École Polytechnique. Also, my keen appreciation goes to Prof. Tobias Friedrich, Dr Timo Kötzing, and Dr Martin S. Krejca for their support during the research stay at Hasso Plattner Institute (HPI), Potsdam University. Furthermore, I would like to thank the AlgoLoG group for the pleasant and friendly working atmosphere.

Last but not least, I would like to thank my loved ones. Nasim, my wife, has been highly supportive and has made numerous sacrifices to help me during this journey. I take the opportunity to say, "I love you". Also, my parents, Behrooz and Maryam, as well as my sister, Kiana, deserve special thanks for supporting me throughout my journey so far. Finally, thanks to my friends and family for being by my side in the up and downs of the journey.

Contents

Chapter 1

Introduction

1.1 Motivation

Natural-inspired computation is a framework for problem-solving methodologies and approaches inspired by natural processes. *Evolutionary algorithms*, which follow Darwin's survival of the fittest principle, *ant colony optimization*, which models ant foraging behavior, and *simulated annealing*, which is inspired by annealing in metallurgy, are all well-known examples of natural-inspired algorithms. These approaches are popular in a variety of industries, including engineering [DM13], healthcare [FC03], and finance [Che02], since they are easy to implement and produce surprise outcomes. In this thesis, we are interested in studying *single-trajectory search heuristics*, a frequent framework in nature-inspired optimization problems, which evolve iteratively a trajectory of solutions to a problem. From now on, we call each solution of the trajectory a search point. The first search point, called *initial search point*, is often chosen randomly from all possible solutions, and each future solution is chosen from previous search points or a modified version of them. A rigorous definition of single-trajectory search heuristics is provided in Section 1.6. This structure for heuristics can be seen in many well-known nature-inspired algorithms discussed in this thesis, such as randomized local search, the $(1 + 1)$-evolutionary algorithm (EA), and simulated annealing.

Runtime analysis of nature-inspired algorithms is a line of research that tries to characterize how fast nature-inspired algorithms solve specific types of problems. This has led to many rigorously proven statements on the speed, so-called the runtime [AD11, Jan13, NW10, DN20]. Nature-inspired algorithms typically come with different parameters that need to be set, e. g., the so-called *mutation rate* (the degree to which existing solutions are changed while developing new ones). Runtime analysis can provide a solution to the often very time-consuming work of parameter tuning in nature-inspired algorithms, which typically come with a variety of parameters that need to be set [DD20]. Also, it has been a subject of ongoing research on how to select promising solutions in nature-inspired algorithms. There are the following two general categories for selection mechanisms: Some algorithms, so-called *elitist* algorithms, keep the best-so-far search point (usually the last one if there are more than one), e. g., the (1+1) EA, while there are *non-elitist* algorithms that also accept inferior solutions sometimes, e. g., simulated annealing.

Leaving *local optima*, where there is no better search point by making a small number of changes to the current solution, is challenging for single-trajectory search heuristics. They need to modify the current search point more than what is usually beneficial for *hill-climbing*, where only a single change in the solution can result in a better solution. In this situation, the choice of the mutation and selection mechanisms is crucial for leaving local optima in an efficient time. We discuss three main approaches for single-trajectory search heuristics to overcome local optima in Chapter 2: global mutations, self-adjusting mechanisms, and non-elitist selections.

In *global mutations*, new search points might be any possible solution, so there is always a positive probability of making a strict improvement if the current search point does not have the optimal fitness value. However, the probability distribution over search points is crucial and has a significant impact on the total optimization time. The *bit-wise mutations* in the (1+1) EA and *fast mutations* in the (1+1) FEA$_\beta$ [DLMN17] are examples of global mutations. However, local mutations can only create a fixed set of offspring points. The *1-bit flip mutation* that often can be found in the Randomized Local Search algorithm (RLS), for example, only reaches a limited number of search points, which may result in being stuck in a local optimum when using the elitist selection. In Section 2.1, we discuss global and local mutations more precisely.

The second approach is adapting the mutation rate by *self-adjusting mechanisms* during the run. Self-adjusting algorithms can learn acceptable or even near-optimal parameters on the fly. There are several self-adjusting mechanisms for unimodal problems [DD18, DGWY19, RW20a, HFS21, DDL21, KLLZ22], while there are a few research works on multimodal problems [LS11, DL16b] (see Section 2.2). On multimodal problems, we study the recently proposed

stagnation detection mechanisms, which gradually increase the mutation rate until leaving a local optimum [RW22a, RW21b, RW21a, DR22]. In Section 2.2, more detailed discussions and concrete runtime results can be found.

The last but not least strategy we study is the non-elitist approach in which an algorithm occasionally accepts inferior solutions. Recently, there has been an increasing amount of literature studying the performance of non-elitist algorithms to escape local optima [Doe20a, DEL21a, DEL21b, FS21]. In the thesis, we mostly investigate two well-known algorithms in practice, *Metropolis algorithm* [MRR$^+$53] and *simulated annealing* [KGJV83]. So far, runtime analyses of these algorithms have mostly appeared as side results in papers focused on other heuristics. In Section 2.3, both algorithms are discussed and analyzed.

1.2 Thesis Overview and Contributions

This thesis is structured as follows: this chapter continues with discussing more studies from the literature in Section 1.3, defining the basic definitions and mathematical tools used in the thesis in Section 1.4 and determining the contexts of optimization problems and heuristics of the thesis in Section 1.5 and 1.6, respectively. Chapter 2 overviews the strategies that can be used for leaving local optima. Afterward, we see the following 6 papers[1] rigorously analyzing different strategies in Chapters 3–8.

- The published journal paper [RW22a] in Chapter 3 as **Paper A: Self-Adjusting Evolutionary Algorithms for Multimodal Optimization**.
- The accepted (with minor revisions) journal paper [RW22c] in Chapter 4 as **Paper B: Stagnation Detection with Randomized Local Search**.
- The submitted journal paper [RW22b] in Chapter 5 as **Paper C: Stagnation Detection in Highly Multimodal Fitness Landscapes**.
- The extended version of [DR22] in Chapter 6 as **Paper D: Stagnation Detection Meets Fast Mutation**.
- The in-preparation paper [DHRW22] in Chapter 7 as **Paper E: How Fast Does the Metropolis Algorithm Leave Local Optima?**

[1]The versions of the papers that appear in this thesis may differ slightly due to typo corrections and minor alterations.

- The extended version of [DRW22] in Chapter 8 as **Paper F: Simulated Annealing is a Polynomial-Time Approximation Scheme for the Minimum Spanning Tree Problem**.

I have published, submitted, or prepared the following articles during my Ph.D. studies. The papers that are used in the thesis are marked by *.

Conference papers:

[RW20b] Amirhossein Rajabi and Carsten Witt. Self-adjusting evolutionary algorithms for multimodal optimization. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1314–1322. ACM, 2020

[RW20a] Amirhossein Rajabi and Carsten Witt. Evolutionary algorithms with self-adjusting asymmetric mutation. In *Parallel Problem Solving from Nature, PPSN 2020, Part I*, pages 664–677. Springer, 2020

[RW21b] Amirhossein Rajabi and Carsten Witt. Stagnation detection with randomized local search. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP 2021*, pages 152–168. Springer, 2021

[RW21a] Amirhossein Rajabi and Carsten Witt. Stagnation detection in highly multimodal fitness landscapes. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 1178–1186. ACM, 2021

[DR22] *Benjamin Doerr and Amirhossein Rajabi. Stagnation detection meets fast mutation. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pages 191–207. Springer, 2022

[DRW22] *Benjamin Doerr, Amirhossein Rajabi, and Carsten Witt. Simulated annealing is a polynomial-time approximation scheme for the minimum spanning tree problem. In *Genetic and Evolutionary Computation Conference, GECCO 2022*. ACM, 2022. To appear

[DHRW22] *Benjamin Doerr, Taha El Ghazi El Houssaini, Amirhossein Rajabi, and Carsten Witt. How fast does the metropolis algorithm leave local optima? To be submitted to a conference, 2022

[FKKR22] Tobias Friedrich, Timo Kötzing, Martin S. Krejca, and Amirhossein Rajabi. Escaping local optima with local search: A theory-driven discussion. In *Parallel Problem Solving From Nature, PPSN 2022*. Springer, 2022. submitted to Parallel Problem Solving from Nature, PPSN 2022

Journal papers:

[RW22a] *Amirhossein Rajabi and Carsten Witt. Self-adjusting evolutionary algorithms for multimodal optimization. *Algorithmica*, pages 1–30, 2022. Preliminary version in GECCO 2020

[RW22c] *Amirhossein Rajabi and Carsten Witt. Stagnation detection with randomized local search. *Evolutionary Computation*, 2022. Preliminary version in EvoCOP 2021, to appear in the EC journal

[RW22b] *Amirhossein Rajabi and Carsten Witt. Stagnation detection in highly multimodal fitness landscapes. *Algorithmica*, 2022. Preliminary version in GECCO 2021, submitted to the Algorithmica journal

1.3 Other approaches

Other than single-trajectory variants, search heuristics with a more complex structure have been used for multimodal optimization. For example, population-based algorithms maintain more than one solution in contrast to single-trajectory search heuristics. The idea is to search different areas of the search space at the same time, avoiding small local optima. An elaborate line of research highlights various aspects of these algorithms [DL16a, LY12, ADY19]. In recent years, there has been an increasing amount of literature studying the performance of non-elitist algorithms to escape local optima [Doe20a, DEL21a, DEL21b, FS21].

When using population-based methods, it is natural to attempt to use variation operators known as *crossover* to integrate advantageous characteristics from different local optima (creating a new solution using at least two solutions from the population). Given good diversity mechanisms, a crossover can be capable of overcoming large fitness valleys [DFK$^+$18, DFK$^+$16a, DFK$^+$16b]. However, also employing crossover without an explicit diversity mechanism speeds up the run time for certain problems [AN21, ABD21, AD20, ADK20, FKK$^+$16b].

Another fundamentally different approaches are *estimation-of-distribution algorithms* (EDAs), which maintain a probabilistic model of the search space instead of sets of solutions. In this approach, the model is constantly updating based on samples from the search space, leading to creating better solutions in the next iteration. In this method, choosing the correct step size is crucial [FKK16a, LSW21]. Given a proper step size, EDAs overcome valleys of a considerable size at almost no extra cost [HS18, Doe19, Wit21]. Also, a number

of authors have investigated the efficiency of another EDA called *UMDA* on escaping local optima on a deceptive fitness function [LN19, DK20].

Moreover, *ant colony optimization* (ACO) algorithms are nature-inspired search heuristics that evolve a probability distribution on the search space. A rigorous run time analysis of a version of ACO so-called 2-MMAS$_{ib}$ on multimodal functions has been conducted in [BBD21b]. *Artificial immune systems* (AISs) are another class of nature-inspired search heuristics that have also been proven to escape from local optima efficiently [COY18, COY20, COY21a, COY21b]. Also, it should be mentioned that the performance of the *Move-Acceptance hyper-heuristic* (MAHH) was rigorously analyzed for multimodal optimisation problems [LOW19].

1.4 Preliminaries

We let $\mathbb{N}$ denote the set of all natural numbers including 0 and $\mathbb{R}$ denote the set of all real numbers. For all $a, b \in \mathbb{N}$, let $[a..b]$ denote the set of natural numbers from at least a to at most b. For each $x \in \{0,1\}^n$, let $\|x\|_1$ denote the number of one-bits in x, and analogously, let $\|x\|_0$ denote its number of zero-bits. Further, for each $i \in [0..n]$, let x_i denote the bit at position i in x. We say that we flip bit i when we refer to the value $1 - x_i$.

We call a pseudo-Boolean function f a *fitness function*, and we refer to bit-strings as *search points*. Assume we want to optimize a pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$ for $n \in \mathbb{N}$. All asymptotic expressions are with respect to n. We call $f(x)$ the *fitness* of the bit-string x. For $x, y \in \{0,1\}^n$, we call $\mathrm{d_H}(x,y) \coloneqq |\{i \in [n] \mid x_i \neq y_i\}|$ the *Hamming distance* of x and y.

Moreover, for all $i \in [0..n]$, we call the set of all search points with Hamming distance i to x the *i-neighborhood of x*. By search points in local neighborhood or neighbors, we mean the 1-neighborhood search points.

We discuss two probability distributions used in this thesis. A random variable X follows a *binomial distribution* with parameters n and p if for all $k \in [0..n]$, we have

$$\Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k},$$

and we write $X \sim \mathrm{Bin}(n,p)$.

In addition, an integer random variable X follows a power-law distribution with parameters β and u if

$$\Pr(X = k) = \begin{cases} C_{\beta,u} k^{-\beta} & \text{if } 1 \leq k \leq u, \\ 0 & \text{otherwise,} \end{cases} \tag{1.1}$$

where $C_{\beta,u} \coloneqq (\sum_{j=1}^{u} j^{-\beta})^{-1}$ is the normalization coefficient, and $C_{\beta,u} = \Theta(1)$ for $\beta > 1$ (we refer to [DLMN17] for more details). The function $\mathrm{pow}(\beta, u)$ returns a sample from this distribution.

1.5 Discrete Optimization Problems

Theoretical research on nature-inspired algorithms often studies simple well-structured functions, which can serve as building blocks of more complicated problems. In this section, we define some of them, which are used in the thesis. In all algorithms and results presented in this thesis, we aim at maximizing the fitness function f except when the minimum spanning tree problem is considered as the optimization problem. Nevertheless, on any minimization problem, we can simply assume that the negated fitness function is maximized.

We use the definition of *unimodal functions* determined in [DJW02]. A search point is a local maximum if no neighbor has a larger fitness value. A fitness function is unimodal if and only if there is only one local maximum. Analogously, a fitness function is *multimodal* if and only if there are more than one local maximum in the function. To study the performance of heuristics on multimodal problems, we should first analyze the behavior of algorithms on unimodal functions as they can usually be seen in the underlying landscapes of multimodal functions. In the next section, we present some well-studied fitness functions in the literature.

1.5.1 Fitness Functions

In this part, we define and explain some of the benchmark functions used in the thesis.

Unimodal functions. The following unimodal benchmark functions OneMax and LeadingOnes have been extensively studied in the literature.

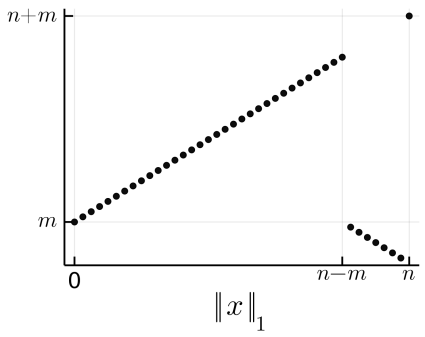


Figure 1.1: $\textsc{Jump}_m$

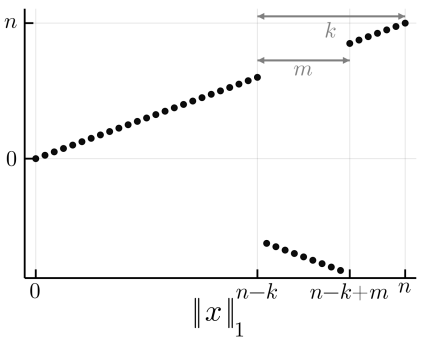


Figure 1.2: $\textsc{Jump}_{k,m}$

They are defined by

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

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 x.

Jump functions families. The well-known $\textsc{Jump}_m$ function [DJW02] with jump size m defined as follows:

$$\textsc{Jump}_m(x) \coloneqq \begin{cases} m + \|x\|_1 & \text{if } \|x\|_1 \leq n - m \text{ or } \|x\|_1 = n, \\ n - \|x\|_1 & \text{otherwise.} \end{cases}$$

This function is multimodal with local optima at all search points with $n - m$ one-bits, and there is only one global optimum 1^n. See also Figure 1.1 for a depiction.

Two papers [BBD21a] and [RW21a] independently defined variants of jump functions in which the place of the jump with size m starts at an earlier point than the Hamming distance m from the global optimum. According to the generalized version of Jump functions illustrated in Figure 1.2 and defined in [BBD21a], formally, for all $x \in \{0,1\}^n$, we have

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

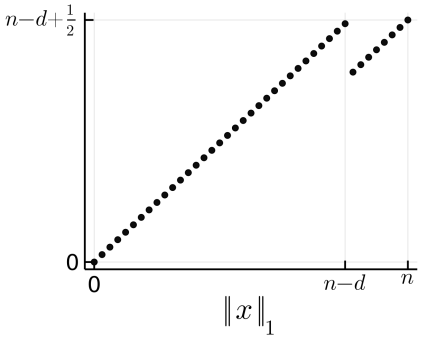


Figure 1.3: $\textsc{Cliff}_d$

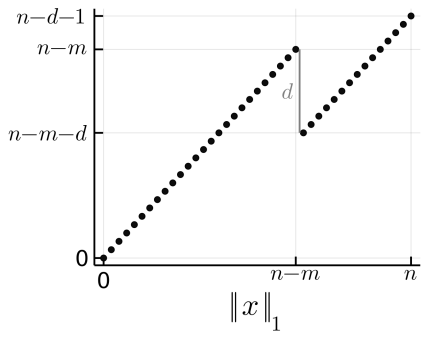


Figure 1.4: $\textsc{Cliff}_{d,m}$

In other words, after the jump, there is a unimodal sub-problem of length $k-m$. The classical Jump function can be considered as a special case of $\textsc{Jump}_{k,m}$ with $k = m$, i. e., $\textsc{Jump}_m = \textsc{Jump}_{m,m}$ by ignoring the differences in the fitness values in the valley. The function $\textsc{Jump}_{k,m}$ is called JumpOff in [RW21a] and JumpOffset in [Wit21] using a fixed $k = n/4$.

Cliff functions. Cliff functions were originally defined with one parameter d determining the distance of the local optimum from the global optimum ([JS07]) (see Figure 1.3).

$$\textsc{Cliff}_d(x) \coloneqq \begin{cases} \|x\|_1 & \text{if } \|x\|_1 \leq n-d, \\ \|x\|_1 - d + 1/2 & \text{otherwise.} \end{cases}$$

We also define this function with two parameters d and m. The function $\textsc{Cliff}_{d,m}$ is increasing as the number of one-bits of the argument increases except for the points with $n-m$ one-bits, where the fitness decreases sharply by d if we add one more one-bit to the search point.

$$\textsc{Cliff}_{d,m}(x) \coloneqq \begin{cases} \|x\|_1 & \text{if } \|x\|_1 \leq n-m, \\ \|x\|_1 - d - 1 & \text{otherwise.} \end{cases}$$

The valley in Jump includes false fitness signals that lead the search back to the local optimum, while the fitness signal in Cliff leads to the global optimum, see Figure 1.4.

The minimum spanning tree (MST) problem. We now define the MST problem and additional notation used later in the thesis. Assume that an undi-

rected, connected, weighted graph $G = (V, E)$ is given, and let n and m denote the number of graph vertices and edges, respectively. Also, let the set of edges be $E = \{e_1, \ldots, e_m\}$. The weight of edge e_i, where $i \in \{1, \ldots, m\}$, is a positive number w_i. We say that the weights $w_1, \ldots, w_m$ are $(1 + \varepsilon)$-*separated* if $w_j \geq (1 + \varepsilon)w_i$ for all $i, j \in \{1, \ldots, n\}$ such that $w_j > w_i$. We define $w_{\min} \coloneqq \min\{w_i \mid i \in \{1, \ldots, m\}\}$ and $w_{\max} \coloneqq \max\{w_i \mid i \in \{1, \ldots, m\}\}$ for the minimum and maximum edge weight. The aim of the problem is to find a subset of edges called E' such that (V, E') is a spanning tree of the graph G having minimal total weight $w(E') = \sum_{e_i \in E'} w_i$. We use the bit-string representation for sets E' of edges, that is, a bit-string $x = (x_1, \ldots, x_m) \in \{0, 1\}^m$ represents the set $E(x) = \{e_i \mid x_i = 1\}$. As objective function, we use the sum of the weights of the selected edges when these form a connected graph on V and ∞ (or an extremely large value) otherwise.

$$f(x) \coloneqq \begin{cases} w_1 x_1 + \cdots + w_m x_m & \text{if } (V, E(x)) \text{ is connected,} \\ \infty & \text{otherwise.} \end{cases}$$

1.6 Single-trajectory Search Heuristics

All algorithms addressed in this thesis follow the framework of *single-trajectory heuristics*, optimizing a fitness function f (Algorithm 1). A single-trajectory heuristic processes a trajectory $(x^{(t)})_{t \in \mathbb{N}}$ of search points (the *current* search point). The initial search point $(x^{(0)})$ is chosen uniformly at random from the search space $\{0, 1\}^n$. In all iterations $t \in \mathbb{N}$, the search point $x^{(t+1)}$ is made via the subroutines: MUTATE and SELECT. We allow mutation and selection to take into account additional information, such as the number of iterations since the last improvement was found.

Algorithm 1: The framework for single-trajectory heuristics, requiring the potentially parameterized subroutines MUTATE and SELECT as well as a fitness function f

$x^{(0)} \leftarrow$ individual drawn uniformly at random from $\{0, 1\}^n$;
for $t \leftarrow 0, 1, 2, \ldots$ **do**
  $y \leftarrow \text{MUTATE}(x^{(t)})$;
  $x^{(t+1)} \leftarrow \text{SELECT}_f(x^{(t)}, y)$;

The subroutine MUTATE: $\{0, 1\}^n \to \{0, 1\}^n$ is a randomized function that returns a modified version of the input. In iteration t, this function gets $x^{(t)}$ as input (the *parent*) and returns $x^{(t)}$, denoted by y (the *offspring*). We call this process *mutation*, and we say that $x^{(t)}$ is *mutated*. In this thesis, we assume

that all mutations are unary unbiased black-box variation operators as rigorously defined in [LW12], which means that the mutation cannot discriminate between bit positions nor between bit values.

After mutation, utilizing f, the subroutine SELECT: $(\{0,1\}^n)^2 \to \{0,1\}^n$ selects either $x^{(t)}$ or y to be assigned to $x^{(t+1)}$, which is a starting point for the next iteration. We refer to this process as *selection*. An *elitist* selection always selects the search points with higher fitness values. In the case that the fitness values are equal, i. e., $f(x^{(t)}) = f(y)$, the algorithm usually selects the offspring y although not always. In elitist algorithms, the current search point is always the search point with the highest-so-far fitness value. On the other side, *non-elitist* selection might accept inferior search points.

The *runtime* or the *optimization time* of a heuristic on a function f is the first point in time where a search point of the global optima has been evaluated. Usually, a black-box perspective is assumed, and time is measured in the number of evaluations of the objective function. Since the heuristics considered in this thesis evaluate one search point in each iteration of their main loop, their runtime equals the number of iterations until an optimum is found plus 1 for the evaluation of the initial search point. Hence, their runtime is simply the smallest value of the iteration counter t such that an optimum is evaluated plus 1. The expected runtime, i. e., the expected value of this time, is often analyzed.

Chapter 2

Strategies for Escaping local optima

This chapter investigates three strategies for leaving a local optimum: global mutations, self-adjusting mechanisms, and non-elitist strategies.

2.1 Mutations

We first study an example of local mutations, called k-bit flip mutation, in Section 2.1.1. Afterward, we investigate two global mutation classes of bit-wise mutations and heavy-tailed mutations in Section 2.1.2 and 2.1.3, respectively.

2.1.1 k-bit flip mutations

The first mutations we address are *k-bit flip* mutations, which are called λ-changes in [JPY88]. A k-bit flip mutation with a fixed integer k flips exactly k out of n bits uniformly at random, which means that the offspring differs from its parent in k places. These mutations are also classified as *local mutations* due to the fact that the set of possible offspring is constrained to a proper subset of the search space. In other words, unlike global mutations, which will be

Algorithm 2: RLS for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$

Select $x^{(0)}$ uniformly at random from $\{0,1\}^n$;
for $t \leftarrow 0, 1, 2, \ldots$ **do**
  Create y by flipping a bit chosen uniformly at random in a copy of $x^{(t)}$;
  if $f(y) \geq f(x^{(t)})$ **then**
    $x^{(t+1)} \leftarrow y$;
  else
    $x^{(t+1)} \leftarrow x^{(t)}$;

discussed later, the mutation is not able to generate all search points in the search space in one iteration.

Algorithm 1 using 1-bit flip as its mutation is called RLS in the literature (displayed in Algorithm 2) and has been studied extensively over the years [MHF93, WW05, NW07, JZ14, DD16]. This algorithm can outperform other unbiased black-box search heuristics on unimodal problems. This is because other algorithms typically spend some time to find the next improvement in larger Hamming distances, which is often not beneficial. However, RLS achieves this superiority by assuming some information about the problem. In other words, we assume that there is always an improvement in the 1-neighborhood. Basically, if we have some specific knowledge about the problem's landscape and can find a suitable value for k, we might achieve an efficient algorithm on that problem.

Taking the minimum spanning tree problem as an example, using some 1-bit flips, we can create a spanning tree from a random solution. Afterward, 1-bit flips are not advantageous anymore, and 2-bit flips can make progress by adding one edge and removing another edge from the solution to keep connectivity in the solution (see [NW10]). The authors in [NW07] introduced a variant of RLS (called $\text{RLS}^{1,2}$ by [NW10] to avoid confusion), which employs both 1-bit and 2-bit flips with equal probability and proved that a minimum spanning tree is found in $O(m^2 \ln(nw_{\max}))$ expected steps. Since this algorithm flips at most 2 bits each time, by replacing all edge weights with their rank in their increasingly sorted sequence, the algorithm can still achieve the same MSTs as in the original graph [RKJ06]. Therefore, because $w_{\max} \leq m$ and $m \leq n^2$, it can be shown that $\text{RLS}^{1,2}$ solves the MST problem in expected time $O(m^2 \ln n)$ (see [RW21b, Section 7] for more detail).

Generally, we are not always fortunate enough to know the proper value for k in k-bit flip mutations. Also, the optimal parameter choice for k might change during the search. For the latter case, we give the above-defined Jump functions

Algorithm 3: The (1+1) EA with mutation probability p for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$

Select $x^{(0)}$ uniformly at random from $\{0,1\}^n$;
for $t \leftarrow 0, 1, 2, \ldots$ **do**
  Create y by flipping each bit in a copy of $x^{(t)}$ independently with probability p;
  if $f(y) \geq f(x^{(t)})$ **then**
    $x^{(t+1)} \leftarrow y$;
  else
    $x^{(t+1)} \leftarrow x^{(t)}$;

as an example. In the beginning, 1-bit flips are efficient until the local optimum is reached. From this state, m-bit flips are required to find the next strict improvement. The size of the jump m typically can be considered as an unknown value to the algorithm. However, even if it is known, the algorithm with m-bit flips operator might not reach the local optimum, starting from the initial search point. These arguments hold for most of the local mutations. They motivate us to design and study global mutations, where any search point in the search space can be produced in a single iteration.

2.1.2 Bit-wise mutations

The *bit-wise* mutations with mutation rates $0 < p < 1$ [JW00, Weg02] are well-known global mutations, which have been studied widely [DLMN17, DG13, BDN10, Wit13, OHY09]. The mutation with a given mutation rate p flips each bit of the search point with probability p. The Hamming distance of the offspring from the parent follows a binomial distribution with parameters n and p, i. e., $\mathrm{d_H}(x^{(t)}, y) \sim \mathrm{Bin}(n, p)$, and the expected number of flipping bits is np. Algorithm 1 with a bit-wise mutation is called the *(1+1) EA*, which has a parameter indicating the mutation rate, see Algorithm 3. Also, let *strength* in mutations denote the expected number of flipping bits, i. e., n times the mutation probability, so a bit-wise mutation with strength k flips each bit with probability $\frac{k}{n}$.

Much research has been conducted to find the best mutation rate for the algorithm (1+1) EA on different problems [DD20]. Based on several studies, such as [Müh92, Bäc93], the mutation rate $1/n$, which is so-called the *standard mutation probability*, is known as a good choice. In [DG13], it was shown that the (1+1) EA with mutation rate c/n for a constant $c > 0$ finds the optimum of the

linear objective function OneMax in expected time $\Theta(n \ln n)$. Later, Witt in [Wit13] rigorously proved that $1/n$ is the optimal rate for all linear functions. On LeadingOnes, the authors of [BDN10] showed that although the mutation rate $p \approx 1.59/n$ is the optimum, the rate $1/n$ is enough to find the global optimum in asymptotic expected runtime $\Theta(n^2)$. Similarly to RLS, we, therefore, see that flipping one bit in expectation is promising on unimodal functions.

The standard mutation probability, however, is not the best choice for multimodal problems. Droste et al. [DJW02] showed that on jump functions as theoretical benchmarks for multimodal problems, the (1+1) EA with the standard mutation probability finds the global optimum in expected runtime $\Theta(n^m + n \ln n)$. However, [DLMN17] proves that the optimum choice of which static parameter for the mutation rate can achieve the expected runtime $\Theta((n/m)^m (n/(n-m))^{n-m}$ on jump functions with jump size m. This results in a speed-up by the factor of roughly $(m/e)^m$. The authors also showed that the mutation rate m/n achieves asymptotically the same runtime although using a problem-specific knowledge (the value m in Jump_m) in black-box algorithms is not recommended [Doe20c]. Using the example of jump functions, we can see that the choice of the mutation rate in the (1+1) EA is crucial on multimodal problems.

Nevertheless, the (1+1) EA with the standard mutation probability can still be a powerful search heuristic in some multimodal scenarios. On MST problems, as a multimodal optimization problem with many local optima, the algorithm can find the optimum in at most $(1+o(1))em^2 \ln(emw_{\max})$ expected steps [DJW12]. Even on the generalized version of jump functions, the (1+1) EA can outperform some mechanisms specifically designed for multimodal problems [BBD21a] in some scenarios.

2.1.3 Heavy-tailed mutations

In the previous section, we mentioned that the number of flipping bits in bit-wise mutations follows a binomial distribution, so they highly concentrate on flipping around np bits by Chernoff's bounds. Thus, we rarely observe a mutation flipping much more or much less than np bits in an iteration. This behavior can be detrimental on multimodal optimization problems. For example, assume that there is only one strict improvement at Hamming distance m. The expected waiting time of the (1+1) EA with $p = 1/n$ to find this improvement is $\Theta(n^m)$ while during this time, each search point in the local neighborhood is evaluated $\Theta(n^{m-1})$ times in expectation, that is, it spends numerous iterations to evaluate inferior search points repeatedly.

Algorithm 4: The (1+1) FEA$_\beta$ with parameters β and u for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$

Select $x^{(0)}$ uniformly at random from $\{0,1\}^n$;
for $t \leftarrow 0, 1, 2, \ldots$ **do**
  $\alpha \leftarrow \mathrm{pow}(\beta, u)$;
  Create y by flipping each bit in a copy of $x^{(t)}$ independently with probability α;
  if $f(y) \geq f(x^{(t)})$ **then**
    $x^{(t+1)} \leftarrow y$;
  else
    $x^{(t+1)} \leftarrow x^{(t)}$;

Doerr et al. in [DLMN17] proposed the *heavy-tailed mutation* or *fast mutation* where a power-law distribution with parameters β and u (i. e., D_u^β) determines the number of flipping bits. We define by the *(1+1) FEA$_\beta$* a single-trajectory search heuristic (Algorithm 1) with bit-wise mutation with probability α/n, where α is sampled from heavy-tailed mutation $D_{n/2}^\beta$ in each iteration [DLMN17] (i. e., $\alpha = \mathrm{pow}(\beta, u)$), see Algorithm 4. When $\beta > 1$, the probability of flipping k bits equals $\Theta(k^{-\beta})$. Thus with a constant probability, the algorithm flips a constant number of bits, which results in efficiency in hill-climbing. In addition, the probability that $\alpha = m$ equals $\Theta(m^{-\beta})$, so it is also beneficial for leaving local optima compared to binomial distributions.

Many studies on the (1+1) FEA$_\beta$ have been carried out in recent years, which shows the advantage of this mutation, compared to the standard mutation probability in a single-trajectory search heuristic [QGWF21, FQW18, FGQW18b, COY18].

2.2 Self-adjusting Mechanisms on Escaping Local Optima: Stagnation Detection Mechanisms

To the best of the author's knowledge, only a few runtime analyses of self-adjusting evolutionary algorithms on multimodal functions exist. There are the work by [LS11] on self-adjusting offspring population sizes resulting in a speedup on the Jump functions and the paper by [DL16b] on a self-adjusting mutation rate in a non-elitist EA with respect to the specifically constructed PeakedLO function. However, since these mechanisms are analyzed on population-based

algorithms, they are beyond the scope of this thesis. In this part, we investigate a mechanism called Stagnation Detection (SD), which has recently been introduced to manage the mutation rate in single-trajectory algorithms on multimodal problems based on the number of failures with a certain mutation setting.

Consider a run of the (1+1) EA with standard mutation probability $1/n$ on a pseudo-Boolean fitness function f. If a strict improvement is not found in the first $en \ln n$ iterations, a given search point in the local neighborhood is observed with probability $1 - (1 - \frac{1}{en})^{en \ln n} \geq 1 - 1/n$, so a large portion (almost all for a sufficiently large n) of the search points in the local neighborhood has already been observed at least once. If the algorithm keeps using this mutation rate, it still creates search points in the neighborhood most of the time (i. e., with probability around $1/e$). In this situation, the algorithm has *stagnated* into looking at Hamming distance 1.

We generalize the arguments for larger strengths as follows. A stagnation at distance k is detected if for a given R, at least a $1 - 1/R$ fraction of the search points at Hamming distance k has been observed in expectation. We call the mechanism that discovers such situations *Stagnation Detection (SD)*. Different strategies can be applied following a stagnation (see restart strategies, for example, in [FKKR22]). However, changing the mutation strength (e. g., increasing it by 1) seems to be sensible and beneficial in elitist algorithms. There are different recommendations of how to set the parameter R best, so we refer the interested readers to see them in the corresponding papers; however, in general, the algorithm is rather robust with respect to setting $R = n^c$ for an arbitrary constant $c > 4$ in most of the algorithms employing SD mechanisms. In the conducted analyses, the value of c only affects the lower order terms of the upper bounds on the runtime.

The idea of stagnation detection has also been successfully used in GSEMO, called SD-GSEMO [DZ21], to compute Pareto-optimal solutions in multi-objective evolutionary algorithms. In the following sections, we summarize the results of the studies conducted for using stagnation detection in several single-trajectory search heuristics.

2.2.1 Stagnation Detection with Bit-wise Mutations

In the first version presented in [RW20b, RW22a], we combine the SD mechanism with bit-wise mutations and propose *the (1+1) EA with stagnation detection* or *the SD-(1+1) EA*. In this algorithm, we start with strength 1 (i. e., the standard mutation probability $1/n$). Each time that the algorithm fails to create a strict improvement with strength r within ℓ_r (discussed below) iterations, it

changes the strength to $\min\{r+1, n/2\}$. We note that since a mutation at a bit should not be more likely than a non-mutation, the strengths above $n/2$ in binary search spaces are considered "ill-natured" [ADK19]. Regarding ℓ_r, in the preliminary version [RW20b] of our work, we proposed that ℓ_r should be $2(en/r)^r \ln(nR)$. However, in the extended work [RW22a], we proved that $\ell_r = (n/r)^r (n/(n-r))^{n-r} \ln(enR)$ gives us a more efficient escaping time from local optima (especially, for large gap sizes), so we only mention the results based on the latter set of threshold values in this thesis.

It is proved in [RW22a, Theorem 3] that this algorithm optimizes the jump benchmark functions with jump size $2 \leq m = o(n/\ln n)$ in expected runtime

$$(1 \pm o(1)) \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m}.$$

This bound is significantly better than the upper bound of the (1+1) EA with strength 1 by a factor of roughly $(m/e)^m$ and equal up to lower-order terms to the bound of the (1+1) EA with the best choice of the static parameter, corresponding to a mutation rate of m/n, up to lower-order terms. Also, the SD-(1+1) EA outperforms the (1+1) FEA$_\beta$ with parameter $\beta > 1$ on the jump functions by a factor of $m^{\beta-1/2}$ [DLMN17].

We will describe these results in detail in Chapter 3.

2.2.2 Stagnation Detection with k-bit Flip Mutations

Next, we aim at using k-bit flip mutations with the stagnation detection mechanism. The first straightforward combination of stagnation detection and k-bit flip mutations is to adjust the strength of the mutation by SD. We call the resulting algorithm *RLS with plain stagnation detection* or *SD-RLS^p* [RW21b]. Starting from strength 1, the algorithm using strength s increases the strength to $s+1$ if it fails to find a strict improvement within $\ell_s = \binom{n}{s} \ln R$ iterations. Let m be the Hamming distance at which there are some strict improvements. After the algorithm has increased the strength to m, it has a high probability to make an improvement in the following $\binom{n}{m}$ iterations; if it fails to do so, the algorithm increases the strength to $m+1$ with probability at most $1/R$. If the algorithm misses strict improvements when the strength is m, it might never make progress in some scenarios, e. g., if there is no strict improvement in larger Hamming distances. However, by setting $R = n^c$ for an arbitrary constant $c > 0$, the probability of failure to make progress at strength s is at most $1/n^c$.

Although this failure probability can be negligible and satisfactory small in many applications, we still prefer to guarantee a finite expected time in making progress. In order to make the algorithm robust, we suggest iteratively returning to lower strengths [RW21b]. In the resulting algorithm, called *RLS with robust stagnation detection* or *SD-RLS*$^{\mathrm{r}}$, given a radius r, the algorithm uses all the strengths between 1 to r with their corresponding number of tries, that is, $\ell_s = \binom{n}{s}$ for strength s. Starting with radius 1, each time the algorithm does not find a strict improvement with radius r, it changes the radius to $r + 1$. Surprisingly, we achieve the same asymptotic upper bound on the expected time of leaving local optima as SD-RLS$^{\mathrm{p}}$ besides having a guarantee for finite expected runtime. It is shown that SD-RLS$^{\mathrm{r}}$ optimizes the jump function with jump size $2 \leq m = o(n/\ln n)$ in expected time $\binom{n}{m}(1 \pm o(1))$, which gives us a speedup of at least $(1 - o(1))\sqrt{2\pi m}$ compared to SD-(1+1) EA.

As further analysis of SD-RLS$^{\mathrm{r}}$, we studied the performance of the algorithm on the minimum spanning tree problem [RW21b]. On this problem, starting with an arbitrary spanning tree, the algorithm cannot make progress with strength 1; however, the algorithm returns to this strength after each strict improvement. Since the algorithm resets the radius to 1 after each strict improvement, there is a term $(2m \ln m)S$ in the upper bound on the expected runtime where S is the total expected number of strict improvements during the run. Since at most $(1+o(1))m^2 \ln m$ iterations with strength 2 are sufficient to find the global optimum in expectation, the number of iterations with strength 1 dominates the total runtime in some scenarios, e. g., on dense graphs. For example, it is not difficult to find an example where $\mathrm{E}(S) = \Omega(m)$, resulting in an upper bound of $cm^2 \ln m$ with a constant c depending on R but larger than 2. Therefore, the algorithm might suffer from iterations in lower strengths in highly multimodal optimization problems. We will describe these results in detail in Chapter 4.

Motivated by this, a mechanism called *radius memory* has been proposed in [RW21a] to manage the starting radius values after strict improvements instead of blindly resetting it to 1. In this mechanism, the algorithm keeps the successful strengths as the initial value for the radius after improvements. Moreover, it considers a maximum budget for the strengths smaller than the radius in the next phase, see Algorithm 15 in Chapter 5. This algorithm is known as *RLS with robust stagnation detection and radius memory mechanism* or *SD-RLS*^{m}. The analysis shows us the upper bound on the expected runtime of SD-RLS$^{\mathrm{m}}$ on the MST problem of $(1 + o(1))m^2 \ln m$, which outperforms the obtained upper bounds of the well-known black-box algorithms, such as RLS^{1,2}, the (1+1) EA, and SD-RLS$^{\mathrm{r}}$. In Chapter 5, more discussions and results of the radius memory mechanism are available.

While the variants of stagnation detection with k-bit flips as mentioned above are very efficient in leaving local optima with a single desired target solution (e. g., on JUMP), they perform poorly when there are several improving solutions, compared to some other heuristics. In a paper by Bambury et al. [BBD21a], it has been proven that we can find δ and m such that SD-RLS$^{\mathrm{r}}$ is slower than the standard (1+1) EA on $\textsc{Jump}_{k,m}$ by a factor polynomial in n of arbitrary degree. The reason for this poor performance is that the algorithm does not flip m bits in the first $\binom{n}{m-1} \ln R$ fitness evaluations until the radius increases to m. However, the (1+1) EA might flip m bits in a given iteration with a positive probability, so there is a chance that the (1+1) EA leaves a local optimum before $\binom{n}{m-1} \ln R$ iterations have elapsed if there is a sufficiently large number of improving solutions at the Hamming distance m.

2.2.3 Stagnation Detection with Heavy-tailed Mutations

Two successful strategies SD-RLS$^{\mathrm{r}}$ and the (1+1) EA with the heavy-tailed mutation have been discussed for leaving local optima. We have seen that when there are a few improving solutions, SD-RLS$^{\mathrm{r}}$ finds a strict improvement in a relatively efficient time, whereas the heavy-tailed mutation becomes more powerful and superior as the number of improving solutions increases. Now, we propose a mutation strategy in SD-FEA$_{\beta,\gamma,R}$ being efficient regardless of the number of improving solutions, using both ideas of the stagnation detection and power-law distributions employed in heavy-tailed mutation.

The SD-FEA$_{\beta,\gamma,R}$ has different phases until finding the strict improvement similarly as in stagnation detection. It starts with phase 1, and after $(1-\gamma)^{-1}\binom{n}{r}\ln(R)$ iterations in phase r without strict improvement, the algorithm increases r by one and enters phase $\min\{r+1, \lfloor \frac{n}{2.1} \rfloor\}$. In phase 1 only, the algorithm accepts search points with equal fitness values. In a given iteration of phase r, the algorithm flips s bits, where s follows the following distribution, see Figure 2.1 for an illustration.

$$s = \begin{cases} r & \text{with probability } 1-\gamma, \\ r + \mathrm{pow}(\beta, n-r) & \text{with probability } \gamma/2, \\ r - \mathrm{pow}(\beta, \max\{1, r-1\}) & \text{with probability } \gamma/2. \end{cases} \tag{2.1}$$

The algorithm SD-FEA$_{\beta,\gamma,R}$ has three parameters β, γ, and R. They might seem challenging to manage, but they are rather robust, and their values can be chosen easily. By setting $\beta > 1$, $\gamma = o(1)$ (e. g., $\gamma = (\log\log n)^{-1}$), and $R = e^{1/\gamma}$, we have achieved some acceptable asymptotic results in our analysis, which can

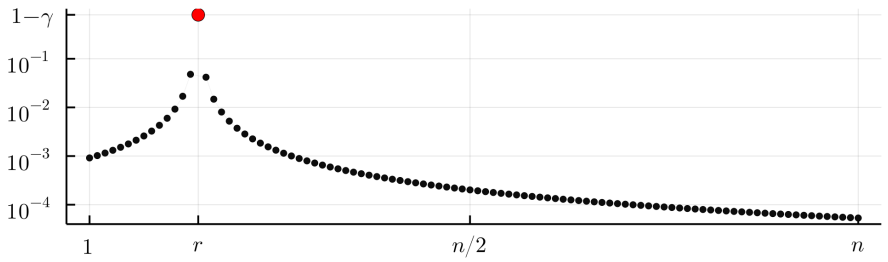


Figure 2.1: An illustration of the probabilities of flipping s out of n bits in phase r in a given iteration of SD-FEA$_{\beta,\gamma,R}$. The graph is based on numerical evaluations of Equation (2.1) with $n = 100$, $r = 15$, $\gamma = 0.2$, $\beta = 1.5$, which are represented in logarithmic scale.

be seen in the next paragraph. In order to read the detailed recommendations for choosing the parameters, see Section 7 in [DR22].

According to our analysis [DR22], SD-FEA$_{\beta,\gamma,R}$ inherits the beneficial properties of previous stagnation detection and fast mutation approaches. The expected optimization time of this algorithm on $\textsc{Jump}_m$ is bounded from above by $(1 + o(1))\binom{n}{m}$ for $2 \leq m = o(n/\ln R)$, and on $\textsc{Jump}_{k,m}$, it is bounded from above by

$$O\left(n \ln n + \binom{n}{m}\binom{k}{m}^{-1}(m - r')^{\beta}/\gamma\right),$$

where $r' = \min\left\{m, \arg\max_r \left\{\binom{n}{r} \leq \binom{n}{m}\binom{k}{m}^{-1}\frac{1}{\gamma}(m-r)^{\beta}\right\}\right\}$. The parameter r' indicates the first phase that the probability of finding an improvement becomes constant, and we have $1 \leq r' \leq m$. As we can see, SD-FEA$_{\beta,\gamma,R}$ achieves the same upper bound on the expected optimization on the classical $\textsc{Jump}_k$ functions as SD-RLS$^{\text{r}}$ and obtains an upper bound on the $\textsc{Jump}_{k,m}$ functions asymptotically less than the (1+1) FEA$_{\beta}$. Moreover, when $k = \omega(1) \cap O(\ln n)$ and $m = k - \Delta$ for a constant $\Delta \geq 2$, the optimization time of SD-FEA$_{\beta,\gamma,R}$ on $\textsc{Jump}_{k,m}$ functions is at most $O\left(\binom{n}{m}\binom{k}{m}^{-1}\gamma^{-1}\right)$, which is an example where the SD-FEA$_{\beta,\gamma,R}$ can asymptotically outperform the (1+1) FEA$_{\beta}$ and SD-RLS$^{\text{r}}$. We will describe these results in detail in Chapter 6.

Algorithm 5: Metropolis algorithm with temperature T for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$. We usually write $\alpha = e^{1/T}$

Select $x^{(0)}$ uniformly at random from $\{0,1\}^n$;
for $t \leftarrow 0, 1, \dots$ **do**
  Create y by flipping a bit of $x^{(t)}$ chosen uniformly at random;
  if $f(y) \geq f(x^{(t)})$ **then**
    $x^{(t+1)} \leftarrow y$;
  else
    $x^{(t+1)} \leftarrow y$ with probability $e^{(f(y)-f(x^{(t)}))/T}$ and
    $x^{(t+1)} \leftarrow x^{(t)}$ otherwise;

2.3 Non-elitist strategies: Metropolis Algorithm and Simulated annealing

2.3.1 Metropolis Algorithm

The *Metropolis Algorithm (MA)* [MRR+53] as defined in Algorithm 5, which is a non-elitist single-trajectory search heuristic, selects a random neighbor to the current search point (i. e., from the 1-neighborhood) and evaluates it. The MA always accepts if the offspring is at least as good as the parent. Also, it may accept inferior solutions with a small probability. More precisely, the inferior offspring is accepted with probability $e^{-\delta/T}$, where δ is the absolute fitness difference with respect to the parent and T is the *temperature*. In this thesis, we use the parameterization $\alpha = e^{1/T}$ for $\alpha > 0$, resulting in the acceptance probability $\alpha^{-\delta}$ for worse solutions. In the next section (Section 2.3.2) we will see that the MA is a special case of simulated annealing with fixed temperature.

On the unimodal benchmark OneMax, although accepting inferior solutions cannot be beneficial, the result in [JW07] shows that only temperatures $\alpha = \Omega(n/\log n)$ lead to polynomial runtimes. In [DHRW22], we have more rigorously shown that if $\alpha = \omega(\sqrt{n})$, the runtime of the MA on OneMax equals

$$(1 \pm o(1))n\ln(n) + \mathbb{1}_{\alpha \leq n}(1 \pm o(1))\alpha e^{n/\alpha},$$

which demonstrates that the best possible runtime of $(1 \pm o(1))n \ln n$ is obtained for $\alpha \geq \frac{n}{\ln \ln n}$, and for all $\alpha = \frac{1}{c}\frac{n}{\ln n}$, the runtime is $(1 \pm o(1))\frac{1}{c}n^{c+1}(\ln n)^{-1}$.

On multimodal problems, which is the main focus of this thesis, the MA has mostly been investigated on the Cliff functions family instead of Jump functions. The main reason is that the deceptive valley in the Jump functions does

not allow the search to get far from the local optimum. As another reason, on JUMP functions, the fitness difference between the local optimum and its neighbors in the valley is of order n, which is unlikely that the algorithm accepts such a fitness decrease. For a lower bound on JUMP functions and a more precise discussion, we refer the interested reader to [LOW19, Theorem 13].

The first analysis of the MA on CLIFF functions is conducted in [LOW19, Theorem 10]. On $\textsc{Cliff}_m$ (i. e., $\textsc{Cliff}_{d,m}$ with fixed $d = m - 3/2$), the authors show a lower bound of

$$\min\left\{\frac{1}{2}\cdot\frac{n-m+1}{m-1}\cdot(n/\log n)^{m-3/2}, n^{\omega(1)}\right\}.$$

To study the performance of MA on multimodal problems, we have conducted a rigorous research in [DHRW22] on the generalized version $\textsc{Cliff}_{d,m}$, resulting in Theorem 7.6 of Chapter 7. Also, in the same work [DHRW22] it is proved that the upper bound on the expected runtime of the (1+1) EA with the optimal parameter choice on $\textsc{Cliff}_{d,m}$ is at most

$$(1+o(1))\binom{m}{\lfloor d\rfloor+2}^{-1}\left(\frac{ne}{\lfloor d\rfloor+2}\right)^{\lfloor d\rfloor+2}.$$

These results show that the $(1+1)$ EA is provably at least as fast and for many parameters faster than the MA, even for an optimal choice of the temperature parameter of the MA, see Section 7.6 for a comprehensive discussion. Following this failed attempt to explain the effectiveness of the MA, it is still an open question of what is the real reason for the MA's success in a wide range of practical applications. [WZD21] showed a good performance of the MA on the DLB problem roughly by a factor of n faster than the (1+1) EA. A number of other results exist for artificially designed problems [DJW00, JW07, OPH+18]. We will describe these results in detail in Chapter 7.

2.3.2 Simulated Annealing

Simulated annealing (SA) proposed in [KGJV83] is another non-elitist simple single-trajectory search heuristic using 1-bit flip mutations, which may also accept inferior solutions with a small probability. A solution with fitness loss δ over the current solution is accepted with probability $e^{-\delta/T}$, where T is the current *temperature*, see Algorithm 6. If we consider a fixed temperature for the whole run, we have a special case of SA called Metropolis Algorithm as discussed in Section 2.3.1.

Algorithm 6: Simulated Annealing (SA) with starting temperature T_0 and cooling factor $\beta \leq 1$ for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$

Select $x^{(0)}$ from $\{0,1\}^n$;
for $t \leftarrow 0, 1, \ldots$ **do**
  Create y by flipping a bit of $x^{(t)}$ chosen uniformly at random;
  if $f(y) \geq f(x^{(t)})$ **then**
    $x^{(t+1)} \leftarrow y$;
  else
    $x^{(t+1)} \leftarrow y$ with probability $e^{(f(y)-f(x^{(t)}))/T_t}$ and
    $x^{(t+1)} \leftarrow x^{(t)}$ otherwise;
  $T_{t+1} \coloneqq T_t \cdot \beta$;

In SA the temperature is reduced during the run. This allows the algorithm to accept worsening moves easily in the early stages of the run, whereas later worsening moves are accepted with smaller probability, bringing the algorithm closer to the algorithm RLS. There are different strategies to reduce the temperature called *cooling schedules*. A popular choice, already proposed in [KGJV83], is a multiplicative cooling schedule; Starting with a given temperature T_0, we reduce the temperature by some factor β in each iteration.

Although we do not see a reducing interest in SA in practice [FS19], there are few research papers conducted to study the theory of SA with variant cooling schedules. Wegener in [Weg05] proposed a construction of an instance of the MST problem where the MA with any fixed temperature fails to find an MST, but SA computes an optimal solution efficiently. In [DRW22], we have shown that simulated annealing is a polynomial-time approximation scheme for the MST problem, thereby proving a conjecture by Wegener [Weg05]. We showed that SA with $T_0 \geq w_{\max}$ and multiplicative cooling schedule with parameter $\beta = 1 - 1/\ell$, where $\ell = \omega(mn\ln(m))$, with probability at least $1 - 1/m$ computes in time $O(\ell(\ln\ln(\ell) + \ln(T_0/w_{\min})))$ a spanning tree with weight at most $1+\kappa$ times the optimum weight, where

$$1 + \kappa = \frac{(1+o(1))\ln(\ell m)}{\ln(\ell) - \ln(mn\ln(m))}.$$

Consequently, stated in a simplified manner, for any $\varepsilon > 0$, we can choose ℓ in such a way that a $(1+\varepsilon)$-approximation is found in time

$$O((mn\ln(n))^{1+1/\varepsilon+o(1)}(\ln\ln n + \ln(T_0/w_{\min}))),$$

with probability at least $1 - 1/m$. Also, the analyses have led to improved results in the case of $(1+\epsilon)$-separated weights (defined in Section 1.5.1) from

Wegener's runtime guarantee of $O(m^{8+8/\varepsilon})$ to

$$O((mn\ln(n))^{1+1/\varepsilon+o(1)}(\ln\ln n + \ln(T_0/w_{\min}))),$$

where simulated annealing yields an optimal solution with high probability. We will describe these results in detail in Chapter 8.

Chapter 3

Paper A: Self-Adjusting Evolutionary Algorithms for Multimodal Optimization

Recent theoretical research has shown that self-adjusting and self-adaptive mechanisms can provably outperform static settings in evolutionary algorithms for binary search spaces. However, the vast majority of these studies focuses on unimodal functions which do not require the algorithm to flip several bits simultaneously to make progress. In fact, existing self-adjusting algorithms are not designed to detect local optima and do not have any obvious benefit to cross large Hamming gaps.

We suggest a mechanism called stagnation detection that can be added as a module to existing evolutionary algorithms (both with and without prior self-adjusting schemes). Added to a simple (1+1) EA, we prove an expected runtime on the well-known Jump benchmark that corresponds to an asymptotically optimal parameter setting and outperforms other mechanisms for multimodal optimization like heavy-tailed mutation. We also investigate the module in the context of a self-adjusting (1+λ) EA.

To explore the limitations of the approach, we additionally present an example where both self-adjusting mechanisms, including stagnation detection, do not help to find a beneficial setting of the mutation rate. Finally, we investigate our module for stagnation detection experimentally.

3.1 Introduction

The runtime analysis of evolutionary algorithms (EAs) is a research area that originated from the analysis of classical randomized algorithms, where the aim is to prove rigorous statements on the expected runtime and approximation quality of the algorithm depending on the problem size; however, a notable difference is that this theoretical research on EAs mostly considers the number of objective function calls as complexity measure. Since the late 1990s, several results on the runtime of simple and moderately complex EAs as well as of other nature-inspired algorithms have emerged [AD11, NW10, Jan13, DN20]. EAs are parameterized algorithms, so it has been ongoing research to understand how to choose their parameters best. Self-adjusting mechanisms address this issue as a non-static parameters control framework that can learn acceptable or even near-optimal parameter settings on the fly.

Recent theoretical research on self-adjusting algorithms in discrete search spaces has produced a remarkable body of results showing that self-adjusting and self-adaptive mechanisms outperform static parameter settings. Examples include an analysis of the well-known $(1+(\lambda,\lambda))$ GA using a 1/5-rule to adjust its mutation rate on OneMax [DD18], of a self-adjusting $(1+\lambda)$ EA sampling offspring with different mutation rates [DGWY19], matching the parallel black-box complexity of the OneMax function, and a self-adaptive variant of the latter [DWY18]. Furthermore, self-adjusting schemes for algorithms over the search space $\{0,\dots,r\}^n$ for $r > 1$ provably outperform static settings [DDK18] of the mutation operator. Self-adjusting schemes are also closely related to hyper-heuristics which, e. g., can dynamically choose between different mutation operators and therefore outperform static settings [LOW20]. Besides the mutation probability, other parameters like the population sizes may be adjusted during the run of an evolutionary algorithm (EA) and analyzed from a runtime perspective [LS11]. Moreover, there is much empirical evidence (e. g. [DYvR+18, DW18, RABD19, Faj19, AM16, DDY16a]) showing that parameters of EAs should be adjusted during its run to optimize its runtime. See also the survey article [DD20] for an in-depth coverage of parameter control, self-adjusting algorithms, and theoretical runtime results.

A common feature of existing self-adjusting schemes is that they use different settings of a parameter (e. g., the mutation rate/strength) and – in some way – measure and compare the progress achievable with the different settings. For example, the 2-rate $(1+\lambda)$ EA from [DGWY19] with the current strength r samples $\lambda/2$ of the offspring with strength $r/2$ (where we define strength as the expected number of flipped bits, i. e., n times the mutation probability) and the other half with strength $2r$. The strength is afterwards adjusted to the one used by a fittest offspring. Similarly, the 1/5-rule [DD18] increases the muta-

tion rate if fitness improvements happen frequently and decreases it otherwise. This requires that the algorithm is likely enough to make *some* improvements with the different parameters tried or, at least, that the smallest disimprovement observed in unsuccessful mutations gives reliable hints on the choice of the parameter. However, there are situations where the algorithm cannot make progress and does not learn from unsuccessful mutations either. This can be the case when the algorithm reaches local optima escaping from which requires an unlikely event (such as flipping many bits simultaneously) to happen. Classical self-adjusting algorithms would observe many unsuccessful steps in such situations and suggest to set the mutation rate to its minimum although that might not be the best choice to leave the local optimum. In fact, the vast majority of runtime results for self-adjusting EAs is concerned with unimodal functions that have no other local optima than the global optimum. An exception is the work [DL16b] which considers a self-adaptive EA allowing two different mutation probabilities on a specifically designed multimodal problem. Altogether, there is a lack of theoretical results giving guidance on how to design self-adjusting algorithms that can leave local optima efficiently.

In this paper, we address this question and propose a self-adjusting mechanism called *stagnation detection* that adjusts mutation rates when the algorithm has reached a local optimum. In contrast to previous self-adjusting algorithms this mechanism is likely to increase the mutation in such situations, leading to a more efficient escape from local optima. This idea has been mentioned before, e. g., in the context of population sizing in stagnation [EMV04]; also, recent empirical studies of the above-mentioned 2-rate (1+λ) EA, handling of stagnation by increasing the variance was explicitly suggested in [YDB19]. Our contribution has several advantages over previous discussion of stagnation detection: it represents a simple module that can be added to several existing evolutionary algorithms with little effort, it provably does not change the behavior of the algorithm on unimodal functions (except for small error terms), allowing the transfer of previous results, and we provide rigorous runtime analyses showing general upper bounds for multimodal functions including its benefits on the well-known Jump benchmark function.

In a nutshell, our stagnation detection mechanism works in the setting of pseudo-Boolean optimization and standard bit mutation. Starting from strength $r = 1$, it increases the strength from r to $r + 1$ after a long waiting time without improvement has elapsed, meaning it is unlikely that an improving bit string at Hamming distance r exists. This approach bears some resemblance with variable neighborhood search (VNS) [HM18]; however, the idea of VNS is to apply local search with a fixed neighborhood until reaching a local optimum and then to adapt the neighborhood structure. There have also been so-called quasirandom evolutionary algorithms [DFW10] that search the set of Hamming neighbors of a search point more systematically; however, these approaches do not change

the expected number of bits flipped. In contrast, our stagnation detection uses the whole time an unbiased randomized global search operator in an EA and just adjusts the underlying mutation probability. Statistical significance of long waiting times is used, indicating that improvements at Hamming distance r are unlikely to exist; this is rather remotely related to (but clearly inspired by) the estimation-of-distribution algorithm sig-cGA [DK18] that uses statistical significance to counteract genetic drift.

This paper is structured as follows: In Section 3.2, we introduce the concrete mechanism for stagnation detection and employ it in the context of a simple, static (1+1) EA and the already self-adjusting 2-rate (1+λ) EA. Moreover, we collect tools for the analysis that are used in the rest of the paper. Section 3.3 deals with concrete runtime bounds for the (1+1) EA with stagnation detection. Besides general upper bounds, we prove a concrete result for the Jump benchmark function that is asymptotically optimal for algorithms using standard bit mutation and outperforms previous mutation-based algorithms for this function like the heavy-tailed EA from [DLMN17]. Elementary techniques are sufficient to show these results. To explore the limitations of stagnation detection and other self-adjusting schemes, we propose in Section 3.4 a function where these mechanisms provably fail to set the mutation rate to a beneficial regime. As a technical tool, we use drift analysis and analyses of occupation times for processes with strong drift. To that purpose, we use a theorem by Hajek [Haj82] on occupation times that, to the best of the knowledge, was not used for the analysis of randomized search heuristics before and may be of independent interest. Finally, in Section 3.5, we add some empirical results, showing that the asymptotically smaller runtime of our algorithm on Jump is also visible for small problem dimensions. We finish with some conclusions.

3.2 Preliminaries

We shall now formally define the algorithms analyzed and present some fundamental tools for the analysis.

3.2.1 Algorithms

We are concerned with pseudo-Boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$ that w. l. o. g. are to be maximized. A simple and well-studied EA studied in many runtime analyses (e. g., [DJW02]) is the (1+1) EA displayed in Algorithm 7. It uses standard bit mutation with strength r, where $1 \leq r \leq n/2$, which means that

every bit is flipped independently with probability r/n. Usually, $r = 1$ is used, which is the optimal strength on linear functions (see [Wit13] for the general case and also [GKS99] for the special case of OneMax). Smaller strengths lead to less than 1 bit being flipped in expectation, and strengths above $n/2$ in binary search spaces are considered "ill-natured" [ADK19] since a mutation at a bit should not be more likely than a non-mutation.

Algorithm 7: (1+1) EA with static strength r

Select x uniformly at random from $\{0,1\}^n$;
for $t \leftarrow 1, 2, \ldots$ **do**
  Create y by flipping each bit in a copy of x independently with probability $\frac{r}{n}$;
  if $f(y) \geq f(x)$ **then**
    $x \leftarrow y$;

The *runtime* (also called *optimization time*) of the (1+1) EA on a function f is the first point of time t where a search point of maximal fitness has been created; often the expected runtime, i. e., the expected value of this time, is analyzed. The (1+1) EA with $r = 1$ has been extensively studied on simple unimodal problems like

$$\textsc{OneMax}(x_1, \ldots, x_n) \coloneqq \|x\|_1 \text{ and } \textsc{LeadingOnes}(x_1, \ldots, x_n) \coloneqq \sum_{i=1}^{n} \prod_{j=1}^{i} x_j,$$

but also on the multimodal $\textsc{Jump}_m$ function with gap size m defined as follows.

$$\textsc{Jump}_m(x_1, \ldots, x_n) = \begin{cases} m + \|x\|_1 & \text{if } \|x\|_1 \leq n - m \text{ or } \|x\|_1 = n, \\ n - \|x\|_1 & \text{otherwise.} \end{cases}$$

The classical (1+1) EA with $r = 1$ optimizes these functions in expected time $\Theta(n \log n)$, $\Theta(n^2)$ and $\Theta(n^m + n \log n)$, respectively (see, e. g., [DJW02]).

The first two problems are unimodal functions, while Jump for $m \geq 2$ is multimodal and has a local optimum at the set of points such that $\|x\|_1 = n - m$. To overcome this optimum, m bits have to be flipped simultaneously. It is well known [DLMN17] that the time to leave this optimum is minimized at strength m instead of strength 1 (see below for a more detailed exposition of this phenomenon). Hence, the (1+1) EA would benefit from increasing its strength when sitting at the local optimum. The algorithm does not immediately know that it sits at a local optimum. However, if there is an improvement at Hamming distance 1 then such an improvement has probability at least $(1/n)(1-1/n)^{n-1}$ with strength 1, and the probability of not finding it in $(1-1/n)^{1-n} n \ln n$ steps

is at most

$$\left(1 - \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1}\right)^{(1-1/n)^{1-n} n \ln n} \leq \frac{1}{n}.$$

Similarly, if there is an improvement that can be reached by flipping k bits simultaneously and the current strength equals k, then the probability of not finding it within $(n/k)^k(1-k/n)^{k-n} \ln n$ steps is at most

$$\left(1 - \left(\frac{k}{n}\right)^{k}\left(1 - \frac{k}{n}\right)^{n-k}\right)^{(n/k)^k(1-k/n)^{k-n} \ln n} \leq \frac{1}{n}.$$

Hence, after $(n/k)^k(1-k/n)^{k-n} \ln n = (n/k)^k(n/(n-k))^{n-k} \ln n$ steps without improvement there is high evidence for that no improvement at Hamming distance k exists.

We put these ideas into an algorithmic framework by counting the number of so-called unsuccessful steps, i. e., steps that do not improve fitness. Starting from strength 1, the strength is increased from r to $r+1$ when the counter exceeds the threshold $((n/r)^r(n/(n-r))^{n-r}) \ln(enR)$ for a parameter R to be discussed shortly. Both counter and strength are reset (to 0 and 1, respectively) when an improvement is found, i. e., a search point of strictly better fitness. In the context of the (1+1) EA, the stagnation detection (SD) is incorporated in Algorithm 8. We see that the counter u is increased in every iteration that does not find a strict improvement. However, search points of equal fitness are still accepted as in the classical (1+1) EA. We note that the strength stays at its initial value 1 if finding an improvement does not take longer than the corresponding threshold $n(n/(n-1)^{n-1}) \ln(enR)$; if the threshold is never exceeded the algorithm behaves identical to the (1+1) EA with strength 1 according to Algorithm 7.

The parameter R can be used to control the probability of failing to find an improvement at the "right" strength. More precisely, the probability of not finding an improvement at distance r with strength r is at most

$$\left(1 - \left(\frac{r}{n}\right)^{r}\left(1 - \frac{r}{n}\right)^{n-r}\right)^{(n/r)^r(n/(n-r))^{n-r} \ln(enR)} \leq \frac{1}{enR}.$$

As shown below in Theorem 3.5, if R is at least set to the maximum of n^3 and the number of fitness values of the underlying function f, i. e., $R \geq \max\{n^3, |\mathrm{Im}(f)|\}$, then the probability of ever missing an improvement at the right strength is sufficiently small throughout the run. We recommend at least $R = n^3$ if nothing is known about the range of f, resulting in a threshold of at least $(n/r)^r(n/(n-r))^{n-r} \ln(en^4)$ at strength r.

Algorithm 8: (1+1) EA with stagnation detection (SD-(1+1) EA)

Select x uniformly at random from $\{0,1\}^n$ and set $r_1 \leftarrow 1$;
$u \leftarrow 0$;
for $t \leftarrow 1, 2, \dots$ **do**
  Create y by flipping each bit in a copy of x independently with probability $\frac{r_t}{n}$;
  $u \leftarrow u + 1$;
  if $f(y) > f(x)$ **then**
    $x \leftarrow y$;
    $r_{t+1} \leftarrow 1$;
    $u \leftarrow 0$;
  else
    if $f(y) = f(x)$ ***and*** $r_t = 1$ **then**
      $x \leftarrow y$;
    if $u > \left(\frac{n}{r_t}\right)^{r_t} \left(\frac{n}{n-r_t}\right)^{n-r_t} \ln(enR)$ **then**
      $r_{t+1} \leftarrow \min\{r_t + 1, n/2\}$;
      $u \leftarrow 0$;
    else
      $r_{t+1} \leftarrow r_t$;

Compared to the conference version of this paper [RW20b], we have optimized the choice of the threshold on the number of unsuccessful steps at strength r. Previously the threshold was $2\left(\frac{en}{r}\right)^r \ln(nR)$, which is larger than the new choice $\left(\frac{n}{r}\right)^r \left(\frac{n}{n-r}\right)^{n-r} \ln(enR)$ in Algorithm 8. We note that

$$\left(\frac{n}{n-r}\right)^{n-r} = \left(1 + \frac{r}{n-r}\right)^{n-r} \leq e^r,$$

so that for $r = o(n)$ the factor $\left(\frac{en}{r}\right)^r$ from the old choice is recovered up to lower order terms. However, for $r = \Omega(n)$ the new choice is asymptotically smaller than before, which results in several improved bounds for scenarios where the mutation strength has to reach $\Omega(n)$, see, e. g., Theorem 3.4 and Corollary 3.9.

Stagnation detection can also be used to other randomized search heuristics – for example, this has been done recently for randomized local search using k-bit flip mutation instead of standard bit mutation [RW21b] and for multi-objective algorithms [DZ21]. We also added stagnation detection to the (1+λ) EA with self-adjusting mutation rate defined in [DGWY19] (adapted to maximization of the fitness function), where half of the offspring are created with strength $r/2$ and the other half with strength $2r$; see Algorithm 9.

Algorithm 9: (1+λ) EA with two-rate standard bit mutation and stagnation detection (SASD-(1+λ) EA)

Select x uniformly at random from $\{0,1\}^n$ and set $r_1 \leftarrow r^{\text{init}}$;
$u \leftarrow 0$;
$g \leftarrow \mathit{False}$; `// boolean variable indicating stagnation detection`
for $t \leftarrow 1, 2, \dots$ **do**
  $u \leftarrow u + 1$;
  if $g = \mathit{True}$ **then**
    ▷ State 1 – Stagnation Detection;
    for $i \leftarrow 1, \dots, \lambda$ **do**
      Create x_i by flipping each bit in a copy of x independently with probability $\frac{r_t}{n}$;
    $y \leftarrow \arg\max_{x_i} f(x_i)$ (breaking ties randomly);
    if $f(y) > f(x)$ **then**
      $x \leftarrow y$;
      $r_{t+1} \leftarrow r^{\text{init}}$;
      $g \leftarrow \mathit{False}$;
      $u \leftarrow 0$;
    else
      if $u > \left(\frac{n}{r_t}\right)^{r_t} \left(\frac{n}{n-r_t}\right)^{n-r_t} \ln(enR)/\lambda$ **then**
        $r_{t+1} \leftarrow \min\{r_t + 1, n/2\}$;
        $u \leftarrow 0$;
      else
        $r_{t+1} \leftarrow r_t$;
  else i. e., $g = \mathit{False}$
    ▷ State 2 – Self-Adjusting (1+λ) EA;
    for $i \leftarrow 1, \dots, \lambda$ **do**
      Create x_i by flipping each bit in a copy of x independently with probability $\frac{r_t}{2n}$ if $i \leq \lambda/2$ and with probability $2r_t/n$ otherwise;
    $y \leftarrow \arg\max_{x_i} f(x_i)$ (breaking ties randomly);
    if $f(y) \geq f(x)$ **then**
      if $f(y) > f(x)$ **then**
        $u \leftarrow 0$;
      $x \leftarrow y$;
    Perform one of the following two actions with prob. 1/2:
      – Replace r_t with the strength that y has been created with;
      – Replace r_t with either $r_t/2$ or $2r_t$, each with probability 1/2;
    $r_{t+1} \leftarrow \min\{\max\{2, r_t\}, n/4\}$;
    if $u > \left(\frac{n}{r_t}\right)^{r_t} \left(\frac{n}{n-r_t}\right)^{n-r_t} \ln(enR)/\lambda$ **then**
      $r_{t+1} \leftarrow 2$;
      $g \leftarrow \mathit{True}$;
      $u \leftarrow 0$;

Unsuccessful mutations are counted in the same way as in Algorithm 8, taking into account that λ offspring are used. The algorithm can be in two states, remotely resembling a hyperheuristic [BGH+13]. Unless the counter threshold

is reached and a strength increase is triggered, the algorithm behaves the same as the self-adjusting (1+λ) EA from [DGWY19] (State 2). If, however, the counter threshold $n \ln(enR)/\lambda$ is reached, then the algorithm changes to the module that keeps increasing the strength until a strict improvement is found (State 1). Since it does not make sense to decrease the strength in this situation, all offspring use the same strength until finally an improvement is found and the algorithm changes back to the original behavior using two strengths for the offspring. The boolean variable g keeps track of the state. From the discussion of these two algorithms, we see that the stagnation detection consisting of a counter for unsuccessful steps, threshold, and strength increase also can be added to other algorithms, while keeping their original behavior unless the counter threshold it reached. In this paper, we investigate this self-adjusting (1+λ) EA with stagnation detection (SASD-(1+λ) EA) mostly experimentally; however, a lower bound on its runtime on an example function is given in Section 3.4.

3.2.2 Mathematical Tools

We now collect frequently used mathematical tools. The first one is a simple summation formula used to analyze the time spent until the strength is increased to a certain value.

Lemma 3.1. *For $1 \leq m < n/2$, we have*

$$\sum_{i=1}^{m} \left(\frac{n}{i}\right)^i \left(\frac{n}{n-i}\right)^{n-i} < \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} \left(\frac{n-m}{n-2m}\right).$$

Proof. By an index transformation for i, we obtain

$$\sum_{i=1}^{m} \left(\frac{n}{i}\right)^i \left(\frac{n}{n-i}\right)^{n-i} = \sum_{i=0}^{m-1} \left(\frac{n}{m-i}\right)^{m-i} \left(\frac{n}{n-m+i}\right)^{n-m+i}$$

$$= \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} \sum_{i=0}^{m-1} \left(\frac{m}{m-i}\right)^{m-i} \left(\frac{n-m}{n-m+i}\right)^{n-m+i} \left(\frac{m}{n-m}\right)^i$$

$$= \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} \cdot \sum_{i=0}^{m-1} \left(1+\frac{i}{m-i}\right)^{m-i} \left(1-\frac{i}{n-m+i}\right)^{n-m+i} \left(\frac{m}{n-m}\right)^i.$$

Via the inequality $1 + x \leq e^x$ for all $x \in \mathbb{R}$, the last expression can be bounded from above by

$$\left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} \sum_{i=0}^{\infty} \frac{e^i}{e^i} \left(\frac{m}{n-m}\right)^i$$
$$= \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} \left(\frac{n-m}{n-2m}\right).$$

The last equation is calculated by the geometric series formula. □

The following result due to Hajek applies to processes with a strong drift towards some target state, resulting in decreasing occupation probabilities with respect to the distance from the target. On top of this occupation probabilities, the theorem bounds *occupation times*, i. e., the number of steps that the process spends in a non-target state over a certain time period.

Theorem 3.2 (Theorem 3.1 in [Haj82])**.** *Let X_t, $t \geq 0$, be a stochastic process adapted to a filtration $\mathcal{F}_t$ on $\mathbb{R}$. Let $a \in \mathbb{R}$. Assume for $\Delta_t = X_{t+1} - X_t$ that there are $\eta > 0, \rho < 1$ and $D > 0$ such that*

1. $\mathrm{E}\left(e^{\eta\Delta} \mid \mathcal{F}_t; X_t > a\right) \leq \rho$,
2. $\mathrm{E}\left(e^{\eta\Delta} \mid \mathcal{F}_t; X_t \leq a\right) \leq D$.

If additionally X_0 is of exponential type (i. e., $\mathrm{E}\left(e^{\lambda X_0}\right)$ is finite for some $\lambda > 0$) then for any constant $\epsilon > 0$ there exist absolute constants $K \geq 0, \delta < 1$ such that for all $b \geq a$ and $T \geq 1$

$$\Pr\Big(\frac{1}{T}\sum_{t=1}^{T} \mathbb{1}_{X_t \leq b} \leq 1 - \epsilon - \frac{1-\epsilon}{1-\rho} D e^{\eta(a-b)}\Big) \;\leq\; K\delta^T.$$

3.3 Analysis of SD-(1+1) EA

In this section, we study the SD-(1+1) EA from Algorithm 8 in greater detail. We show general upper bounds on multimodal functions and then analyze the special case of Jump more precisely. We also show the important result that on unimodal functions, the SD-(1+1) EA with high probability behaves in the same way as the classical (1+1) EA with strength 1. Moreover, this result asymptotically transfers bounds on the expected optimization time from the (1+1) EA to the SD-(1+1) EA.

3.3.1 Expected Times to Leave Local Optima

In the following, given a fitness function $f\colon \{0,1\}^n \to \mathbb{R}$, we call the *fitness level gap* (or in short, *gap*) of a point $x \in \{0,1\}^n$ the maximum of all individual gap sizes in the fitness level of x, where the *individual gap* is the minimum Hamming distance to points with strictly larger fitness function value, i. e.,

$$\begin{aligned}&\text{IndividualGap}(x) \coloneqq \min\{H(x,y) : f(y) > f(x), y \in \{0,1\}^n\},\\ &\text{gap}(x) = \text{FitnessLevelGap}(x) \coloneqq \max_{\{y \mid f(y)=f(x)\}} \text{IndividualGap}(y).\end{aligned}$$

If the algorithm creates a point of $\text{gap}(x)$ distance from the current search point x, we can make progress with a positive probability. Note that $\text{gap}(x) = 1$ is allowed, so the definition also covers points that are not local optima.

Let phase r consist of all points of time where strength r is used in the algorithm with stagnation counter. Let E_r be the event of **not** finding a strict improvement in phase r, and U_r be the event of not finding a strict improvement during phases 1 to $r-1$ and finding in phase r. In other words, $U_r = E_1 \cap \cdots \cap E_{r-1} \cap \overline{E_r}$.

The following lemma will be used throughout this section. It shows that the probability of not finding a search point with larger fitness value in phases at least the real gap size is small; however, by definition, phase $n/2$ is not finished before the algorithm finds an improvement. In the statement of the lemma, recall that the parameter R controls the threshold for the number of unsuccessful steps in stagnation detection.

Lemma 3.3. *Consider the SD-(1+1) EA on a pseudo-Boolean fitness function $f\colon \{0,1\}^n \to \mathbb{R}$. Assume that $x \in \{0,1\}^n$ be the search point at the beginning of phase $r \geq 1$, i. e., immediately after the counter u has been (re)set to 0 and r becomes the current strength. Then*

$$\Pr(E_r) \leq \begin{cases} \frac{1}{enR} & \text{if } \text{gap}(x) \leq r < n/2, \\ 0 & \text{if } r = n/2. \end{cases}$$

Proof. The algorithm spends $(\frac{n}{r})^r (\frac{n}{n-r})^{n-r} \ln(enR)$ steps at strength r until it increases the counter. Then, the probability of not improving at strength $r \geq m$ is at most

$$\Pr(E_r) = \left(1 - \left(1 - \frac{r}{n}\right)^{n-m} \left(\frac{r}{n}\right)^m\right)^{n^r/r^r (n/(n-r))^{n-r} \ln(enR)} \leq \frac{1}{enR}.$$

During phase $n/2$, the algorithm does not increase the strength, and it continues to mutate each bit with probability of $1/2$. As each point in the search space is accessible in this phase, the probability of eventually failing to find the improvement is 0. □

We turn the previous observation into a general theorem on improvement times.

Theorem 3.4. *Consider the SD-(1+1) EA with $R \geq n^3$ on 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. Define T_x as the time to find a strict improvement from x and let $L_{x,m} \coloneqq \mathrm{E}(T_x \mid \mathrm{gap}(x) = m)$. Then for all $m \in \{1,\dots,n\}$, we have $L_{x,m} = O(2^n n \ln(enR))$. Moreover, for any constant $0 < \epsilon < 1$, if $m < (1-\epsilon)n/2$, then*

$$L_{x,m} \leq \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} \left(1 + O\left(\frac{m}{n}\ln(enR)\right)\right).$$

Note that while waiting for a strict improvement, the current search point of the SD-(1+1) EA may change to another search point with the same fitness value in phase 1, i. e., if $r = 1$; however, this does not change the value $m = \mathrm{gap}(x)$ considered in the theorem.

Proof of Theorem 3.4. Let I_r be the number of iterations spent in phase r. Using linearity of expectation, we have

$$\mathrm{E}(T_x) = \sum_{r=1}^{\lceil n/2\rceil - 1} \mathrm{E}(I_r) + \mathrm{E}\left(I_{n/2}\right).$$

We first prove the upper bound for the case that there is a constant $0 < \epsilon < 1$ such that $m < (1-\epsilon)n/2$. In order to accomplish this, we bound the summation $\sum_{r=1}^{\lceil n/2\rceil - 1} \mathrm{E}(I_r) + \mathrm{E}\left(I_{n/2}\right)$ for different ranges of the strengths, where $0 \leq r < m$, $r = m$, and $r < m$. In the first case, we assume that the strength is less than m, i. e. $r < m$. Then, we have $\mathrm{E}(I_r)$ at most the threshold value in phase r. Thus, by using Lemma 3.1, we compute

$$\begin{aligned}
&\sum_{r=1}^{m-1} \mathrm{E}(I_r) \leq \sum_{r=1}^{m-1} \left(\frac{n}{r}\right)^r \left(\frac{n}{n-r}\right)^{n-r} \ln(enR) \\
&< \left(\frac{n}{m-1}\right)^{m-1} \left(\frac{n}{n-m+1}\right)^{(n-m+1)} \left(\frac{n-m+1}{n-2m+2}\right) \ln(enR)
\end{aligned}$$

$$
\begin{aligned}
&= \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{(n-m)} \\
&\left(\left(\frac{m^m}{(m-1)^{m-1}}\right)\left(\frac{(n-m)^{n-m}}{(n-m+1)^{n-m+1}}\right)\left(\frac{n-m+1}{n-2m+2}\right)\ln(enR)\right) \\
&= \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{(n-m)}\left(\frac{m}{n-m}\left(1+\frac{1}{m-1}\right)^{m-1}\left(1-\frac{1}{n-m+1}\right)^{n-m+1}\right. \\
&\qquad \left.\left(\frac{n-m+1}{n-2m+2}\right)\ln(enR)\right) \\
&\leq \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{(n-m)}\left(\frac{m}{n-m}\cdot\frac{e}{e}\cdot\frac{n-m+1}{n-2m+2}\ln(enR)\right).
\end{aligned}
$$

Then, for $m < (1-\epsilon)\frac{n}{2}$, we can bound the last expression from above and compute

$$
\sum_{r=1}^{m-1} \mathrm{E}\left(I_r\right) = O\left(\left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{(n-m)} \cdot \frac{m\ln R}{n}\right).
$$

When the strength is m, i. e., the mutation probability is m/n, within an expected number of at most $\left((m/n)^m(1-m/n)^{n-m}\right)^{-1}$ steps, a better point will be found or the phase ends since we are dealing with a truncated geometric distribution with success probability $(m/n)^m(1-m/n)^{n-m}$. Thus,

$$
\mathrm{E}\left(I_m\right) \leq \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m}.
$$

For $r > m$, with probability $\Pr\left(U_r\right)$, the algorithm does not make progress with strengths less than r, and the strength is increased to r. In phase r, the number of iterations is at most the threshold value. This is because for $r < n/2$, the algorithm changes the strength after the threshold value is exceeded and for $r = n/2$, the algorithm makes progress within 2^n iterations in expectation via the geometric distribution, which is less than $(n/r)^r(n/(n-r))^{n-r}\ln(enR)$. Thus, for all strengths $r > m$, we have

$$
\mathrm{E}\left(I_r\right) \leq \Pr\left(U_r\right) \cdot \left(\frac{n}{r}\right)^r \left(\frac{n}{n-r}\right)^{n-r} \ln(enR).
$$

Using Lemma 3.3, we have $\Pr\left(U_r\right) < \prod_{j=m}^{r-1} \Pr\left(E_j\right) < (enR)^{-(r-m)}$. Hence, we can bound

$$
\mathrm{E}\left(I_r\right) \leq (enR)^{-(r-m)} \cdot \left(\frac{n}{r}\right)^r \left(\frac{n}{n-r}\right)^{n-r} \ln(enR)
$$

$$= (enR)^{-(r-m)} \cdot \left(\frac{n}{r}\right)^r \left(\frac{n}{n-m}\right)^{n-r} \left(1 + \frac{r-m}{n-r}\right)^{n-r} \ln(enR).$$

Since $(1 + (r-m)/(n-r))^{n-r} \leq e^{r-m}$ using the inequality $1 + x \leq e^x$ for all $x \in \mathbb{R}$, the last expression is less than

$$\begin{aligned}
&(nR)^{-(r-m)} \cdot \left(\frac{n}{r}\right)^r \left(\frac{n}{n-m}\right)^{n-r} \ln(enR) \\
&= \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} \cdot \frac{m^m}{r^r} \left(\frac{n}{n-m}\right)^{m-r} R^{m-r} \ln(enR).
\end{aligned}$$

Since $r > m$ and $(n/(n-m))^{m-r} \leq 1$, the last expression is bounded from above by

$$\left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} \cdot R^{m-r} \ln(enR),$$

and since $R \geq n^3$, it is at most

$$o\left(\frac{1}{n} \cdot \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m}\right).$$

Altogether, we achieve

$$\begin{aligned}
\mathrm{E}(T_x) &= \sum_{r=1}^{\lceil n/2 \rceil - 1} \mathrm{E}(I_r) + \mathrm{E}\left(I_{n/2}\right) \\
&= \sum_{r=1}^{m-1} \mathrm{E}(I_r) + \mathrm{E}(I_m) + \sum_{r=m+1}^{\lceil n/2 \rceil - 1} \mathrm{E}(I_r) + \mathrm{E}\left(I_{n/2}\right) \\
&= \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{(n-m)} O\left(1 + \frac{m \ln(enR)}{n}\right).
\end{aligned}$$

Now, we investigate an upper bound on the improvement time for all gap sizes including the cases that for all constants $0 < \epsilon < 1$, $m > (1-\epsilon)n/2$. For $r \leq \lceil n/2 \rceil - 1$, $\mathrm{E}(I_r)$ is at most the threshold value considered for strength r. When the strength is $n/2$ (i. e., the mutation probability is $1/2$), the algorithm makes progress within 2^n iterations in expectation through the geometric distribution with success probability 2^{-n}. We compute

$$\mathrm{E}(T_x) \leq \sum_{r=1}^{\lceil n/2 \rceil - 1} \left(\frac{n}{r}\right)^r \left(\frac{n}{n-r}\right)^{n-r} \ln(enR) + 2^n$$

$$\leq O(n/2)\left(\frac{n}{\lceil n/2\rceil - 1}\right)^{\lceil n/2\rceil - 1}\left(\frac{n}{n - \lceil n/2\rceil + 1}\right)^{n-\lceil n/2\rceil+1} \ln(enR) + 2^n$$
$$= O(2^n n \ln(enR)).$$ □

We now present the above-mentioned important "simulation result" implying that on unimodal functions, the stagnation detection of SD-(1+1) EA is unlikely ever to trigger a strength increase during its run so that the algorithm behaves like the (1+1) EA then. Moreover, for a wide range of runtime bounds obtained via the fitness level method [Weg02, Sud13], we show that these bounds transfer to the SD-(1+1) EA up to vanishingly small error terms. The proof carefully estimates the probability of the strength ever exceeding 1.

Lemma 3.5. *Let $f\colon \{0,1\}^n \to \mathbb{R}$ be a unimodal function and consider the SD-(1+1) EA with $R \geq \max\{|\mathrm{Im}(f)|, n^3\}$. Then, with probability $1 - o(1)$, the SD-(1+1) EA never increases the strength and behaves stochastically like the (1+1) EA until finding an optimum of f.*

Denote by T the runtime of the SD-(1+1) EA on f. Let f_i be the i-th fitness value of an increasing order of all fitness values of f and s_i be a lower bound for the probability that (1+1) EA finds an improvement from a search point with fitness value f_i, then

$$\mathrm{E}(T) \leq (1 + o(1)) \sum_{i=1}^{|\mathrm{Im}(f)|-1} \frac{1}{s_i}.$$

Proof. As above, E_1 denotes the probability of not finding an improvement within phase 1. As on unimodal functions the gap of all points is 1, we have by Lemma 3.3 that $\Pr(E_1) \leq \frac{1}{enR}$. This argumentation holds for each improvement that has to be found. Since at most $|\mathrm{Im}(f)| \leq R$ improving steps happen before finding the optimum, by a union bound the probability of the SD-(1+1) EA ever increasing the strength beyond 1 is at most $R\frac{1}{enR} = o(1)$, which proves the first claim of the lemma.

Regarding the second claim, we consider all fitness levels $A_1, \ldots, A_{|\mathrm{Im}(f)|}$ such that A_i contains search points with fitness value f_i. Using strength 1, the worst-case time to leave fitness level A_i is $1/s_i$ as for the (1+1) EA. Let $I_r^{(i)}$ be the number of iterations spent in phase r after the search point being selected from the fitness level i for the first time. Hence, for each fitness level i, we bound $I_1^{(i)}$ from above by the waiting time on the fitness level for the (1+1) EA, which is given by $1/s_i$, and for $r > 1$, we bound $I_r^{(i)}$ from above by considering the probability of missing the improvement for the strengths less than r with the

maximum iterations that can be spent in phase r, similarly to the proof of Theorem 3.4; formally, let

$$I^{(i)} = \sum_{r=1}^{\lceil n/2 \rceil - 1} I_r^{(i)} + I_{n/2}^{(i)}.$$

In other words, $I^{(i)}$ is the number of all iterations spent to leave the fitness level i since observing the first search point in the fitness level. we have

$$\mathrm{E}(T) = \sum_{i=1}^{|\mathrm{Im}(f)|-1} \mathrm{E}\left(I^{(i)}\right).$$

When the strength is 1, the algorithm makes progress in $1/s_i$ steps in expectation where the current fitness level is i, i. e., $\mathrm{E}\left(I_1^{(i)}\right) \leq 1/s_i$. For strengths r larger than 1, we estimate $\mathrm{E}\left(I_r^{(i)}\right)$ similarly to the analogous part in the proof of Theorem 3.4. The algorithm does not make progress with strengths less than r with probability $\Pr(U_r)$, and the number of iterations with strength $r < n/2$ is at most the threshold value, and for $r = n/2$, is at most 2^n iterations in expectation, which is less than $(n/r)(n/(n-r))^{n-r}\ln(enR)$. So, for all strengths $r > 1$, we have

$$\mathrm{E}\left(I_r^{(i)}\right) \leq \Pr(U_r) \cdot \left(\frac{n}{r}\right)^r \left(\frac{n}{n-r}\right)^{n-r} \ln(enR).$$

Via Lemma 3.3, we have $\Pr(U_r) < \prod_{j=1}^{r-1} \Pr(E_j) < (enR)^{1-r}$. Then, we have

$$\begin{aligned}\mathrm{E}\left(I_r^{(i)}\right) &\leq (enR)^{1-r} \cdot \left(\frac{n}{r}\right)^r \left(\frac{n}{n-r}\right)^{n-r} \ln(enR) \\ &= (enR)^{1-r} \cdot \left(\frac{n}{r}\right)^r \left(1 + \frac{r}{n-r}\right)^{n-r} \ln(enR).\end{aligned}$$

Since $(1 + r/(n-r))^{n-r} \leq e^r$ through the inequality $1 + x \leq e^x$ for all $x \in \mathbb{R}$, the last expression is less than

$$e \cdot (nR)^{1-r} \cdot \left(\frac{n}{r}\right)^r \ln(enR) \leq \frac{en\ln(enR)}{R} = o(1/n),$$

where $r \geq 2$ and $R \geq n^3$.

Hence for each fitness level and strength larger than 1 we have at most $o(1/n)$ extra iterations in expectation, which results in at most $o(|\mathrm{Im}(f)|)$ extra iterations in expectation for the total optimization time.

Altogether, we have

$$\begin{aligned}\mathrm{E}(T) &= \sum_{i=1}^{|\mathrm{Im}(f)|-1} \mathrm{E}\left(I^{(i)}\right) \leq \sum_{i=1}^{|\mathrm{Im}(f)|-1} \frac{1}{s_i} + o(|\mathrm{Im}(f)|) \\ &\leq (1+o(1)) \sum_{i=1}^{|\mathrm{Im}(f)|-1} \frac{1}{s_i},\end{aligned}$$

where we have $1/s_i \geq 1$, resulting in $\sum_{i=1}^{|\mathrm{Im}(f)|-1} \frac{1}{s_i} \geq |\mathrm{Im}(f)| - 1$. □

3.3.2 Analysis on JUMP

It is well known that strength 1 for the (1+1) EA leads to an expected runtime of $\Theta(n^m)$ on $\textsc{Jump}_m$ if $m \geq 2$ [DJW02]. The asymptotically dominating term comes from the fact that m bits must flip simultaneously to leave the local optimum at $n-m$ one-bits. To minimize the waiting time for such an escaping mutation, the mutation rate m/n is optimal [DLMN17], leading to an expected time of $(1+o(1))(n/m)^m(1-m/n)^{m-n}$ to optimize JUMP, which is $\Theta((en/m)^m)$ for $m = O(\sqrt{n})$. However, a static rate of m/n cannot be chosen without knowing the gap size m. Therefore, different heavy-tailed mutation operators have been proposed for the (1+1) EA [DLMN17, FQW18], which most of the time choose strength 1 but also use strength r, for arbitrary $r \in \{1, \dots, n/2\}$ with at least polynomial probability. This results in optimization times on JUMP of $\Theta((en/m)^m \cdot p(n))$ for some small polynomial $p(n)$ (roughly, $p(n) = \omega(\sqrt{m})$ in [DLMN17] and $p(n) = \Theta(n)$ in [FQW18]). Similar polynomial overheads occur with hypermutations as used in artificial immune systems [COY18]; in fact such overheads cannot be completely avoided when using random choices of the mutation strength, as proved in [DLMN17]. We also remark that JUMP can be optimized faster than $O((en/m)^m)$ if crossover is used [WVHM18, RA19], by simple estimation-of-distribution algorithms [Doe19] or specific black-box algorithms [BDK16]. In addition, in [ADK20], the expected optimization time of $n^{(m+1)/2}e^{O(m)}m^{-m/2}$ is shown for the $(1+(\lambda,\lambda))$ GA to optimize JUMP with $2 < m < n/16$ and a carefully set algorithm parameter depending on m; with a heavy-tailed version of the algorithm eliminating this parameter, almost matching upper bounds can be achieved [AD20]. All of this is outside the scope of this study that concentrates on mutation-only algorithms.

We now state our main result, implying that the SD-(1+1) EA achieves an asymptotically optimal runtime on $\textsc{Jump}_m$ for $m = o(n/\ln n)$, hence being faster than the heavy-tailed mutations mentioned above. Recall that this does not come at a significant extra cost for simple unimodal functions like ONEMAX according to Lemma 3.5.

Theorem 3.6. *Let $n \in \mathbb{N}$. For all $2 \leq m = o(n/\ln n)$, the expected runtime $\mathrm{E}(T)$ of the SD-(1+1) EA with $R = n^3$ on $\textsc{Jump}_m$ satisfies*

$$\left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} (1 - o(1)) \leq \mathrm{E}(T) \leq \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} (1 + o(1)).$$

Proof. It is well known that the (1+1) EA with mutation rate $1/n$ finds the optimum of the n-dimensional OneMax function in an expected number of at most $O(n \ln n)$ iterations [Weg02].

Until reaching the local optimum consisting of all points of $n-m$ one-bits, Jump is equivalent to OneMax; hence, according to Lemma 3.5, the expected time until SD-(1+1) EA reaches the local optimum is at most $O(n \ln n)$ (noting that this bound was obtained via the fitness level method with $s_i = e^{-1}/(n-i)$ as minimum probability for leaving the set of search points with i one-bits).

Every local optimum x with $n - m$ one-bits satisfies $\mathrm{gap}(x) = m$ according to the definition of Jump. Thus, using the upper bound stated in Theorem 3.4, the algorithm finds the global optimum from the local optimum within expected time at most

$$\left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} (1 + o(1)).$$

This dominates the expected time of the algorithm before the local optimum.

For the lower bound, we first claim that with probability at least $1 - o(1)$, the local optimum is reached. Let X be the number of one-bits in the initial random search point. Then, using Chernoff's bounds [DD20, Subsection 1.10], for $m = o(n/\ln n)$, $\Pr(X \leq n - m)$, i. e., the probability that the initial search point is at a distance greater than m from the optimum, is $1 - o(1)$. If this holds, as long as it has less than $n - m$ one-bits, the algorithm does not increase the strength and behaves like the (1+1) EA with mutation rate $1/n$ using Lemma 3.5 with probability $1 - o(1)$. If this also happens, there are at most $O((n - m + 1) \cdot n \ln(enR)) = O\left(n^2 \ln(enR)\right)$ iterations in expectation before reaching a local optimum. The probability of reaching the global optimum is at most $(1/n)^{m+1}$ in each iteration before reaching the local optimum. Thus, the probability of reaching the global optimum before a local optimum is at most $O\left(n^{1-m} \ln(enR)\right) = o(1)$ for $m \geq 2$ using union bounds. Altogether, with probability at least $1 - o(1)$, the local optimum is reached in a run.

From a search point in the local optimum, the expected number of iterations for finding the global optimum is at least $p^{-m}(1-p)^{-(n-m)}$ for any mutation rate p. Using the same arguments as in the analysis of the (1+1) EA on Jump in [DLMN17], since $\frac{m}{n}$ is the unique minimum point in the interval $[0, 1]$, the

expected iterations to find an improvement is at least $\left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m}$. Considering the probability of reaching the local optimum and using the law of total probability, we achieve the lower bound

$$\left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} (1 - o(1)). \qquad \square$$

It is easy to see (similarly to the analysis of Theorem 3.6) that for all $m = O(n)$, the expected runtime $\mathrm{E}(T)$ of the SD-(1+1) EA on $\textsc{Jump}_m$ is at most $L_{x,m} + O(n \ln n)$, where $L_{x,m}$ is defined in Theorem 3.4.

3.3.3 General Bounds

The Jump function only has one local optimum that usually has to be overcome on the way to the global optimum. We generalize the previous analysis to functions that have multiple local optima of possibly different gap sizes, using the canonical partition of the search space into fitness levels. As a use case, we can asymptotically recover the expected runtime on the LeadingOnes function in Corollary 3.8.

Theorem 3.7. *Given a pseudo-Boolean fitness function f having k different fitness values $f_1 < \cdots < f_k$, let $F_i := \{x \in \{0,1\}^n \mid f(x) = f_i\}$, where $i \in \{1, \ldots, k-1\}$, be the set of all search points in the i-th non-optimal fitness level. Let $g_i = \mathrm{gap}(x_i^*)$ for an arbitrary but fixed $x_i^* \in F_i$, noting that g_i is identical for all $x \in F_i$, and let $L_i = L_{x_i^*, g_i}$ with $L_{x_i^*, g_i}$ as defined in Theorem 3.4.*

Then the expected runtime of the SD-(1+1) EA on f is at most

$$\mathrm{E}(T) \leq \sum_{i=1}^{k-1} L_i.$$

Proof. In order to find an optimum point, each non-optimal fitness level has to be left at most once towards a search point of strictly higher fitness. For any $i \in \{1, \ldots, k-1\}$ and any $x \in F_i$ as the first search point reached in level i, the expected time to find a strictly better search point is $\mathrm{E}(T_x) \leq L_{x_i^*, g_i} = L_i$ according to Theorem 3.4, noting again that $\mathrm{gap}(x) = g_i$ for all $x \in F_i$. Since the strength r of the SD-(1+1) EA is reset to 1 after each improvement, the total expected optimization time is bounded by the sum of the L_i as suggested. $\square$

Corollary 3.8. *The expected runtime of the SD-(1+1) EA with $R = n^3$ on* LeadingOnes *is at most $O(n^2)$.*

Proof. On LeadingOnes, all n non-optimal fitness levels contain points of gap size 1 only. Hence, according to Theorem 3.7, the expected runtime is $O(n^2)$. □

Corollary 3.8 follows also from Lemma 3.5 since LeadingOnes is unimodal and the $O(n^2)$ bound for the (1+1) EA can be inferred via the fitness level method with improvement probabilities $s_i \geq 1/(en)$. The alternative proof given here does not use the improvement probabilities from fitness levels explicitly but arguably, these probabilities implicitly influence the bounds L_i.

We finally specialize Theorem 3.7 into a result for the well-known, multimodal Trap function [DJW02] that is identical for OneMax except for the all-zeros string that has optimal fitness $n + 1$. We also note that Trap is isomorphic to Jump_n; however, Theorem 3.6 does not apply for $m = n$.

We obtain a bound of $O(2^n n \ln n)$ for the runtime of the SD-(1+1) EA on Trap instead of the $\Theta(n^n)$ bound for the classical (1+1) EA. We note that our result is close to the 2^n bound that would be obtained by uniform search and superior to the bound for the fast GA with $\beta > 1$ from [DLMN17] optimizing this function in $O(2^n n^\beta)$.

Corollary 3.9. *The expected runtime of SD-(1+1) EA with $R = n^3$ on* Trap *is at most $O(2^n n \ln n)$.*

Proof. On Trap, there are $n - 1$ non-optimal fitness levels containing points of gap size 1 only and one non-optimal level of gap size n. So according to Theorem 3.7, the expected runtime is $O\big(n^2 + 2^n n \ln n\big) = O(2^n n \ln n)$. □

3.4 An Example Where Self-Adaptation Fails

While our previous analyses have shown the benefits of the self-adjusting scheme, in particular highlighting stagnation detection on multimodal functions, it is clear that our scheme also has limitations. In this section, we present an example of a pseudo-Boolean function where stagnation detection does not help to find its global optimum in polynomial time; moreover, the function is hard for other self-adjusting schemes since measuring the number of successes does not hint on the location of the global optimum. In fact, the function demonstrates a more general effect where the behavior is very sensitive with respect to the choice of the mutation probability. More precisely, a plain (1+1) EA with mutation probability $1/n$ with overwhelming probability gets stuck in a local optimum

from which it needs exponential time to escape while the (1+1) EA with mutation probability $2/n$ and also above finds the global optimum in polynomial time with overwhelming probability. Since the function is unimodal except at the local optimum, our self-adjusting (1+1) EA with stagnation detection fails as well.

To the best of our knowledge, a phase transition with respect to the mutation probability where an increase by a small constant factor leads from exponential to polynomial optimization time has been unknown in the literature of runtime analysis so far and may be of independent interest. We are aware of opposite phase transitions on monotone functions [Len18] where increasing the mutation rate is detrimental; however, we feel that our function and the general underlying construction principle are easier to understand than these specific monotone functions.

The construction of our function, called NeedHighMut, is based on a general principle that was introduced in [Wit03] to show the benefits of populations and was subsequently applied in [JW04] to separate a coevolutionary variant of the (1+1) EA from the standard (1+1) EA. Section 5 of the latter paper also beautifully describes the general construction technique that involves creating two differently pronounced gradients for the algorithms to follow. Further applications are given in [Wit06] and [Wit08] to show the benefit of populations in elitist and non-elitist EAs. Also [RLY09] use very similar construction technique for their Balance function that is easier to optimize in frequently changing than slowly changing environments; however, they did not seem to be aware that their approach resembles earlier work from the papers above.

We now describe the construction of our function NeedHighMut. The crucial observation is that strength 1 (i. e., probability $p = 1/n$) makes it more likely to flip exactly one specific bit than strength 2 – in fact strength 1 is asymptotically optimal since the probability of flipping one specific bit is $p(1-p)^{n-1} \approx pe^{-pn}$, which is maximized for $p = 1/n$. However, to flip specific two bits, which has probability $p^2(1-p)^{n-2} \approx p^2e^{-pn}$, the choice $p = 2/n$ is asymptotically optimal and clearly better than $1/n$. Now, given a hypothetical time span of T, we expect approximately $T_1(p) \coloneqq Tpe^{-p/n}$ specific one-bit and $T_2(p) \coloneqq Tp^2e^{-p/n}$ specific two-bit flips. Assuming the actual numbers to be concentrated and just arguing with expected values, we have $T_1(1/n) \gg nT_2(1/n)$ but $nT_2(2/n) \gg T_1(2/n)$, i. e., there will be (after scaling with n) considerably more two-bit flips at strength 2 than at strength 1 and considerably less 1-bit flips. The fitness function will account for this. It leads to a trap at a local optimum if a certain number of one-bit flips is exceeded before a certain minimum number of two-bit flips has happened; however, if the number of one-bit flips is low enough before the minimum number of two-bit flips has been reached, the process is on track to the global optimum.

We proceed with the formal definition of NeedHighMut, making these ideas precise and overcoming technical hurdles. Since we have at most n specific one-bit flips but a specific two-bit flip is already by a factor of $O(1/n)$ less likely than a one-bit flip, we will work with two-bit flips happening in small blocks of size $\sqrt[4]{n}$, leading to a probability of roughly $n^{-3/2}$ for a two-bit flip in a block. In the following, we will imagine a bit string x of length n as being split into a prefix $a := a(x)$ of length $n-m$ and a suffix $b := b(x)$ of length m, where m still has to be defined. Hence, $x = a(x) \circ b(x)$, where $\circ$ denotes the concatenation.

The prefix $a(x)$ is called *valid* if it is of the form 1^i0^{n-m-i}, i. e., i leading ones and $n-m-i$ trailing zeros. The prefix fitness $\textsc{pre}(x)$ of a string $x \in \{0,1\}^n$ with valid prefix $a(x) = 1^i0^{n-m-i}$ equals just i, the number of leading ones. The suffix consists of $\lceil \frac{2}{3}\xi\sqrt{n} \rceil$, where $\xi \geq 1$ is a parameter of the function, consecutive blocks of $\lceil n^{1/4} \rceil$ bits each, altogether $m \leq \xi\frac{2}{3}n^{3/4} = o(n)$ bits. Such a block is called *valid* if it contains either 0 or 2 one-bits; moreover, it is called *active* if it contains 2 and *inactive* if it contains 0 one-bits. A suffix where all blocks are valid and where all blocks following the first inactive block are also inactive is called valid itself, and the suffix fitness $\textsc{suff}(x)$ of a string x with valid suffix $b(x)$ is the number of leading active blocks before the first inactive block. Finally, we call a string $x \in \{0,1\}^n$ valid if both its prefix and suffix are valid.

Our final fitness function is a weighted combination of $\textsc{pre}(x)$ and $\textsc{suff}(x)$. We define for $x \in \{0,1\}^n$, where $x = a \circ b$ with the above-introduced a and b,

$$\textsc{NeedHighMut}_\xi(x) := \begin{cases} n^2\textsc{suff}(x) + \textsc{pre}(x) & \text{if } \textsc{pre}(x) \leq \frac{9(n-m)}{10} \wedge x \text{ valid}, \\ n^2\lceil \frac{2}{3}\xi\sqrt{n} \rceil + \textsc{pre}(x) + \textsc{suff}(x) - n - 1 & \text{if } \textsc{pre}(x) > \frac{9(n-m)}{10} \wedge x \text{ valid}, \\ -\textsc{OneMax}(x) & \text{otherwise.} \end{cases}$$

We note that all search points in the second case have a fitness of at least $n^2m - n - 1$, which is bigger than $n^2(m-1)+n$, an upper bound on the fitness of search points that fall into the first case without having m leading active blocks in the suffix. Hence, search points x where $\textsc{pre}(x) = n - m$ and $\textsc{suff}(x) = \lceil \frac{2}{3}\xi\sqrt{n} \rceil$ represent local optima of second-best overall fitness. The set of global optima equals the points where $\textsc{pre}(x) = 9(n-m)/10$ and $\textsc{suff}(x) = \lceil \frac{2}{3}\xi\sqrt{n} \rceil$, which implies that $(n-m)/10 = \Omega(n)$ bits have to be flipped simultaneously to escape from the local towards the global optimum.

The parameter $\xi \geq 1$ controls the target strength that allows the algorithm to find the global optimum with high probability. In the simple setting $\xi = 1$, strength 1 usually leads to the local optimum first while strengths above 2 usually lead directly to the global optimum. Using larger ξ increases the threshold

for the strength necessary to find the global optimum instead of being trapped in the local one.

We now formally show with respect to different algorithms that NEEDHIGHMUT is challenging to optimize without setting the right mutation probability in advance. We start with an analysis of the classical (1+1) EA, where we for simplicity only show the negative result for $p = 1/n$ even though it would even hold for ξ/n.

Theorem 3.10. *Consider the plain (1+1) EA with mutation probability p on* NEEDHIGHMUT_ξ *for a constant $\xi \geq 1$. If $p = 1/n$ then with probability $1 - 2^{-\Omega(n)}$, its optimization time is $n^{\Omega(n)}$. If $p = (c\xi)/n$ for any constant $c \geq 2$ then the optimization time is $O(n^2)$ with probability $1 - 2^{-\Omega(n^{1/3})}$.*

Proof. It is easy to see (similarly to the analysis of the SUFSAMP function from [JJW05]) that the first valid search point (i. e., search point of non-negative fitness) has both PRE- and SUFF-value of at most $n^{1/3}$ with probability $2^{-\Omega(n^{1/3})}$. This follows from the fact that the function is symmetric on invalid search points and that from each level set of i one-bits, only $O(1)$ search points are valid. In the following, we tacitly assume that we have reached a valid search point of the described maximum PRE- and SUFF-value and note that this changes the required number of improvements to reach local or global maximum only by a $1 - o(1)$ factor. For readability this factor will not be spelt out any more.

We prepare the main analysis by bounding the probability of a mutation being accepted after a valid search point has been reached. Even if a mutation changes up to $o(n)$ consecutive bits of the prefix or suffix, it must maintain $n - o(n)$ prefix bits in order to result in a valid search point. Hence, the probability of an accepted step at mutation probability c/n (valid for any constant c) is at most $(1 - c/n)^{n-m-o(n)} = (1 + o(1))e^{-c}$. Steps flipping $\Omega(n)$ consecutive bits have probability $n^{-\Omega(n)}$ and are subsumed by the failure probabilities stated in this theorem. Clearly, the probability of an accepted step is at least $(1 - 1/n)^n = (1 - o(1))e^{-c}$.

Using this knowledge of accepted steps, we shall now prove the statement for $p = 1/n$. The probability of improving the PRE-value is at least e^{-1}/n since it is sufficient to flip the leftmost zero of the prefix to 1. In a phase of length $\frac{11}{10}emn$ steps, there are at least m prefix-improving mutations with probability $1 - 2^{-\Omega(n)}$ by Chernoff bounds. All these mutations improve the function value and are accepted unless the SUFF-value increases to m before the PRE-value exceeds $9n/10$.

The probability of improving the leftmost inactive block of the suffix by 1 is at most $\binom{n^{1/4}}{2}\frac{1}{n^2}e^{-1}(1+o(1)) \leq (1+o(1))(e^{-1}/2)n^{-3/2}$ since it is necessary to flip two zeros into ones and to have an accepted mutation. By the same reasoning, steps that activate $k = o(n)$ blocks simultaneously have a probability of at most $(1 + o(1))(e^{-1}/2n^{-3/2})^k$. We consider a phase of $s := \frac{11}{10}emn$ steps and bound the number of accepted steps increasing the SUFF-value by k by applying Chernoff bounds since this number is bounded by a binomial distribution with parameter s and $p_k := (1+o(1))(e^{-1}/2n^{-3/2})^k$. Hence, the number of accepted steps activating one suffix block in $\frac{11}{10}emn \leq \frac{11}{10}en^2$ steps is less than $\frac{3}{5}\sqrt{n}$ with probability $1 - 2^{-\Omega(\sqrt{n})}$. The expected number of accepted steps activating $k \geq 2$ suffix blocks is already $O(n^{-1/2})$, and by Chernoff bounds the actual number is at most $n^{1/3}$ with probability $1-2^{-\Omega(n^{1/3})}$. Hence, by a union bound over $k \in \{2, \ldots, n^{1/9}\}$, the steps adding more than one valid suffix block increase the SUFF-value by at most $n^{1/3+1/9} = n^{4/9}$ with probability $1-2^{-\Omega(n^{1/3})}$. Steps adding $k > n^{1/9}$ valid blocks have probability $O(2^{-\Omega(n^{1/9})})$ and are subsumed by the failure probability. If none of the failure events occurs, the total increase of the SUFF-value is at most $\frac{3}{5}\sqrt{n}+n^{4/3} < \frac{2}{3}\sqrt{n}$. Also, with probability $1-2^{-\Omega(\sqrt{n})}$, the PRE-value decreases by altogether at most $O(\sqrt{n})$ in the $O(\sqrt{n})$ mutations that improve the suffix, which can be subsumed in a lower-order term in the above analysis of PRE-improving steps.

Altogether, with overwhelming probability $1 - 2^{-\Omega(n^{1/9})}$ the prefix is optimized before the suffix. The probability of reaching the global optimum from the local one is $n^{-\Omega(n)}$ since it is necessary to flip $m/10$ bits simultaneously to leave the local optimum. In a phase of $n^{c'n}$ steps for a sufficiently small constant c' this does not happen with probability $1 - 2^{-\Omega(n)}$. This completes the proof of the statement for the case $p = 1/n$.

For $p = c/n$, where $c \geq 2\xi$, we argue similarly with inverted roles of prefix and suffix. The probability of activating a block in the suffix is at least $(1 - o(1))((c^2/2)e^{-c}n^{-3/2})$ now. In a phase of $(7/4)\xi(e^2/c^2)mn$ steps, we expect $(7/8)\xi\sqrt{n}$ activated blocks and with overwhelming probability we have at least $(2/3)\xi\sqrt{n}$ such blocks. The probability of improving the PRE-value by k is only $(1+o(1))ce^{-c}/n^k$, amounting to a total expected number of improvements by k of at most $(1+o(1))(7/4)(e^2/e^c)(\xi/c)mn^{1-k} \leq (1+o(1))(7/4)(\xi/c)n^{2-k} \leq (1 + o(1))(7/8)n^{2-k}$ since $c \geq 2\xi \geq 2$, and, using similar Chernoff and union bounds as above, the probability of at least $(9/10)m$ PRE-improving steps in the phase is $2^{-\Omega(n^{1/3})}$. □

The previous analysis can be transferred to the SD-(1+1) EA with stagnation detection, showing that this mechanism does not help to increase the success probability significantly compared to the plain (1+1) EA with $p = 1/n$. The

proof shows that the SD-(1+1) EA with high probability does not behave differently from the (1+1) EA. The only major difference is visible after reaching the local optimum of NeedHighMut, where stagnation detection kicks in. This results in the bound $2^{\Omega(n)}$ in the following theorem, compared to $n^{\Omega(n)}$ in the previous one.

Theorem 3.11. *With probability at least $1 - O(1/n)$, the SD-(1+1) EA with $R \geq n$ needs at least $2^{\Omega(n)}$ steps to optimize* NeedHighMut_ξ *for $\xi \geq 1$.*

Proof. We follow the analysis of the case $p = 1/n$ from the proof of Theorem 3.10. In a phase of $\frac{11}{10}emn$ steps, there are at least m pre-improving mutations (having probability at least $1/(en)$ each) with probability $1 - 2^{-\Omega(n)}$ by Chernoff bounds. For each of these improving mutations, the probability that it does not happen within the threshold of $en\ln(enR) \geq en\ln(n^2)$ iterations is at most $(1 - 1/(en))^{en\ln(en^2)} \leq 1/n^2$. By a union bound, the probability that at least one of the mutations does not happen within this number of iterations is at most $1/n$. Together with the analysis of the number of suff-increasing mutations, this means that the strength stays at 1 until the local optimum is reached, and that the local optimum is reached first, with probability at least $1 - O(1/n)$.

Leaving the local optimum requires a mutation flipping at least $m/10 = \Omega(n)$ bits simultaneously. As already analyzed in Theorem 3.4, even at optimal strength this requires $2^{\Omega(n)}$ steps with probability $1 - 2^{-\Omega(n)}$. Taking a union bound over all failure probabilities completes the proof. □

Finally, we also show that the self-adaptation scheme of the SASD-(1+λ) EA does not help to concentrate the mutation rate on the right regime for NeedHighMut_ξ if ξ is a sufficiently large constant and λ is not too large. This still applies in connection with stagnation detection.

Theorem 3.12. *Let ξ be a sufficiently large constant and assume $\lambda = o(n)$ and $\lambda = \omega(1)$. Then with probability at least $1 - O(1/n)$, the SASD-(1+λ) EA with stagnation detection (Algorithm 9) needs at least $2^{\Omega(n)}/\lambda$ generations to optimize* NeedHighMut_1.

The proof of this theorem uses more advanced techniques, more precisely Theorem 3.2 to analyze the distribution of mutation strength in the offspring over time. This technique allows us that only a small constant fraction of steps uses strength that are more beneficial for the suffix than the prefix.

Proof. The idea is to show that the strength has a drift towards its minimum and then apply Theorem 3.2 to bound the number of steps at which a mutation

rate is taken that could be beneficial. Then, since most of the steps use small mutation rates, the prefix is optimized before the suffix with high probability and a local optimum is reached.

To make these ideas precise, we pick up and extend the analysis of the acceptance and improvement probabilities from Theorem 3.10. Hence (with respect to the creation of a single offspring):

- The probability of creating a valid offspring at strength $r = o(n)$ is $(1 \pm o(1))e^{-r}$ since only $o(n)$ bits flip with probability $1-e^{-\omega(r)}$ and $(1-o(1))n$ bits have to be preserved (not flipped) with probability $1 - 2^{-\Omega(n)}$. At strength $r = \Omega(n)$ the probability is $2^{-\Omega(n)}$ which can be seen as follows: the probability of improving the PRE-value by $m/2$ is $2^{-\Omega(n)}$ since $m/2$ consecutive bits have to be set to 1; otherwise, at least $m/2$ bits must be preserved, which has probability at most $e^{-\Omega(r)}$.
- the probability of improving the PRE-value by $k=o(n)$ is $(1\pm o(1))(\frac{r}{n})^k e^{-r}$.
- the probability of improving the SUFF-value by $k=o(n)$ is $(1\pm o(1))(\frac{r}{n})^k e^{-r}$.

Clearly, the probability that at least one out of λ offspring is improving the function value is at most λ times as large. Since we have $\lambda = o(n)$ offspring and each improvement has probability $p_i = O(1/n)$, the probability of having at least one improving offspring is at least $1-(1-p_i)^\lambda = 1-(1-(1-o(1))\lambda p_i$, hence also by a factor at least $(1-o(1))\lambda$ larger.

Using these bounds on the acceptance and improvement probabilities, we now use ideas similar to the analysis of the near region in [DGWY19] to show a drift of the strength towards small values. We distinguish between three cases.

Case 1: $r_t \leq (\ln \lambda)/4 =: L$: then the probability of creating a copy of the parent at strength $r_t/2$ is at least $(1-o(1))e^{-(\ln\lambda)/8} = (1-o(1))\lambda^{-1/8}$. This probability is by a factor $(1-o(1))e^4$ smaller at strength $2r_t$. Using Chernoff bounds and exploiting $\lambda = \omega(1)$ we have that with probability $1-o(1)$, the number of copies produced at strength $r_t/2$ is by a constant factor larger than the one produced at strength $2r_t$, and there is at least one copy produced from strength $r_t/2$. Due to the uniform choice of the individual adjusting the strength in case of ties, the probability of increasing the strength is at most $1/2-\epsilon$ for some constant $\epsilon > 0$.

Case 2: $r_t \geq 4\ln\lambda =: U$: then with probability $1-o(1)$, all offspring are invalid in prefix or suffix and therefore worse than the parent. The fitness function is $-$OneMax in this case. Now, since the minimum number of bits flipped at

strength $2r_t$ is with probability $1 - o(1)$ larger than the maximum number of bits flipped at strength $r_t/2$ (using Chernoff and union bounds), with probability $1 - o(1)$ an offspring produced from strength $r_t/2$ has best fitness and adjusts the strength. Hence, the probability of increasing the strength is at most $1/2 - \epsilon$ again.

Case 3: $L \leq r_t \leq U$: here we only know that the probability of decreasing the strength is at least $1/4$ due to the random steps of the SASD-(1+λ) EA. However, a constant number of such decreasing steps is enough to reach strength at most L from the smallest possible strength above U. To accommodate for this, we shall define a potential function with an exponentially falling slope in the range $[L, U]$ like in [DGWY19]. Since the change of r_t is of a multiplicative kind, our potential function will consider the logarithmized state $\log_2(r_t)$, which lives on the non-negative integers and changes by an absolute value of at most 1 in every step. In more detail, we define

$$g(r_t) \coloneqq \begin{cases} \log_2(r_t) & \text{if } r_t \leq L, \\ \log_2(L) + \sum_{i=1}^{\lceil \log_2(r_t/L) \rceil} 2^{-i} & \text{if } L < r_t < U, \\ \log_2(L) + \sum_{i=1}^{\log_2(U/L)+1} 2^{-i} + 2^{-\log_2(U/L)-1} \log_2(r_t) & \text{otherwise,} \end{cases}$$

assuming that L and U have been rounded down and up to the closest power of 2, respectively.

To apply Theorem 3.2, we have to analyze the moment-generating function of the drift of g. Therefore, we write $\Delta_t \coloneqq g(X_{t+1}) - g(X_t)$ and will bound $\mathrm{E}\left(e^{\eta \Delta_t}\right)$ for a sufficiently small constant $\eta > 0$.

In Case 1, if $r_t \leq L$ and additionally $r_t > 2$, the probability of decreasing the strength is at least $1/2 + \epsilon$ and the probability of increasing it is at most $1/2 - \epsilon$ for a sufficiently small constant $\epsilon > 0$. The g-value changes by ± 1 in this case. Hence, using $e^x \leq 1 + x + x^2/2$ as well as $e^{-x} \leq 1 - x + x^2/2$ for $0 \leq x \leq 1$, we have

$$\mathrm{E}\left(e^{\eta \Delta_t} \mid \mathcal{F}_t; 2 < r_t \leq L\right) = e^{-\eta}\left(\frac{1}{2} + \epsilon\right) + e^{\eta}\left(\frac{1}{2} - \epsilon\right) \leq 1 - 2\eta\epsilon + \eta^2 \leq \rho$$

for a constant $\rho < 1$ if η is chosen as a sufficiently small constant (depending on the constant ϵ). Similarly, considering Case 3, we have the same bounds on the probabilities for increasing and decreasing the g-value. Now the change is $\pm 2^{-\log_2(U/L)-1}$ in both cases if $r_t > U$ and even stronger in negative direction if $r_t = U$. Since $2^{-\log_2(U/L)-1} = c^*$ for a constant c^*, we have

$$\mathrm{E}\left(e^{\eta \Delta_t} \mid \mathcal{F}_t; r_t \geq U\right) = e^{-\eta c^*}\left(\frac{1}{2} + \epsilon\right) + e^{\eta c^*}\left(\frac{1}{2} - \epsilon\right),$$

which again is at most $\rho < 1$ for an appropriate choice of η.

Finally, considering Case 3, i. e., $L < r_t < U$, the probability of decreasing r_t can only be bounded from below by $1/4$ thanks to the random steps. The g-value increases by $2^{-1-\log_2(r_t/L)}$ if r_t increases and decreases by $2^{1-\log_2(r_t/L)}$ otherwise; hence the increase is by a factor of 4 larger. We obtain

$$\mathrm{E}\left(e^{\eta\Delta_t} \mid \mathcal{F}_t; L < r_t < U\right) \leq \frac{1}{4}e^{-\eta(2r_t/L)} + \frac{3}{4}e^{\eta r_t/(2L)},$$

which, writing $\eta' = \eta r_t/L$, can be bounded from above by

$$\left(1 - 2\eta' + \frac{(2\eta')^2}{2}\right)\frac{1}{4} + \left(1 + (\eta/2)' + \frac{(\eta'/2)^2}{2}\right)\frac{3}{4},$$

which is again bounded from above by the constant $\rho < 1$ if η is chosen as a sufficiently small constant; here we have used that $\log(r_t/L)$ is bounded by a constant as well.

Altogether, we have shown

$$\mathrm{E}\left(e^{\eta\Delta_t} \mid \mathcal{F}_t; r_t > 2\right) \leq \rho$$

for a constant $\rho < 1$ if η is chosen as a sufficiently small constant (depending on the constant ϵ). Similarly, given this choice of η, we immediately have

$$\mathrm{E}\left(e^{\eta\Delta_t} \mid \mathcal{F}_t; X_t \leq 2\right) \leq D$$

for a constant $D > 0$. If we choose b in Theorem 3.2 as a sufficiently large constant, we obtain, noting $a = 2$,

$$1 - \epsilon - \frac{1-\epsilon}{1-\rho}De^{\eta(a-b)} \geq \frac{9}{10}.$$

Hence, the theorem states that in a phase of length T, the number of generations where $X_t > b$ holds, is at most $T/10$ with probability $1 - 2^{-\Omega(T)}$. Let $b^* = 2^b$, i. e., the strength corresponding to $X_t = b$. We set $T := (\frac{12}{10})e^{b^*}\frac{mn}{b^*\lambda}$. Since a PRE-improving mutation has probability at least $(1-o(1))\lambda(b^*/n)e^{-b^*}$, we have an expected number of at least $(1-o(1))(27/25)m$ such mutations in the phase, and with probability $1 - 2^{-\Omega(n)}$ we have at least m such mutations by Chernoff bounds. This is sufficient to reach the local optimum unless there are at least $(2/3)\xi\sqrt{n}$ SUFF-improving mutations in the phase. Note that the choice of the constant ξ only impacts the length of the prefix in lower-order terms that vanish in O-notation.

We bound the number of SUFF-improving mutations separately for the points in time (i. e., generations) where $X_t \leq b$ and where $X_t > b$. For the first

set of time points, we note that the probability of a SUFF-improving mutation by $k \geq 1$ is at most $(1 + o(1))\lambda(2/n^{3/2})^k e^{-2}$ since the term x^2/e^{-x} takes its maximum at $x = 2$. Using similar arguments based on Chernoff and union bounds as in the proof of Theorem 3.10, we bound the total improvement of the SUFF-value in at most $T \leq (12/10)e^{b^*}n^2/(\lambda b^*)$ steps where $X_t \leq b$ by $i_1 \coloneqq (25/10)e^{b^*-2}\sqrt{n}/b^*$ with probability $1 - 2^{-\Omega(n^{1/9})}$. For the points of time where $X_t > b$ the probability of a PRE-improving mutation is maximized (up to lower-order terms) at strength b^* since the function x^2/e^{-x} is monotonically decreasing for $x > 2$. Assuming at most $(12/100)e^{b^*}n^2/b^*$ such time points (which assumption holds with probability at least $1 - 2^{-\Omega(n^2)}$), we obtain an expected number of SUFF-improving mutations by 1 of at most

$$\frac{12}{100}e^{b^*}\frac{n^2}{b^*}\frac{b^*}{n^{3/2}}e^{-b^*} = \frac{12}{100}\sqrt{n}$$

and using Chernoff and union bounds we bound the total improvement of the SUFF-value in these generations by $i_2 = (13/100)\sqrt{n}$ with $1 - 2^{-\Omega(n^{1/9})}$. Now, if we choose ξ large enough, then

$$i_1 + i_2 \leq \frac{2}{3}\xi\sqrt{n}$$

so that the prefix is optimized before the suffix with probability altogether $1 - 2^{-\Omega(n^{1/3})}$.

Together with the analysis in Theorem 3.11 for the case that the stagnation counter exceeds its threshold, this means that with probability $1 - O(1/n)$ the local optimum is reached before the global one. Again arguing in the same way as in the proof of Theorem 3.11, the time to reach the global optimum from the local one is $2^{\Omega(n)}/\lambda$ with probability $1 - 2^{-\Omega(n)}$. The sum of all failure probabilities is $O(1/n)$. □

The algorithms considered in this paper use standard-bit mutation only. It would be an interested subject to study algorithms incorporating other mutation operators (e. g., fast mutation [DLMN17]) on NEEDHIGHMUT.

3.5 Experiments

Our theoretical results are asymptotic. In this section, we show the results of the experiments we did in order to see how the different algorithms perform in practice for small n.

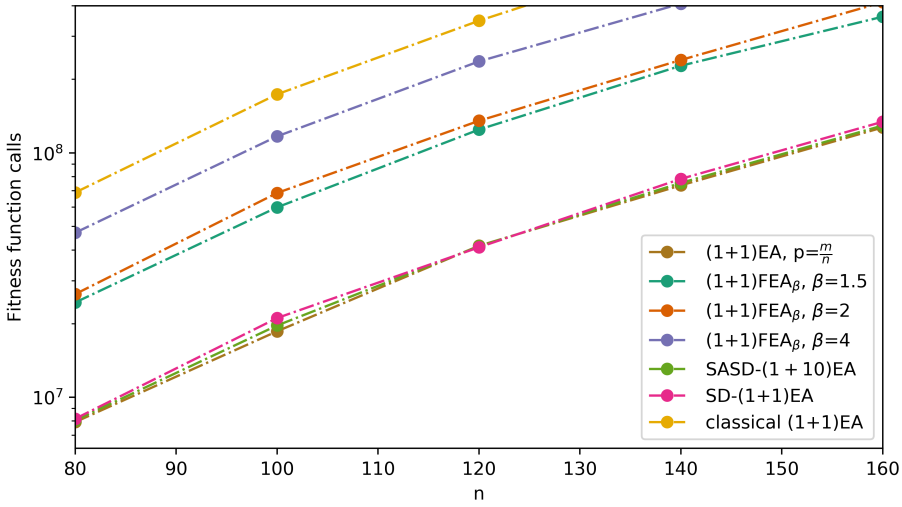


Figure 3.1: Average number of fitness calls (over 1000 runs) the mentioned algorithms take to optimize $\textsc{Jump}_4$.

In the first experiment, we ran an implementation of Algorithms 8 (i. e., SD-(1+1) EA) and 9 (i. e., SASD-(1+λ) EA) with $\lambda = 10$ on the JUMP fitness function with jump size $m = 4$ and n varying from 80 to 160. We compared our algorithms against (1+1) EA with standard mutation rate $1/n$, (1+1) EA with mutation probability m/n, and Algorithm (1+1) FEA$_\beta$ from [DLMN17] with three different values of $\beta = \{1.5, 2, 4\}$.

In Figures 3.1 and more precisely 3.2, we observe that stagnation detection technique makes the algorithm faster than the algorithms with heavy-tailed mutation operator (1+1) FEA$_\beta$. Also, Algorithm SD-(1+1) EA and SASD-(1+λ) EA perform roughly the same compared to the (1+1) EA with mutation probability $\frac{m}{n}$ even though it does not need the gap size.

In the second experiment, we ran our algorithms and the classic (1+1) EA with different mutation probabilities on $\textsc{NeedHighMut}_\xi$ with different $n = \{200, 400, 600, 800, 1000\}$ and $\xi = 3$.

The outcomes support that the theory from Section 3.4 already holds for small n. In Table 3.1, one can see that for $\xi = 3$, the (1+1) EA with $p = 6/n$ and $8/n$ is much more successful to find global optimum points than the rest of the algorithms.

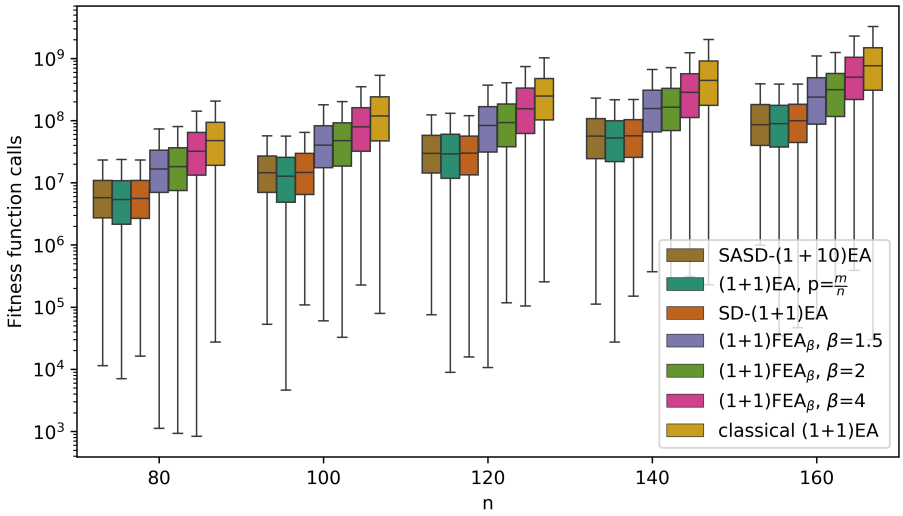


Figure 3.2: Box plots comparing number of fitness calls (over 1000 runs) the mentioned algorithms take to optimize $\textsc{Jump}_4$.

Conclusions

We have designed and analyzed self-adjusting EAs for multimodal optimization. In particular, we have proposed a module called *stagnation detection* that can be added to existing EAs without essentially changing their behavior on unimodal (sub)problems. Our stagnation detection keeps track of the number of unsuccessful steps and increases the mutation rate based on statistically significant waiting times without improvement. Hence, there is high evidence for being at a local optimum when the strength is increased.

Theoretical analyses reveal that the (1+1) EA equipped with stagnation detection optimizes the Jump function in asymptotically optimal time corresponding to the best static choice of the mutation rate. Moreover, we have proved a general upper bound for multimodal functions that can recover asymptotically runtimes on well-known example functions, and we have shown that on unimodal functions, the (1+1) EA with stagnation detection with high probability never deviates from the classical (1+1) EA. Finally, to show the limitations of the approach we have presented a function on which all of our investigated self-adjusting EAs provably fail to be efficient.

In the future, we would like to investigate our module for stagnation detection in other EAs and study its benefits on combinatorial optimization problems.

	(1+1) EA				SD-$(1+1)$EA	SASD-$(1+\ln n)$EA
n	$p=\frac{1}{n}$	$p=\frac{2}{n}$	$p=\frac{6}{n}$	$p=\frac{8}{n}$		
200	0.000	0.000	0.011	0.193	0.000	0.000
400	0.000	0.000	0.338	0.874	0.001	0.000
600	0.000	0.000	0.424	0.859	0.000	0.000
800	0.000	0.000	0.840	0.972	0.000	0.002
1000	0.000	0.000	0.807	0.979	0.000	0.001

Table 3.1: Ratio of successfully achieved global optimum where $\xi = 3$ over 1000 runs instead of getting stuck in a local optimum from which it needs exponential time to escape.

Acknowledgement

This work was supported by a grant by the Danish Council for Independent Research (DFF-FNU 8021–00260B).

Chapter 4

Paper B: Stagnation Detection with Randomized Local Search

Recently a mechanism called stagnation detection was proposed that automatically adjusts the mutation rate of evolutionary algorithms when they encounter local optima. The so-called SD-(1+1) EA introduced by Rajabi and Witt (GECCO 2020) adds stagnation detection to the classical (1+1) EA with standard bit mutation. This algorithm flips each bit independently with some mutation rate, and stagnation detection raises the rate when the algorithm is likely to have encountered a local optimum.

In this paper, we investigate stagnation detection in the context of the k-bit flip operator of randomized local search that flips k bits chosen uniformly at random and let stagnation detection adjust the parameter k. We obtain improved runtime results compared to the SD-(1+1) EA amounting to a speedup of at least $(1 - o(1))\sqrt{2\pi m}$, where m is the so-called gap size, i. e., the distance to the next improvement. Moreover, we propose additional schemes that prevent infinite optimization times even if the algorithm misses a working choice of k due to unlucky events. Finally, we present an example where standard bit mutation still outperforms the k-bit flip operator with stagnation detection.

4.1 Introduction

Evolutionary algorithms (EAs) are parameterized algorithms, so it has been a subject of ongoing research to discover how to choose their parameters best. A poor choice of a parameter may result in inefficient optimization times, and it can be challenging to find an optimal parameter setting, both from a theoretical and empirical perspective. Also, given a specific problem, there might be different scenarios during the optimization, e. g., success probabilities that decrease with the distance to the optimum. This can result in relative inefficiency of one static parameter configuration for the whole run, see, e. g., the $(1 + (\lambda, \lambda))$ GA on OneMax [DD18]. Self-adjusting mechanisms address this issue by learning acceptable or even near-optimal parameter settings on the fly. See also the survey article by [DD20] for a detailed coverage of static and non-static parameter control.

Many studies have been conducted on frameworks that adjust the mutation rate of different mutation operators on unimodal functions. For example, the above-mentioned $(1 + (\lambda, \lambda))$ GA using the 1/5-rule can adjust its mutation strength (and also its crossover rate) on the well-known OneMax function [DD18], resulting in asymptotic speedups compared to static settings. Likewise, the self-adjusting mechanism in the $(1+\lambda)$ EA with two rates proposed in [DGWY19] performs on OneMax as efficiently as the best λ-parallel unary unbiased black-box algorithm. Another approach based on reinforcement learning has been presented in [DDY16b], which adjusts the mutation strength of RLS (i. e., the k in the k-bit flip operator) on OneMax.

The self-adjusting frameworks mentioned above have been mainly analyzed on unimodal functions such as the OneMax benchmark. Generally, it is not very clear how they can suggest efficient parameter settings on multimodal functions. Since the above frameworks mainly work based on the past number of successes, they do not receive clear signals on promising parameters choices when an algorithm gets stuck in a local optimum and does not make progress for a long time. On multimodal functions, where a specific number of bits has to flip to make progress, *stagnation detection (SD)* introduced in [RW22a] can efficiently overcome local optima in some black-box optimization scenarios. This module can be added to many existing algorithms to leave local optima without any significant increase of the optimization time on unimodal (sub)problems.

To our knowledge, there are only a few other runtime analyses of self-adjusting mechanisms on multimodal functions, most notably the work by [LS11] on self-adjusting offspring population sizes that yield a speedup on, e. g., the well-known Jump function, and the one by [DL16b] on a self-adjusting mutation rate in a

non-elitist evolutionary algorithm with respect to the specifically constructed PeakedLO function.

In the broader context of mutation-based randomized search heuristics, the heavy-tailed mutation presented in [DLMN17] is able to leave a local optimum in a much more efficient time than the standard bit mutation does. Moreover, in the context of artificial immune systems [COY18] and hyperheuristics [LOW19], there are proofs that specific search operators and selection of low-level heuristics can speed up multimodal optimization compared to the classical mutation operators. Altogether, so far there are only few runtime analyses that study local optima from a general perspective and aim at designing and analyzing self-adjusting mutation operators to escape local optima.

Recent theoretical research on evolutionary algorithms in discrete search spaces mainly considers global mutations which can create all possible points in one iteration. These mutations have been functional in optimization scenarios where information about the difficulties of the local optima is not available. For example, the standard bit mutation, which flips each bit independently with a non-zero probability, can produce any point in the search space. However, local mutations can only create a fixed set of offspring points. The 1-bit flip mutation that often can be found in the Randomized Local Search algorithm (RLS) can only reach a limited number of search points, which may result in being stuck in a local optimum when using the elitist selection. Nevertheless, local mutations may outperform global mutations on unimodal functions and multimodal functions with known gap sizes. It is of special interest to use advantages of local mutations on unimodal (sub)functions additionally to overcome local optima efficiently.

This paper investigates k-bit flip mutation as a local mutation in the context of the above-mentioned stagnation detection mechanism. This mechanism detects when the algorithm is likely to be stuck in a local optimum and gradually increases the mutation strength (i. e., the number of flipped bits) to a value the algorithm needs to leave the local optimum. Similarly, we aim to show that the algorithms using k-bit flip mutation can use stagnation detection to tune the parameter k. One of the key benefits of such algorithms is using the efficiency of RLS, which performs very well on unimodal (sub)problems, without fear of infinite running time in local optima. An additional advantage of using k-bit flip mutation accompanied by stagnation detection is that it overcomes Hamming gaps of size m, i. e., search points with Hamming distance m to the closest improvement, more efficiently than global mutations (see below for a detailed calculation of the speedup). Moreover, our outcomes point out the advantages and practicability of our self-adjusting approach that makes local-mutation algorithms able to optimize functions that have been intractable to solve so far.

We propose two globally searching algorithms combining stagnation detection with local mutations. The first algorithm called SD-RLS$^{\mathrm{p}}$ gradually increases the mutation strength when the current strength has been unsuccessful in finding improvements for a significantly long time. In the most extreme case, the strength ends at n, i. e., mutations flipping all bits. With high probability, SD-RLS$^{\mathrm{p}}$ has a runtime that is by a factor of $(\frac{en}{m})^m/\binom{n}{m} \geq \sqrt{2\pi m}$ (up to lower-order terms) smaller on Jump functions with Hamming gaps of size m than the SD-(1+1) EA previously considered in [RW22a]. Although it is unlikely that the algorithm fails to find an improvement when the current strength allows this, there is a risk that this algorithm misses the "right" strength and therefore it can have infinite expected runtime. To address this, we propose a more robust algorithm called SD-RLS$^{\mathrm{r}}$ that repeatedly loops over smaller strengths than the last one tried. This results in an expected finite optimization time on all problems and only increases the typical runtime by lower-order terms compared to SD-RLS$^{\mathrm{p}}$. We also observe that the algorithms we obtain can still follow the same search trajectory as the classical RLS when one-bit flips are sufficient to make improvements. In those cases, well-established techniques for the analysis of RLS like the fitness-level method carry over to our variant enhanced with stagnation detection. This is not necessarily the case in related approaches like variable neighborhood search [HM18] and quasirandom evolutionary algorithms [DFW10], both of which employ more determinism and do not generally follow the trajectory of RLS.

We shall investigate the two suggested algorithms on unimodal functions and functions with local optima of different gap sizes, defined as the number of bits that need to be flipped to escape from the optima. Many results are obtained following the analysis of the SD-(1+1) EA [RW22a] which uses a global operator with self-adjusted mutation strength. In conclusion, globally searching algorithms with the self-adjusting local mutation seem to be the preferred alternative to the SD-(1+1) EA with global mutation. However, we will also investigate carefully chosen scenarios where the simple (1+1) EA with global mutation outperforms our algorithms with self-adjusting local mutation, despite the fact that they are globally searching and cannot get stuck in local optima.

This paper is structured as follows: in Section 4.2, we state the classical RLS algorithm and introduce our self-adjusting variants with stagnation detection; moreover, we collect important mathematical tools. Section 4.3 shows runtime results for the simpler variant SD-RLS$^{\mathrm{p}}$, concentrating on the probability of leaving local optima, while Section 4.4 gives a more detailed analysis of the variant SD-RLS$^{\mathrm{r}}$ on benchmark functions like OneMax and Jump. Section 4.5 analyzes an example function which the standard (1+1) EA with standard bit mutation can solve in polynomial time with high probability whereas the k-bit flip mutation with stagnation detection needs exponential time. Through improved upper bounds, we give in Section 4.6 indications for that our approach

may also be superior to static settings on instances of the minimum spanning tree problem. This problem and other scenarios are investigated experimentally in Section 4.7 before we finally conclude the paper.

Recent, related work. In a recent conference article, [RW21a] extend SD-RLS^p with a memory mechanism that gives improved running times for problems containing a large number of local optima like the MST problem. The present paper describes the original implementation of stagnation detection in randomized local search as originally announced at EvoCOP 2021 [RW21b].

4.2 Preliminaries

4.2.1 Algorithms

Let $f\colon \{0,1\}^n \to \mathbb{R}$ be the fitness function we want to optimize. This paper describes all algorithms and results taking the perspective that one aims to maximize the fitness function f except when the minimum spanning tree problem is considered as optimization problem (in Section 4.6). Nevertheless, on any minimization problem, we can still use the results from Sections 4.2–4.5 analogously by simply assuming that the negated fitness function is maximized.

One of the most simple randomized search heuristics studied in the literature is *randomized local search* (RLS) displayed in Algorithm 10. To the best of our knowledge, the first appearance of the name "Random(ized) Local Search" in the literature goes back to [WW05]. They also mention that the algorithm was called "random mutation hill-climbing" in [MHF93]. This heuristic starts with a random search point and then repeats mutating the point by flipping s uniformly chosen bits (without replacement) and replacing it with the offspring if it is not worse than the parent.

Algorithm 10: RLS with static strength s for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$

```
Select x uniformly at random from {0,1}^n;
for t ← 1, 2, ... do
    Create y by flipping s bit(s) in a copy of x;
    if f(y) ≥ f(x) then
        x ← y;
```

The *runtime* or the *optimization time* of a heuristic on a function f is the first point in time where a search point of maximal fitness has been created. Usually, a black-box perspective is assumed and time is measured in the number of evaluations of the objective function. Since the algorithms considered in this paper evaluate one search point in each iteration of their main loop, their runtime equals the number of iterations until an optimum is found plus 1 for the evaluation of the initial search point. Hence, their runtime is simply the smallest value of the iteration counter t such that an optimum is evaluated plus 1. Very often the expected runtime, i. e., the expected value of this time, is analyzed.

We use the definition of *unimodal functions* provided in [DJW02]. A function is unimodal if and only if there is only one local maximum, where a local maximum is defined as a search point such that no Hamming neighbor has a larger fitness value. An immediate result of this definition is that on unimodal functions, any search point except the global optimum has a better point in its Hamming neighborhood.

Theoretical research on evolutionary algorithms often studies simple unimodal benchmark problems, such as

$$\textsc{OneMax}(x) \coloneqq \|x\|_1,$$

where $\|x\|_1$ denotes the number of one-bits in the bit string x, and multimodal functions, like the $\textsc{Jump}_m$ function [DJW02] with jump size m defined as follows:

$$\textsc{Jump}_m(x) \coloneqq \begin{cases} m + \|x\|_1 & \text{if } \|x\|_1 \leq n - m \text{ or } \|x\|_1 = n, \\ n - \|x\|_1 & \text{otherwise.} \end{cases}$$

These simple, well-structured functions can serve as building blocks of more complicated problems.

The mutation used in RLS is a local mutation as it only produces a limited number of offspring. This mutation, which we call s-bit flip (in the introduction, we used the classical name k-bit flip), flips exactly s bits randomly chosen from the bit string of length n, so for any point $x \in \{0,1\}^n$, RLS with strength s can just sample from $\binom{n}{s}$ possible points. As a result, the s-bit flip mutation is often more efficient compared to global mutations when we know the difficulty of making progress since the algorithm just looks at a certain part of the search space.

To be more precise, we define two terms describing the difficulty of a local optimum in terms of the distance from improving solutions. Let the *individual gap* of $x \in \{0,1\}^n$ be the minimum Hamming distance of x to points with strictly larger fitness function value, i. e.,

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

By the *fitness level* of x, we mean all the search points with fitness value $f(x)$. Since the algorithm might replace the current search point with another search point in its fitness level before creating a strict improvement, we need to take into account the individual gap of all the search points in the fitness level. Hence, 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.,

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

If the algorithm creates a point at Hamming distance $\text{IndividualGap}(x)$ from the current search point x, with positive probability a strict improvement can be found. We also note that $\text{FitnessLevelGap}(x) = 1$ is allowed, so the definition also covers search points that are not local optima. As long as no strict improvement is made, the FitnessLevelGap remains the same, although the current search point might be replaced with another search point in the same fitness level.

However, in general, quantities such as $\text{FitnessLevelGap}(x)$ are unknown, and benefiting from domain knowledge to determine the most promising strength is not always feasible in the perspective of black-box optimization. Therefore, despite the advantages of the s-bit flip operator, global mutations, which can produce any point in the search space with positive probability, have been used in the literature frequently. For example, the well-known (1+1) EA [DJW02] uses the same framework as Algorithm 10 except for the mutation operator, which follows the so-called standard bit mutation where each bit is flipped independently with probability p. Hence, standard bit mutation implicitly uses the binomial distribution with parameters n and p to determine how many bits must flip. Consequently, even if the algorithm uses strength 1 (i. e., mutation rate $p = 1/n$), with a positive probability, the algorithm can escape from any local optimum.

We now study the search and success probability of Algorithm 10 and its relation to stagnation detection more closely. With similar arguments as presented in [RW22a], if the individual gap of the current search point is 1, then the algorithm makes a strict improvement with probability $1/n$ at strength 1, and the probability of not finding it in $n \ln R$ steps is at most $(1-1/n)^{n \ln R} \leq 1/R$ (where R is a parameter to be discussed). Similarly, if we use the strength k, the probability of not finding a strict improvement for a point with individual gap of k within $\binom{n}{k} \ln R$ steps is at most

$$\left(1 - \frac{1}{\binom{n}{k}}\right)^{\binom{n}{k} \ln R} \leq \frac{1}{R}.$$

Hence, after $\binom{n}{k} \ln R$ steps, there is a probability of at least $1 - 1/R$ that a possible improvement at that distance would have been found.

Algorithm 11: RLS with plain stagnation detection (SD-RLS$^{\text{P}}$) with parameter R for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$

Select x uniformly at random from $\{0,1\}^n$ and set $s_1 \leftarrow 1$;
$u \leftarrow 0$;
for $t \leftarrow 1, 2, \ldots$ **do**
  Create y by flipping s_t bits in a copy of x uniformly;
  $u \leftarrow u + 1$;
  if $f(y) > f(x)$ **then**
    $x \leftarrow y$;
    $s_{t+1} \leftarrow 1$;
    $u \leftarrow 0$;
  else
    if $f(y) = f(x)$ ***and*** $s_t = 1$ **then**
      $x \leftarrow y$;
    if $u \geq \binom{n}{s_t} \ln R$ **then**
      $s_{t+1} \leftarrow \min\{s_t + 1, n\}$;
      $u \leftarrow 0$;
    else
      $s_{t+1} \leftarrow s_t$;

We consider this idea to develop the first algorithm. We add the *plain* stagnation detection mechanism to RLS to manage the strength s. As shown in Algorithm 11, hereinafter called SD-RLS$^{\text{P}}$, the initial strength is 1. Also, there is a counter u for counting the number of unsuccessful steps, i. e., steps without a strict improvement, to find the next after the last success. When the counter exceeds the threshold of $\binom{n}{s} \ln R$, the strength s is increased by one, and when the algorithm finds a strict improvement using appropriate strengths, e. g., a strength equal to the individual gap of the current search point, the counter and the strength are reset to their initial values. Non-strict improvements, i. e., search points of equal fitness, are accepted if the current strength equals 1 to maintain the search behavior of the classical RLS algorithm with 1-bit flip mutation. They also increase the counter.

In the case that the algorithm fails to have a success at a strength allowing a strict improvement, the algorithm might miss the chance of making further progress. Therefore, with probability up to $1/R$, the optimization time would be infinite. Choosing a sufficiently large R to have an overwhelming large probability of making progress could be a solution to this problem. However, we next propose another algorithm that resolves this issue, although the running time is not always as efficient as with Algorithm 11.

Algorithm 12: RLS with robust stagnation detection (SD-RLS$^{\mathrm{r}}$) with parameter R for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$

Select x uniformly at random from $\{0,1\}^n$ and set $r_1 \leftarrow 1$ and $s_1 \leftarrow 1$;
$u \leftarrow 0$;
for $t \leftarrow 1, 2, \ldots$ **do**
  Create y by flipping s_t bits in a copy of x uniformly;
  $u \leftarrow u + 1$;
  if $f(y) > f(x)$ **then**
    $x \leftarrow y$;
    $s_{t+1} \leftarrow 1$;
    $r_{t+1} \leftarrow 1$;
    $u \leftarrow 0$;
  else
    if $f(y) = f(x)$ ***and*** $r_t = 1$ **then**
      $x \leftarrow y$;
    if $u \geq \binom{n}{s_t} \ln R$ **then**
      if $s_t = r_t$ **then**
        if $r_t \leq \lfloor n/2 \rfloor - 1$ **then** $r_{t+1} \leftarrow r_t + 1$ **else** $r_{t+1} \leftarrow n$;
        $s_{t+1} \leftarrow 1$;
      else
        $r_{t+1} \leftarrow r_t$;
        $s_{t+1} \leftarrow s_t + 1$;
      $u \leftarrow 0$;
    else
      $s_{t+1} \leftarrow s_t$;
      $r_{t+1} \leftarrow r_t$;

In Algorithm 12, hereinafter called SD-RLS$^{\mathrm{r}}$, where the label r denotes *robust*, we introduce a new variable r called radius. This parameter determines the largest Hamming distance from the current search point that the algorithm must investigate. In detail, when the radius is r, the algorithm explores all strengths at most r (i. e., strengths from 1 to r). This results in a more robust behavior since possibly promising strengths are revisited regularly. In the case that the threshold is exceeded and the current strength equals the radius, the radius is increased by one to cover a more expanded space. Also, when the radius exceeds $n/2$, the algorithm sets it to n, which means that the algorithm covers all possible strengths between 1 and n. Apart from that, the robust algorithm SD-RLS$^{\mathrm{r}}$ follows the plain variant SD-RLS$^{\mathrm{p}}$ as far as possible, including acceptance of equally good search points at radius 1.

The reason for not considering radii from $\lfloor n/2 \rfloor + 1$ to $n - 1$ is mostly for the sake of simplicity in the analyses. Since the threshold values $\binom{n}{s_t} \ln R$ are not increasing for strengths at least $n/2$, the first inequality in Lemma 4.1 does not hold anymore so we would need to consider another case for such gap sizes. Hence, we immediately increase the radius to n without substantially sacrificing the performance of the algorithm. We also note that the strategy of repeatedly returning to lower strengths remotely resembles the 1/5-rule with rollbacks proposed in [BBS21].

In the preliminary version of this work [RW21b], we considered a decreasing order for the strengths in such a way that the algorithm starts with strength r (i. e., $s = r$), and when the threshold is exceeded, it decreases the strength by one as long as the strength is greater than 1. However, in this paper, for the sake of simplicity of the algorithm, we consider the increasing order of the strengths instead of the decreasing one, i. e., the algorithm increases the strength from 1 to the radius r. In any case, in the proofs in this paper, we pessimistically assume that the successful strength making progress is attempted last after all other strengths. Thus, the runtime bounds derived are independent of the order in which the algorithm traverses the strength.

We finally discuss the parameter R in more detail, which is related to the probability of failing to find a strict improvement at the "right" strength. More precisely, as proved in Theorem 4.2 and Lemma 4.4 (for SD-RLS$^{\mathrm{p}}$ and SD-RLS$^{\mathrm{r}}$, respectively), the probability of not finding a strict improvement when there is a potential of making progress is at most $1/R$. Assume that S is an upper bound on the number of strict improvements during the run. In our upcoming analyses, we will recommend $R \geq S$ for SD-RLS$^{\mathrm{p}}$, and for an arbitrary constant $\epsilon > 0$, $R \geq \max\{n^{4+\epsilon}, S\}$ for SD-RLS$^{\mathrm{r}}$, resulting in that the probability of ever missing a strict improvement at the right strength is sufficiently small throughout the run.

The choice of S is not very crucial since only the logarithm of R determines the phase lengths of the algorithms, but can make a difference. Obviously, we can always set $S = |\mathrm{Im}\, f|$ (where $\mathrm{Im}\, f$ is the image set of f); however, sometimes we may have tighter estimates on the number of strict improvements. Consider the binary value problem, i. e., $\textsc{BinVal}(x_1, \ldots, x_n) = \sum_{i=1}^{n} 2^{i-1} x_i$, as an example. This function has 2^n different function values, i. e., $|\mathrm{Im}\, f| = 2^n$, but can be optimized by RLS by making at most n strict improvements.

4.2.2 Mathematical tools

The following lemma containing some combinatorial inequalities will be used in the analyses of the algorithms SD-RLS$^{\mathrm{p}}$ and SD-RLS$^{\mathrm{r}}$. The first part of the lemma seems to be well known and has already been proved in [Lug17] and is also a consequence of Lemma 1.10.38 in [Doe20b]. The second part follows from elementary manipulations.

Lemma 4.1. *We have*

1. $\sum_{i=1}^{m} \binom{n}{i} \leq \frac{n-(m-1)}{n-(2m-1)} \binom{n}{m}$ *for any integer* $m \leq n/2$,
2. $\binom{n}{M} \leq \left(\frac{n-m}{m+1}\right)^{M-m} \binom{n}{m}$ *for* $m \leq M \leq n$.

Proof. For (a), we use the following proof due to [Lug17]. Through the equation $\binom{n}{k-1} = \frac{k}{n-k+1}\binom{n}{k}$, which comes from the definition of the binomial coefficient, and the infinite geometric series sum formula, we achieve the following result:

$$\begin{aligned}
\frac{\sum_{i=1}^{m} \binom{n}{i}}{\binom{n}{m}} &= \frac{\binom{n}{m}}{\binom{n}{m}} + \frac{\binom{n}{m-1}}{\binom{n}{m}} + \cdots + \frac{\binom{n}{1}}{\binom{n}{m}} \\
&= 1 + \frac{m}{n-m+1} + \frac{m(m-1)}{(n-m+1)(n-m+2)} + \cdots + \frac{m(m-1)\cdots 2}{(n-m+1)\cdots(n-1)} \\
&\leq 1 + \frac{m}{n-m+1} + \left(\frac{m}{n-m+1}\right)^2 + \cdots = \frac{n-(m-1)}{n-(2m-1)}.
\end{aligned}$$

Regarding (b), for $m = M$, the inequality holds trivially. For $m < M$, by using $\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$ multiple times, we have

$$\binom{n}{M} = \frac{(n-M+1)\cdots(n-m)}{M\cdots(m+1)}\binom{n}{m} \leq \left(\frac{n-m}{m+1}\right)^{M-m}\binom{n}{m}. \qquad \square$$

4.3 Analysis of the Algorithm SD-RLS$^{\mathrm{p}}$

In this section, we study the first algorithm called SD-RLS$^{\mathrm{p}}$, see Algorithm 11. In the beginning of the section, in Theorem 4.2, we will show upper bounds on the time to escape from local optima, conditioned on an event that holds with high probability depending on the choice of the parameter R. Then in Theorem 4.3, we will show the important result that on unimodal functions, SD-RLS$^{\mathrm{p}}$ with high probability behaves in the same way as RLS with strength 1,

including the same asymptotic bound on the expected optimization time using the fitness-level method.

We shall now introduce the notation used in the analysis of SD-RLS$^{\text{p}}$. Let x be the initial search point or the search point immediately following a strict improvement in a run of the algorithm on an arbitrary fitness function. Let *phase* s consist of all points of time where strength s is used in the algorithm. Let ℓ_s denote the number of iterations with strength s (in this section, it equals the number of iterations in phase s). We note that $\ell_s \leq \binom{n}{s} \ln R$ and that equality holds if no strict improvement is found in phase s.

We let P denote the random phase number in which SD-RLS$^{\text{p}}$ finds a strict improvement. Since the phase number is nothing else than the current strength used by the algorithm, intuitively, there should be a high probability that P is no larger than the fitness level gap of the search point x we consider. Concretely, if $m = \text{FitnessLevelGap}(x)$, the following theorem shows that $\Pr(P \leq m) \geq 1 - 1/R$ and gives us upper bounds on the expected time SD-RLS$^{\text{p}}$ takes to make progress, conditional on the event that $\{P \leq m\}$. We note that SD-RLS$^{\text{p}}$ might not be able to find a strict improvement in a finite number of steps, and $P = \infty$ can occur with positive probability. The reason is that there is a positive probability that the algorithm fails to make progress in situations when it is possible to find improvements, and when it increases the strength to larger values, it may fail to create a strict improvement forever. However, we have $\Pr(P = \infty) \leq \Pr(P > m) \leq 1/R$.

In the theorem, we also will make the assumption that no search point in the fitness level has individual gap 1. The reason is the following. The algorithm SD-RLS$^{\text{p}}$ selects search points in the same fitness level (and not only strict improvements) in phase 1, and these search points might have different individual gap sizes. Assume that in phase 1, the search points accepted have individual gaps of larger than 1 except in the last iteration of the phase, where a search point y with individual gap 1 is chosen. From the next iteration in phase 2, the algorithm only accepts strict improvements, so the algorithm might not make progress if there are only improvements in the local neighborhood (i. e., Hamming distance 1) of y. Then, we cannot guarantee that the algorithm spends enough iterations with "right" strength. The assumption of no search point in the fitness level having individual gap 1 is met on the typical benchmarks we consider, e. g., the local optima of the Jump function.

Theorem 4.2. *Let $0 < \epsilon < 1$ be a constant. Consider SD-RLS$^{\text{p}}$ on a pseudo-Boolean 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 $m =$ FitnessLevelGap(x). Assume that there is no search point in the fitness level*

of x with individual gap 1, i. e.

$$\forall y \in \{0,1\}^n \left(f(y) = f(x) \Rightarrow \mathrm{IndividualGap}(y) > 1\right).$$

Let P denote the phase in which the algorithm creates a strict improvement. We define T_m as the time to create a strict improvement. Then we have the upper-bound $\mathrm{E}\left(T_m \mid P \leq m\right) = O(2^n \ln R)$. If $m < (1-\epsilon)n/2$, we have

$$\mathrm{E}\left(T_m \mid P \leq m\right) \leq \binom{n}{m}\left(1 + O\left(\frac{m \ln R}{n}\right)\right).$$

Moreover, $\Pr\left(P \leq m\right) \geq 1 - 1/R$.

Compared to the corresponding theorems in [RW22a], the bound in Theorem 4.2 is by a factor of $(\frac{en}{m})^m / \binom{n}{m}$ (up to lower-order terms) smaller. In more detail, on unimodal functions, we have a speedup of e. For general m, using Stirling's inequality $m! > \sqrt{2\pi m}(\frac{m}{e})^m$, we compute a speedup of at least

$$\frac{(en/m)^m}{(1+o(1))\binom{n}{m}} \geq (1-o(1))\frac{(en/m)^m}{\frac{n^m}{m!}} > (1-o(1))\sqrt{2\pi m}. \tag{4.1}$$

Proof of Theorem 4.2. The algorithm SD-RLS$^{\mathrm{p}}$ can make a strict improvement when the current strength equals the individual gap of the current search point. Let y be the current search point at the beginning of phase 2. This point might be different from x because in phase 1 the algorithm accepts offspring with the same fitness value. However, in phase 2 and larger, y is fixed until a strict improvement is found.

We know that $g_y \coloneqq \mathrm{IndividualGap}(y) \geq 2$ because of the assumption in the theorem. There is no strict improvement in phases less than g_y. Since the algorithm spends $\binom{n}{g_y}$ iterations with strength g_y (or a strict improvement is found), we have

$$\Pr\left(P > g_y\right) \leq \left(1 - \binom{n}{g_y}^{-1}\right)^{\binom{n}{g_y}\ln R} \leq \frac{1}{R}.$$

Thus we have $\Pr\left(P \leq m\right) \geq \Pr\left(P \leq g_y\right) = 1 - \Pr\left(P > g_y\right) \geq 1 - 1/R$ as claimed.

We now prove the statements on $\mathrm{E}\left(T_m \mid P \leq m\right)$. We recall that ℓ_s denotes the number of iterations with strength s, i. e., $\ell_s \leq \binom{n}{s}\ln R$. Let $g_y \leq k \leq m$. We have

$$\mathrm{E}\left(T_m \mid P = k\right) \leq \underbrace{\sum_{s=1}^{k-1} \ell_s}_{=:S_1} + \underbrace{\binom{n}{k}}_{=:S_2},$$

where S_1 is the number of iterations for increasing the strength to k and S_2 is an upper bound on the expected number of iterations needed to make a strict improvement when the strength equals k. For S_2, we note that the algorithm finds a strict improvement in phase k as we condition on the event $\{P = k\}$. In each step, the probability of finding this improvement is at least $\binom{n}{k}^{-1}$. Hence, the algorithm makes progress in at most $\binom{n}{k}$ iterations in expectation using the truncated geometric distribution.

Since using the law of total probability, we have

$$\begin{aligned}\mathrm{E}\left(T_m \mid P \leq m\right) &= \sum_{k=g_y}^{m} \mathrm{E}\left(T_m \mid P = k\right) \Pr\left(P = k \mid P \leq m\right) \\ &\leq \max_{g_y \leq k \leq m} \mathrm{E}\left(T_m \mid P = k\right)\end{aligned}$$

and $g_y \leq m$, we conclude

$$\mathrm{E}\left(T_m \mid P \leq m\right) \leq \sum_{s=1}^{m-1} \ell_s + \binom{n}{m}, \tag{4.2}$$

where we have used $\binom{n}{b} \leq \binom{n}{a}$ for $b \leq a \leq n/2$. By using Lemma 4.1 for $m < n/2$, we have

$$\begin{aligned}\mathrm{E}\left(T_m \mid P \leq m\right) &\leq \sum_{s=1}^{m-1} \binom{n}{s} \ln R + \binom{n}{m} \leq \frac{n-m+2}{n-2m+3}\binom{n}{m-1} \ln R + \binom{n}{m} \\ &= \frac{n-m+2}{n-2m+3} \cdot \frac{m}{n-m+1}\binom{n}{m} \ln R + \binom{n}{m} \\ &= \binom{n}{m}\left(\frac{n-m+2}{n-2m+3} \cdot \frac{m}{n-m+1} \ln R + 1\right).\end{aligned}$$

If $m < (1-\epsilon)n/2$, we have

$$\mathrm{E}\left(T_m \mid P \leq m\right) \leq \binom{n}{m}\left(1 + O\left(\frac{m \ln R}{n}\right)\right),$$

which proves the bound on $\mathrm{E}\left(T_m \mid P \leq m\right)$ for $m < (1-\epsilon)n/2$.

To prove the general bound, since we have $\sum_{s=1}^{n} \binom{n}{s} < 2^n$, we can compute an upper bound on Equation (4.2):

$$\mathrm{E}\left(T \mid P \leq m\right) = \sum_{s=1}^{m-1} \ell_s + \binom{n}{m} = \sum_{s=1}^{m-1} \binom{n}{s} \ln R + \binom{n}{m} = O(2^n \ln R). \quad \square$$

Next, we obtain the following result that allows us to reuse existing results for RLS on unimodal functions. In particular, runtime bounds obtained via the well-known fitness-level method [Weg02] can be carried over.

Theorem 4.3. *Let $\epsilon > 0$ be a constant. Let $f\colon \{0,1\}^n \to \mathbb{R}$ be a unimodal function and consider SD-RLS^p with $R \geq S$, where S is an upper bound on the number of strict improvements during the run, e. g., $S = |\operatorname{Im} f|$. Then there is an event G happening with probability at least $1 - S/R$, such that conditioned on G, SD-RLS^p never increases the radius and behaves stochastically like RLS, also conditioned on G, before finding an optimum of f.*

Denote by T the runtime of SD-RLS^p with $R \geq n^{4+\epsilon}$ on f. Let f_i be the i-th fitness value of an increasing order of all fitness values of f and s_i be a lower bound on the probability that RLS finds a strict improvement from search points with fitness value f_i. Then it holds that

$$\mathrm{E}\left(T \mid G\right) \leq \sum_{i=1}^{|\operatorname{Im} f|-1} \frac{1}{s_i}.$$

Proof. As on unimodal functions, the individual gap of all points is 1, the probability of not finding a strict improvement from any non-optimal search point within phase 1 consisting of $\binom{n}{1} \ln R$ steps is at most

$$\left(1 - \binom{n}{1}^{-1}\right)^{\binom{n}{1}\ln R} \leq \frac{1}{R}.$$

This argumentation holds for each improvement that has to be found. Let G be the event that all strict improvements until finding a global optimum are found in phase 1. On G, SD-RLS^p always uses strength 1 and is stochastically identical to RLS, also conditioned on G. Since at most S improving steps happen before finding the optimum, by a union bound we have that $\Pr\left(G\right) \geq 1 - S/R$. This proves the first claim.

To prove the second claim, we use a similar approach as for the analysis of S_2 in the proof of Theorem 4.2. Conditioned on G, we bound the time to leave fitness level f_i from above by a truncated geometric distribution with success probability at least s_i. Summing up the upper bounds $1/s_i$ on expected waiting times, we obtain the second claim. □

We remark that s_i in Lemma 4.3 can be set to N_i/n, where N_i is the minimum number of local strict improvements (i. e., strict improvements at Hamming distance 1) of the search points in fitness level i.

With these two general results, we conclude the analysis of SD-RLS$^{\mathrm{p}}$ and turn to the variant SD-RLS$^{\mathrm{r}}$ that always has finite expected optimization time. In fact, we will present similar results in general optimization scenarios and supplement them by analyses on specific benchmark functions. It is possible to analyze the simpler SD-RLS$^{\mathrm{p}}$ on these benchmark functions as well, but we do not feel that this gives additional insights.

4.4 Analysis of the Algorithm SD-RLS$^{\mathrm{r}}$

In this section, we turn to the algorithm SD-RLS$^{\mathrm{r}}$ that iteratively returns to lower strengths to avoid missing the "right" strength. In a nutshell, we will obtain the following results: Theorem 4.6 will show that the algorithm will overcome (fitness level) gaps of size m in expected time roughly $(1 + o(1))\binom{n}{m}$, i. e., essentially corresponding to the size of the m-bit neighborhood. Unlike the analysis in Theorem 4.2, this expected time is not conditional on an event corresponding to making progress with the "right strength". To prove Theorem 4.6, we will formulate two helper results in Lemmas 4.4 and 4.5. The purpose of the first lemma is to bound the probability of missing a strict improvement at the "right" strength, similar to the proof of Theorem 4.2, while the second one bounds the contribution of steps using "too high" strengths to the total expected runtime. Afterwards, we re-use the helper results to prove Theorem 4.7, a result dealing with the behavior on unimodal functions. The statement and results are similar to Theorem 4.3; however, again the expected runtime is finite in total and not only conditional on a success event. We will conclude this section with an analysis of SD-RLS$^{\mathrm{r}}$ on the jump function class, where we will make use of both Theorem 4.6 and Theorem 4.7.

We now define the crucial notions in the analysis of SD-RLS$^{\mathrm{r}}$, including the distinction of so-called phases and subphases. As in the previous section, let x be the initial search point or the search point immediately following a strict improvement in a run of SD-RLS$^{\mathrm{r}}$ on a fitness function f. We consider the epoch starting from the first point in time where x is the current search point until the next strict improvement. Let *phase* r consist of all points of time of the epoch, starting immediately after a reset of the strength to 1 (or the beginning of the epoch), before the strength is reset to 1 for the r^{th} time or the epoch ends. In other words, the epoch begins with phase 1, and any phase $r \geq 1$ starts with strength 1. When the counter exceeds the threshold, if the strength is smaller than the radius, it increases the strength by one; otherwise, the next phase $r+1$ begins, the radius increases to $r + 1$ and the strength is reset to 1. Hence, for $r \leq \lfloor n/2 \rfloor$, we have that phase r starts from the first time with radius r and ends before the radius reaches $r + 1$ or a strict improvement is found.

In the case that $r+1 > \lfloor n/2 \rfloor$, the next radius is n instead of $r+1$ but the phase number increases from $\lfloor n/2 \rfloor$ to $\lfloor n/2 \rfloor + 1$. Eventually, the phase number may even become larger than the radius. For example, when the radius has become n for the first time and the strength has increased from 1 to n without a success at any strength, the strength will be reset to 1 again, which starts phase $\lfloor n/2 \rfloor + 2$ etc.

Let *subphase* s consist of all iterations with strength s in a phase. We have that for $r \in [1..\lfloor n/2 \rfloor]$, phase r consists of subphases $s \in [1..r]$ and any phase larger than $\lfloor n/2 \rfloor$ consists of subphases $s \in [1..n]$.

Carrying over the notation from the previous section, let ℓ_s denote the number of iterations in subphase s in a phase, so we have $\ell_s \leq \binom{n}{s} \ln R$. Moreover, let L_i denote the number of iterations in phase i. Then we have $L_i \leq \sum_{s=1}^{i} \ell_s$ for $i \leq \lfloor n/2 \rfloor$ and $L_i \leq \sum_{s=1}^{n} \ell_s$ for $r > \lfloor n/2 \rfloor$. We note that the upper bounds on ℓ_s and L_i hold with equality if no strict improvement is found in subphase s or phase i, respectively. We should also note that ℓ_s already appeared in the previous section, where a different notion of phases was used; more precisely, the phases in Section 4.3 correspond to subphases in this section. However, mathematically, the ℓ_s are the same in both sections.

Recall that according to the definition of the algorithm SD-RLS^r, in phase 1, i. e., when the radius is 1, the algorithm might change the current search point if another one with the same fitness value $f(x)$ is found. However, from phase 2 onwards, the algorithm keeps the current search point until a strict improvement is found.

Our main result in this section, Theorem 4.6 below, hinges on an analysis of the phase in which the algorithm makes progress. Let E_r be the event of **not** finding an optimum within phase r. Let the random variable P denote the phase in which the algorithm finds a strict improvement, that is, $\Pr(P = i) = \Pr\left(E_1 \cap \cdots \cap E_{i-1} \cap \overline{E_i}\right)$. In our analyses, we will bound the distribution of P and show that it essentially is dominated by a geometric distribution with success probability $1 - 1/R$ plus the gap size of the current search point minus 1. This is a consequence of the following result on the failure probability E_r.

Lemma 4.4. *Consider SD-RLS^r on a pseudo-Boolean 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* $m = \mathrm{FitnessLevelGap}(x)$ *and* $m \leq \lfloor n/2 \rfloor$. *Then for* $r \geq m$, $\Pr(E_r) \leq 1/R$.

Proof. Let y be the current search point in the beginning of phase r and $g_y \coloneqq \text{IndividualGap}(y)$. For $r \geq 2$, the algorithm does not change the current search point during phase r. However, for $m = 1$ and $r = 1$, although the algorithm might change the current search point, the individual gap of all selected search points is still 1. Therefore, in both cases, there is a strict improvement at Hamming distance g_y during phase r. We have that $g_y \leq m$ since by definition, the fitness level gap is the maximum of all individual gap sizes in the fitness level.

During phase r, the algorithm spends ℓ_{g_y} steps at strength g_y until it changes the strength or enters phase $r+1$ (unless it already has found a strict improvement with strength g_y). Then the probability of not finding a strict improvement in subphase g_y of phase r is at most

$$\Pr(E_r) \leq \left(1 - \binom{n}{g_y}^{-1}\right)^{\ell_{g_y}} = \left(1 - \binom{n}{g_y}^{-1}\right)^{\binom{n}{g_y}\ln R} \leq \frac{1}{R}. \quad \square$$

Intuitively, the proof of our main result shows that SD-RLS$^{\text{r}}$ has a high probability of finding a strict improvement in phase m, where m corresponds to the gap size of the current search point. The following lemma bounds the time to leave a local optimum conditional on that the "right" strength was missed. In the proof of our main theorem, we will combine this lemma with Lemma 4.4 to obtain an expected number of at most $o(\binom{n}{m})$ extra iterations at "too high" strengths, which is dominated by the upper bound $\binom{n}{m}$ on the expected number of iterations sufficient for making a strict improvement at strength at most m.

Lemma 4.5. *Let $\epsilon > 0$ be a constant. Consider SD-RLS$^{\text{r}}$ with $R \geq n^{4+\epsilon}$ on 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 $m \leq \lfloor n/2 \rfloor$. By T_m we define the time to create a strict improvement. Let P denote the phase in which the algorithm makes progress. Then we have*

$$\mathrm{E}(T_m \mid P > m) = o\left(\frac{R}{n} \cdot \binom{n}{m}\right).$$

Proof. Using the law of total probability with respect to the random variable P defined above, we have

$$\begin{aligned}\mathrm{E}(T_m \mid P > m) &= \sum_{i=1}^{\infty} \mathrm{E}(T_m \mid P = i \cap P > m) \Pr(P = i \mid P > m) \\ &= \sum_{i=m+1}^{\infty} \mathrm{E}(T_m \mid P = i) \Pr(P = i \mid P > m).\end{aligned}$$

The variable i represents the earliest phase in which the algorithm finds a strict improvement. The event $\{P = i \mid P > m\}$ implies that no strict improvement is made in phases $m+1$ to $i-1$. Thus in order to estimate $\Pr(P = i \mid P > m)$ for $i \geq m+1$, we have

$$\Pr(P = i \mid P > m) \leq \Pr(E_{m+1} \cap \cdots \cap E_{i-1}) \leq \prod_{j=m+1}^{i-1} \Pr(E_j) \leq R^{-(i-m-1)},$$

where we have used Lemma 4.4 to bound $\Pr(E_j)$.

Let y be the current search point from phase 2 and $g_y \coloneqq \text{IndividualGap}(y)$. For $m+1 \leq i \leq \lfloor n/2 \rfloor$, we estimate, using the notation L_r defined above, that

$$\mathrm{E}(T_m \mid P = i) \leq \sum_{r=1}^{i} L_r \leq \sum_{r=1}^{i} \sum_{s=1}^{r} \binom{n}{s} \ln R \leq n^2 \binom{n}{i} \ln R \leq n^{i-m+2} \binom{n}{m} \ln R,$$

where we have used $\binom{n}{i} \leq n^{i-m}\binom{n}{m}$ using Lemma 4.12. For $i > \lfloor n/2 \rfloor$, we compute

$$\mathrm{E}(T_m \mid P = i) \leq \sum_{r=1}^{i} L_r \leq \sum_{r=1}^{\lfloor n/2 \rfloor} \sum_{s=1}^{r} \binom{n}{s} \ln R + (i - \lfloor n/2 \rfloor) \sum_{s=1}^{n} \binom{n}{s} \ln R,$$

recalling that radius $\lfloor n/2 \rfloor$ is immediately follows by radius n. Since for all $1 \leq k \leq n$, we have $\binom{n}{k} \leq \binom{n}{\lfloor n/2 \rfloor}$, we bound the last expression from above by

$$\begin{aligned} &\frac{n^2}{4}\binom{n}{\lfloor n/2 \rfloor} \ln R + (i - \lfloor n/2 \rfloor) n \binom{n}{\lfloor n/2 \rfloor} \ln R \\ &\quad < (i - \lfloor n/2 \rfloor) n^2 n^{\lfloor n/2 \rfloor - m} \binom{n}{m} \ln R. \end{aligned}$$

Altogether, since $(i - \lfloor n/2 \rfloor) < n^{i - \lfloor n/2 \rfloor}$ for sufficiently large n, we have

$$\begin{aligned} \sum_{i=m+1}^{\infty} \mathrm{E}(T_m \mid P = i) \Pr(P = i \mid P > m) &< \sum_{i=m+1}^{\infty} \binom{n}{m} \frac{n^{i-m+2} \ln R}{R^{i-m-1}} \\ &= n^3 \ln R \cdot \binom{n}{m} \left(\sum_{i=0}^{\infty} (n/R)^i \right). \end{aligned}$$

Using the fact that $R \geq n^{4+\epsilon}$ and $i > m$, the last expression is bounded from above by

$$O\left(n^3 \ln R \cdot \binom{n}{m}\right) = o\left(\frac{R}{n} \cdot \binom{n}{m}\right). \qquad \square$$

We are now ready for formulate and prove our main result in the following Theorem 4.6. The theorem and its proof are similar to Theorem 4.2 but require a more careful analysis to cover the repeated use of smaller strengths. We note that the bounds differ from Theorem 4.2 only in lower-order terms. Hence, we again achieve the speedup calculated in Equation 4.1 for SD-RLS$^{\mathrm{r}}$, which is $\sqrt{2\pi m}$ roughly, compared to SD-(1+1) EA.

Theorem 4.6. *Let $\hat{\epsilon} > 0$ and $0 < \epsilon < 1$ be constants. Consider SD-RLS$^{\mathrm{r}}$ with $R \geq n^{4+\hat{\epsilon}}$ on a pseudo-Boolean 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 $m = \mathrm{FitnessLevelGap}(x)$. By T_m we define the time to create a strict improvement. Then $\mathrm{E}(T_m) = O(2^n n \ln R)$. Moreover, if $m < (1-\epsilon)n/2$, we have*

$$\mathrm{E}(T_m) \leq \binom{n}{m}\left(1 + O\left(\frac{m \ln R}{n}\right)\right).$$

Proof. We first prove the statement where we have $m < (1-\epsilon)n/2$. Here we will exploit that in phase m, there is a subphase of index $g_y \leq m$ where the algorithm can potentially make progress, i. e., create a strict improvement, in each iteration of the subphase. If $m = 1$, we have $g_y = m = 1$, because all selected search points in phase and subphase 1 have the individual gap 1, so the algorithm can make progress in subphase 1. If $m \geq 2$, let y be the search point in the beginning of phase 2. This point might be different from x since the algorithm accepts equal-fitness search points in phase 1. However, in phase 2 and after, the current search point y remains fixed until a strict improvement is found. Thus, the algorithm is able to create a strict improvement in subphase $g_y \coloneqq \mathrm{IndividualGap}(y)$. We recall that $g_y \leq m$ since by definition, the fitness level gap is the maximum of all individual gap sizes in the fitness level.

We recall the random variable P denoting the phase in which the algorithm creates a strict improvement. The aim is to decompose the expected time $\mathrm{E}(T_m)$ according to the event $P \leq m$, which is the more typical event of making progress in a phase no larger than the fitness level gap of the search point x, and the opposite event $P > m$, whose contribution to $\mathrm{E}(T_m)$ will be proved small. Formally, using the law of total probability with respect to P, we have

$$\mathrm{E}(T_m) = \underbrace{\mathrm{E}(T_m \mid P \leq m)\Pr(P \leq m)}_{=:S_1} + \underbrace{\mathrm{E}(T_m \mid P > m)\Pr(P > m)}_{=:S_2},$$

and we are left with bounding S_1 and S_2.

Regarding S_1, we pessimistically assume that a strict improvement is not found in phases less than m. Thus, it take $\sum_{r=1}^{m-1} L_r$ steps until SD-RLS$^{\mathrm{r}}$ increases the

radius to m. Then it takes $\sum_{s=1}^{g_y-1} \ell_s$ steps to increase the strength to g_y. When the strength is g_y, within an expected number of at most $\binom{n}{g_y}$ steps, a better point is found or the subphase is terminated by using a truncated geometric distribution. Thus, since $g_y \leq m$,

$$\begin{aligned} S_1 &= \Pr(P \leq m)\,\mathrm{E}(T_m \mid P \leq m) \leq \mathrm{E}(T_m \mid P \leq m) \leq \mathrm{E}(T_m \mid P = m) \\ &\leq \sum_{r=1}^{m-1} L_r + \sum_{s=1}^{g_y-1} \ell_s + \binom{n}{g_y} \leq \sum_{r=1}^{m-1}\sum_{s=1}^{r} \binom{n}{s} \ln R + \sum_{s=1}^{m-1} \binom{n}{s} \ln R + \binom{n}{m}. \end{aligned}$$

Using Lemma 4.1, we can bound the last expression from above by

$$\begin{aligned} &\sum_{r=1}^{m-1} \frac{n-(r-1)}{n-(2r-1)} \binom{n}{r} \ln R + \sum_{s=1}^{m-1} \binom{n}{s} \ln R + \binom{n}{m} \\ &\leq \left(1 + \frac{m-1}{n-2m+3}\right) \sum_{r=1}^{m-1} \binom{n}{r} \ln R + \sum_{s=1}^{m-1} \binom{n}{s} \ln R + \binom{n}{m} \\ &\leq \left(2 + \frac{m-1}{n-2m+3}\right)\left(1 + \frac{m-1}{n-2m+3}\right) \binom{n}{m-1} \ln R + \binom{n}{m} \\ &\leq \left(2 + \frac{m-1}{n-2m+3}\right)\left(1 + \frac{m-1}{n-2m+3}\right) \frac{m \cdot \ln R}{n-m+1} \binom{n}{m} + \binom{n}{m}. \end{aligned}$$

Then for $m < (1-\epsilon)\frac{n}{2}$, the last expression is bounded from above by

$$\binom{n}{m}\left(1 + O(\frac{m \ln R}{n})\right).$$

In regard to S_2, where the optimum is not found by the end of phase m, there are in expectation at most $o((R/n)\binom{n}{m})$ iterations to find the optimum through Lemma 4.5. Also, the event $\{P > m\}$ happens if the algorithm fails to make a strict improvement in phase m, so $\Pr(P > m) \leq \Pr(E_m) \leq 1/R$ using Lemma 4.4. Altogether, for $m < (1-\epsilon)\frac{n}{2}$, we have

$$\begin{aligned} \mathrm{E}(T_m) &= \Pr(P \leq m)\,\mathrm{E}(T_m \mid P \leq m) + \Pr(P > m)\,\mathrm{E}(T_m \mid P > m) \\ &\leq \binom{n}{m}\left(1 + O\left(\frac{m \ln R}{n}\right)\right) + \frac{1}{R} \cdot o\left(\frac{R}{n}\binom{n}{m}\right) \\ &\leq \binom{n}{m}\left(1 + O\left(\frac{m \ln R}{n}\right) + o(1/n)\right). \end{aligned}$$

We are left with the proof of the general result $\mathrm{E}(T_m) = O(2^n n \ln R)$. Here we pessimistically assume that the radius r increases to n as no strict improvement has been found in phases at most $\lfloor n/2 \rfloor$. Afterwards, in each phase, there is a

subphase using the strength of the individual gap of the current search point. Thus according to Lemma 4.4, the algorithm makes a strict improvement with probability at least $1 - 1/R$; so by the geometric distribution, there are in expectation at most $R/(R-1)$ phases with radius n. Finally, recalling that we pessimistically ignore all progress in phases at most $\lfloor n/2 \rfloor$, we bound

$$\begin{aligned}
\mathrm{E}(T_m) &\leq \sum_{r=1}^{\lfloor n/2 \rfloor} L_r + \mathrm{E}(T_m \mid P > \lfloor n/2 \rfloor) \leq \sum_{r=1}^{\lfloor n/2 \rfloor} L_r + \frac{R}{R-1} \sum_{s=1}^{n} \ell_s \\
&= \sum_{r=1}^{\lfloor n/2 \rfloor} \sum_{s=1}^{r} \binom{n}{s} \ln R + \frac{R}{R-1} \sum_{s=1}^{n} \binom{n}{s} \ln R \\
&\leq \sum_{r=1}^{n} \sum_{s=1}^{n} \binom{n}{s} \ln R + \frac{R}{R-1} \sum_{s=1}^{n} \binom{n}{s} \ln R = O\left(2^n n \ln R\right),
\end{aligned}$$

where we used $R \geq n^{4+\hat{\epsilon}}$. □

Similarly to Lemma 4.3, we obtain a relation to RLS on unimodal functions and can re-use existing upper bounds based on the fitness-level method. The first part of the following theorem and its proof are the same as in Theorem 4.3. However, since the runtime of SD-RLS$^{\mathrm{r}}$ is finite, in the second part of the following theorem, we prove an upper bound on the optimization time of the algorithm using runtime bounds for RLS.

Theorem 4.7. *Let $\epsilon > 0$ be a constant. Let $f\colon \{0,1\}^n \to \mathbb{R}$ be a unimodal function and consider SD-RLS$^{\mathrm{r}}$ with $R \geq S$, where S is an upper bound on the number of strict improvements during the run, e. g., $S = |\operatorname{Im} f|$. Then there is an event G happening with probability at least $1 - S/R$, such that conditioned on G, SD-RLS$^{\mathrm{r}}$ never increases the radius and behaves stochastically like RLS, also conditioned on G, before finding an optimum of f.*

Denote by T the runtime of SD-RLS$^{\mathrm{r}}$ with $R \geq n^{4+\epsilon}$ on f. Let f_i be the i-th fitness value of an increasing order of all fitness values in f and s_i be a lower bound on the probability that RLS finds a strict improvement from search points with fitness value f_i. Then

$$\mathrm{E}(T) \leq (1 + o(1)) \sum_{i=1}^{|\operatorname{Im} f|-1} \frac{1}{s_i}.$$

Proof. Using the same arguments as in the proof of Theorem 4.3, we prove the first part.

To prove the second claim, we consider all fitness levels $A_1, \dots, A_{|\operatorname{Im} f|}$ such that A_i contains search points with fitness value f_i and sum up upper bounds on the expected times to leave each of these fitness levels. Let T_i denote the random time spent in level A_i. We have $T \leq \sum_{i=1}^{|\operatorname{Im} f|-1} T_i$. Using the truncated geometric distribution, within at most $1/s_i$ iterations in expectation, the algorithm leaves the fitness level by finding a strict improvement or increases the radius to 2 (i. e., the phase 1 ends). Using Lemma 4.4, the probability of that the phase is terminated before creating a strict improvement is at most $1/R$, and using Lemma 4.5, if the algorithm enters the second phase, the expected time to find a strict improvement is $o((R/n)\binom{n}{1}) = o(R)$. Altogether, since $s_i \leq 1$, we have

$$\mathrm{E}(T_i) \leq \frac{1}{s_i} + \frac{1}{R} \cdot o(R) = (1 + o(1))\frac{1}{s_i}.$$

Then

$$\mathrm{E}(T) \leq (1 + o(1)) \sum_{i=1}^{|\operatorname{Im} f|-1} \frac{1}{s_i}. \quad \square$$

In the same way as for Theorem 4.3, s_i in Theorem 4.7 can be set to N_i/n, where N_i is the minimum number of local strict improvements of the search points in fitness level i.

Finally, we use the results developed so far to prove a bound on the Jump function with gap size m. This bound is essentially the size of the m-bit flip neighborhood and seems to be the best available for mutation-based hill-climbers. For example, the expected optimization time of the (1+1) EA is $\Omega(n^m)$ [DJW02], which is by an asymptotic factor $\Omega(m^m)$ worse. The fast (1+1) EA from [DLMN17] is by a factor polynomial in m slower, as is the SD-(1+1) EA [RW22a].

Theorem 4.8. *Let $n \in \mathbb{N}$ and $\epsilon > 0$ be a constant. For all $2 \leq m = o(n)$, the expected runtime $\mathrm{E}(T)$ of SD-RLS^r with $R \geq n^{4+\epsilon}$ on $\textsc{Jump}_m$ satisfies*

$$\binom{n}{m}(1 - o(1)) \leq \mathrm{E}(T) \leq \binom{n}{m}\left(1 + O\left(\tfrac{m}{n}\ln R\right)\right).$$

Proof. Before reaching the plateau consisting of all points of $n - m$ one-bits, Jump is equivalent to OneMax; hence, according to Theorem 4.7, the expected time SD-RLS^r takes to reach the plateau at $n-m$ one-bits is at most $O(n \ln n)$. Note that this bound was obtained via the fitness level method with $s_i = (n - i)/n$ as minimum probability for leaving the set of search points with i one-bits via a one-bit flip.

Every plateau point x with $n - m$ one-bits satisfies

$$\text{IndividualGap}(x) = \text{FitnessLevelGap}(x) = m,$$

according to the definition of Jump. Thus, using Theorem 4.6, starting from the plateau, the algorithm finds the optimum within expected time

$$\mathrm{E}(T) \leq \binom{n}{m} \left(1 + O\left(\tfrac{m}{n} \ln R\right)\right).$$

Since $m \geq 2$, this dominates the expected time of the algorithm before the plateau point and results in the running time in the theorem.

Regarding the lower bound, we show that with high probability, the initial search point is not the global optimum and is not a search point at Hamming distance 1 to the global optimum. Using Chernoff's bounds, the probability that the initial search point is at Hamming distance at least m from the optimum is $1 - o(1)$. Using Theorem 4.7, with probability $1 - o(1)$, the algorithm finds at most n strict improvements in phase 1 until it reaches the local optimum (within an expected number of $\Theta(n \ln n)$ iterations). To move from the local optimum to the global optimum, the algorithm first needs to increase the radius and strength to m. From that strength, the algorithm requires at least $\binom{n}{m}$ iterations in expectation since only one out of the search points at Hamming distance m yields a strict improvement. Hence, SD-RLS$^{\mathrm{r}}$ needs at least $(1 - o(1))\binom{n}{m}$ iterations in expectation. □

4.5 An Example Where Global Mutations are Necessary

While our s-bit flip mutation along with stagnation detection can outperform the (1+1) EA on Jump functions, it is clear that its different search behavior may be disadvantageous on other examples. Concretely, we will present a function that has a unimodal path to a local optimum with a large Hamming distance to the global optimum. SD-RLS$^{\mathrm{p}}$ will with high probability follow this path and incur exponential optimization time. However, the function has a second gradient that requires two-bit flips to make progress. The classical (1+1) EA will be able to follow this gradient and to arrive at the global optimum before one-bit flips have reached the end of the path to the local optimum.

In a broader context, our function illustrates an advantage of global mutation operators. By a simple swap of local and global optimum, it immediately turns into the direct opposite, i. e., an example where using global instead of local

mutations is highly detrimental and increases the runtime from polynomial to exponential with overwhelming probability. An example of such a function was previously presented in [DJK08]; however, both the underlying construction and the proof of exponential runtime for the (1+1) EA seem much more complicated than our example.

The construction of our function is based on a general principle that was introduced in [Wit03] to show the benefits of populations and was subsequently applied in [JW04] to separate a coevolutionary variant of the (1+1) EA from the standard (1+1) EA. Section 5 of the latter paper also beautifully describes the general construction technique that involves creating two differently pronounced gradients for the algorithms to follow. A further applications was given in [Wit06] to show the benefit of populations in elitist and non-elitist EAs. Also, [RLY09] use very similar construction technique for their BALANCE function that is easier to optimize in frequently changing than slowly changing environments; however, they did not seem to be aware that their approach resembles earlier work from the papers above. Recently, the construction technique inspired the function NEEDHIGHMUT that [RW22a] used to show disadvantages of stagnation detection adjusting the rate of a global mutation operator. In fact, our function NEEDGLOBALMUT is more or less immediately obtained from NEEDHIGHMUT. The only change is to adjust the length of the suffix part of the function, which rather elegantly allows us to re-use the previous technique of construction and a major part of the analysis.

We now described the function NEEDGLOBALMUT formally. In the following, we will imagine any bit string x of length n as being split into a prefix $a := a(x)$ of length $n-m$ and a suffix $b := b(x)$ of length m, where m is defined below. Hence, $x = a(x) \circ b(x)$, where $\circ$ denotes the concatenation. The prefix $a(x)$ is called *valid* if it is of the form $1^i 0^{n-m-i}$, i. e., i leading ones and $n-m-i$ trailing zeros. The prefix fitness $\textsc{pre}(x)$ of a string $x \in \{0,1\}^n$ with valid prefix $a(x) = 1^i 0^{n-m-i}$ equals i, the number of leading ones. The suffix consists of $\lceil \frac{1}{3}\sqrt{n} \rceil$ consecutive blocks of $\lceil n^{1/4} \rceil$ bits each, altogether $m \leq (\frac{1}{3}\sqrt{n}+1)(\lceil n^{1/4} \rceil) = O(n^{3/4})$ bits. Such a block is called *valid* if it contains either 0 or 2 one-bits; moreover, it is called *active* if it contains 2 and *inactive* if it contains 0 one-bits. A suffix where all blocks are valid and where all blocks following the first inactive block are also inactive is called valid itself, and the suffix fitness $\textsc{suff}(x)$ of a string x with valid suffix $b(x)$ is the number of leading active blocks before the first inactive one. Finally, we call $x \in \{0,1\}^n$ valid if both its prefix and suffix are valid.

The final fitness function is a weighted combination of $\textsc{pre}(x)$ and $\textsc{suff}(x)$. We define for $x \in \{0,1\}^n$, where $x = a \circ b$ with the above-introduced a and b,

$\textsc{NeedGlobalMut}(x) :=$

$$\begin{cases} n^2\textsc{suff}(x) + \textsc{pre}(x) & \text{if } \textsc{pre}(x) \leq \frac{9(n-m)}{10} \wedge x \text{ valid} \\ n^2\lceil \frac{1}{3}\sqrt{n} \rceil + \textsc{pre}(x) + \textsc{suff}(x) - n - 1 & \text{if } \textsc{pre}(x) > \frac{9(n-m)}{10} \wedge x \text{ valid} \\ -\textsc{OneMax}(x) & \text{otherwise.} \end{cases}$$

The function NeedGlobalMut equals NeedHighMut$_\xi$ from [RW22a] for the setting $\xi = 1/2$ (ignoring that $\xi < 1$ was disallowed there for technical reasons). We note that all search points in the second case have a fitness of at least $n^2\lceil \frac{1}{3}\sqrt{n} \rceil - n - 1$, which is bigger than $n^2(\lceil \frac{1}{3}\sqrt{n} \rceil - 1) + n$, an upper bound on the fitness of search points that fall into the first case without having m leading active blocks in the suffix. Hence, search points x where $\textsc{pre}(x) = n - m$ and $\textsc{suff}(x) = \lceil \frac{1}{3}\sqrt{n} \rceil$ represent local optima of second-best overall fitness. The set of global optima equals the points where $\textsc{pre}(x) = 9(n-m)/10$ and $\textsc{suff}(x) = \lceil \frac{1}{3}\sqrt{n} \rceil$, which implies that $(n-m)/10 = \Omega(n)$ bits have to be flipped simultaneously to escape from the local toward the global optimum.

Theorem 4.9. *With probability $1-o(1)$, both SD-RLS$^{\text{p}}$ and SD-RLS$^{\text{r}}$ with $R \geq n^2$ need $2^{\Omega(n)}$ steps to optimize* NeedGlobalMut. *The (1+1) EA optimizes this function in time $O(n^2)$ with probability $1 - 2^{-\Omega(n^{1/3})}$.*

Proof. As in the proof of Theorem 4.1 in [RW22a], we have that the first valid search point (i. e., search point of non-negative fitness) of both SD-RLS$^{\text{p}}$, SD-RLS$^{\text{r}}$ and (1+1) EA has both PRE- and SUFF-value value of at most $n^{1/3}$ with probability $2^{-\Omega(n^{1/3})}$. In the following, we tacitly assume that we have reached a valid search point of the described maximum PRE- and SUFF-value and note that this changes the required number of improvements to reach local or global maximum only by a $1-o(1)$ factor. For readability this factor will not be spelt out anymore.

We now consider SD-RLS$^{\text{p}}$ and SD-RLS$^{\text{r}}$, for both of which the proof is identical. As long as the counter threshold is not exceeded, the algorithm behaves like RLS. We follow the argumentation from Theorem 4.3 until the point in time where $\textsc{pre}(x) = n - m$ since it is possible to improve the function value by one-bit flips before. Hence, using a union bound over at most $n - m$ improvements, the probability of ever increasing the SUFF-value before $\textsc{pre}(x) = n - m$ is at most $(n-m)/R \leq 1/n$. The fitness can only be further improved if at least $(n-m)/10$ bits flip simultaneously. This requires the rate to be increased $(n-m)/10 - 1$ times; in particular the last of the increases happens only after a phase of length at least $\binom{n}{(n-m)/10-1} = 2^{\Omega(n)}$. This proves the statement for both SD-RLS$^{\text{p}}$ and SD-RLS$^{\text{r}}$.

We now analyze the success probability of the (1+1) EA. To this end, we first bound the probability of a mutation being accepted after a valid search point

has been reached. Even if a mutation changes up to $\Theta(n^{3/4})$ consecutive bits of the prefix or suffix, it must maintain $n - \Theta(n^{3/4})$ prefix bits in order to result in a valid search point. Hence, the probability of an accepted step at mutation probability $1/n$ is at most $(1-1/n)^{n-m-\Theta(n^{3/4})} = (1+o(1))e^{-1}$. Since the probability of flipping $\Omega(n^{3/4})$ bits is $n^{-\Omega(n^{3/4})}$, the probability of an accepted step is altogether, by the law of total probability, $(1 \pm o(1))(1-1/n)^n = (1 \pm o(1))e^{-1}$. By similar arguments, the probability of a mutation improving the PRE-value by k at most $(1+o(1))e^{-1}/n^k$ and the probability of improving the SUFF-value is at least $(1-o(1))(e^{-1}/2)n^{-3/2}$ since there are $\binom{n^{1/4}}{2} = (1-o(1))n^{1/2}/2$ choices of probability at least e^{-1}/n^2 each.

We now consider a phase of $(3/4)emn$ steps. Using the bound on improving the SUFF-value, we expect $(1-o(1))(3/8)\sqrt{n}$ activated blocks. By Chernoff bounds, with overwhelming probability we have at least $\frac{1}{3}\sqrt{n}$ such blocks. The probability of improving the PRE-value by $k \geq 1$ is only $(1+o(1))e^{-1}n^{-k}$, amounting to an expected number of improvements by k of $(1+o(1))(3/4)mn^{1-k} = (1+o(1))(3/4)n^{2-k}$. Using Chernoff bounds and union bounds over all values of k such that $k = o(n)$, the probability of improving the PRE-value by at least $(9/10)m$ during the phase is $2^{-\Omega(n^{1/3})}$. □

4.6 Minimum Spanning Trees

Our self-adjusting s-bit flip mutation operator can also have advantages on classical combinatorial optimization problems. We reconsider the minimum spanning tree (MST) problem on which EAs and RLS were analyzed before [NW07]. The known bounds for the globally searching (1+1) EA are not tight. More precisely, they depend on $\log(w_{\max})$, the logarithm of the largest edge weight. This is different with RLS variants that flip only one or two bits due to an equivalence first formulated in [RKJ06]: if only up to two bits flip in each step, then the MST instance becomes indistinguishable from the MST instance formed by replacing all edge weights with their rank in their increasingly sorted sequence. This results in a tight upper bound of $O(m^2 \ln m)$, where m is the number of edges, for $\text{RLS}^{1,2}$, an algorithm that uniformly at random decides to flip either one or two uniformly chosen bits. Note that this algorithm is simply called RLS in [NW07]; however, to avoid confusion, we use the naming $\text{RLS}^{1,2}$ inspired by Neumann and Witt [NW10].

Although not spelled out by Neumann and Wegener [NW07], it is easy to see that the leading term in the polynomial $O(m^2 \ln m)$ is at most 2, i. e., that the expected time for $\text{RLS}^{1,2}$ to create an MST is at most $(2+o(1))m^2 \ln m$.

The leading coefficient 2 stems from the logarithm of sum of the weight ranks, which can be in the order of m^2; more precisely, following the multiplicative drift analysis in [DJW12, Theorem 15] with $\delta = 1/m^2$ and $\ln(X^{(t)}) \leq \ln(1+\cdots+m) \leq 2\ln m$ gives the bound $2m^2 \ln m$ on the expected time to transform an arbitrary spanning tree to an MST. With the fitness functions from [NW07], the expected time for $\mathrm{RLS}^{1,2}$ to arrive at a first spanning tree, starting from an arbitrary initial search point, is $O(m \log m)$ and therefore of lower order.

We will see that the leading coefficient of 2 can, in some sense, be avoided in our SD-$\mathrm{RLS}^{\mathrm{r}}$. The following theorem bounds the optimization time of SD-$\mathrm{RLS}^{\mathrm{r}}$ in the case that the algorithm has reached a spanning tree and the fitness function only allows spanning trees to be accepted. Since the expected time to find the first spanning tree is $O(m \log m)$ also for SD-$\mathrm{RLS}^{\mathrm{r}}$ with $R \geq m^{4+\epsilon}$, we do not consider this lower-order term further. However, our bound comes with an additional term related to the number of strict improvements. We will discuss this term and implications on the use of multiplicative drift analysis for SD-$\mathrm{RLS}^{\mathrm{r}}$ after the proof.

Theorem 4.10. *The expected optimization time of SD-$\mathrm{RLS}^{\mathrm{r}}$ with $R = m^5$ on the MST problem with m edges, starting with an arbitrary spanning tree, is at most*

$$(1+o(1))\big((m^2/2)(1+\ln(r_1+\cdots+r_m)) + (10m\ln m)\mathrm{E}(S)\big) \leq (1+o(1))\big(m^2\ln m + (10m\ln m)\mathrm{E}(S)\big),$$

where r_i is the rank of the ith edge in the sequence sorted by increasing edge weights, and $\mathrm{E}(S)$ is the total expected number of strict improvements until reaching the optimum conditioned on that the strength never exceeds 2 during this time.

Proof. We aim at using multiplicative drift analysis using $g(x) = \sum_{i=1}^{m} x_i r_i$ as potential function. As long as at most two bits flip, implying that $g(x)$ is indistinguishable from $\sum_{i=1}^{m} w_i x_i$ (cf. [RKJ06]), $g(x)$ can be seen as the canonical potential function used in multiplicative drift analysis with respect to the MST [DJW12, Th. 15].

Since SD-$\mathrm{RLS}^{\mathrm{r}}$ can be in the state of strength 1 but at least two bits must be flipped simultaneously to transform a spanning tree into another one, there is not always a positive drift to the optimum. However, at strength 1 no mutation is accepted since the fitness function from [NW07] gives a huge penalty to non-trees. Hence, our plan is to conduct the drift analysis conditioned on that the strength is exactly 2 and account for the steps spent at strength 1 separately. Cases where the radius exceeds 2 will be handled by an error analysis and a restart argument.

Let $X^{(t)} := g(x^t) - g(x_{\text{opt}})$ for the current search point $x^{(t)}$ and an optimal search point x_{opt}. Moreover, assume that the strength never exceeds 2 before finding an optimum. Then the algorithm behaves stochastically the same on the original fitness function and the potential function g. We obtain that $\mathrm{E}\left(X^{(t)} - X^{(t+1)} \mid X^{(t)}\right) \geq X^{(t)}/\binom{m}{2} \geq 2X^{(t)}/m^2$ since the g-value can be decreased by altogether $g(x^t) - g(x_{\text{opt}})$ via a sequence of at most $\binom{m}{2}$ disjoint two-bit flips; see also the proof of Theorem 15 in [DJW12] for the underlying combinatorial argument. Let T denote the number of steps at strength 2 until g is minimized, assuming no larger strength to occur. Using the multiplicative drift theorem, we have $\mathrm{E}(T) \leq (m^2/2)(1 + \ln(r_1 + \cdots + r_m)) \leq (m^2/2)(1 + \ln(m^2))$ and by the tail bounds for multiplicative drift [DG13] it holds that $\Pr\left(T > (m^2/2)(\ln(m^2) + \ln(m^2))\right) \leq e^{-\ln(m^2)} = 1/m^2$. Note that this bound on T is below the threshold for strength 2 since $\binom{m}{2} \ln R = (m^2 - m)/2 \ln(m^5) \geq (m^2/2)(4 \ln m)$ for m large enough. Hence, with probability at most $1/m^2$ the algorithm fails to find the optimum before the strength can change from 2 to a different value due to the threshold being exceeded.

Next, we bound the expected number of steps spent at larger strengths. Since each increase of the radius implies an unsuccessful phase at strength 2, the probability that radius r, where $3 \leq r \leq m/2$, is selected before finding the optimum is at most $(1/m^2)^{r-2}$. According to Lemma 4.1, the number of steps spent for each such radius is at most $\frac{m-(r-1)}{m-(2r-1)}\binom{m}{r}$. By the law of total probability, the expected number of steps at larger strengths than 2 is at most

$$\sum_{r=3}^{m/2} \frac{m-(r-1)}{m-(2r-1)} \binom{m}{r} \left(\frac{1}{m^2}\right)^{r-2} = o(m^2)$$

and contributes only a lower-order term captured by the $o(1)$ in the statement of the theorem. If the strength exceeds 2, we wait for it become 2 again and restart the previous drift analysis, which is conditional on strength at most 2. Since the probability of a failure is at most $1/m^2$, this accounts for an expected number of at most $1/(1 - m^{-2})$ restarts, which is $1 + o(1)$ as well.

It remains to bound the number of steps at strength 1 when the radius equals 1 or 2. For each strict improvement, the radius and consequently, the strength, is reset to 1. Also, when the radius is increased to 2, the algorithm uses strength 1 before increasing the strength to 2. Thereafter, $2m \ln R$ steps pass before the strength becomes 2 again. Hence, if the strength does not exceed 2 before the optimum is reached, this adds a term of $(2m \ln R)S$, where S is the number of strict improvements in the run, to the running time. The expected number of strict improvements is bounded by $\mathrm{E}(S)$, where we assume a random starting point of the algorithm and count the number of strict improvement after reaching the first tree. If an error occurs and the strength exceeds 2, the remaining expected number of strict improvements will not be bigger. □

The term $\mathrm{E}(S)$ appearing in the previous theorem is not easy to bound. If $\mathrm{E}(S) = o(m)$, the upper bound suggests that SD-RLS$^{\mathrm{p}}$ may be more efficient than the classical RLS^{1,2} algorithm; with the caveat that we are talking about upper bounds only. However, it is not difficult to find examples where $\mathrm{E}(S) = \Omega(m)$, e. g., on the worst-case graph used for the lower-bound proof in [NW07], which we will study below experimentally, and we cannot generally rule out that $\mathrm{E}(S)$ is asymptotically bigger than m on certain instances. However, empirically SD-RLS$^{\mathrm{r}}$ can be faster than RLS^{1,2} and the (1+1) EA on MST instances, as we will see in Section 4.7. In any case, although the algorithm can search globally, the bound in Theorem 4.10 does not suffer from the $\log(w_{\max})$ factor appearing in the analysis of the (1+1) EA.

Technically, the proof of Theorem 4.10 shows that it is not straightforward to apply drift analysis in algorithms using stagnation detection. More precisely, since the drift of SD-RLS$^{\mathrm{r}}$ not only depends on the current search point, but also on the current strength, it may be challenging to derive good drift bounds. In the proof, we could address this challenge by essentially ignoring all improvements happening at strengths different from 2.

We also considered designing variants of SD-RLS$^{\mathrm{r}}$ that do not reset the strength to 1 after each strict improvement and would therefore, be able to work with strength 2 for a long while on the MST problem. However, such an approach is risky in scenarios where, e. g., both one-bit flips and two-bit flips are possible and one-bit flips should be exploited for the sake of efficiency. Instead, we think that a combination of stagnation detection and selection hyperheuristics [War19] based on the s-bit flip operator or the learning mechanism from [DDY16b], which performs very well on the MST, would be more promising here.

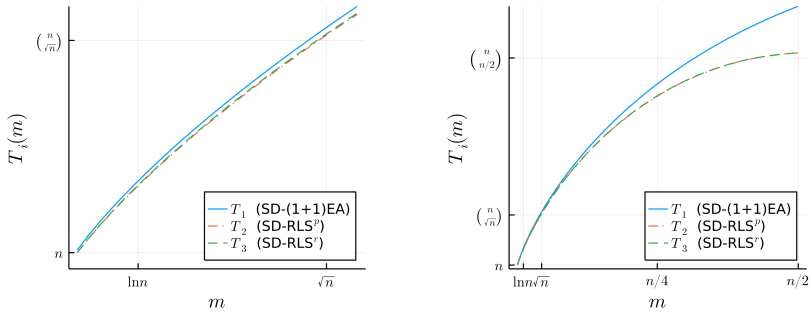
4.7 Experiments

In this section, we present the results of the experiments[1] conducted to see the empirical performance of the proposed algorithms. On the problems we will study in this section, the individual gap of a given search point equals its fitness level gap, so we use the term gap instead for short.

In order to investigate the escaping time of different algorithms using stagnation detection, in a numerical simulation, we compared the expressions

- $T_1(m) \coloneqq \sum_{s=1}^{m-1} 2(en/s)^s \ln(en^6) + (en/m)^m$,

[1] `https://github.com/DTUComputeTONIA/SDRLS`



(a) For small gap sizes

(b) For gap sizes at most $n/2$

Figure 4.1: Evaluations of the functions T_1, T_2, and T_3 with a fixed but large n ($n = 500$) and $R = n^5$ as some estimators for the number of steps the mentioned algorithms take to escape local optima of different gap sizes m. The data is represented in logarithmic scale.

- $T_2(m) \coloneqq \sum_{s=1}^{m-1} \binom{n}{s} \ln n^5 + \binom{n}{m}$,
- $T_3(m) \coloneqq \sum_{r=1}^{m-1} \sum_{s=1}^{r} \binom{n}{s} \ln n^5 + \sum_{s=1}^{m-1} \binom{n}{s} \ln n^5 + \binom{n}{m}$,

representing roughly the expected number of steps of SD-(1+1) EA[2], SD-RLS^p, and SD-RLS^r, respectively, to leave a local optimum with the gap size m in finite time. We considered $R = n^5$, which satisfies the recommendations for the parameter R in different algorithms. For the sake of simplicity, we did not consider the number of iterations in the case of missing the improvement at the right strength in the estimators T_i. However, since those events only contribute to lower-order terms (except for SD-RLS^p, which results in infinite runtime), we believe that ignoring them does not change the results significantly. The results can be seen in Figure 4.1, where the x-axis shows the asymptotic growth rate of m, i. e., the gap size, and the y-axis indicates the value of each aforementioned expression, i. e., the estimated escaping time.

Although in SD-RLS^r (T_3), the algorithm looks into smaller strengths each time to guarantee finite expected running time, there is no apparent increase in the bounds on improvement times compared to SD-RLS^p (T_2). However, the bounds for both variants outperform the bound for the SD-(1+1) EA (T_1).

[2]Note that in this section, we always use the threshold value from the preliminary version of the SD-(1+1) EA presented in GECCO 2020, i. e., $2(en/r)^r \ln(nR)$ for the strength r.

We also conducted two experiments to see the performance of the proposed algorithms for small problem dimensions. This experimental design was employed because our theoretical results are asymptotic.

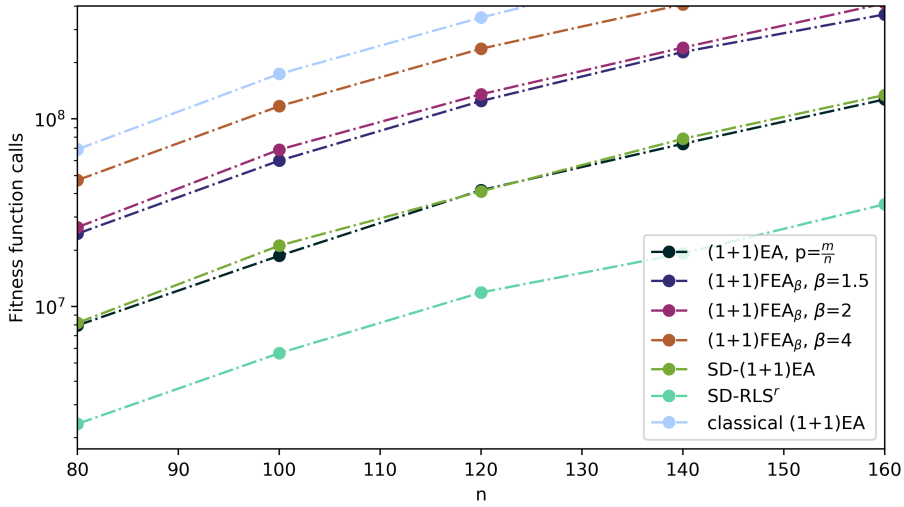


Figure 4.2: Average number of fitness calls (over 1000 runs) the mentioned algorithms took to optimize $\textsc{Jump}_4$.

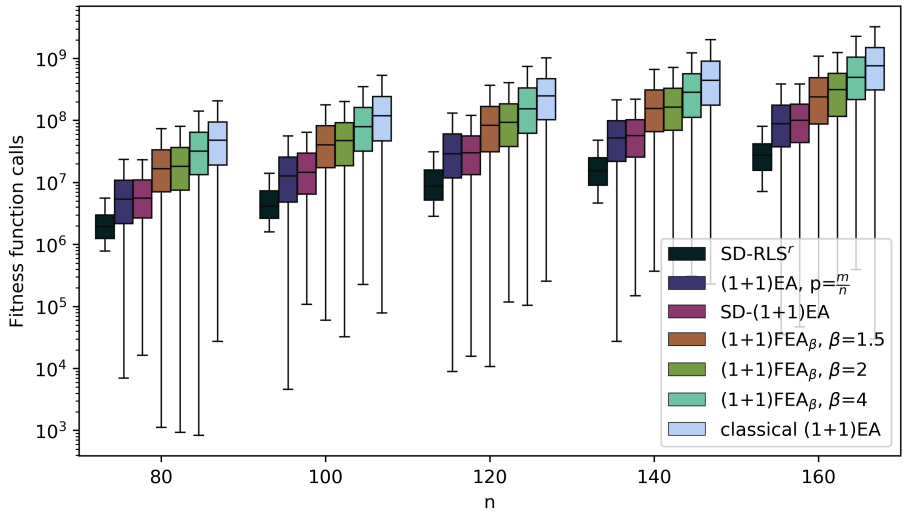


Figure 4.3: Box plots comparing number of fitness calls (over 1000 runs) the mentioned algorithms took to optimize $\textsc{Jump}_4$.

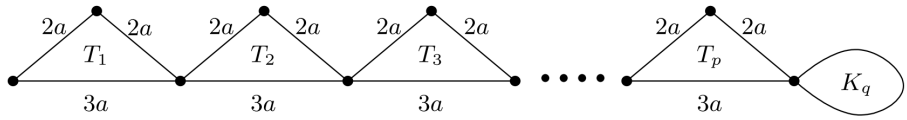


Figure 4.4: Example graph TG with $p = n/4$ connected triangles and a complete graph on q vertices with edges of weight 1. The source of the figure is [NW07].

In the first experiment, we ran an implementation of Algorithm 12 (SD-RLS$^{\mathrm{r}}$) on the Jump fitness function with jump size $m = 4$ and n varying from 80 to 160. We compared our algorithm against the (1+1) EA with standard mutation rate $1/n$, the (1+1) EA with mutation probability m/n, Algorithm (1+1) FEA$_\beta$ from [DLMN17] with three different $\beta = \{1.5, 2, 4\}$, and the SD-(1+1) EA presented in [RW22a]. In Figure 4.2, we observe that SD-RLS$^{\mathrm{r}}$ outperforms the rest of the algorithms based on average values of 1000 independent runs.

In the second experiment, we ran an implementation of four algorithms SD-RLS$^{\mathrm{r}}$, (1+1) FEA$_\beta$ with $\beta = 1.5$ from [DLMN17], the standard (1+1) EA and RLS^{1,2} from [NW07] on the MST problem with the fitness function from [NW07] for two types of graphs called TG and Erdős–Rényi.

The graph TG with n vertices and $m = 3n/4 + \binom{n/2}{2}$ edges contains a sequence of $p = n/4$ triangles which are connected to each other, and the last triangle is connected to a complete graph on $q = n/2$ vertices. Regarding the weights, the edges of the complete graph have the weight 1, and we set the weights of edges in triangles to $2a$ and $3a$ for the side edges and the main edge, respectively (See Figure 4.4). In this paper, we consider $a = n^2$. The graph TG is used for estimating lower bounds on the expected runtime of the (1+1) EA and RLS in the literature [NW07]. In this experiment, we use $n = \{24, 36, 48, 60\}$. As can be seen in Figure 4.5, the (1+1) FEA$_\beta$ is faster than the rest of the algorithms.

An Erdős–Rényi graph as defined in [ER60] is a random graph of size n such that each edge appears with probability p. We produced several random Erdős–Rényi graphs with $p = (2 \ln n)/n$ and assigned each edge an integer weight in the range $[1, n^2]$ uniformly at random. We also checked that the graphs obtained were connected. Then we ran the implementation to find the MST of these graphs. The obtained results can be seen in Figure 4.6. As we discussed in Section 4.6, SD-RLS$^{\mathrm{r}}$ is outperformed by the (1+1) EA and RLS^{1,2} on the MST problems when the number of strict improvements in SD-RLS$^{\mathrm{r}}$ is large. However, since relatively fewer strict improvements seem to be sufficient on the TG graphs than on the Erdős–Rényi graphs, we can see in Figure 4.5 that SD-RLS$^{\mathrm{r}}$ is slightly better than the (1+1) EA and RLS^{1,2}.

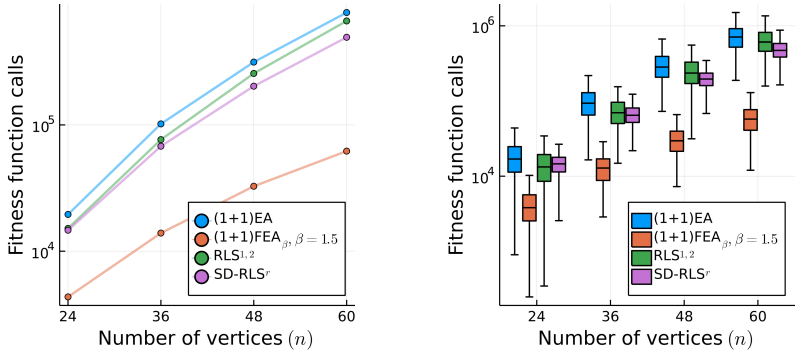


Figure 4.5: Average number of fitness calls (over 400 runs) the mentioned algorithms took to optimize the fitness function MST of the TG graphs.

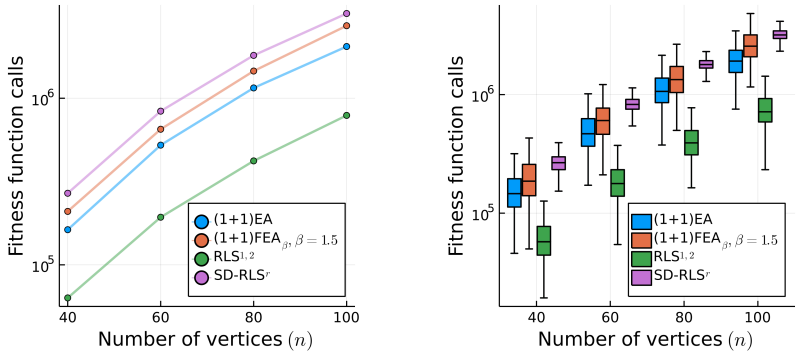


Figure 4.6: Average number of fitness calls (over 400 runs) the mentioned algorithms took to optimize the fitness function MST of the Erdős–Rényi graphs.

Conclusions

We have transferred stagnation detection, previously proposed for EAs with standard bit mutation, to the operator flipping exactly s uniformly randomly chosen bits as typically encountered in randomized local search. We have also introduced techniques that make the algorithm robust if it, due to its randomized nature, misses the right number of bits flipped, and analyzed scenarios where global mutations are still preferable.

Through theoretical runtime analyses, we have shown that this combination of stagnation detection and local search can leave local optima located at Hamming distance k from the closest improvement in expected time roughly $\binom{n}{k}$, i. e., the size of the k-bit flip neighborhood, which outperforms the previously considered variant of stagnation detection with global mutation. Moreover, in experimental studies, our proposed algorithms outperform some well-known algorithms on Jump functions. However, it can be seen that on problems where many strict improvements are needed, e. g., MST problems with dense instances, the number of iterations that the robust variant SD-RLS$^{\mathrm{r}}$ spends at too small strengths has a considerable negative impact on the total optimization time.

In the future, we would like to investigate stagnation detection more thoroughly on instances of classical combinatorial optimization problem like the minimum spanning tree problem, for which the present paper only gives preliminary but promising results.

Acknowledgement

This work was supported by a grant by the Danish Council for Independent Research (DFF-FNU 8021-00260B).

Chapter 5

Paper C: Stagnation Detection in Highly Multimodal Fitness Landscapes

Stagnation detection has been proposed as a mechanism for randomized search heuristics to escape from local optima by automatically increasing the size of the neighborhood to find the so-called gap size, i. e., the distance to the next improvement. Its usefulness has mostly been considered in simple multimodal landscapes with few local optima that could be crossed one after another. In multimodal landscapes with a more complex location of optima of similar gap size, stagnation detection suffers from the fact that the neighborhood size is frequently reset to 1 without using gap sizes that were promising in the past.

In this paper, we investigate a new mechanism called *radius memory* which can be added to stagnation detection to control the search radius more carefully by giving preference to values that were successful in the past. We implement this idea in an algorithm called SD-RLS$^{\mathrm{m}}$ and show compared to previous variants of stagnation detection that it yields speed-ups for linear functions under uniform constraints and the minimum spanning tree problem. Moreover, its running time does not significantly deteriorate on unimodal functions and a generalization of the Jump benchmark. Finally, we present experimental results carried out to study SD-RLS$^{\mathrm{m}}$ and compare it with other algorithms.

5.1 Introduction

The theory of self-adjusting evolutionary algorithms (EAs) is a research area that has made significant progress in recent years [DD20]. For example, a self-adjusting choice of mutation and crossover probability in the algorithm so-called $(1+(\lambda,\lambda))$ GA allows an expected optimization time of $O(n)$ on OneMax, which is not possible with any static setting [DDE15, DD18]. Many studies focus on unimodal problems, while self-adjusting EAs for multimodal problems in discrete search spaces, more precisely for pseudo-Boolean optimization, were investigated only recently from a theoretical runtime perspective. *Stagnation detection* proposed in [RW20b] addresses a shortcoming of classical self-adjusting EAs, which try to learn promising parameter settings from fitness improvements. Despite the absence of a fitness improvements when the best-so-far solution is at a local optimum, stagnation detection learns from the number of unsuccessful steps and adjusts the mutation rate if this number exceeds a certain threshold. Thanks to this mechanism, the so-called SD-(1+1) EA proposed in [RW20b] optimizes the classical Jump function with n bits and gap size m in expected time $O((en/m)^m)$, which corresponds asymptotically to the best possible time achievable through standard bit mutation, more precisely when each bit is flipped independently with probability m/n. It is worth pointing out that stagnation detection does not have any prior information about the gap size m.

Although leaving a local optimum requires a certain number of bits to be flipped simultaneously, which we call the *gap size*, the SD-(1+1) EA mentioned above still performs independent bit flips. Therefore, even for the best setting of the mutation rate, only the expected number of flipping bits equals the gap size while the actual number of flipping bits may be different. This has motivated Rajabi and Witt [RW21b] to consider the k-bit flip operator flipping a uniform random subset of k bits as known from randomized local search (RLS) [DD18] and to adjust k via stagnation detection. Compared to the SD-(1+1) EA, this allows a speed-up of $(\frac{ne}{m})^m/\binom{n}{m}$ (up to lower-order terms) on functions with gap size m and a speed-up of up to roughly $e = 2.718\ldots$ on unimodal functions while still being able to search globally.

Rajabi and Witt [RW21b] emphasize that their RLS with self-adjusting k-bit flip operator resembles variable neighborhood search [HM18] but features less determinism by drawing the k bits to be flipped uniformly at random instead of searching the neighborhood in a fixed order. The random behavior still maintains many characteristics of the original RLS, including independent stochastic decisions which ease the runtime analysis. If the bit positions to be flipped follow a deterministic scheme as in quasirandom EAs [DFW10], dependencies complicate the analysis and make it difficult to apply tools like drift analysis. However, a drawback of the randomness is that the independent, uniform choice

of the set of bits to be flipped leaves a positive probability of missing an improvement within the given time a specific parameter value k is tried. Therefore, the first RLS variant with stagnation detection proposed in [RW21b] and called SD-RLS there has infinite expected runtime in general, but is efficient with high probability, where the success probability of the algorithm is controlled via the threshold value for the number of steps without fitness improvement that trigger a change of k. We remark here that the problem with infinite runtime is not existent with the independent bit flips as long as each bit is flipped with probability in the open interval $(0, 1)$.

In this paper, we denote by SD-RLS$^{\mathrm{p}}$ the simple SD-RLS just proposed. To guarantee finite expected optimization time, Rajabi and Witt [RW21b] introduce a second variant that repeatedly returns to lower so-called (mutation) strengths, i. e., number of bits flipped, while the algorithm is still waiting for an improvement. The largest neighborhood size (i. e., number of bits flipped) is denoted as *radius* r and, in essence, the strength s is decreased in a loop from r to 1 before the radius is increased. Interestingly, the additional time spent at exploring smaller strengths in this loop, with the right choice of phase lengths, contributes only a lower-order term to the typical time that SD-RLS$^{\mathrm{p}}$ has in the absence of errors. The resulting algorithm (Algorithm 14) is called SD-RLS* in [RW21b], but in this paper referred to by SD-RLS$^{\mathrm{r}}$, where the label r denotes *robust*.

As already explained in [RW21b], SD-RLS$^{\mathrm{r}}$ (and also the plain SD-RLS$^{\mathrm{p}}$) return to strength 1 after every fitness improvement and try this strength for a sufficiently large time to find an improving one-bit flip with high probability. This behavior can be undesired on highly multimodal landscapes where progress is typically only made via larger strengths. As an example, the minimum spanning tree (MST) problem as originally considered for the (1+1) EA and an RLS variant in [NW07] requires two-bit flips to make progress in its crucial optimization phase. Both theoretically and experimentally, Rajabi and Witt [RW21b] observed that SD-RLS$^{\mathrm{r}}$ is less efficient than the RLS variant from [NW07] since low, useless strengths (here 1) are tried for a too long period of time. On the other hand, it can also be risky exclusively to proceed with the strength that was last found to be working if the fitness landscape becomes easier at some point and progress can again be made using smaller strengths.

In this paper, we address this trade-off between exploiting high strengths that were necessary in the past and again trying smaller strengths for a certain amount of time. We propose a mechanism called *radius memory* that uses the last successful strength value to assign a reduced budget of iterations to smaller strengths. This budget is often much less than the number stereotypically tried in SD-RLS$^{\mathrm{r}}$ after every fitness improvement. However, the budget must be balanced carefully to allow the algorithm to adjust itself to gap sizes becoming

smaller over the run of the algorithm. Our choice of budget is based on the number of iterations (which is the same as the number of fitness evaluations) passed to find the latest improvement and assigns the same combined amount of time, divided by $\ln n$, to smaller strengths tried afterwards.

This choice, incorporated in our new algorithm SD-RLS$^{\mathrm{m}}$, basically limits the time spent at unsuccessful strengths by less than the waiting time for an improvement with the last successful strength but is still big enough to adjust to smaller strength sufficiently quickly. On the one hand, it (up to lower-order terms) preserves the runtime bounds on general unimodal function classes and jump functions shown for SD-RLS$^{\mathrm{r}}$ in [RW21b]. On the other hand, it significantly reduces the time for the strength to return to larger values on two highly multimodal problems, namely optimization of linear functions under uniform constraints and the MST. Although these ideas are implemented in a simple RLS maintaining one individual only, we implicitly consider stagnation detection as a module that can be added to other algorithms as shown in [RW20b] and very recently in [DZ21] for multi-objective optimization. Concretely, we could also use the stagnation detection with radius memory in population-based algorithms.

This paper is structured as follows. In Section 5.2, we define the algorithms considered and collect some important technical lemmas. Section 5.3 presents time bounds for the new algorithm SD-RLS$^{\mathrm{m}}$ to leave local optima and applies these to obtain bounds on the expected optimization time on unimodal and jump functions. Moreover, it includes in Lemma 5.5 the crucial analysis of the time for the strength to settle at smaller values when an improvement is missed. Thereafter, these results are used in Section 5.4 to analyze SD-RLS$^{\mathrm{m}}$ on linear functions under uniform constraints and to show a linear-time speedup compared to the SD-RLS$^{\mathrm{r}}$ algorithm in [RW21b]. Section 5.5 shows that SD-RLS$^{\mathrm{m}}$ optimizes MST instances on graphs with m edges in expected time at most $(1 + o(1))m^2 \ln m$, which is by an asymptotic factor of 2 faster than the bound for RLS^{1,2} from [NW07] and represents, to the best of our knowledge, the first asymptotically tight analysis of a globally searching (1+1)-type algorithm on the problem. In Section 5.6, we present an example where the radius memory is detrimental and leads to exponential optimization time with probability $1 - o(1)$ while the original SD-RLS$^{\mathrm{r}}$ from [RW21b] is efficient. Section 5.7 presents experimental supplements to the analysis of SD-RLS$^{\mathrm{r}}$ and SD-RLS$^{\mathrm{m}}$ and comparisons with other algorithms from the literature before we finish with some conclusions.

5.2 Preliminaries

5.2.1 Algorithms

We start by describing a class of classical RLS algorithms and the considered extensions with stagnation detection. Algorithm 13 is a simple hill climber that uses a static strength s and always flips s bits uniformly at random. The special case where $s = 1$, i. e., using one-bit flips, has been investigated thoroughly in the literature [DD16] and is mostly just called RLS.

Algorithm 13: RLS with static strength s for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$

```
Select x uniformly at random from {0,1}^n;
for t ← 1, 2, ... do
    Create y by flipping s bits in a copy of x;
    if f(y) ≥ f(x) then
        x ← y;
```

The algorithm RLS^{1,2} (which also is just called RLS in [NW07]) is an extension of this classical RLS (i. e., Algorithm 13 with strength 1) choosing strength $s \in \{1,2\}$ uniformly before flipping s bits. This extension is crucial for making progress on the MST problem, as further explained in Section 5.5.

In [RW21b], RLS is enhanced by stagnation detection, leading to Algorithm 14. In a nutshell, the algorithm increases its strength after a certain number of unsuccessful steps according to the threshold value $\binom{n}{s} \ln R$ which has been chosen to bound the so-called failure probability at strength s, i. e., the probability of not finding an improvement at Hamming distance s, by at most $\left(1-1/\binom{n}{s}\right)^{\binom{n}{s}\ln R} \leq 1/R$. It also incorporates logic to return to smaller strengths repeatedly by maintaining the so-called radius value r. All variables and parameters will be discussed in detail below when we come to our extension with radius memory. Algorithm 14 is called SD-RLS* in [RW21b] since that paper also discusses a simpler variant called SD-RLS without the logic related to the radius variable. However, that variant is not robust and has infinite expected optimization time in general even on unimodal problems. We call Algorithm 14 SD-RLS$^{\mathrm{r}}$ since, as argued in [RW21b], the radius makes the algorithm *robust*.

In the following, we present in Algorithm 15 the new algorithm SD-RLS$^{\mathrm{m}}$ using stagnation detection and radius memory. It extends SD-RLS$^{\mathrm{r}}$ by adding logic for setting the helper variable B and by using B to minimize the original thresh-

Algorithm 14: RLS with robust stagnation detection (SD-RLS$^{\mathrm{r}}$) for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$

```
Select x uniformly at random from {0,1}^n;
r ← 1, s ← 1, u ← 0;
for t ← 1, 2, ... do
    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 ← 1, s ← 1, u ← 0;
    else if f(y) = f(x) and r = 1 then
        x ← y;
    if u > (n choose s) ln R then
        if s = 1 then
            if r < n/2 then r ← r + 1 else r ← n;
            s ← r ;
        else
            s ← s − 1;
        u ← 0;
```

old $\binom{n}{s} \ln R$ for the number of unsuccessful steps with $\frac{u}{(\ln n)(r-1)}$, where u is explained below. Another minor change is that it increases the strength from 1 to the radius r instead of decreasing it for the sake of simplicity of the new algorithm and the proofs. We describe the algorithm in more detail now.

After a strict improvement with strength s (which becomes the initial radius r for the next search point), the algorithm uses all strengths $s' < r$ for

$$\min\left\{\frac{u}{(\ln n)(r-1)}, \binom{n}{s'} \ln R\right\}$$

attempts, where u is the value of the counter at the time that the previous improvement happened. Once the current strength becomes equal to the current radius, the threshold becomes $\min\{\infty, \binom{n}{s'} \ln R\} = \binom{n}{s'} \ln R$ for the rest of iterations with the current search point. Therefore, the cap at $\frac{u}{(\ln n)(r-1)}$ is only effective as long as the current radius equals r and the current strengths are smaller than r.

For technical reasons, the radius increases directly to n when it has passed $n/2$. Moreover, as another technical detail, we accept search points of equal fitness only if the current radius is one (leading to the same acceptance behavior as in

classical RLS, see Algorithm 13), whereas only strict improvements are accepted at larger radii.

The factor $1/\ln n$ appearing in the first argument of the minimum

$$\min\left\{\frac{u}{(\ln n)(r-1)}, \binom{n}{s'}\ln R\right\}$$

is a parameter choice that has turned out robust and useful in our analyses. The choice $\frac{u}{r-1}$, i. e., an implicit constant of 1, could seem more natural here since then the algorithm would look at smaller strengths as often as the last successful strength was tried; however, this would make our forthcoming bounds worse by a constant factor.

As mentioned above, stagnation detection has also a parameter R to bound the probability of failing to find an improvement at the "right" strength. We formally prove in Lemma 5.3 that the probability of not finding an improvement where there is a potential of making progress is at most $1/R$. The recommendation of R for SD-RLS$^{\mathrm{r}}$ in [RW21b] is still valid for SD-RLS$^{\mathrm{m}}$, i. e., $R \geq n^{3+\epsilon} \cdot |\operatorname{Im} f|$ for a constant ϵ (where $\operatorname{Im} f$ is the image set of f), resulting in that the probability of ever missing an improvement at the right strength is sufficiently small throughout the run. However, in this paper, by improving some analyses, we recommend a tighter value for R, namely $R \geq \max\{S, n^{3+\epsilon}\}$ for an arbitrary constant $\epsilon > 0$ where S is an upper bound on the number of strict improvements during the run. Obviously, we can always choose $S = |\operatorname{Im} f|$.

The *runtime* or the *optimization time* of a search heuristic on a function f is the first point time t where a search point of optimal fitness has been created; often the expected runtime, i. e., the expected value of this time, is analyzed.

5.2.2 Mathematical tools

In the following lemma, which has been taken from [RW21b], we have some combinatorial inequalities that will be used in the analyses of the algorithms. Part 1 in Lemma 5.1 seems to be well known and has already been proved in [Lug17] and is also a consequence of Lemma 1.10.38 in [Doe20b]. Part 2 follows from elementary manipulations.

Lemma 5.1 (Lemma 1 in [RW21b])**.** *We have*

1. $\sum_{i=1}^{m}\binom{n}{i} \leq \frac{n-(m-1)}{n-(2m-1)}\binom{n}{m}$ *for any integer* $m \leq n/2$,
2. $\binom{n}{M} \leq \binom{n}{m}\left(\frac{n-m}{m}\right)^{M-m}$ *for* $m < M < n/2$.

Algorithm 15: RLS with robust stagnation detection and radius memory mechanism (SD-RLS$^{\mathrm{m}}$) for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$

Select x uniformly at random from $\{0,1\}^n$;
$r \leftarrow 1$, $s \leftarrow 1$, $u \leftarrow 0$, $B \leftarrow \infty$;
for $t \leftarrow 1, 2, \ldots$ **do**
  Create y by flipping s bits in a copy of x uniformly at random;
  $u \leftarrow u + 1$;
  if $f(y) > f(x)$ **then**
    $x \leftarrow y$;
    $r \leftarrow s$, $s \leftarrow 1$;
    if $r > 1$ **then** $B \leftarrow \frac{u}{(\ln n)(r-1)}$ **else** $B \leftarrow \infty$;
    $u \leftarrow 0$;
  else if $f(y) = f(x)$ ***and*** $r = 1$ **then**
    $x \leftarrow y$;
  if $u > \min\{B, \binom{n}{s} \ln R\}$ **then**
    if $s = r$ **then**
      if $r < n/2$ **then** $r \leftarrow r + 1$ **else** $r \leftarrow n$;
      $s \leftarrow 1$;
    else
      $s \leftarrow s + 1$. **if** $s = r$ **then** $B \leftarrow \infty$;
    $u \leftarrow 0$;

Proof. For Part 1, we use the following proof due to [Lug17]. Through the equation $\binom{n}{k-1} = \frac{k}{n-k+1}\binom{n}{k}$, which comes from the definition of the binomial coefficient and the summation formula for geometric series, we achieve the following result:

$$\begin{aligned}\frac{\sum_{i=1}^{m}\binom{n}{i}}{\binom{n}{m}} &= \frac{\binom{n}{m}}{\binom{n}{m}} + \frac{\binom{n}{m-1}}{\binom{n}{m}} + \cdots + \frac{\binom{n}{1}}{\binom{n}{m}} \\ &< 1 + \frac{m}{n-m+1} + \frac{m(m-1)}{(n-m+1)(n-m+2)} + \ldots \\ &\leq 1 + \frac{m}{n-m+1} + \left(\frac{m}{n-m+1}\right)^2 + \cdots = \frac{n-(m-1)}{n-(2m-1)}.\end{aligned}$$

Regarding Part 2, for $t \geq m$, by using $\binom{n}{m} = \frac{n-m+1}{m}\binom{n}{m-1}$, we have

$$\binom{n}{M} = \frac{(n-M+1)\ldots(n-M+t)}{M\ldots(M-t+1)}\binom{n}{M-t} \leq \left(\frac{n-M+t}{M-t}\right)^t \binom{n}{M-t}.$$

Thus, by setting $m = M - t$, we obtain the statement. □

5.3 Analysis of the Algorithm SD-RLS$^{\mathrm{m}}$

In this section, we shall show bounds on the optimization time of SD-RLS$^{\mathrm{m}}$ in addition to useful technical lemmas used in different analyses in the paper.

5.3.1 Expected Times to Leave a Search Point

In this subsection, we will prove some bounds on the time to leave a search point that has a Hamming distance larger than 1 to all improvements. Let us define by the *epoch* of x the sequence of iterations where x is the current search point. In contrast to the previously proposed algorithm SD-RLS$^{\mathrm{r}}$ in [RW21b], the optimization time of SD-RLS$^{\mathrm{m}}$ for making progress with the current search point x is also dependent to the progress in the epoch of the second-to-last different search point, i. e., the parent of x. In detail, the algorithm in epoch x starts with parameters r_0 and B_0, which are set to the strength escaping the parent of x and the number of fitness calls at that strength divided by $(r_0 - 1)\ln n$, respectively. Therefore, to analyze the running time, we also need to consider those parameters. Hereinafter, we define T_{x,r_0,B_0} as the number of steps SD-RLS$^{\mathrm{m}}$ takes to find an improvement from the current search point x with starting radius r_0 (i. e., at the beginning of the epoch) and budget B_0 in the current epoch.

We recall the so-called *gap* of the point $x \in \{0,1\}^n$ defined in [RW20b] as the minimum Hamming distance to points with the strictly larger fitness function value. Formally, $\mathrm{gap}(x^*) = \infty$ where x^* is an optimum, and for the rest of the points, we define

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

It is not possible to make progress by flipping less than $\mathrm{gap}(x)$ bits of the current search point x, but if the algorithm uses the s-flip with $s = \mathrm{gap}(x)$, it can make progress with a positive probability.

In order to estimate the escape time bounds, we consider two cases where $\mathrm{gap}(x) \geq r_0$ and where $\mathrm{gap}(x) < r_0$. In the first case, it costs B_0 fitness function calls for the algorithm to increase the radius r to r_0. Then, the analysis of the rest of the iterations is the same as Theorem 3 for the algorithm SD-RLS$^{\mathrm{r}}$ (the algorithm without radius memory) in [RW21b]. Obviously, the algorithm is asymptotically as efficient as SD-RLS$^{\mathrm{r}}$ in this case. However, in the second case where $\mathrm{gap}(x) < r_0$, the proposed algorithm can be outperformed by SD-RLS$^{\mathrm{r}}$ since if it fails to improve at every radius, the algorithm meets larger strengths, which are costly. It means that although the gap size of the current search

point is less than its parent, the escape time is the same or even longer in the worst case scenario. However, in the rest of the paper, we show that this is not harmful to the optimization time because it is captured by the escape time of the previous epoch, and after a short time, the radius "recovers" and is set to the gap of the current search point, as proved in Lemma 5.5 below.

Concretely, we present the next theorem presenting escape time bounds and prove it by the end of this subsection.

Theorem 5.2. *Let $x \in \{0,1\}^n$ be the first search point after a strict improvement of SD-RLS^m with starting radius r_0, budget B_0, and $R \geq n^{3+\epsilon}$ for an arbitrary constant $\epsilon > 0$ on a pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$. Define T_{x,r_0,B_0} as the time to create a strict improvement. Let $m = \operatorname{gap}(x)$. Then, for an arbitrary constant $\hat{\epsilon}$, we have*

$$\mathrm{E}\left(T_{x,r_0,B_0}\right) \leq \begin{cases} \binom{n}{m}\left(1 + O(\frac{m \ln R}{n})\right) & \text{if } r_0 \leq m \leq (1-\hat{\epsilon})n/2, \\ O\left(\binom{n}{r_0} \ln R\right) & \text{if } m < r_0 \leq (1-\hat{\epsilon})n/2, \\ O(2^n n \ln R) & \text{otherwise.} \end{cases}$$

For proving Theorem 5.2, we need some definitions and lemmas as follows. Let **phase** r consist of all points in time where radius r is used in the algorithm. Let E_r be the event of **not** finding the optimum within phase r. In Lemma 5.3, we show that the failure probability is at most $1/R$ in phases r with $m \leq r < n/2$ and zero in the last phase (i. e., at radius n). In the following lemma, we show that in each radius which is at least the gap size, the algorithm makes progress with high probability.

Lemma 5.3. *Let $x \in \{0,1\}^n$ be the current search point of SD-RLS^m on a pseudo-Boolean fitness function $f\colon \{0,1\}^n \to \mathbb{R}$ and let $m = \operatorname{gap}(x)$ and $B \geq \binom{n}{m} \ln R$. Then*

$$\Pr\left(E_r\right) \leq \begin{cases} 1 & \text{if } r < m, \\ \frac{1}{R} & \text{if } m \leq r < \frac{n}{2}, \\ 0 & \text{if } r = n. \end{cases}$$

Proof. According to the definition of the gap, the algorithm can not make progress where the strength is less than the gap size, so in this case, $\Pr\left(E_r\right) = 1$. Now, assume $r \geq m$. During phase r (i. e., at radius r), the algorithm spends $\min\{B, \binom{n}{m} \ln R\} = \binom{n}{m} \ln R$ steps at strength m until it changes the strength or radius. Then, the probability of not improving at strength $s = m$ is at most

$$\Pr\left(E_r\right) = \left(1 - \binom{n}{m}^{-1}\right)^{\binom{n}{m} \ln R} \leq \frac{1}{R}.$$

During phase n, the algorithm does not change the radius anymore, and it continues to flip s bits with different s containing m until making progress, so the probability of eventually failing to find the improvement in this phase is 0. □

Let E_i^j for $j > i$ be the event of not finding the optimum during phases i to j. In other words, $E_i^j = E_i \cap \cdots \cap E_j$. Also, we remark that $E_i^i = E_i$ and $E_i^j = \emptyset$ for $j < i$.

The next lemma is used to analyze situations where the algorithm does not find an improvement in the first k phases, where k is some number depending on the application of the lemma. In the proof of the lemma, we pessimistically do not consider possible improvements at distances larger than the gap size.

Lemma 5.4. *Let $x \in \{0,1\}^n$ with $m = \operatorname{gap}(x) < n/2$ be the current search point of SD-RLS^m with $R \geq n^{3+\epsilon}$ for an arbitrary constant $\epsilon > 0$ on a pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$. Given r_0 as starting radius and B_0 as budget, T'_{x,r_0,B_0} is defined as the number of steps to find a strict improvement. Let $k < (1-\hat{\epsilon})n/2$ for an arbitrary constant $0 < \hat{\epsilon} < 1$ such that $k \geq m-1$ and $k \geq r_0$ in the case that $r_0 > m$. In other words, if the first k phases do not find a strict improvement, the algorithm uses the threshold $\binom{n}{m} \ln R$ at strength m in phase $k+1$. Then we have*

$$\mathrm{E}\left(T'_{x,r_0,B_0} \mid E_1^k\right) \leq \binom{n}{m} + O\left(\frac{k \ln R}{n}\binom{n}{k+1}\right),$$

where E_i^j is the event of not finding an optimum during phases i to j (included).

Proof. Using the law of total probability, we have

$$\mathrm{E}\left(T'_{x,r_0,B_0} \mid E_1^k\right) = \underbrace{\sum_{i=k+1}^{\lceil n/2\rceil - 1} \mathrm{E}\left(T'_x \mid E_{k+1}^{i-1} \cap \overline{E_i}\right) \Pr\left(E_{k+1}^{i-1} \cap \overline{E_i}\right)}_{=:S_1}$$
$$+ \underbrace{\mathrm{E}\left(T'_x \mid E_{k+1}^{\lceil n/2\rceil - 1} \cap \overline{E_n}\right) \Pr\left(E_{k+1}^{\lceil n/2\rceil - 1} \cap \overline{E_n}\right)}_{=:S_2}.$$

To interpret the last formula, we recall phase r as all points of time where radius r is used. In each term in S_1, variable i represents the phase in which the algorithm makes progress (i. e., $\overline{E_i}$) and not in smaller phases, i. e., phases from $k+1$ to $i-1$ (i. e., E_{k+1}^{i-1}). Thus, all cases where making improvement happens in one of the phases ranging from $k+1$ to $\lceil n/2\rceil - 1$ are considered in

S_1, and the last case of phase n is computed in S_2. In this manner, we consider all possible cases of success.

In order to estimate $\Pr\left(E_{k+1}^{i-1} \cap \overline{E_i}\right)$, because of the assumptions on k, the budget B_0 is not effective in the threshold value. Hence we can use Lemma 5.3, which results in

$$\Pr\left(E_{k+1}^{i-1} \cap \overline{E_i}\right) < \Pr\left(E_{k+1}^{i-1}\right) = \prod_{j=k+1}^{i-1} \Pr\left(E_j\right) < R^{-(i-k-1)}.$$

Note that we consider only improvements at the Hamming distance of the gap size. Now, for S_1, we compute

$$\begin{aligned}
&\sum_{i=k+1}^{\lceil n/2 \rceil - 1} \mathrm{E}\left(T'_x \mid E_{k+1}^{i-1} \cap \overline{E_i}\right) \Pr\left(E_{k+1}^{i-1} \cap \overline{E_i}\right) \\
&\quad \leq \sum_{i=k+1}^{\lceil n/2 \rceil - 1} \left(\sum_{r=1}^{i-1} \sum_{s=1}^{r} \binom{n}{s} \ln R + \sum_{s=1}^{m-1} \binom{n}{s} \ln R + \binom{n}{m} \right) \cdot R^{-(i-k-1)}.
\end{aligned}$$

This is because in the last phase, the success can happen with strength m, so we do not consider the strengths larger than m in the last phase. Also, in the last phase, the algorithm makes progress in $\binom{n}{m}$ iterations in expectation.

Now, since we have $\sum_{s=1}^{m-1} \binom{n}{s} \leq \sum_{r=1}^{i-1} \sum_{s=1}^{r} \binom{n}{s}$ for $i \geq m$, the last expression is bounded from above by

$$\begin{aligned}
&\sum_{i=k+1}^{\lceil n/2 \rceil - 1} \left(2 \sum_{r=1}^{i-1} \sum_{s=1}^{r} \binom{n}{s} \ln R + \binom{n}{m} \right) \cdot R^{-(i-k-1)} \\
&\leq 2 \sum_{i=k+1}^{\lceil n/2 \rceil - 1} \left(\sum_{r=1}^{i-1} \frac{n-r+1}{n-2r+1} \binom{n}{r} \ln R + \binom{n}{m} \right) R^{-(i-k-1)} \\
&\leq 2 \sum_{i=k+1}^{\lceil n/2 \rceil - 1} \left(\left(\frac{n-i+2}{n-2i+3} \right)^2 \binom{n}{i-1} \ln R + \binom{n}{m} \right) R^{-(i-k-1)} \\
&= 2 \sum_{i=k+1}^{\lceil n/2 \rceil - 1} \left(\left(1 + \frac{i-1}{n-2i+3} \right)^2 \binom{n}{i-1} \ln R + \binom{n}{m} \right) R^{-(i-k-1)},
\end{aligned}$$

where in the second and third inequalities, we apply the first inequality in Lemma 5.1 to eliminate the inner summations.

Now, via the second inequality in Lemma 5.1, and then excluding the first term from the summation, we bound the last expression from above by

$$2\left(1+\frac{k}{n-2k+1}\right)^2\binom{n}{k}\ln R+\binom{n}{m} \\ +4\sum_{i=k+2}^{\lceil n/2\rceil-1}\left(1+\frac{i-1}{n-2i+3}\right)^2\left(\frac{n-k}{k}\right)^{i-k-1}\binom{n}{k}\ln R\cdot R^{-(i-k-1)}.$$

We have $k/(n-2k+1)=O(1)$ because $k<(1-\hat{\epsilon})n/2$ for a constant $\hat{\epsilon}$, giving the upper bound

$$O\left(\binom{n}{k}\ln R\right)+\binom{n}{m}+4\ln R\binom{n}{k}\sum_{i=k+2}^{\lceil n/2\rceil-1}n^2\left(\frac{n}{k}\right)^{i-k-1}\cdot R^{-(i-k-1)}.$$

By using $\binom{n}{k}=\frac{k+1}{n-k}\binom{n}{k+1}$, we get

$$O\left(\frac{k+1}{n-k}\ln R\binom{n}{k+1}\right)+\binom{n}{m}+4\frac{k+1}{n-k}\ln R\binom{n}{k+1}\sum_{i=k+2}^{\lceil n/2\rceil-1}\frac{n^2\left(\frac{n}{k}\right)^{i-k-1}}{R^{i-k-1}}.$$

Using the fact that $R\geq n^{3+\epsilon}$, the third term in the last expression is bounded from above by $O(k\ln R/n\binom{n}{k+1})$. Then, we have

$$\binom{n}{m}+O\left(\frac{k\ln R}{n}\binom{n}{k+1}\right).$$

Regarding S_2, when radius r is increased to n, the algorithm mutates s bits of the current search point for all possible strengths s from 1 to n periodically. In each cycle through different strengths, according to Lemma 5.3, the algorithm escapes from the local optimum with probability at least $1-1/R$, so there are at most $R/(R-1)$ cycles in expectation via the geometric distribution. Besides, each cycle of radius n costs $\sum_{s=1}^{n}\binom{n}{s}\ln R$. Overall, we have $\frac{R}{R-1}\sum_{s=1}^{n}\binom{n}{s}\ln R$ extra fitness function calls if the algorithm fails to find the optimum in the first $\lceil n/2\rceil-1$ phases, happening with probability at most $R^{-(\lceil n/2\rceil-k-1)}$. Thus, we have

$$\mathrm{E}\left(T'_{x,r_0,B_0}\mid E_{k+1}^{\lceil n/2\rceil-1}\cap\overline{E_n}\right)\Pr\left(E_{k+1}^{\lceil n/2\rceil-1}\cap\overline{E_n}\right) \\ \leq\left(\sum_{r=1}^{\lceil n/2\rceil-1}\sum_{s=1}^{r}\binom{n}{s}\ln R+\frac{R}{R-1}\sum_{s=1}^{n}\binom{n}{s}\ln R\right)R^{-(\lceil n/2\rceil-k-1)}$$

$$\leq \left(3 \sum_{r=1}^{\lceil n/2 \rceil - 1} \sum_{s=1}^{r} \binom{n}{s} \ln R\right) R^{-(\lceil n/2 \rceil - k - 1)}$$
$$\leq \left(3n^2 \binom{n}{\lceil n/2 \rceil - 1} \ln R\right) R^{-(\lceil n/2 \rceil - k - 1)}$$
$$\leq \left(3n^2 \left(\frac{n-k}{k}\right)^{\lceil n/2 \rceil - k - 1} \binom{n}{k} \ln R\right) R^{-(\lceil n/2 \rceil - k - 1)} = o\left(\binom{n}{k}\right).$$

Altogether, we finally have $\mathrm{E}\left(T'_{x,r_0,B_0} \mid E_1^k\right) = S_1 + S_2$, resulting in the statement. □

Now, by using Lemma 5.4, we prove Theorem 5.2.

Proof of Theorem 5.2. We have $\mathrm{E}\left(T_{x,r_0,B_0}\right) \leq \mathrm{E}\left(T_{x,r_0,B_0} \mid E_1^k\right)$. If $r_0 \leq m \leq (1-\hat{\epsilon})n/2$, we use Lemma 5.4 with $k = m-1$. Otherwise, if $m < r_0 \leq (1-\hat{\epsilon})n/2$, we use Lemma 5.4 with $k = r_0$. Then we have

$$\mathrm{E}\left(T_{x,r_0,B_0} \mid E_1^k\right) \leq \binom{n}{m} + O\left(\frac{k \ln R}{n} \binom{n}{k+1}\right).$$

In the first case, the last expression is bounded from above by $\binom{n}{m}\left(1+O(\frac{m \ln R}{n})\right)$. However, in the latter case, it is bounded from above by $O\left(\binom{n}{r_0} \ln R\right)$.

If $\max\{m, r_0\} > (1-\hat{\epsilon})n/2$, we pessimistically assume that the algorithm is not able to make an improvement for radius r less than $n/2$. As radius r is increased to n, the algorithm mutates m bits of the current search point for all possible strengths of 1 to n periodically. Thus, according to Lemma 5.3, the algorithm escapes from the local optimum with probability at least $1 - 1/R$, so there are at most $R/(R-1)$ cycles in expectation in this phase (i. e., at radius n) by using the geometric distribution. Finally, we compute

$$\mathrm{E}\left(T_{x,r_0,B_0}\right) < \sum_{r=1}^{\lceil n/2 \rceil - 1} \sum_{s=1}^{r} \binom{n}{s} \ln R + \frac{R}{R-1} \sum_{s=1}^{n} \binom{n}{s} \ln R \leq O(2^n n \ln R). \square$$

5.3.2 Expected Optimization Times

In this subsection, we prove a crucial technical lemma on recover times and use it to obtain bounds on the expected optimization time on unimodal and jump functions.

Recover times for strengths. In the previous subsection, we analyzed the time of SD-RLS$^{\mathrm{m}}$ for leaving only a single search point. We observed that the duration of epochs depended on the starting radius denoted as r_0 set from the previous epoch. This can be inconvenient to estimate an upper bound on the running time on an arbitrary function. Therefore, in the following lemma, we show that if the algorithm uses larger strengths than the gap size to make progress, after a relatively small number of iterations the algorithm chooses the gap size of a current search point as the strength.

Lemma 5.5. *Let $x \in \{0,1\}^n$ with $m = \operatorname{gap}(x) < n/2$ be the current search point of SD-RLS$^{\mathrm{m}}$ with $R \geq n^{3+\epsilon}$ for an arbitrary constant $\epsilon > 0$ on a pseudo-Boolean function $f\colon \{0,1\}^n \to \mathbb{R}$. Assume that the radius is $k > m$. Define S_x as the number of iterations spent from that point in time on until the algorithm sets the radius to at most the gap size of the current search point. Assume that B_0 is the value of the variable B in the beginning. Then, for $B_0 \geq \binom{n}{m} \ln R$, we have*

$$\mathrm{E}\,(S_x) = o\left(\binom{n}{k-1}\right),$$

and for $B_0 < \binom{n}{m} \ln n$, we have

$$\mathrm{E}\,(S_x) = o\left(R\binom{n}{k-1}\right).$$

The idea of the proof is that in the case of making progress with larger strengths than m or failing to improve and increasing strength and radius even further, the algorithm also tries all smaller strengths often enough, more precisely, as often as the threshold value $\binom{n}{\operatorname{gap}(x)} \ln R$ that would hold if the current search point was x, $s = \operatorname{gap}(x)$ and $B = \infty$ in a phase of Algorithm 15. Thus, the algorithm can make progress when the strength equals the current gap with good probability.

Proof of Lemma 5.5. We recall the epoch of x as the sequence of iterations where x is the current search point. We assume that the gap size of the current search point does not become equal to or larger than the current radius value in all epochs. Otherwise, S_x is bounded from above by the following estimation.

Assume x' is the current search point and r' is a radius value larger than $g_{x'} \coloneqq \operatorname{gap}(x')$. Note that $g_{x'}$ equals m in the beginning (in the first epoch), but it may be different when a strict improvement is made.

We now claim that for each at most $\ln n \cdot r'\binom{n}{r'} \ln R$ iterations with strength r' in phase r', i. e., at radius r', (even in different epochs), the algorithm uses

smaller strengths including strength $g_{x'}$ for $\binom{n}{g_{x'}} \ln R$ iterations. After proving the claim, we can show that with probability $1 - 1/R$ the algorithm makes progress with the strength which equals the gap size of the current search point via Lemma 5.3.

To prove this claim, we consider two cases. First, if the algorithm does not make progress with strength r', then in the next phase the algorithm uses strengths smaller than r' including $g_{x'}$ in $\binom{n}{g_{x'}} \ln R$ iterations. Thus, $\binom{n}{r'} \ln R$ iterations with strength r' are enough for satisfying the claim in this case.

In the second case, assume that the algorithm makes an improvement with strength r' in its u^{th} attempt. For the next epoch, the algorithm tries strength $g_{x'}$ for $(1/\ln n) \cdot u/(r'-1)$ times (according to the variable B in Algorithm 15). Assume that $u_1, u_2, \ldots$ are the counter values where the algorithm makes progress with strength r'. Thus, after at most ℓ improvements with $\sum_{i=1}^{\ell} u_i$ iterations with strength r' such that $\sum_{i=1}^{\ell} u_i \leq (\ln n) \cdot (r'-1)\binom{n}{g_{x'}} \ln R - 1$, the number of iterations with strength $g_{x'}$ is at least $\sum_{i=1}^{\ell} (1/\ln n) \cdot u_i/(r'-1) \geq \binom{n}{g_{x'}} \ln R - 1$. In the next epoch, there are at most $\binom{n}{r'} \ln R$ iterations with strength r' for having the last required iteration.

Overall, it costs less than $(\ln n) \cdot r' \binom{n}{g_{x'}} \ln R + \binom{n}{r'} \ln R < (\ln n) \cdot r' \binom{n}{r'}$ iterations to observe $\binom{n}{g_{x'}} \ln R$ iterations with strength $g_{x'}$, and consecutively with a probability of at least $1 - 1/R$, the algorithm makes progress with strength $g_{x'}$ via Lemma 5.3.

If we pessimistically assume that after each $\binom{n}{g_{x'}} \ln R$ iterations with strength $g_{x'}$, the radius is increased by one, by the law of total probability and Lemma 5.1, the expected number of steps at larger strengths than the gap of the current search point is at most

$$\begin{aligned} &\sum_{r=k}^{\lfloor n/2-1 \rfloor} \frac{n-(r-1)}{n-(2r-1)} r \ln n \binom{n}{r} \ln R \cdot R^{-(r-k-1)} \\ &< R \cdot \sum_{r=k}^{\lfloor n/2-1 \rfloor} n^2 \ln n \cdot \left(\frac{n-k+1}{k-1}\right)^{r-k+1} \binom{n}{k-1} \ln R \cdot R^{-(r-k)} \end{aligned}$$

for the phases ranging from k to $\lfloor n/2 - 1 \rfloor$ and at most

$$\frac{R}{R-1} \cdot 2n \ln n \binom{n}{\lfloor n/2 \rfloor} \ln R \cdot R^{-(r-m-1)}$$

$$< R \cdot 4n \ln n \left(\frac{n-k+1}{k-1}\right)^{\lfloor n/2 \rfloor - k + 1} \binom{n}{k-1} \ln R \cdot R^{-(\lfloor n/2 \rfloor - k - 2)}$$

for the last phase (i. e., at radius n). Since $R > n^{3+\epsilon}$, both are bounded from above by $o\Big(R\binom{n}{k-1}\Big)$.

However, in phase k, before reaching the strength k, the algorithm uses the strength m B_0 times. If $B_0 \geq \binom{n}{m} \ln R$, then with probability at least $1 - 1/R$, the algorithm finds an improvement at the Hamming distance corresponding to the gap size, resulting in

$$\mathrm{E}\,(S_x) \leq \frac{1}{R} \cdot o\left(R\binom{n}{k-1}\right) = o\left(\binom{n}{k-1}\right). \qquad \square$$

Analysis on unimodal functions On unimodal functions, the gap of all points in the search space (except for global optima) is one, so the algorithm can make progress with strength 1. In the following theorem, we show how SD-RLS$^{\mathrm{m}}$ behaves on unimodal functions compared RLS using an upper bounds based on the fitness-level method [Weg02]. The proof is similar to the proof of Lemma 4 in [RW21b].

Theorem 5.6. *Let $f\colon \{0,1\}^n \to \mathbb{R}$ be a unimodal function and consider SD-RLS^m with $R \geq \max\{S, n^{3+\epsilon}\}$ for an arbitrary constant $\epsilon > 0$ where S is an upper bound on the number of strict improvements during the run, e. g., $S = |\mathrm{Im}\, f|$. Then, with probability at least $1 - S/R^2$, SD-RLS^m never uses strengths larger than 1 and behaves stochastically like RLS before finding an optimum of f.*

Denote by T the runtime of SD-RLS^m on f. Let f_i be the i-th fitness value of an increasing order of all fitness values in f and s_i be a lower bound for the probabilities that RLS finds an improvement from search points with fitness value f_i, then $\mathrm{E}\,(T) \leq \sum_{i=1}^{|\mathrm{Im}\, f|} 1/s_i + o(n)$.

Proof. The algorithm SD-RLS$^{\mathrm{m}}$ uses strength 1 for $\binom{n}{m} \ln R$ times when the radius is 1 and $\binom{n}{m} \ln R$ times when the radius is 2 but the strength is still 1. (Only considering the first case would not be sufficent for the result of this lemma.) Overall, the algorithm tries $2\binom{n}{m} \ln R$ steps with strength 1 before setting the strength to 2.

As on unimodal functions, the gap of all points is 1, the probability of not finding and improvement is

$$\left(1 - \binom{n}{m}^{-1}\right)^{2\binom{n}{m}\ln R} \leq \frac{1}{R^2}.$$

This argumentation holds for each improvement that has to be found. Since at most $|\operatorname{Im} f|$ improving steps happen before finding the optimum, by a union bound the probability of SD-RLS$^{\mathrm{m}}$ ever increasing the strength beyond 1 is at most $S\frac{1}{R^2}$, which proves the lemma.

To prove the second claim, we consider all fitness levels $A_1, \ldots, A_{|\operatorname{Im} f|}$ such that A_i contains search points with fitness value f_i and sum up upper bounds on the expected times to leave each of these fitness levels. Under the condition that the strength is not increased before leaving a fitness level, the worst-case time to leave fitness level A_i is $1/s_i$ similarly to RLS. Hence, we bound the expected optimization time of SD-RLS$^{\mathrm{m}}$ from above by adding the waiting times on all fitness levels for RLS, which is given by $\sum_{i=1}^{|\operatorname{Im} f|} 1/s_i$.

We let the random set W contain the search points from which SD-RLS$^{\mathrm{m}}$ does not find an improvement within phase 1 (i. e., while $r = 1$) so the radius is increased. Assume T_x is the number of iterations spent where the radius is larger than 1 and increasing the radius happening where x is the current search point; formally,

$$\mathrm{E}(T) \leq \sum_{i=1}^{|\operatorname{Im} f|} \frac{1}{s_i} + \sum_{x \in W} \mathrm{E}(T_x).$$

Each search point selected by the algorithm contributes with probability $\Pr(E_1)$ to W. Hence, as S is an upper bound on the number of improvements, $\mathrm{E}(|W|) \leq S \cdot \Pr(E_1)$. As on unimodal functions, the gap of all points is 1, by Lemma 5.5, we compute

$$\begin{aligned}\sum_{x \in W} \mathrm{E}(T_x) &\leq S \cdot \Pr(E_1) \cdot \mathrm{E}(T_x \mid \operatorname{gap}(x) = 1) \\ &\leq S \cdot R^{-1} o\left(\binom{n}{1}\right) = o(n).\end{aligned}$$

Thus, we finally have

$$\mathrm{E}(T) \leq \sum_{i=1}^{|\operatorname{Im} f|} \frac{1}{s_i} + o(n),$$

as suggested. □

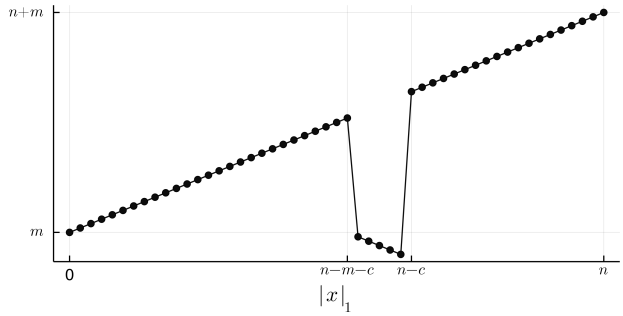


Figure 5.1: The function JumpOffset$_{m,c}$.

Analysis on JumpOffset We now use the results developed so far to prove a bound on a newly designed function called *Jump with Offset* or JumpOffset illustrated in Figure 5.1 with two parameters m and c. The function JumpOffset can be considered as a variant of well-known Jump benchmark [DJW02], which the location of the jump with size m is moved to an earlier point. Then, after the jump, there is a unimodal sub-problem behaving like OneMax of length c. The Jump function is a special case of JumpOffset with $c = 0$, i. e., JumpOffset$_{m,0}$ = Jump$_m$. Formally,

$$\textsc{JumpOffset}_{m,c} \coloneqq \begin{cases} m + \|x\|_1 & \text{if } \|x\|_1 \leq n - m - c \text{ or } \|x\|_1 \geq n - c, \\ n - \|x\|_1 - c & \text{otherwise.} \end{cases}$$

This generalized version of Jump was also proposed independently in [BBD21a] to investigate how quickly various mutation-based algorithms overcome a gap that is not adjacent to the global optimum. In another recent study, Witt in [Wit21] analyzes the performance of other algorithms on the function JumpOffset.

The following theorem shows that SD-RLS$^{\mathrm{m}}$ optimizes JumpOffset in a time that is essentially determined by the time to overcome the gap only. The proof idea is that the algorithm can quickly re-adapt its radius value to the gap size of the current search point after escaping the local optimum.

Theorem 5.7. *Let $n \in \mathbb{N}$. For all $2 \leq m < O(\ln n)$ and $0 \leq c < O(\ln n)$, the expected runtime $\mathrm{E}(T)$ of SD-RLS$^{\mathrm{m}}$ with $R \geq n^{3+\epsilon}$ for an arbitrary constant $\epsilon > 0$ on JumpOffset$_{m,c}$ satisfies $\mathrm{E}(T) = O\big(\binom{n}{m}\big)$, conditioned on an event that happens with probability $1 - o(1)$.*

Proof. Before reaching the local optimum consisting of all points with $n-m-c$ one-bits, JumpOffset$_{m,c}$ is equivalent to OneMax; hence, according to

Lemma 5.6, the expected time SD-RLS$^{\mathrm{m}}$ takes to reach the local optimum is at most $O(n \ln n)$. Note that this bound was obtained via the fitness level method with $s_i = (n-i)/n$ as the minimum probability for leaving the set of search points with i one-bits.

Every local optimum x with $n-m-c$ one-bits satisfies $\mathrm{gap}(x) = m$ according to the definition of JumpOffset$_{m,c}$. Thus, using Theorem 5.2, the algorithm finds one of the $\binom{m+c}{m}$ improvements within expected time at most $\binom{n}{m}$ iterations. According to Lemma 5.3, this success happens with strength m (and not larger) with probability at least $1 - 1/R$.

After making progress over the jump, the starting radius r_0 is at least m, although an improvement can be found at the Hamming distance of 1, i. e., $\mathrm{gap}(x) = 1$. If we show that $B_0 \geq \binom{n}{1} \ln R$, the algorithm sets the radius to 1 within $o(\binom{n}{m-1})$ steps in expectation via Lemma 5.5 with $k = m$. Now, we compute the probability that $B_0 < n \ln R$, resulting from making progress within less steps than $(m-1)n \ln n \ln R$ in the previous epoch. Hence, let u be the number of iterations with the strength the algorithm makes progress in the epoch with the local optimum. We have

$$\begin{aligned}
&\Pr\left(u < (m-1)n \ln n \ln R\right) \leq 1 - \left(1 - \frac{\binom{m+c}{m}}{\binom{n}{m}}\right)^{(m-1)n \ln n \ln R} \\
&\leq (m-1)n \ln n \ln R \frac{\binom{m+c}{m}}{\binom{n}{m}} \leq (m-1)n \ln n \ln R \frac{(e(m+c))^m}{n^m}.
\end{aligned}$$

According to the assumption on m and c, the last term is bounded from above by

$$(m-1)n \ln n \ln R \frac{O(\ln^m n)}{n^m} \leq O\left(\frac{m(\ln^m n)}{n^{m-1}}\right) = o(1).$$

This means that with probability at least $1 - o(1)$, the variable B is not effective in the beginning of the next epoch with strength 1, so we use the first case in Lemma 5.5.

After recovering the radius to 1, the algorithm needs to optimize a sub-problem like OneMax of length at most c, so similarly to the first part, its expected time again can be obtained from Lemma 5.6, which is at most $O(n \ln n)$.

Altogether, $\mathrm{E}(T) \leq O(n \ln n) + \binom{n}{m} + \binom{n}{m-1} + O(n \ln n)$, conditioned on the mentioned events of having enough iterations with the strength passing the jump part and escaping from the local optimum with strength m, happening with probability $1 - o(1)$. □

5.4 Speed-ups By Using Radius Memory

In this section, we consider the problem of minimizing a linear function under a uniform constraint as analyzed in [NPW19]: given a linear pseudo-Boolean function $f(x_1, \dots, x_n) = \sum_{i=1}^{n} w_i x_i$, the aim is to find a search point x minimizing f under the constraint $\|x\|_1 \geq B$ for some $B \in \{1, \dots, n\}$. W. l. o. g., $w_1 \leq \cdots \leq w_n$.

Neumann, Pourhassan and Witt [NPW19] obtain a tight worst-case runtime bound $\Theta(n^2)$ for RLS^{1,2} and a bound for the (1+1) EA which is $O(n^2 \log B)$ and therefore tight up to logarithmic factors. We will see in Theorem 5.8 that with high probability, SD-RLS$^{\mathrm{m}}$ achieves the same bound $O(n^2)$ despite being able to search globally like the (1+1) EA. Afterwards, we will identify a scenario where SD-RLS$^{\mathrm{r}}$ is by a factor of $\Omega(n)$ slower.

We start with the general result on the worst-case expected optimization time, assuming the set-up of [NPW19].

Theorem 5.8. *Starting with an arbitrary initial solution, the expected optimization time of SD-RLS$^{\mathrm{m}}$ with $R \geq n^{3+\epsilon}$, where $\epsilon > 0$ is an arbitrary constant, on a linear function with a uniform constraint is $O(n^2)$ conditioned on an event that happens with probability $1 - O(1/n)$.*

Proof. We follow closely the proof of Theorem 4.2 in [NPW19] which analyzes RLS^{1,2}. The first phase of optimization (covered in Lemma 4.1 of the paper) deals with the time to reach a feasible search point and proves this to be $O(n \log n)$ in expectation. Since the proof uses multiplicative drift, it is easily seen that the time is $O(n \log n)$ with probability at least $1 - O(1/n)$ thanks to the tail bounds for multiplicative drift [Len20]. The second phase deals with the time to reach a tight search point (i. e., containing B one-bits, which is $O(n \log n)$ with probability at least $1 - O(1/n)$ by the very same type of arguments. The analyses so far rely exclusively on one-bit flips so that the bounds also hold for SD-RLS$^{\mathrm{m}}$ thanks to Lemma 5.6, up to a failure event of $O(S/R^2) = o(1/n)$ since it holds for the number of improvements S that $S \leq n$. By definition of the fitness function, only tight search points will be accepted in the following.

The third phase in the analysis from [NPW19] considers the potential function $\phi(x) = |\{x_i = 1 : i > B\}|$ denoting the number of one-bits outside the B least significant positions. At the same time, $\phi(x)$ describes the number of zero-bits at the B optimal positions. Given $\phi(x) = i > 0$ for the current search point, the probability of improving the potential is $\Theta(i^2/n^2)$ since there are i^2 improving two-bit flips (i choices for a one-bit to be flipped to 0 at the non-optimal positions

and i choices for a zero-bit to be flipped to 1 at the other positions). This results in an expected optimization time of at most $\sum_{i=1}^{\infty} O(n^2/i^2) = O(n^2)$ for RLS^{1,2}.

SD-RLS$^{\mathrm{m}}$ can achieve the same time bound since after every two-bit flip, the radius memory only allocates the time for the last improvement (via two-bit flips) to iterations trying strength 2. Hence, as long as no strengths larger than 2 are chosen, the expected optimization time of SD-RLS$^{\mathrm{m}}$ is $O(n^2)$.

To estimate the failure probability, we need a bound on the number of strict improvements of f, which may be larger than the number of improvements of the ϕ-value since steps that flip a zero-bit and a one-bit both located in the prefix or suffix may be strictly improving without changing ϕ. Let us assign a value to each zero-bit, representing the number of one-bits to its left (at higher indices). In other words, each of these values shows the number of one-bits that can be flipped with the respective zero-bit to make a strict improvement. Let us define by S the sum of these values. Clearly, $S \leq n(n-1)/2 = O(n^2)$. Now, we claim that each strict improvement decreases S by at least one, resulting in bounding the number of strict improvements from above by $O(n^2)$. Assume that in a strict improvement, the algorithm flips one one-bit at position i and one zero-bit at position j. Obviously, $j < i$. The corresponding value for the zero-bit at the new position i is less than the corresponding number for the zero-bit at position j before flipping because there is at least one one-bit less for the new zero-bit, which was the one-bit at the position of i. Altogether, the number of strict improvements at strength 2 is at most $O(n^2)$.

The proof is completed by noting that the strength never exceeds 2 with probability at least $1 - O(n^2)/R = 1 - O(1/n)$, using Lemma 5.3 and a union bound. □

With more effort, including an application of Lemma 5.5, the bound from Theorem 5.8 could be turned into a bound on the expected optimization time. We will see an example of such arguments later in the proof of Theorem 5.9 and in the proof of Theorem 5.11.

We now illustrate why the original SD-RLS$^{\mathrm{r}}$ is less efficient on linear functions under uniform constraints than SD-RLS$^{\mathrm{m}}$. To this end, we study the following instance: the weights of the objective function are n pairwise different natural numbers (sorted increasingly), and the constraint bound is $B = n/2$, i. e., only search points having at least $n/2$ one-bits are valid. Writing search points in big-endian as $x = (x_n, \ldots, x_1)$, we assume the point $1^{n/2}0^{n/2}$ as starting point of our search heuristic. The optimum is then $0^{n/2}1^{n/2}$ since the $n/2$ one-bits are at the least significant positions. We call the latter positions the suffix and the other the prefix.

Considering the potential $\phi(x)$ defined as the number of one-bits in the prefix, we note that the expected time for $\text{RLS}^{1,2}$ and for SD-RLS^{m} (up to a failure event) to reduce the potential from its initial value $n/4$ to $n/8$ is $O(n)$ since during this period there is always an improvement probability of at least $\Omega((n/8)^2/n^2) = \Omega(1)$. We claim that SD-RLS^{r} needs time $\Omega(n^2 \log n)$ for this since after each improvement the strength and radius are reset to 1, resulting in $\Omega(n)$ phases where the algorithm is forced to iterate unsuccessfully with strength 1 until the threshold $\binom{n}{1} \ln R$ is reached. By contrast, SD-RLS^{m} will spend $O(n \ln n)$ steps to set the strength and radius to 2. Afterwards, during the $n/4$ improvements it will only spend an expected number of $O(1)$ steps at strength 1 before it returns to strength 2 and finds an improvement in expected time $O(1)$. The total time is $O(n \log n + n) = O(n \log n)$. Based on these ideas, we formulate the following theorem.

Theorem 5.9. *Consider a linear function with n pairwise different weights under uniform constraint with $B = n/2$ and let $S_b = \{\phi(y) \leq b \mid y \in \{0,1\}^n\}$. Starting with a feasible solution x such that x contains B one-bits and $\phi(x) = a$, the expected time to find a search point in S_b for SD-RLS^{m} with $R \geq n^{3+\epsilon}$ for an arbitrary constant $\epsilon > 0$ is at most*

$$O\left(n \ln R + n^2 \sum_{i=b}^{a} \frac{1}{i^2}\right).$$

For SD-RLS^{r} with $R \geq n^{3+\epsilon}$ for an arbitrary constant $\epsilon > 0$ it is at least

$$\Omega\left(P \cdot n \ln R + n^2 \sum_{i=b}^{a} \frac{1}{i^2}\right),$$

where P is the number of improvements with strengths larger than 1 and $P > a - b$ with probability $1 - o(1/n)$.

Proof. We first find an upper bound for SD-RLS^{m}. First, SD-RLS^{m} spends $O(n \ln R)$ steps to set the strength and radius to 2. Afterwards, the number of iterations with strength 1 is $1/\ln n$ times of the number of iterations with strength 2, conditioned on not exceeding the threshold when $r = 2$, which will be studied later.

When the strength equals 2, the probability of improving the potential is $\Theta(\frac{i^2}{n^2})$, resulting in the expected time of $\Theta(n^2/i^2)$ for each improvement. Thus, in expectation, there are $n^2 \sum_{i=b}^{a} 1/i^2$ iterations to find a search point in S_b. In the case that the counter exceeds the threshold, happening with probability $1/R$ for each improvement using Lemma 5.3, it costs $o(\binom{n}{2})$ iterations with different

strengths in expectation, according to Lemma 5.5, to set the radius to 2 again. Since the number of fitness improvements at strength 2 is at most n^2 (see the proof of Th. 5.9), we obtain a failure probability of at most $n^2 \cdot 1/R = o(1/n)$ for the event of exceeding the threshold. Hence, the expected number iterations with different strengths is $o((1/n)\binom{n}{2}) = o(n)$ and therefore a lower-order term of the claimed bound on the expected time to reach S_b.

In case of a failure, we repeat the argumentation. Hence, after an expected number of $1/(1 - o(1/n)) = 1 + o(1)$ repetitions no failure occurs and S_b is reached.

Overall, the expected time for SD-RLS$^{\mathrm{m}}$ to find a search point in S_b is at most

$$(1 + o(1)) \left(n \ln R + (1 + 1/\ln n) \cdot \sum_{i=b}^{a} \frac{n^2}{i^2} \right) = O\left(n \ln R + n^2 \sum_{i=b}^{a} \frac{1}{i^2} \right).$$

In order to compute a lower bound on the optimization time of SD-RLS$^{\mathrm{r}}$, we claim that the number of potential improvements is at least $a - b$ with probability $1 - o(1/n)$, i. e., each improvement decreases the potential function roughly by at most one in expectation.

As long as the strength does not become greater than 2, the number of one-bits remains B, so when the strength is at most 2, the algorithm can only improve the potential by 1 since it cannot make progress by flipping at least two zero-bits in B least significant positions. The radius becomes 3 with probability at most $1/R$ for each improvement at strength 2. Since there are at most n^2 improvements, the probability of not increasing the current radius to 3 during the run is at least $1 - n^2/R = 1 - o(1/n)$.

Now, since the algorithm spends $n \ln R$ steps for each improvement, the number of steps with strength 1 is $\Omega((a-b)n \ln R)$ with probability $1-o(1/n)$. Also, the number of steps with strength 2 is $\Theta(n^2 \sum_{i=b}^{a} 1/i^2)$ in expectation. Note that we ignore the number of iterations with strengths larger than 2 for the lower bound.

Overall, with probability at least $1 - o(1/n)$ the time of SD-RLS$^{\mathrm{r}}$ is at least

$$\Omega\left((a - b)n \ln R + n^2 \sum_{i=b}^{a} \frac{1}{i^2} \right).$$

□

In the following corollary, it can be seen that SD-RLS$^{\mathrm{m}}$ is faster than SD-RLS$^{\mathrm{r}}$ in the middle of the run by a factor of roughly n.

Corollary 5.10. *The relative speed-up of SD-RLS^m with $R \geq n^{3+\epsilon}$ for an arbitrary constant $\epsilon > 0$ compared to SD-RLS^r with $R \geq n^{3+\epsilon'}$ for an arbitrary constant $\epsilon' > 0$ to find a search point in S_b with $b = n/8$ for a starting search point x with $\phi(x) = n/4$ is $\Omega(n)$ with probability at least $1 - o(1/n)$.*

Proof. Assume that T_r and T_m are the considered hitting times of SD-RLS$^{\mathrm{r}}$ and SD-RLS$^{\mathrm{m}}$, respectively, with $R \geq n^{3+\epsilon}$ for an arbitrary constant $\epsilon > 0$. Using Theorem 5.9 with $a = n/4$ and $b = n/8$, we have

$$\frac{\mathrm{E}\,(T_r)}{\mathrm{E}\,(T_m)} \geq \frac{\Omega(n^2 \ln R)}{O(n \ln R)} = \Omega(n),$$

with probability at least $1 - o(1/n)$. □

5.5 Minimum Spanning Trees

In Theorem 6 in [RW21b], the authors studied SD-RLS$^{\mathrm{r}}$ on the MST problem as formulated in [NW07]: we are given an undirected, weighted graph $G = (V, E)$, where $n = |V|$, $m = |E|$ and the weight of edge e_i, where $i \in \{1, \ldots, m\}$, 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 [NW07], to be minimized, is defined by

$$f(x) \coloneqq M^2(c(x) - 1) + M\left(\sum_{i=1}^{m} x_i - (n-1)\right) + \sum_{i=1}^{m} w_i x_i$$

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 [RW21b] showed that SD-RLS$^{\mathrm{r}}$ with $R = m^4$ can find an MST starting with an arbitrary spanning tree 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 reason behind $\mathrm{E}\,(S)$ is that for each improvement with strength 2, the algorithm resets the radius to one for the next epoch and explores this radius more or less completely in $m \ln R$ iterations. This can be costly for the graphs requiring many improvements.

However, with SD-RLS$^{\mathrm{m}}$, we do not need to include the number of improvements for estimating the number of iterations with strength 1 in the runtime

bound since with the radius memory mechanism, the number of iterations with strength 1 is asymptotically in the order of $1/\ln n$ times of the number of successes. This leads to the following simple bound.

Theorem 5.11. *Consider an instance to the MST problem, modeled with the classical fitness function from [NW07]. The expected optimization time of SD-RLS^m with $R = m^4$, starting with an arbitrary spanning tree, is at most*

$$(1 + o(1))\big((m^2/2)(1 + \ln(r_1 + \cdots + r_m))\big) \leq (1 + o(1))\big(m^2 \ln m\big),$$

where r_i is the rank of the ith edge in the sequence sorted by increasing edge weights.

The proof is similar to the proof of Theorem 6 in [RW21b] by using drift multiplicative analysis [Len20]. However, we show that the radius memory mechanism controls the number of iterations with strength 1, and we apply Lemma 5.5 to show that if the algorithm uses strengths larger than 2, the algorithm shortly after makes an improvement with strength 2 again.

Proof of Theorem 5.11. We aim at using multiplicative drift analysis using $g(x) = \sum_{i=1}^{m} x_i r_i$ as potential function. As shown in [RKJ06], RLS behaves stochastically identical on the original fitness function f and the function g if at most two bits may flip simultaneously. Since SD-RLS$^{\mathrm{m}}$ has different states, we do not have the same lower bound on the drift towards the optimum as with the classical RLS^{1,2} from [NW07]. However, at strength 1 no mutation is accepted since the fitness function from [NW07] gives a huge penalty to non-trees. Hence, our plan is to conduct the drift analysis conditioned on that the strength is at most 2 and account for the steps spent at strength 1 separately. Cases where the strength exceeds 2 will be handled by an error analysis and a restart argument.

Let $X^{(t)} \coloneqq g(x^t) - g(x_{\mathrm{opt}})$ for the current search point $x^{(t)}$ and an optimal search point x_{opt}. Since the algorithm behaves stochastically the same on the original fitness function f and the potential function g, we obtain that $\mathrm{E}\left(X^{(t)} - X^{(t+1)} \mid X^{(t)}\right) \geq X^{(t)}/\binom{m}{2} \geq 2X^{(t)}/m^2$ since the g-value can be decreased by altogether $g(x^t) - g(x_{\mathrm{opt}})$ via a sequence of at most $\binom{m}{2}$ disjoint two-bit flips; see also the proof of Theorem 15 in [DJW12] for the underlying combinatorial argument. Let T denote the number of steps at strength 2 until g is minimized, assuming no larger strength to occur. Using the multiplicative drift theorem, we have $\mathrm{E}\,(T) \leq (m^2/2)(1+\ln(r_1+\cdots+r_m)) \leq (m^2/2)(1+\ln(m^2))$ and by the tail bounds for multiplicative drift (e. g., [Len20]) it holds that $\Pr\left(T > (m^2/2)(\ln(m^2) + \ln(m^2))\right) \leq e^{-\ln(m^2)} = 1/m^2$. Note that this bound on T is below the threshold for strength 2 since $\binom{m}{2} \ln R = (m^2 - m)\ln(m^4) \geq (m^2/2)(4 \ln m)$ for m large enough. Hence, with probability at most $1/m^2$ the

algorithm fails to find the optimum before the strength can change from 2 to a different value due to the threshold being exceeded.

We next bound the expected number of steps spent at larger strengths. By Lemma 5.5, if the algorithm fails to find an improvement with the right radius, i. e. when the radius becomes $r = 3 > \text{gap}(x) = 2$, then, in at most $o(\binom{m}{r-1})$ iterations in expectation, the radius is set to the gap size of the current search point at that time. Thus, by using Lemma 5.5, an increase of radius above 2 costs at most $o\big(\binom{m}{2}\big)$ iterations to make improvements with strength 2 again and set the radius to two. This time is only a lower-order term of the runtime bound claimed in the theorem. If the strength exceeds 2, we wait for it to become 2 again and restart the previous drift analysis, which is conditional on strength at most 2. Since the probability of a failure is at most $1/m^2$, this gives an expected number of at most $1/(1 - m^{-2})$ restarts, which is $1 + o(1)$ as well.

It remains to bound the number of steps at strength 1. For each strict improvement, B is set to $1/\ln n \cdot u$ where u is the counter value, the counter is reset, and radius r is set to 2. Thereafter, B steps pass before the strength becomes 2 again. Hence, if the strength does not exceed 2 before the optimum is reached, this contributes a term of $(1 + 1/\ln n)$ to the number of iterations with strength 2 in the previous epoch. Also, in the beginning of the algorithm, there is a complete phase with strength 1 costing $m \ln R$, which only contributes a lower-order term. □

Theorem 5.11 is interesting since it is asymptotically tight and does not suffer from the additional $\log(w_{\max})$ factor known from the analysis of the classical (1+1) EA [NW07]. In fact, this seems to be the first asymptotically tight analysis of a globally searching (1+1)-type algorithm on the MST. So far a tight analysis of evolutionary algorithms on the MST was only known for $\text{RLS}^{1,2}$ with one- and two-bit flip mutations [RKJ06]. Our bound in Theorem 5.11 is by a factor of roughly 2 better since it avoids an expected waiting time of 2 for a two-bit flip. On the technical side, it is interesting that we could apply drift analysis in its proof despite the algorithm being able to switch between different mutation strengths influencing the current drift.

5.6 Radius Memory can be Detrimental

After a high mutation strength has been selected, e. g., to overcome a local optimum, the radius memory decreases the threshold values for phase lengths related to lower strengths. As we have seen in Lemma 5.5, SD-RLS^{m} can often

return to a smaller strength quickly. However, we can also point out situations where using the smaller strength with their original threshold values as used in the original SD-RLS$^{\mathrm{r}}$ from [RW21b] is crucial.

Our example is based on a general construction principle that can be traced back to [Wit03] and was picked up in [RW20b] to show situations where stagnation in the context of the (1+1) EA is detrimental; see that paper for a detailed account of the construction principle. In [RW21b], the idea was used to demonstrate situations where bit-flip mutations outperforms SD-RLS$^{\mathrm{r}}$. Roughly speaking, the functions combine two gradients one of which is easier to exploit for an algorithm A while the other is easier to exploit for another algorithm B. By appropriately defining local and global optima close to the end of the search space in direction of the gradients, either Algorithm A significantly outperforms Algorithm B or the other way round.

We will now define a function on which SD-RLS$^{\mathrm{m}}$ is exponentially slower than SD-RLS$^{\mathrm{r}}$. In the following, we will imagine any bit string x of length n as being split into a prefix $a \coloneqq a(x)$ of length $n-m$ and a suffix $b \coloneqq b(x)$ of length m, where m is defined below. Hence, $x = a(x) \circ b(x)$, where $\circ$ denotes the concatenation. The prefix $a(x)$ is called *valid* if it is of the form $1^i 0^{n-m-i}$, i. e., i leading ones and $n-m-i$ trailing zeros. The prefix fitness $\textsc{pre}(x)$ of a string $x \in \{0,1\}^n$ with valid prefix $a(x) = 1^i 0^{n-m-i}$ equals i, the number of leading ones. The suffix consists of $\lceil n^{1/8} \rceil$ consecutive blocks of $\lceil n^{3/4} \rceil$ bits each, altogether $m = O(n^{7/9})$ bits. Such a block is called *valid* if it contains either 0 or 2 one-bits; moreover, it is called *active* if it contains 2 and *inactive* if it contains 0 one-bits. A suffix where all blocks are valid and where all blocks following first inactive block are also inactive is called valid itself, and the suffix fitness $\textsc{suff}(x)$ of a string x with valid suffix $b(x)$ is the number of leading active blocks before the first inactive one. Finally, we call $x \in \{0,1\}^n$ valid if both its prefix and suffix are valid.

The final fitness function is a weighted combination of $\textsc{pre}(x)$ and $\textsc{suff}(x)$. We define for $x \in \{0,1\}^n$, where $x = a \circ b$ with the above-introduced a and b,

$$\textsc{PreferOneBitFlip}(x) \coloneqq \begin{cases} n - m - \textsc{pre}(x) + \textsc{suff}(x) & \text{if } \textsc{suff}(x) \leq n^{1/9} \wedge x \text{ valid} \\ n^2 \textsc{pre}(x) + \textsc{suff}(x) & \text{if } n^{1/9} < \textsc{suff}(x) \leq n^{1/8}/2 \wedge x \text{ valid} \\ n^2(n-m) + \textsc{suff}(x) - n - 1 & \text{if } \textsc{suff}(x) > n^{1/8}/2 \wedge x \text{ valid} \\ -\textsc{OneMax}(x) & \text{otherwise.} \end{cases}$$

We note that all search points in the third case have a fitness of at least $n^2(n-m) - n - 1$, which is bigger than $n^2(n-m-1) + n$, an upper bound on the

fitness of search points that fall into the second case without having m leading active blocks in the suffix. Hence, search points x where $\text{PRE}(x) = n - m$ and $\text{SUFF}(x) = \lceil n^{1/8} \rceil$ represent local optima of second-best overall fitness. The set of global optima equals the points where $\text{SUFF}(x) = \lfloor n^{1/8}/2 \rfloor$ and $\text{PRE}(x) = n-m$, which implies that at least $n^{1/8}$ bits (two from each block) have to be flipped simultaneously to escape from the local toward the global optimum. The first case is special in that the function is decreasing in the PRE-value as long as $\text{SUFF}(x) \leq n^{1/9}$. Typically, the first valid search point falls into the first case. Then two-bit flips are essential to make progress and the radius memory of SD-RLS$^{\text{m}}$ will be used when waiting for the next improvement. After leaving the first case, since two-bit flips happen quickly enough, the memory will make progress via one-bit flips unlikely, leading to the local optimum.

We note that function PREFERONEBITFLIP shares some features with the function NEEDHIGHMUT from [RW20b] and the function NEEDGLOBALMUT from [RW21b]. However, it contains an extra case for small suffix values, uses different block sizes and block count for the suffix, and inverts roles of prefix and suffix by leading to a local optimum when the suffix is optimized first.

In the following, we make the above ideas precise and show that SD-RLS$^{\text{r}}$ outperforms SD-RLS$^{\text{m}}$ on PREFERONEBITFLIP.

Theorem 5.12. *With probability at least* $1 - 1/n^{1/8}$*, SD-RLS*^{m} *with* $R \geq n^{3+\epsilon}$ *for an arbitrary constant* $\epsilon > 0$ *needs time* $2^{\Omega(n^{1/8})}$ *to optimize* PREFERONEBITFLIP*. With probability at least* $1 - 1/n$*, SD-RLS*^{r} *with* $R \geq n^{3+\epsilon}$ *optimizes the function in time* $O(n^2)$*.*

Proof. As in the proof of Theorem 4.1 in [RW20b], we have that the first valid search point (i. e., search point of non-negative fitness) of both SD-RLS$^{\text{r}}$ and SD-RLS$^{\text{m}}$ has both PRE- and SUFF-value value of at most $n^{1/9}/2$ with probability $2^{-\Omega(n^{1/9})}$. In the following, we tacitly assume that we have reached a valid search point of the described maximum PRE- and SUFF-value and note that this changes the required number of improvements to reach local or global maximum only by a $1 - o(1)$ factor. For readability this factor will not be spelt out any more.

Given the situation with a valid search point x where $\text{SUFF}(x) < n^{1/9}/2$, fitness improvements only possible by increasing the SUFF-value. Since the probability of a SUFF-improving steps is at least $\binom{n^{3/2}}{2}/n^2 = \Omega(n^{-1/2})$ at strength 2, the time for both algorithms to reach the second case of the definition of PREFERONEBITFLIP is $O(n^{1/9}n^{1/2}) = O(n^{11/18})$ according to Lemma 5.4, and by repeating independent phases and Markov's inequality, the time is $O(n)$ with probability exponentially close to 1. Afterwards, one-bit flips increasing the

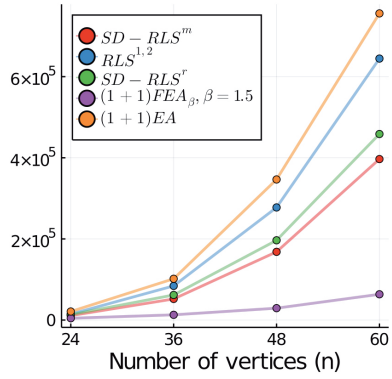
PRE-value are strictly improving and happen while the strength is 1 with probability at least $1 - 1/R$ in SD-RLS$^{\mathrm{r}}$, which does not have radius memory. Therefore, by a union bound over all $O(n)$ improvements, with probability at least $1-1/n$, SD-RLS$^{\mathrm{p}}$ increases the PRE-value to its maximum before the SUFF-value becomes greater than $\lceil n^{1/9} \rceil$. The time for this is $O(n^2)$ even with probability exponentially close to 1 by Chernoff bounds. Afterwards, by reusing the above analysis to leave the first case, with probability at least $1 - 1/n$ a number of $o(n^2)$ steps is sufficient for SD-RLS$^{\mathrm{p}}$ to find the global optimum. This proves the second statement of the theorem.

To prove the first statement, i. e., the claim for SD-RLS$^{\mathrm{m}}$, we first note that the probability of improving the PRE-value at strength 2 is $O(n^2)$ since two specific bits would have to flip. Hence, such steps never happen in $\Theta(n)$ steps with probability $1 - O(1/n)$. By contrast, a SUFF-improving step at strength 2, which has probability $\Omega(n^{-1/2})$, happens within $O(n^{3/4})$ steps with probability at least $1 - 1/n^{1/4}$ according to Markov's inequality. In this case, the radius memory of SD-RLS$^{\mathrm{m}}$ will set a threshold of $B = n^{3/4}$ for the subsequent iterations at strength 1. The probability of improving the PRE-value within this time is $O(1/n^{-1/4})$ by a union bound, noting the success probability of at most $1/n$. Hence, the probability of having at least $n^{1/8}$ improvements of the SUFF-value within $n^{3/4}$ steps each before an improvement of the PRE-value (at strength 1) happens, is at least $1 - 1/n^{1/8}$ by a union bound. If all this happens, the algorithm has to flip at least $n^{1/8}$ bits simultaneously, which requires $2^{\Omega(n^{1/8})}$ steps already to reach the required strength. The total failure probability is $O(1/n^{1/8})$. □

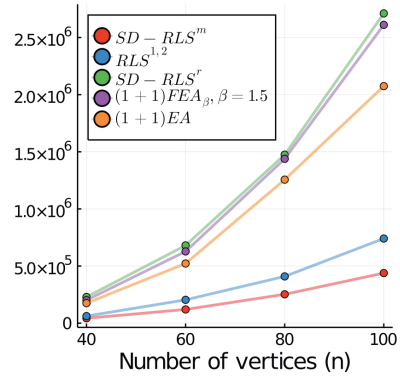
5.7 Experiments

We ran an implementation of five algorithms SD-RLS$^{\mathrm{m}}$, SD-RLS$^{\mathrm{r}}$, (1+1) FEA$_\beta$ with $\beta = 1.5$ from [DLMN17], the standard (1+1) EA and RLS^{1,2} on the MST problem with the fitness function from [NW07] for three types of graphs called TG, Erdős–Rényi with $p = (2 \ln n)/n$, and K_n. We carried out a similar experiment to [RW21b] with the more additional class of complete graphs K_n to illustrate the performance of the new algorithm and compare with the other algorithms.

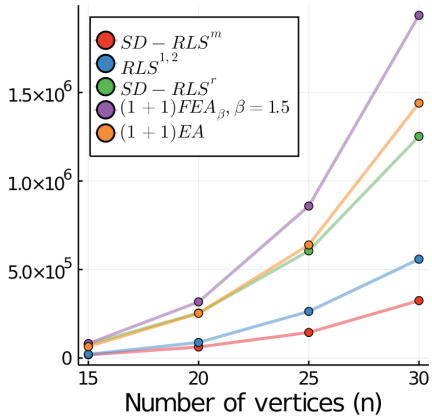
The graph TG represented in Figure 5.3 with n vertices and $m = 3n/4 + \binom{n/2}{2}$ edges contains a sequence of $p = n/4$ triangles which are connected to each other, and the last triangle is connected to a complete graph of size $q = n/2$. Regarding the weights, the edges of the complete graph have the weight 1, and we set the weights of edges in triangle to $2a$ and $3a$ for the side edges and the



(a) Graphs TG.



(b) Graphs Erdős–Rényi.



(c) Graphs K_n.

Figure 5.2: Average number of fitness calls (over 200 runs) the mentioned algorithms took to optimize the fitness function MST of the graphs.

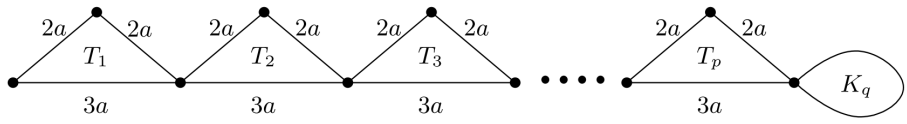


Figure 5.3: Example graph TG with $p = n/4$ connected triangles and a complete graph on q vertices with edges of weight 1 [NW07].

main edge, respectively. In this paper, we consider $a = n^2$. The graph TG is used for estimating lower bounds on the expected runtime of the (1+1) EA and RLS in the literature [NW07]. As can be seen in Figure 5.2a, (1+1) EA with heavy-tailed mutation (i. e., (1+1) FEA$_\beta$) with $\beta = 1.5$ outperformed the rest of the algorithms. However, SD-RLS$^{\mathrm{r}}$ and SD-RLS$^{\mathrm{m}}$ also outperformed the standard (1+1) EA and RLS^{1,2}.

Regarding the graphs Erdős–Rényi, we produced some Erdős–Rényi graphs randomly with $p = (2 \ln n)/n$ and assigned each edge an integer weight in the range $[1, n^2]$ uniformly at random. We also checked that the graphs are certainly connected. Then, we ran the implementation to find the MST of these graphs. The obtained results can be seen in Figure 5.2b. As discussed in Section 6 in [RW21b], SD-RLS$^{\mathrm{r}}$ does not outperform the (1+1) EA and RLS^{1,2} on MST with graphs when the number of strict improvements in SD-RLS$^{\mathrm{r}}$ is large. However, the proposed algorithm in this paper, SD-RLS$^{\mathrm{m}}$ outperformed the rest of the algorithms, although there can be a relatively large number of improvements on such graphs. We can also see this superiority in Figure 5.2c for the complete graphs K_n with random edge weights in the range $[1, n^2]$.

For statistical tests, we ran the algorithms on the graphs TG and Erdős–Rényi 200 times, and all p-values obtained from a Mann-Whitney U-test between the algorithms, with respect to the null hypothesis of identical behavior, are less than 10^{-2} except for the results regarding the smallest size in each set of graphs.

Conclusions

We have investigated stagnation detection with the s-bit flip operator as known from randomized local search and introduced a mechanism called *radius memory* that allows continued exploitation of large s values that were useful in the past. Improving earlier work from [RW21b], this leads to tight bounds on complex multimodal problems like linear functions with uniform constraints and the minimum spanning tree problem, while still optimizing unimodal and jump functions as efficiently as in earlier work. The bound for the MST is the first tight runtime bound for a global search heuristics and improves upon the runtime of classical RLS algorithms by a factor of roughly 2. We have also pointed out situations where the radius memory is detrimental for the optimization process. In the future, we would like to investigate the concept of stagnation detection with radius memory in population-based algorithms and plan analyses on further combinatorial optimization problems.

Acknowledgement

This work was supported by a grant by the Danish Council for Independent Research (DFF-FNU 8021–00260B).

Chapter 6

Paper D: Stagnation Detection Meets Fast Mutation

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.

6.1 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 [DLMN17, 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") [DLMN17] 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 [RW20b]. An improved version of this stagnation-detection approach [RW21b], 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 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 6.5 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 [BBD21a] (see again Section 6.5 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 [RW21b], 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 16 for the full details). Different from [RW21b], 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 16) 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 [RW21b], 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 6.5). 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 [RW21b]. The basic SD-RLS algorithm proposed in [RW21b] 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 [RW21b] 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 6.4. 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 $O\left(\binom{n}{\delta-1}\ln(R)\right)$, the one of the fast $(1+1)$ EA is $O\left(\delta^{\beta-0.5}(en/\delta)^\delta n^{-\Delta}\right)$, and the one of our algorithm is at most $O\left(\binom{n}{\delta}n^{-\Delta}\gamma^{-1}\right)$ (Corollary 6.11). 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 6.2, we introduce our new algorithm in Section 6.3. In Section 6.4, 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 6.5 to prove several runtime results, among others, for generalized jump functions. We present some experimental results in Section 6.6. In Section 6.7, 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 6.8.

6.2 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 [DD20]) and how evolutionary algorithms can leave local optima (here we refer to [Doe20a, Section 2.1] for a discussion of non-elitist approaches and to the introduction of [DFK+18] 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 [DLMN17] (the earlier work [Prü04] 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 [DLMN17] 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 [FQW18, FGQW18b, FGQW18a, WQT18, ABD20a, ABD20b, AD20, ABD21, DZ21, COY21a, COY21b].

A different way to cope with local optima was proposed in [RW20b]. 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 [RW20b, RW21a, RW21b]. As shown in [RW20b], 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 [RW21b]. The simple one, obtained from just replacing standard bit mutation in [RW20b] 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 [RW21b], which by cyclically reverting to smaller mutation strengths overcomes the problem that, with small probability, a given solution in distance m is not found in the phase which uses m-bit flips. In [RW21a], 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

Algorithm 16: 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 $r_1 \leftarrow 1$;
$u \leftarrow 0$;
for $t \leftarrow 1, 2, \ldots$ **do**
  Set $s = r_t$ with probability $1-\gamma$ or
  $s = r_t + \text{pow}(\beta, n - r_t)$ with probability $\gamma/2$ or
  $s = r_t - \text{pow}(\beta, \max\{1, r_t - 1\})$ with probability $\gamma/2$;
  Create y by flipping s bits in a copy of x uniformly at random;
  $u \leftarrow u + 1$;
  if $f(y) > f(x)$ **then**
    $x \leftarrow y$;
    $r_{t+1} \leftarrow 1$;
    $u \leftarrow 0$;
  else
    if $f(y) = f(x)$ and $r_t = 1$ **then**
      $x \leftarrow y$;
    if $u \geq \ell_{r_t}$ **then**
      $r_{t+1} \leftarrow \min\{r_t + 1, \lfloor \frac{n}{2.1} \rfloor\}$;
      $u \leftarrow 0$;
    else
      $r_{t+1} \leftarrow r_t$;

detection has also been successfully used in multi-objective evolutionary computation [DZ21].

6.3 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 16. The function $\text{pow}(\beta, u)$ samples from a power-law distribution with exponent β and range $[1..u]$ as defined in Equation (6.1) 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

$$\Pr[X = i] = \begin{cases} C_{\beta,u} i^{-\beta} & \text{if } 1 \leq i \leq u, \\ 0 & \text{otherwise,} \end{cases} \tag{6.1}$$

where $C_{\beta,u} := (\sum_{j=1}^{u} j^{-\beta})^{-1}$ is the normalization coefficient. The function $\mathrm{pow}(\beta, u)$ used in Algorithm 16 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 6.1 below.

In this paper, we use maximum phase lengths of

$$\ell_r = (1-\gamma)^{-1} \binom{n}{r} \ln(R). \tag{6.2}$$

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 (6.2) 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 6.7, 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.

6.4 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,

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

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 16) 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 6.1. *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 16. 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 $r=1$, $\Pr[X=0] = \gamma/2$.

Proof. It is immediately visible from Algorithm 16 that $\Pr[X = r] = 1 - \gamma$. For $1 \leq \alpha < r$, we have

$$\begin{aligned}\Pr[X = \alpha] &= \Pr[X < r] \cdot \Pr[X = \alpha \mid X < r] \\ &= \Pr[X < r] \cdot \Pr[\mathrm{pow}(\beta, r-1) = r - \alpha] \\ &= (\gamma/2) \cdot C_{\beta,r-1}(r-\alpha)^{-\beta}.\end{aligned}$$

For $\alpha > r$, we similarly obtain

$$\begin{aligned}\Pr[X = \alpha] &= \Pr[X > r] \cdot \Pr[X = \alpha \mid X > r] \\ &= \Pr[X > r] \cdot \Pr[\mathrm{pow}(\beta, n-r) = \alpha - r] \\ &= (\gamma/2) \cdot C_{\beta,n-r}(\alpha-r)^{-\beta}.\end{aligned}$$

For $r = 1$, $\Pr[X = 0] = \gamma/2$ because in this case, $\mathrm{pow}(\beta, \max\{1, r_t - 1\}) = \mathrm{pow}(\beta, 1)$ returns 1 only. □

In a run of the algorithm SD-FEA$_{\beta,\gamma,R}$, assume w is the current search point immediately following a strict improvement or the initial search point. Since in phase 1 the algorithm accepts offspring with the same fitness value, the current search point is changing and might be different from w. However, in phase 2 and larger, the current search point is fixed until a strict improvement is found, see Algorithm 16. Let x be the selected search point at the beginning of phase 2, which is basically the latest selected search point in fitness level of w. From phase 2, the algorithm needs to flip at least m bits to find a strict improvement, where $m \coloneqq \mathrm{IndividualGap}(x)$ and $m \leq \mathrm{FitnessLevelGap}(w)$ according to their definitions. That is the reason why in the following results, we also consider the latest selected search point in the fitness level.

Also, in the case that $\mathrm{FitnessLevelGap}(w) > 1$ but $\mathrm{IndividualGap}(x) = 1$, some of the selected search points might have individual gaps larger than 1 but the last one called x in phase 1. Then, we cannot claim that the algorithm spends sufficient iterations in the promising phase 1 for x. Hence this case is excluded in the following results.

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 6.2. *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 $w \in \{0,1\}^n$ be the current search point immediately following a strict improvement or the initial search point. Let $x \in \{0,1\}^n$ be the latest selected search point in fitness level of w and $m = \mathrm{IndividualGap}(x)$. Let E_1^{r-1} denote the probability of not*

finding an improvement in phases 1 to $r-1$ and $m < r \leq \lfloor \frac{n}{2.1} \rfloor$. Then for either $m \geq 2$ or $\mathrm{FitnessLevelGap}(w) = 1$*, we have*

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

Proof. Let p_r be a lower bound on the probability of making progress in phase r in one iteration. Then we have

$$\begin{aligned}\Pr[E_1^{r-1}] &\leq \Pr[E_m \cap \cdots \cap E_{r-1}] = \prod_{i=m}^{r-1} \Pr[E_i] \leq \prod_{i=m}^{r-1} (1-p_i)^{\ell_i} \\ &\leq \exp\left(-\sum_{i=m}^{r-1} p_i \ell_i\right), \end{aligned} \tag{6.3}$$

where we use the inequality $1 + x \leq e^x$ for all $x \in \mathbb{R}$.

In the following paragraphs, we aim at bounding $p_i \ell_i$ from below. Note that the search point x might be different from w because in phase 1 the algorithm accepts offspring with the same fitness value. However, in phase 2 and larger, x is fixed until a strict improvement is found.

For $i = m$, the promising phase to make progress, if $m \geq 2$, w is the fixed current search point in phase m until making a strict improvement or increasing the phase. Otherwise, if $\mathrm{FitnessLevelGap}(x) = 1$ resulting in $m = 1$ and the individual gap 1 for all search points in the fitness level x, the algorithm can always make progress by changing a bit although the current search point might change in phase 1. In both cases either $\mathrm{FitnessLevelGap}(x) = 1$ or $m \geq 2$, the success probability in all iterations in phase m equals $(1-\gamma)\binom{n}{m}^{-1}$ via Lemma 6.1, so

$$p_m \ell_m \geq (1-\gamma)\binom{n}{m}^{-1} \cdot (1-\gamma)^{-1}\binom{n}{m}\ln(R) = \ln(R).$$

For $m < i \leq \frac{n}{2.1}$, again using Lemma 6.1, we have

$$p_i \geq (\gamma/2) C_{\beta,i-1}(i-m)^{-\beta}\binom{n}{m}^{-1},$$

and thus

$$p_i \ell_i \geq (\gamma/2) \cdot C_{\beta,i-1} \frac{\binom{n}{i}\ln(R)}{(1-\gamma)(i-m)^{\beta}\binom{n}{m}} \geq (\gamma/2) \cdot C_{\beta,n} \frac{\binom{n}{i}\ln(R)}{(i-m)^{\beta}\binom{n}{m}},$$

where we have used $C_{\beta,n} \leq C_{\beta,i-1}$. The last expression is bounded from below by

$$(\gamma/2) \cdot C_{\beta,n} \frac{\ln(R)}{(i-m)^\beta} \cdot \frac{\binom{n}{i}}{\binom{n}{i-1}} \cdots \frac{\binom{n}{m+1}}{\binom{n}{m}} \geq (\gamma/2) \cdot C_{\beta,n} \frac{\ln(R)}{(i-m)^\beta} (1.1)^{i-m}, \quad (6.4)$$

where we have used $\binom{n}{k}/\binom{n}{k-1} = \frac{n-k+1}{k} \geq 1.1$ for $k \leq \lfloor \frac{n}{2.1} \rfloor$.

We finally show that $1.1^k/k^\beta \geq (\ln(1.1)/\beta)^\beta$ for $k \in \mathbb{N}_{\geq 1}$. To prove this, let $f(x) = 1.1^x/x^\beta$. For $x > 0$, its derivative, i. e., $f'(x)$, has only one root, namely $\hat{x} = \frac{\beta}{\ln 1.1}$. Before and after this point the function is decreasing and increasing, respectively, so $f(\hat{x})$ is the minimum value of the function for $x > 0$. We have

$$f(\hat{x}) = \frac{1.1^{\beta/\ln(1.1)}}{(\beta/\ln(1.1))^\beta} \geq \left(\frac{\ln(1.1)}{\beta}\right)^\beta.$$

Thus, Equation (6.4) is bounded from below by $(\gamma/2) \cdot C_{\beta,n} (\ln(1.1)/\beta)^\beta \ln(R)$.

From Equation (6.3), we obtain

$$\Pr[E_1^{r-1}] \leq \exp\left(-\sum_{i=m}^{r-1} p_i \ell_i\right) \leq R^{-1-(\gamma/2)\cdot\left(\frac{\ln(1.1)}{\beta}\right)^\beta C_{\beta,n}(r-m-1)}. \qquad \square$$

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 6.3. *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 $w \in \{0,1\}^n$ be the current search point immediately following a strict improvement or the initial search point. Let $x \in \{0,1\}^n$ be the latest selected search point in fitness level of w. Assume $m = \text{IndividualGap}(x)$ 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 point x conditional on flipping m bits. Then, if either $m \geq 2$ or $\text{FitnessLevelGap}(w) = 1$, the expected number of iterations spent with strengths larger than m is at most*

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

Proof. Let I_r be the number of iterations spent in phase r and $E[I_{>m}]$ denote the expected number of iterations spent with strengths larger than m. Then

$$E[I_{>m}] = \sum_{r=m+1}^{\lfloor \frac{n}{2.1} \rfloor} E[I_r].$$

With probability $\Pr[E_1^{r-1}]$, the algorithm does not make progress with strengths less than r. In phase r, the probability of finding an improvement is at least $C_{\beta,r-1}(\gamma/2)(r-m)^{-\beta} \cdot s_m$ in each iteration, by Lemma 6.1. Thus, for all strengths $r > m$, using the law of total probability, we have

$$\begin{aligned} E[I_r] &= \Pr\left[E_1^{r-1}\right] E[I_r \mid E_1^{r-1}] + \Pr\left[\overline{E_1^{r-1}}\right] E\left[I_r \mid \overline{E_1^{r-1}}\right] \\ &\leq \Pr[E_1^{r-1}] \cdot (C_{\beta,r-1})^{-1} 2\gamma^{-1} \cdot \frac{1}{s_m}(r-m)^{\beta} + \Pr\left[\overline{E_1^{r-1}}\right] \cdot 0 \\ &= \Pr[E_1^{r-1}] \cdot (C_{\beta,r-1})^{-1} 2\gamma^{-1} \cdot \frac{1}{s_m}(r-m)^{\beta}. \end{aligned}$$

Using Lemma 6.2 and $R \geq e^{1/\gamma}$, we can bound

$$\begin{aligned} E[I_r] &\leq R^{-1-(\gamma/2)\left(\frac{\ln(1.1)}{\beta}\right)^{\beta} C_{\beta,n}(r-m-1)} (C_{\beta,r-1})^{-1} 2\gamma^{-1} \frac{1}{s_m}(r-m)^{\beta} \\ &= O\left(R^{-1}\gamma^{-1} \frac{1}{s_m} \frac{(r-m)^{\beta}}{\exp\left[(1/2) \cdot \left(\frac{\ln(1.1)}{\beta}\right)^{\beta} C_{\beta,n}(r-m-1)\right]} \right), \end{aligned}$$

where we have used $(C_{\beta,r-1})^{-1} = O(1)$ for $\beta > 1$. This results in

$$\begin{aligned} \sum_{r=m+1}^{\lfloor \frac{n}{2.1} \rfloor} E[I_r] &\leq O\left(R^{-1}\gamma^{-1} \frac{1}{s_m} \sum_{r=m+1}^{\lfloor \frac{n}{2.1} \rfloor} \frac{(r-m)^{\beta}}{\exp\left[(1/2) \cdot \left(\frac{\ln(1.1)}{\beta}\right)^{\beta} C_{\beta,n}(r-m)\right]} \right) \\ &\leq O\left(R^{-1}\gamma^{-1} \frac{1}{s_m} \right), \end{aligned}$$

where we estimated

$$\begin{aligned} \sum_{r=m+1}^{\lfloor \frac{n}{2.1} \rfloor} \frac{(r-m)^{\beta}}{\exp\left[(1/2) \cdot \left(\frac{\ln(1.1)}{\beta}\right)^{\beta} C_{\beta,n}(r-m)\right]} &= \sum_{r=m+1}^{\lfloor \frac{n}{2.1} \rfloor} \frac{(r-m)^{\beta}}{e^{\Theta(r-m)}} \\ &\leq \sum_{k=1}^{\infty} \frac{k^{\beta}}{e^{\Theta(k)}} = O(1). \end{aligned}$$

Therefore, we obtain

$$E[I_{>m}] = \sum_{r=m+1}^{\lfloor \frac{n}{2.1} \rfloor} E[I_r] = O\left(R^{-1}\gamma^{-1} \frac{1}{s_m} \right)$$

as claimed. □

The following lemma, a combinatorial inequality taken from [RW21b], will be used to count the number of iterations spent with strengths smaller than the fitness level gap.

Lemma 6.4 (Lemma 1 in [RW21b])**.** *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 6.5. *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 $w \in \{0,1\}^n$ be the current search point immediately following a strict improvement or the initial search point. Define T as the time SD-FEA$_{\beta,\gamma,R}$ takes to create a strict improvement. Let $x \in \{0,1\}^n$ be the latest selected search point in fitness level of w and $m = \mathrm{IndividualGap}(x)$. If $2 \leq m \leq n/2.1$, 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

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

Proof. Let I_r be the number of iterations spent in phase r. Using linearity of expectation, we have

$$E[T] = \sum_{r=1}^{\lfloor \frac{n}{2.1} \rfloor - 1} E[I_r] + E[I_{\lfloor \frac{n}{2.1} \rfloor}].$$

Let first $m \leq n/2.1$. For $r < m$, we use that $E[I_r]$ is at most the maximum length of phase r, i. e., $\ell_r = (1-\gamma)^{-1}\binom{n}{r}\ln(R)$. Thus, with Lemma 6.4, we compute

$$\begin{aligned}\sum_{r=1}^{m-1} E[I_r] &\leq \sum_{r=1}^{m-1} \binom{n}{r}\frac{\ln(R)}{1-\gamma} \\ &\leq \binom{n}{m-1}\frac{\ln(R)}{1-\gamma} \cdot \frac{n-(m-2)}{n-(2m-3)}\end{aligned}$$

$$= \binom{n}{m} \frac{\ln(R)}{1-\gamma} \cdot \frac{m}{n-m+1} \cdot \frac{n-(m-2)}{n-(2m-3)}.$$

Since $m \leq \frac{n}{2.1}$, the last expression is bounded from above by

$$\sum_{r=1}^{m-1} E[I_r] = O\left(\binom{n}{m} \frac{m \ln(R)}{(1-\gamma)n}\right).$$

When the strength r equals m, with probability $1-\gamma$, the algorithm flips exactly m bits (Lemma 6.1). When m bits are flipped, with probability at least $\binom{n}{m}^{-1}$ an improvement is found. Regarding a truncated geometric distribution with success probability $(1-\gamma)\binom{n}{m}^{-1}$, within at most $(1-\gamma)^{-1}\binom{n}{m}$ iterations in expectation the algorithm finds a better point or the phase is terminated. Thus

$$E[I_m] \leq \frac{\binom{n}{m}}{(1-\gamma)}.$$

For $r > m$, using Lemma 6.3 with $s_m \geq \binom{n}{m}^{-1}$, we obtain

$$E[I_{>m}] = \sum_{r=m+1}^{\lfloor \frac{n}{2.1} \rfloor} E[I_r] = O\left(R^{-1}\gamma^{-1}\binom{n}{m}\right).$$

Altogether, we have

$$\begin{aligned} E[T] &= \sum_{r=1}^{\lfloor \frac{n}{2.1} \rfloor} E[I_r] = \sum_{r=1}^{m-1} E[I_r] + E[I_m] + \sum_{r=m+1}^{\lfloor \frac{n}{2.1} \rfloor} E[I_r] \\ &\leq \binom{n}{m}\left(\frac{1}{1-\gamma} + O\left(\frac{m\ln(R)}{(1-\gamma)n} + R^{-1}\gamma^{-1}\right)\right). \end{aligned}$$

To prove the second claim, since for $r \leq \lfloor \frac{n}{2.1} \rfloor - 1$, we have that $E[I_r]$ is at most the maximum length of phase r, we have

$$E[T] \leq \sum_{r=1}^{\lfloor \frac{n}{2.1} \rfloor - 1} \ell_r + E[I_{\lfloor \frac{n}{2.1} \rfloor}] = \sum_{r=1}^{\lfloor \frac{n}{2.1} \rfloor - 1} \binom{n}{r} (1-\gamma)^{-1} \ln(R) + E[I_{\lfloor \frac{n}{2.1} \rfloor}].$$

In phase $\lfloor \frac{n}{2.1} \rfloor$, the algorithm no longer increases the strength until finding an improvement. Using Lemma 6.1, the improvement is found with probability at

least

$$\Omega\left((\gamma/2)\cdot|\lfloor\tfrac{n}{2.1}\rfloor-m|^{-\beta}\cdot\binom{n}{m}^{-1}\right)$$

in each iteration. Using the geometric distribution with this success probability, we obtain

$$\begin{aligned}E[T] &\leq \sum_{r=1}^{\lfloor\frac{n}{2.1}\rfloor-1}\binom{n}{r}(1-\gamma)^{-1}\ln(R)+O\left(\gamma^{-1}\binom{n}{m}|\lfloor\tfrac{n}{2.1}\rfloor-m|^{\beta}\right)\\ &= O\left(2^n\frac{\ln(R)}{1-\gamma}+\gamma^{-1}\binom{n}{m}|\lfloor\tfrac{n}{2.1}\rfloor-m|^{\beta}\right),\end{aligned}$$

where we have used $\sum_{i=0}^{n}\binom{n}{i}=2^n$. The second part is proven as desired. □

Theorem 6.5 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 6.6. *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 $w\in\{0,1\}^n$ be the current search point immediately following a strict improvement or the initial search point. Let x be the latest selected search point in fitness level of w and $m=\mathrm{IndividualGap}(x)$. Let s_m be a lower bound on the probability that a strict improvement is found from search point $x\in\{0,1\}^n$ conditional on flipping m bits. Define T as the time SD-FEA$_{\beta,\gamma,R}$ takes to create a strict improvement. If $2\leq m\leq n/2.1$, 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 $r'=\min\left\{m,\arg\max_r\left\{\binom{n}{r}\leq\frac{1}{s_m}\frac{1}{\gamma}(m-r)^{\beta}\right\}\right\}$.

Proof. Let I_r be the number of iterations spent in phase r. Using linearity of expectation, we have

$$E[T]=\sum_{r=1}^{\lfloor\frac{n}{2.1}\rfloor}E[I_r].$$

For $r < r'$, we use that $E[I_r]$ is at most the maximum length of phase r. Thus, by Lemma 6.4, we have

$$\begin{aligned}\sum_{r=1}^{r'-1} E[I_r] &\leq \sum_{r=1}^{r'-1} \binom{n}{r}(1-\gamma)^{-1}\ln(R)\\ &\leq \binom{n}{r'-1}\frac{\ln(R)}{(1-\gamma)}\frac{n-(r'-2)}{n-(2r'-3)}\\ &= \frac{r'}{n-r'+1}\cdot\binom{n}{r'}\frac{\ln(R)}{(1-\gamma)}\frac{n-(r'-2)}{n-(2r'-3)}.\end{aligned}$$

Since $r' \leq m \leq \frac{n}{2.1}$, the last expression is bounded from above by

$$O\left(\binom{n}{r'}\frac{r'\ln(R)}{(1-\gamma)n}\right).$$

Since $\binom{n}{r'} \leq \frac{1}{s_m}\frac{1}{\gamma}(m-r')^{\beta}$ by definition of r', we estimate

$$\sum_{r=1}^{r'-1} E[I_r] = O\left(\frac{1}{s_m}\cdot\frac{1}{\gamma}(m-r')^{\beta}\cdot\frac{r'\ln(R)}{(1-\gamma)n}\right).$$

In the phases from r' to $m-1$, the probability of finding an improvement is at least $s_m(\gamma/2)C_{\beta,n-r'}(m-r')^{-\beta}$, see Lemma 6.1. Hence the expected time spent in phases r' to $m-1$ is

$$\sum_{r=r'}^{m-1} E[I_r] = O\left(\frac{1}{s_m}\cdot\frac{1}{\gamma}(m-r')^{\beta}\right).$$

In phase m, where the strength is m, exactly m bits are flipped with probability $1-\gamma$ (Lemma 6.1), and in this phase an improvement is found with probability at least s_m when m bits are flipped. Thus

$$E[I_m] \leq \frac{1}{s_m}\cdot\frac{1}{(1-\gamma)}.$$

For $r > m$, using Lemma 6.3 with s_m, we obtain

$$\sum_{r=m+1}^{\lfloor\frac{n}{2.1}\rfloor} E[I_r] = O\left(R^{-1}\gamma^{-1}s_m^{-1}\right).$$

Altogether, we have

$$E[T] = \sum_{r=1}^{\lfloor\frac{n}{2.1}\rfloor} E[I_r]$$

$$
\begin{aligned}
&= \sum_{r=1}^{r'-1} E[I_r] + \sum_{r=r'}^{m-1} E[I_r] + E[I_m] + \sum_{r=m+1}^{\lfloor \frac{n}{2.1} \rfloor} E[I_r] \\
&\leq O\left(\frac{1}{s_m} \cdot \frac{1}{\gamma} (m-r')^{\beta} \cdot \frac{r' \ln(R)}{(1-\gamma)n} + \frac{1}{s_m} \cdot \frac{1}{\gamma} (m-r')^{\beta} + \frac{1}{s_m(1-\gamma)} + \frac{R^{-1}}{\gamma s_m} \right) \\
&\leq \frac{1}{s_m} \cdot \frac{1}{\gamma} (m-r')^{\beta} \cdot O\left(1 + \frac{r' \ln(R)}{(1-\gamma)n} \right). \qquad \square
\end{aligned}
$$

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. A maximization function is called *unimodal* in [DJW02] if and only if there is only one local maximum, where a local maximum is defined as a search point with no better neighbors. In this paper, we use this definition of unimodal functions. Thus, 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 [Weg01]. The theorem and its proof are similar to the second part of Lemma 4 in [RW21b], and with a good choice of parameters γ and R, the same asymptotic result can be achieved (see the following corollary).

Theorem 6.7. *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 $|\mathrm{Im}(f)|$ 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 $A_1, \ldots, A_{|\mathrm{Im}(f)|}$ 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*

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

Proof. We define by $I^{(i)}$ the number of all iterations spent to leave the fitness level i. Using linearity of expectation, we have

$$
E[T] = \sum_{i=1}^{|\mathrm{Im}(f)|-1} E[I^{(i)}].
$$

Let $I_r^{(i)}$ be the number of iterations spent in phase r after a search point for A_i was found. Then

$$I^{(i)} = \sum_{r=1}^{\lfloor \frac{n}{2.1} \rfloor} I_r^{(i)}.$$

As long as the strength is 1, the algorithm flips exactly one bit with probability at least $1-\gamma$ (Lemma 6.1). The worst-case time to leave fitness level i is at most $\frac{1}{(1-\gamma)s_i}$ using the geometric distribution with success probability $s_i(1-\gamma)$. Hence, for each fitness level i, we bound $E[I_1^{(i)}]$ from above by $\frac{1}{(1-\gamma)s_i}$, and for $r > 1$, we bound $E[I_r^{(i)}]$ from above by using Lemma 6.3 with $s_m = s_i$. Thus

$$E[I_{>1}^{(i)}] = O\left(R^{-1}\gamma^{-1}\frac{1}{s_i}\right).$$

Altogether, we have

$$\begin{aligned} E[T] &= \sum_{i=1}^{|\mathrm{Im}(f)|-1} E[I^{(i)}] \\ &\leq \sum_{i=1}^{|\mathrm{Im}(f)|-1} \left(\frac{1}{s_i(1-\gamma)} + O\left(R^{-1}\gamma^{-1}\frac{1}{s_i}\right)\right) \\ &\leq \left(\frac{1}{1-\gamma} + O\left(R^{-1}\gamma^{-1}\right)\right) \sum_{i=1}^{|\mathrm{Im}(f)|-1} \frac{1}{s_i}. \end{aligned}$$

□

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

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

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 6.7 applied on the unimodal functions OneMax with $s_i = (n-(i-1))/n$ and LeadingOnes with $s_i = 1/n$.

Corollary 6.8. *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 $(1+o(1))n^2$.*

6.5 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 6.1 for a depiction.

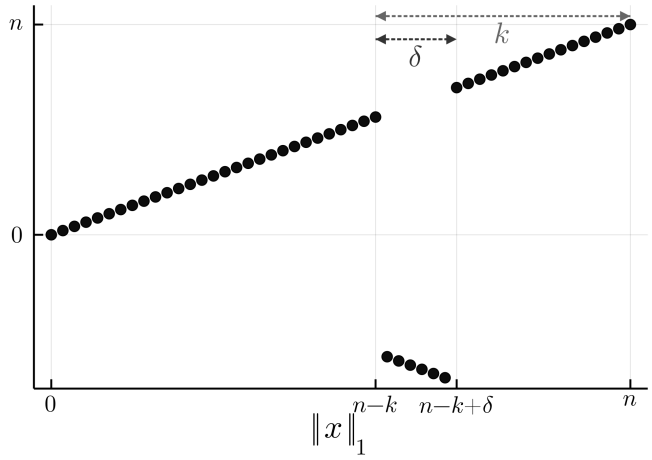


Figure 6.1: The function JUMP$_{k,\delta}$.

This function is based on the well-known JUMP benchmark [DJW02], 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

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

We refer the interested reader to see [BBD21a] 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$^{\text{r}}$) are carefully analyzed. Also, Rajabi and Witt [RW21a] 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$^{\text{m}}$) after leaving the local optimum. Recently, Witt in [Wit21] analyzes the performance of some other algorithms on the function 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 6.5. The result is formally proven in Theorem 6.9.

Theorem 6.9. *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 $2 \leq \delta = o\left(n/\ln(R)\right)$ satisfies*

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

Proof. Before reaching a local optimum with $n-m$ one-bits, $\textsc{Jump}_\delta$ is equivalent to OneMax. Thus, the expected time until SD-FEA$_{\beta,\gamma,R}$ reaches the local optimum is at most $O(n \ln n)$ via Theorem 6.7 with $s_i = (n - (i-1))/n$.

For a local optimum x we have $\text{FitnessLevelGap}(x) = \delta$ according to the definition of $\textsc{Jump}$. Hence, using Theorem 6.5, the algorithm finds the global optimum from the local optimum within the expected time at most

$$\binom{n}{\delta}(1 + o(1)).$$

This dominates the expected time the algorithm spends before reaching the local optimum. □

For $\gamma = \Theta(1)$, by closely following the analysis of Theorem 6.9, 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 6.10. *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 $\delta = o\left(n/\ln(R)\right)$ satisfies*

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

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

Proof. Until reaching the local optimum with $n-k$ one-bits, $\text{Jump}_{k,\delta}$ is equivalent to OneMax. Thus, the expected time until SD-FEA$_{\beta,\gamma,R}$ reaches the local optimum is at most $O(n \ln n)$ via Theorem 6.7 with $s_i = (n-(i-1))/n$.

For a local optimum x, we have $\text{FitnessLevelGap}(x) = \delta$ according to the definition of $\text{Jump}_{k,\delta}$. Using Theorem 6.6 with $s_m = \binom{n}{\delta}^{-1}\binom{k}{\delta}$, the algorithm finds a strict improvement with at least $n-k+\delta$ one-bits from the local optimum within expected time at most

$$O\left(\binom{n}{\delta}\binom{k}{\delta}^{-1}(\delta - r')^{\beta} \cdot \gamma^{-1}\right),$$

where we used our assumption $\delta = o(n/\ln(R))$.

After leaving the local optimum, $\text{Jump}_{k,\delta}$ is again equivalent to OneMax on the second slope. Using the same arguments as in the beginning of the proof, the expected time until SD-FEA$_{\beta,\gamma,R}$ reaches the global optimum is at most $O(n \ln n)$ via Theorem 6.7 with $s_i = (n-(i-1))/n$. □

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 6.11. *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 $\text{Jump}_{k,\delta}$ with $k = \omega(1) \cap O(\ln n)$ and $\delta = k - \Delta$ satisfies*

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

Proof. We show that r' defined in Theorem 6.10 is at least $k-2\Delta$. To this aim, we show that for $r \leq k-2\Delta$, we have

$$\frac{\binom{n}{r}}{\binom{n}{\delta}\binom{k}{\delta}^{-1}\gamma^{-1}(\delta-r)^{\beta}} \leq \gamma\frac{\binom{n}{k-2\Delta}}{\binom{n}{k-\Delta}\binom{k}{\Delta}^{-1}\Delta^{\beta}} \leq \gamma\frac{(en/(k-2\Delta))^{k-2\Delta}(ek/\Delta)^{\Delta}}{(n/(k-\Delta))^{k-\Delta}\Delta^{\beta}},$$

where we have used $\delta = k-\Delta$ and the inequality $(n/m)^m \leq \binom{n}{m} \leq (en/m)^m$. The last expression equals

$$\gamma\frac{e^{k-\Delta}k^{\Delta}}{n^{\Delta}\Delta^{\Delta+\beta}}\frac{(k-\Delta)^{k-\Delta}}{(k-2\Delta)^{k-2\Delta}} = \gamma\frac{e^{k-\Delta}k^{\Delta}}{n^{\Delta}\Delta^{\Delta+\beta}}(k-\Delta)^{\Delta}\left(1+\frac{\Delta}{k-2\Delta}\right)^{k-2\Delta}$$

$$\leq \gamma \frac{e^k k^\Delta}{n^\Delta \Delta^{\Delta+\beta}} (k-\Delta)^\Delta = o(1),$$

where we use the assumption $k \leq \ln n$ and the estimate $1+x \leq e^x$ for all $x \in \mathbb{R}$. Thus for $r \leq k - 2\Delta$ and a large enough n, we have

$$\binom{n}{r} \leq \binom{n}{\delta}\binom{k}{\delta}^{-1} \gamma^{-1}(\delta - r)^\beta,$$

which means that $r' \geq k - 2\Delta$. Therefore, using the result of Theorem 6.10 with $r' \geq k - 2\Delta$, we obtain

$$E[T] = O\left(\binom{n}{\delta}\binom{k}{\delta}^{-1} \Delta^\beta \cdot \gamma^{-1} + n \ln n\right) = O\left(\binom{n}{\delta}\binom{k}{\delta}^{-1} \gamma^{-1}\right),$$

where the $O(n \ln n)$ term is subsumed by the first term according to our assumptions. □

6.6 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 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 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 [DLMN17] with $\beta = 1.5$, the SD-(1+1) EA presented in [RW20b] with $R = n^2$, and SD-RLS$^{\mathrm{r}}$ from [RW21b] 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 6.2, 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

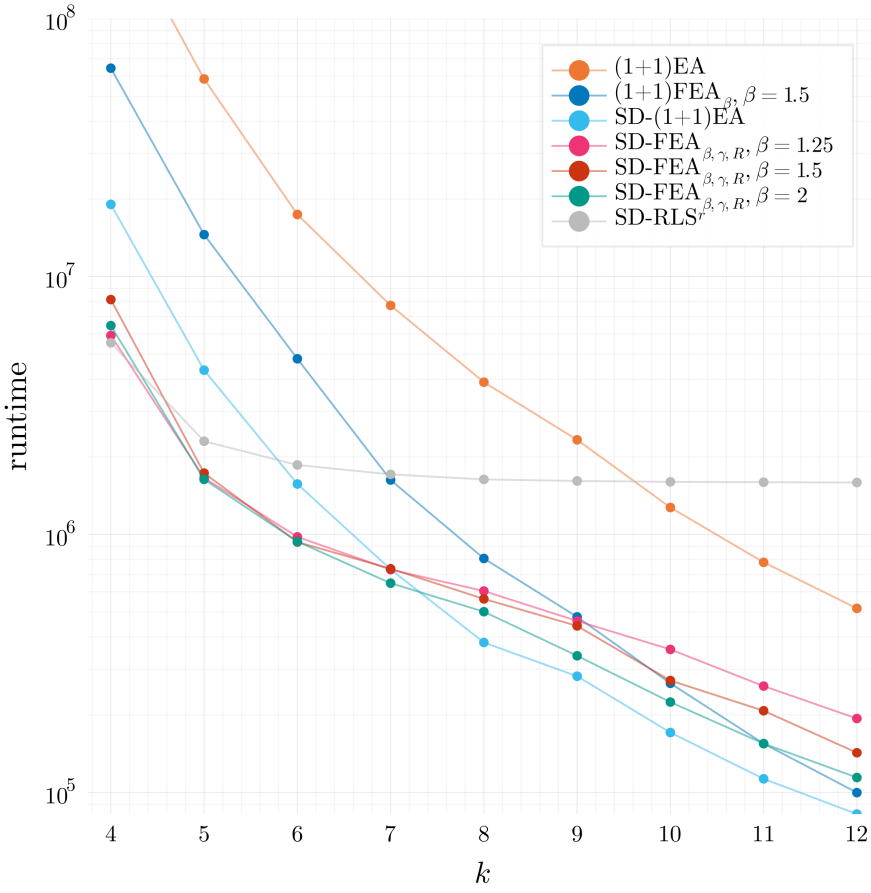


Figure 6.2: Average number (over 200 runs) of fitness calls the mentioned algorithms spent to optimize $\textsc{Jump}_{k,4}$ ($n = 100$) with different values for k.

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^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 $\gamma/2$.

6.7 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 [DLMN17], there is little indication that the precise value is important. The value $\beta = 1.5$ suggested in [DLMN17] 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 6.2). 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 6.5 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., $\gamma = 0.25$.

6.8 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 [RW21b]) 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 [RW21b], 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 $\Omega\left(n^{-m}\gamma\right)$ to find this particular improvement. Hence the probability that finding this improvement takes $\Omega(n^{m+1})$ time, is only $(1 - \Omega\left(n^{-m}\gamma\right))^{\Omega(n^{m+1})} \leq \exp(-\Omega(n\gamma))$.

As discussed in Section 6.7, 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.

Chapter 7

Paper E: How Fast Does the Metropolis Algorithm Leave Local Optima?

The Metropolis algorithm (MA) is a classical local search heuristic. It avoids getting stuck in local optimal by occasionally accepting inferior solutions. To better and in a rigorous manner understand this ability, we conduct a mathematical runtime analysis of the MA on the CLIFF benchmark. Apart from one local optimum, cliff functions are monotonically increasing towards the global optimum. Consequently, to optimize a cliff function, the MA only once needs to accept an inferior solution. Despite apparently being an ideal benchmark for the MA to profit from its main working principle, our runtime analysis shows that this hope does not come true. Even with the optimal temperature (the only parameter of the MA), the MA does not optimize most cliff functions faster than simple elitist evolutionary algorithms (EAs), which can only leave the local optimum by generating a superior solution possibly far away. This result suggests that our understanding of why the MA is often very successful in practice is not yet complete. It also suggests to try the MA with global mutation operators, an idea supported by our preliminary experiments.

7.1 Introduction

A major difficulty faced by many search heuristics is that the heuristic might run into a local optimum and then finds it hard to escape from it. A number of mechanisms have been proposed to overcome this difficulty, e. g., restart mechanisms, discarding good solutions (non-elitism), tabu mechanisms, global mutation operators (which can, in principle, generate any solution as offspring), or diversity mechanisms (which prevent a larger population to fully converge into a local optimum). While all these ideas have been successfully used in practice, a rigorous understanding of how these mechanisms work and in which situation to employ which one, is still largely missing.

To shed some light on this important question, we analyze how the Metropolis algorithm (MA) profits from its mechanism to leave local optima. The MA is a simple randomized hillclimber except that can also accept an inferior solution. This happens with some small probability which depends on the degree of inferiority and the *temperature*, the only algorithm parameter. Choosing the right temperature is a delicate problem – a too low temperature makes it hard to leave local optima, whereas a too high temperature lets the algorithm lose profitable solutions too quickly.

From this description of the MA one might speculate that the MA copes particularly well with local optima that have neighbors from which improving paths lead away from the local optimum. In this case, the local optimum can be left by just once accepting an inferior solution. The main result of this work is that this speculation does not come true. By conducting a rigorous runtime analysis of the MA on the Cliff benchmark (in which the above-mentioned property is very pronounced), we observe that the MA even with the optimal temperature finds the optimum of most Cliff functions not much faster than a simple elitist mutation-based algorithm called $(1 + 1)$ EA. If the $(1 + 1)$ EA uses an optimized mutation rate, then it outperforms the MA with optimal temperature on almost all cliff functions. Our experimental results support this finding and show that several simple heuristics using global mutation clearly outperform the MA on cliff functions. These results have motivated us to conduct preliminary experiments on the MA equipped with a global mutation operator instead of the usual one-bit flips. While not fully conclusive, these experiments show a good performance of the MA with global mutation operators on Cliff.

This paper is structured as follows. After a description of previous work in Section 7.2, we define in Section 7.3 the algorithms and benchmark problems under consideration. Sections 7.4 and 7.5 are devoted to the mathematical runtime analysis of OneMax and Cliff, respectively. Section 7.6 is the mathematical comparison of the MA and $(1 + 1)$ EA. Section 7.7 presents experimental

supplements to the theoretical analysis, emphasizing that the performance difference of the algorithms is rather pronounced for small problem sizes already.

7.2 Previous Works

The mathematical runtime analysis of randomized search heuristics has produced a decent number of results on how elitist evolutionary algorithm cope with local optima, but much fewer on other algorithms. The majority of results on evolutionary algorithms concern mutation-based algorithms. Results derived from the JUMP benchmark suggest that higher mutation rates or a heavy-tailed random mutation rate [DLMN17] as well as a stagnation-detection mechanism [RW20b] are preferable when the local optimum can only be left by moving to a more distant solution. Some elitist crossover-based algorithms showed a significant superiority in leaving local optima [JW02, DHK12, DFK+18], but overall crossover is not too well understood from the theoretical perspective.

There are a few runtime results on non-elitist evolutionary algorithms, however, they do not give a very positive picture. The results in [JS07, Leh10, Leh11, RS14, Doe20a] show that in many situations, there is essentially no room between a regime with low selection pressure, in which the algorithm cannot optimize any function with unique optimum efficiently, and a regime with large selection pressure, in which the algorithm essentially behaves like its elitist counterpart. With a very careful parameter choice, one can profit from non-elitism in a small middle regime, e. g., with a population size being neither too small nor too large in the order $\Theta(\log n)$, the (1,λ) EA optimizes the function $\textsc{Cliff}_{\frac{n}{3}-\frac{3}{2},\frac{n}{3}}$ in polynomial time [FS21]. While these results show that the non-elitism of the (1,λ) EA can be helpful, it has to be noted that the exponential dependence of the runtime on λ, roughly 6.20^{λ} in [FS21], implies that this algorithm parameter has to be chosen very carefully. Other examples of successful applications of non-elitism in evolutionary algorithms exist, e.g., [DEL21a], but most of these works consider artificial problems designed to demonstrate that a particular behavior can happen, but not giving much information on how widespread this behavior might be. Outside the range of well-established search heuristics, [PHST17] show that the strong-selection weak-mutation process from biology can optimize some functions faster than elitist evolutionary algorithms. [LOW19] show that the move-acceptance hyper-heuristic proposed in [LO13] can optimize cliff functions in cubic time.

For the MA algorithm, the rigorous understanding is less developed than for EAs. The classic result [SH88] shows that it can compute good approximations for the maximum matching problem. An analogous result was shown for the

$(1+1)$ EA [GW03], demonstrating that this problem can also be solved via elitist methods. [JS98] showed that the MA can solve certain random instances of the minimum bisection problem in quadratic time. [JW07] conducted a runtime analysis on the OneMax benchmark. While it is not surprising that the MA does not profit from its ability to accept inferior solutions on this unimodal benchmark, their result shows that only very small temperatures (namely such that the probability of accepting an inferior solution is at most $O(n/\log n)$) lead to polynomial runtimes. Again a number of results exist for artificially designed problems [DJW00, JW07, OPH+18]. [WZD21] show a good performance of the MA on the DLB problem (roughly by a factor of n faster than elitist EAs). This problem has (many) local optima, however, these are easy to leave since they all have a strictly better solution in Hamming distance two.

7.3 Preliminaries

7.3.1 The Metropolis Algorithm and the $(1+1)$ EA

The MA [MRR+53] is a simple single-trajectory search heuristic for pseudo-boolean optimization. It selects and evaluates a random neighbor of the current solution and accepts it (i) always if it is at least as good as the parent, and (ii) with probability $e^{-\delta/T}$ if its fitness is by δ is worse than the fitness of the current solution. Here T, often called *temperature*, is the single parameter of the MA. See Algorithm 17 for the pseudocode. To ease our later analyses, we use the parameterization $\alpha = e^{1/T}$, that is, the parameter $\alpha > 0$ fixes the probability $\alpha^{-\delta}$ of accepting a solution worse than the parent by δ. The MA and its generalization *Simulated Annealing* have found numerous successful applications in various areas, see, e.g., [vLA87].

Algorithm 17: Metropolis algorithm with temperature T for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$. We usually write $\alpha = e^{1/T}$.

Select $x^{(0)}$ uniformly at random from $\{0,1\}^n$;
for $t \leftarrow 0, 1, \ldots$ **do**
  Create y by flipping a bit of $x^{(t)}$ chosen uniformly at random;
  if $f(y) \geq f(x^{(t)})$ **then**
    $x^{(t+1)} \leftarrow y$;
  else
    $x^{(t+1)} \leftarrow y$ with probability $e^{(f(y)-f(x^{(t)}))/T}$ and
    $x^{(t+1)} \leftarrow x^{(t)}$ otherwise;

The $(1+1)$ EA (Algorithm 18) is a simple stochastic hillclimber. It follows a single search trajectory as the MA; however, it never accepts search points of inferior fitness. Since the $(1+1)$ EA flips each bit independently of the others, the mutation operator is global and can reach any search point with positive probability. In particular, it is able to escape from local optima by flipping the required number of bits for an improvement. The $(1+1)$ EA is intensively studied in the theory of evolutionary computation [DJW02] and serves as the basis for the study of more advanced evolutionary algorithms. In our formulation, it comes with the parameter p for the mutation rate. The setting $p = 1/n$ is known as the *standard mutation rate*, which is a default, rather robust choice used in many studies [DJW02, Wit13].

Algorithm 18: $(1+1)$ EA with mutation rate p for the maximization of $f\colon \{0,1\}^n \to \mathbb{R}$.

Select $x^{(0)}$ uniformly at random from $\{0,1\}^n$;
for $t \leftarrow 0, 1, \ldots$ **do**
  Create y by flipping each bit of $x^{(t)}$ independently with probability p;
  if $f(y) \geq f(x^{(t)})$ **then**
    $x^{(t+1)} \leftarrow y$;
  else
    $x^{(t+1)} \leftarrow x^{(t)}$;

The *runtime* (synonymously, *optimization time*) of the algorithms is the random first point in time t where an optimum has been sampled. Usually, its expected value, called expected runtime/optimization time, is analyzed.

7.3.2 The Cliff and OneMax Functions

The aim of this paper is to study how efficient the MA is at optimizing functions with a local optimum. Two well-established and well-studied benchmark functions to model situations with local optima are Jump [DJW02] and Cliff [JS07]. In this research, we investigate the Metropolis algorithm on Cliff instead of Jump for two primary reasons. Firstly, on Jump functions, since the difference between the fitness of the local optimum and its neighbors is of order n, it is unlikely that the algorithm accepts such a fitness decrease. Also, the deceptive valley in the function Jump does not allow the search to get far from the local optimum because of accepting all improvements. We refer the interested reader to [LOW19, Theorem 13] for a lower bound on the optimization time of Metropolis on Jump. We should mention that the authors in [OPH+18] also

studied the Metropolis algorithm on the function so-called Valley, which is a multimodal problem containing both increasing and decreasing valleys.

In contrast to Jump functions, for Cliff functions the valley of low fitness is more shallow and the fitness inside the valley is not deceptive, that is, the gradient is pointing towards the optimum. These properties could let them appear like an easy optimization problem for the Metropolis algorithm, but as our precise analysis for the full spectrum of temperatures will show, this is not true.

Cliff functions were originally defined with only one parameter determining the distance of the local optimum from the global optimum [JS07]. However, since the Metropolis algorithm is sensitive to function values when it accepts worse solutions, we are interested in analyzing different depths for the valley in the fitness function. That is why we define an additional parameter to express the depth of the cliff.

Let again $n \in \mathbb{N}$ denote the problem size. As before, we shall usually suppress this parameter from our notation. Let $m \in \mathbb{N}_{\geq 1}$ and $d \in \mathbb{R}_{>0}$ such that $m < n$ and $d < m - 1$. Then the function $\textsc{Cliff}_{d,m}$ is increasing as the number of one-bits of the argument increases except for the points with $n - m$ one-bits, where the fitness decreases sharply by d if we add one more one-bit to the search point. Formally,

$$\textsc{Cliff}_{d,m}(x) \coloneqq \begin{cases} \|x\|_1 & \text{if } \|x\|_1 \leq n - m, \\ \|x\|_1 - d - 1 & \text{otherwise.} \end{cases}$$

See Figure 7.1 for a graphical sketch. Note that the original cliff function can be obtained with fixed parameter $d = m - 3/2$.

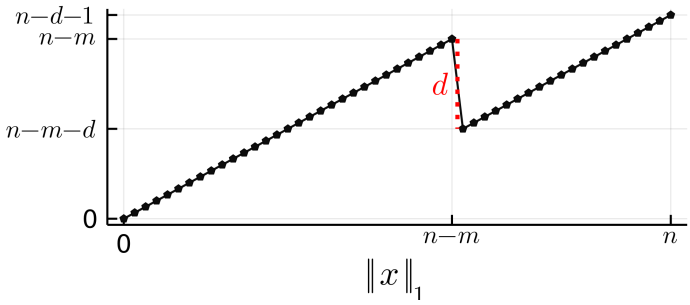


Figure 7.1: The function $\textsc{Cliff}_{d,m}$.

To the best of our knowledge, the only available analysis of the Metropolis Algorithm on Cliff functions is conducted in [LOW19, Theorem 10]. On $\textsc{Cliff}_{d,m}$

with fixed $d = m - 3/2$, the authors show a lower bound of

$$\min\left\{\frac{1}{2}\cdot\frac{n-m+1}{m-1}\cdot(n/\log n)^{m-3/2}, n^{\omega(1)}\right\},$$

indicating that even for reasonable settings of the parameter α the expected runtime grows exponentially in m, with the bound being capped a some super-polynomial function. However, if depth and location of the cliff are not linked, our analysis will show polynomial runtimes for big m as well.

On the two slopes of Cliff, the function is only depending on the number of one-bits of the search point and monotonically increasing in this number. If this applies to the whole search space, we obtain the function

$$\textsc{OneMax}(x) \coloneqq \|x\|_1,$$

which is intensively studied in the theory of evolutionary algorithms [DJW02, Wit13]. Understanding the MA on OneMax is crucial for the analysis on Cliff. Intuitively, if the MA is run on Cliff, assuming $m \ll n/2$, it first of all has to optimize a OneMax-like function to reach the cliff, accept the drop to jump down the cliff and then again optimize a OneMax-like function to reach the global optimum. However, it may happen that the MA returns to the cliff point or even points left of the cliff again after having overcome it for the first time.

7.4 Analysis of OneMax

In this section, we study the performance of the Metropolis algorithm on the OneMax benchmark. OneMax is the possibly best-studied benchmark in the theory of randomized search heuristics. For a given *problem size* $n \in \mathbb{N}$, the OneMax function is the mapping $f : \{0,1\}^n \to \mathbb{N}$ defined by $f(x) = \|x\|_1 = \sum_{i=1}^n x_i$ for all $x = (x_1, \dots, x_n) \in \{0,1\}^n$. This is an easy benchmark representing problems or parts of problems where the gradient points into the direction of the global optimum $x^* = (1, \dots, 1)$.

It is safe to say that $\Theta(n \log n)$ is a typical runtime of a randomized search heuristic optimizing OneMax. This runtime, more precisely, $(1 + o(1))n \ln$ [DDY20], was proven for the *randomized local search* heuristic, a randomized hillclimber that flips a single random bit and accepts the new solution if it is at least as good as the previous one. Many simple evolutionary algorithms also solve OneMax in time $\Theta(n \log n)$ with suitable parameters, e.g., the mutation-based $(\mu+\lambda)$ EA [Müh92, DJW02, Wit06, AD21].

It might appear surprising at first that it takes time $\Omega(n \log n)$ to find the correct value of n bits for a simple function like OneMax, where the correct value each bit can be found from the discrete partial derivative at any search point (i.e., by comparing the fitness of the search point and the search point obtained by flipping this bit). The reason is that many randomized search heuristics flip bits chosen at random and then the so-called coupon-collector effect implies that it takes $\Omega(n \log n)$ time until each bit was flipped at least once. It is clear that this problem can be overcome, and an $O(n)$ runtime can be obtained, by flipping the bits in a given order, however, as shown in [DFW10], this can lead to unexpected difficulties when trying to design algorithms that not always flip single bits. Linear runtimes on OneMax have also been obtained via crossover-based EAs [DD18, ADK20]. The black-box complexity of OneMax, that is, the best performance a black-box optimization algorithm can have on the class of all functions isomorphic to OneMax, is $\Theta(n/\log n)$ [ER63]. Despite the fact that these faster performances have been shown for particular algorithms, it still appears appropriate to call $\Theta(n \log n)$ the typical runtime of a general-purpose search heuristic on OneMax. In fact, Lehre and Witt [LW12] have shown that any unary unbiased black-box algorithm, that is, any black-box algorithm that treats the bit positions $1, \dots, n$ and the bit values 0 and 1 in a symmetric fashion (unbiasedness) and that creates new solution only from one parent (unary), takes time at least $\Omega(n \log n)$ on OneMax. A precise tight bound of $(1 \pm o(1))n \ln n$ was given in [DDY20].

We now conduct a precise analysis of how the Metropolis algorithm with different values of the parameter α performs on the OneMax problem. The only previous work on this question [JW07] has shown the following three results for the number T of iterations taken to find the optimum.

- If $\alpha \geq \varepsilon n$ for any positive constant ε, then $E[T] = O(n \log n)$.
- If $\alpha = o(n)$, then $E[T] = \Omega(\alpha 2^{n/3\alpha})$.
- $E[T]$ is polynomial in n if and only if $\alpha = \Omega(n/\log n)$.

This first work clearly shows that a relatively large value of α is necessary to efficiently optimize OneMax.

Our main result (Theorem 7.5) is very precise analysis of the runtime of the Metropolis algorithm on OneMax showing that for all $\alpha = \omega(\sqrt{n})$, we have

$$E[T] = (1 \pm o(1))n \ln(n) + (1 \pm o(1))\alpha \exp(\tfrac{n}{\alpha}).$$

This result covers the most interesting regime describing the transition from polynomial to exponential runtimes. Our methods would also allow to prove results for smaller values of α, but in the light of the previously shown $\exp(\Omega(n/\alpha))$

lower bound, these appear less interesting and consequently we do not explore this further.

Different from the previous work, our runtime result is tight apart from lower order terms for all $\alpha = \omega(\sqrt{n})$ and thus, in particular, for the phase transition between $(1+o(1))n \ln n$ and runtimes exponential in $\frac{n}{\alpha}$. From this, we learn that we have a runtime of $(1\pm o(1))n \ln(n)$ if $\alpha \geq \frac{n}{\ln \ln n}$, but that the runtime becomes $\omega(n \log n)$ when $\alpha \leq (1-\varepsilon)\frac{n}{\ln \ln n}$ for any constant $\varepsilon > 0$. Recall from above that $(1 \pm o(1))n \ln n$ is the best runtime a unary unbiased black-box algorithm can have on OneMax (and in fact any function $f : \{0,1\}^n \to \mathbb{R}$ with unique global optimum), so this insight characterizes the optimal parameter settings for the Metropolis algorithm on OneMax.

Our result also implies the known result that the runtime is polynomial in n if and only if $\alpha = \Omega(\frac{n}{\log n})$, however, we also make precise the runtime behavior in this critical phase: For all $\alpha = \frac{1}{c}\frac{n}{\ln n}$, the runtime is $(1 \pm o(1))\frac{1}{c}n^{c+1}(\ln n)^{-1}$.

We show this result not only because precise runtime results give a better picture of the performance of an algorithm, but also because our alternative analysis method gives additional insights on where this runtime stems from. In particular, we observe that a OneMax fitness of $\lceil n - \frac{n}{\alpha+1} \rceil$ is always obtained very efficiently (in expected time $(1+o(1))n \ln n$ at most). Hence as long as $\alpha = \omega(1)$, an almost optimal solution of fitness $(1 - o(1))n$ is found in that time. A third motivation for this detailed analysis on OneMax is that we need similar arguments in the next section, where we study the performance of the Metropolis algorithm to see how well it copes with local optima.

7.4.1 Preliminaries and Notation

From the symmetry of the Metropolis algorithm and the OneMax function, it is clear that all search points $x \in \{0,1\}^n$ having the same number of ones, that is, the same OneMax value, and thus the same number of zeroes, that is, same distance

$$d(x) := n - \textsc{OneMax}(x)$$

from the optimum, behave equivalently. For this reason, let us, for all $i \in [0..n]$, denote by $L_i = \{x \in \{0,1\}^n \mid d(x) = i\}$ the set of search points in distance i. Since the Metropolis algorithm creates new solutions by flipping single bits, an iteration starting with a solution $x \in L_i$ for some i can only end with a solution in L_{i-1}, L_i, or L_{i+1}. By definition of the algorithm, the probability for reducing the fitness distance from a solution $x \in L_i$ (that is, creating an offspring in L_{i-1})

is

$$p_i^- := \frac{i}{n}$$

and the probability for increasing the distance (that is, creating an offspring in L_{i+1} and accepting it as new solution) is

$$p_i^+ := \frac{n-i}{\alpha n}.$$

We note that this notation is different from the one used in [JW07], where the notation was based on the fitness and not on the distance. Hence our p_i^+ equals the p_{n-i}^- used there. We prefer to work with the distance since the more critical part of the optimization process is close to the optimum.

With the transition probabilities just defined, we can use simple Markov chain arguments to, in principle, compute the expected runtime. For $i \geq j$, denote by E_i^j the expected time the Metropolis algorithm (with some given parameter α suppressed in this notation) takes to find a solution in L_j when started with a solution in L_i. We abbreviate $E_i = E_i^{i-1}$. Then, by elementary properties of Markov processes,

$$E_i^j = \sum_{\ell=i}^{j+1} E_\ell. \tag{7.1}$$

From the one-step equation $E_i = 1 + p_i^+(E_{i+1} + E_i) + (1 - p_i^+ - p_i^-)E_i$, we derive the following equation, which was also used in [JW07].

$$E_i = \frac{1}{p_i^-} + \frac{p_i^+}{p_i^-} E_{i+1}. \tag{7.2}$$

The analysis of the runtime of the Metropolis algorithm on OneMax in [JW07] was solely based on the above two equations (with, of course, non-trivial estimates of the arising sums). In this work, we partially take a different route by separately analyzing the part of the process in which the algorithm has a positive expected fitness gain per iteration. This is when the fitness distance is still large and thus is it easy to find improving solutions. In this part of the process, we can conveniently use multiplicative drift analysis, a tool presented a few years after [JW07]. For the remainder of the process, we use arguments similar to those in [JW07], however, we profit from the fact that we need to cover only a smaller range of fitness levels.

7.4.2 Pseudo-linear Time in the Regime with Positive Drift

We start our analysis with the part of the process where the expected progress per iteration is positive. We recall that we denote by $d(x)$ the fitness distance (and Hamming distance) of x to the optimum, in other words, the number of zeros in x. Recalling that $x^{(t)}$ denotes the current solution at the end of iteration t (and $x^{(0)}$ the random initial solution), and defining $D_t = d(x^{(t)})$ for convenience, we see that the expected progress in one iteration satisfies

$$E[D_t - D_{t+1} \mid D_t] = p^-_{D_t} + \frac{1}{\alpha} p^+_{D_t} = \frac{D_t}{n} - \frac{n - D_t}{\alpha n} = D_t \left(\frac{1}{n} + \frac{1}{\alpha n}\right) - \frac{1}{\alpha}\,. \quad (7.3)$$

In particular, the expected progress is positive when $D_t > \frac{n}{\alpha+1} =: k^*$ and negative when $D_t < k^*$. An expected progress towards a target can be translated into estimates on the hitting time of this target, usually via so-called drift theorems [Len20], and this is our approach to show the following result.

Theorem 7.1. *Let $k^* = \frac{n}{\alpha+1}$ and $k = \lceil k^* \rceil$. Then the first time T such that the Metropolis algorithm with parameter α finds a solution $x^{(T)}$ with $d(x^{(T)}) \leq k$ satisfies*

$$E[T] \leq \frac{\alpha}{\alpha+1} n(\ln(n) + 1).$$

If $k = o(n)$, then we also have $E[T] \geq (1 - o(1)) n \ln(\frac{n}{k})$.

Proof. To derive a situation with multiplicative drift, we regard a shifted version of the process (D_t). Let $X_t = D_t - k^*$ for all t. By (7.3), we have

$$\begin{aligned} E[X_t - X_{t+1} \mid X_t] &= E[D_t - D_{t+1} \mid D_t] \\ &= D_t \left(\frac{1}{n} + \frac{1}{\alpha n}\right) - \frac{1}{\alpha} \\ &= \left(X_t + \frac{n}{\alpha+1}\right)\left(\frac{1}{n} + \frac{1}{\alpha n}\right) - \frac{1}{\alpha} \\ &= X_t \left(\frac{1}{n} + \frac{1}{\alpha n}\right) =: X_t \delta, \end{aligned}$$

that is, we have an expected multiplicative progress towards 0 in the regime $X_t \geq 0$ (which is the regime $D_t \geq k^*$).

To apply the multiplicative drift theorem, we require a process in the non-negative numbers, having zero as target, and such that the smallest positive value is bounded away from zero. For this reason, we define (Y_t) by $Y_t = 0$ if $X_t < k + 1 - k^* =: y_{\min}$ and $Y_t = X_t$ otherwise (in other words, for $D_t \geq k + 1$,

the processes (X_t) and (Y_t) agree, and we have $Y_0 = 0$ otherwise). Since (X_t) changes by at most one per step and since $y_{\min} \geq 1$, in an iteration t such that $X_t = Y_t \geq y_{\min}$ we have $Y_{t+1} \leq X_{t+1}$ and thus $E[Y_t - Y_{t+1} \mid Y_t] \geq E[X_t - X_{t+1} \mid X_t] = X_t\delta = Y_t\delta$, that is, we have the same multiplicative progress. We can thus apply the multiplicative drift theorem from [DJW12] (also found as Theorem 11 in the survey [Len20]) and derive that the first time T such that $Y_T = 0$ satisfies $E[T] \leq \frac{1+\ln(n/y_{\min})}{\delta} \leq \frac{\alpha}{\alpha+1} n(\ln(n)+1)$ as claimed.

For the lower bound, we argue as follows, very similar to the proof of [JW07, Proposition 5]. Let D_0 be the fitness distance of the initial random search point. We condition momentarily on a fixed outcome of D_0 that is larger than k. Consider in parallel a run of the randomized local search heuristic RLS [DDY20] on OneMax, starting with a fitness distance of D_0. Note that this is equivalent to saying that we start a second run of the Metropolis algorithm with parameter $\alpha = \infty$. Denote the fitness distances of this run by $\tilde{D}_t$. This is again a Markov chain with one-step changes in $\{-1, 0, 1\}$, however, with transition probabilities $\tilde{p}_i^- = p_i^-$ and $\tilde{p}_i^+ = 0 \leq p_i^+$. Consequently, a simple induction shows that D_t stochastically dominates $\tilde{D}_t$. In particular, the first hitting time $\tilde{T}$ of k of this chain is a lower bound for T, both in the stochastic domination sense and in expectation. We therefore analyze $\tilde{T}$. Since $\tilde{D}_t$ in each step either decreases by one or remains unchanged, we can simply sum up the waiting times for making a step towards the target, that is, $E[\tilde{T}] = \sum_{i=D_0}^{k-1} \frac{1}{p_i^-} = \sum_{i=D_0}^{k-1} \frac{n}{i} = n(H_{D_0} - H_{k-1})$, where $H_m := \sum_{i=1}^{m} \frac{1}{i}$ denotes the m-th Harmonic number. Using the well-known estimate $\ln(m) \leq H_m \leq \ln(m) + 1$, we obtain $E[\tilde{T}] \geq n(\ln(D_0) - \ln(k) - 1)$. Recall that this estimate was conditional on a fixed value of D_0. Since D_0 follows a binomial distribution with parameters n and $\frac{1}{2}$, we have $D_0 \geq \frac{n}{2} - n^{3/4}$ with probability $1 - o(1)$, and in this case, $E[T \mid D_0 \geq \frac{n}{2} - n^{3/4}] \geq n(\ln(\frac{n}{2} - n^{3/4} - 1) - \ln(k) - 1) = (1 - o(1))n\ln(\frac{n}{k})$, where the last estimate exploits our assumption $k = o(n)$. Just from the contribution of this case, we obtain $E[T] \geq (1 - o(1))E[T \mid D_0 \geq \frac{n}{2} - n^{3/4}] = (1 - o(1))n\ln(\frac{n}{k})$. □

We note that there is a non-vanishing gap between our upper and lower bound in the theorem above when $k = n^{\Omega(1)}$. The reason, most likely, is the argument used in the lower bound proof that the Metropolis algorithm cannot be faster than randomized local search, which ignores any negative effect of accepting inferior solutions. For our purposes, the theorem above is sufficient, since for all but very large values of α (where the gap is negligible) the runtime of the Metropolis algorithm is dominated by the second part of the optimization process starting from a solution x with $d(x) = k$. The reason why we could not prove a tighter bound for all values of α is that the existing multiplicative drift theorems for lower bounds, e.g., Theorem 2.2 in [Wit13] or Theorem 3.7 in [DKLL20], either are not applicable to our process or necessarily lead to a constant-factor gap to

the upper bound obtained from multiplicative drift. Applying the variable drift theorem from [DFW11] to the process $Z_t = \min\{Y_s \mid s \leq t\}$ appears to be a promising way to overcome these difficulties, but since we do not need such a precise bound, we do not follow this route any further.

7.4.3 Progress Starting the Equilibrium Point

In the regime with negative drift, we use elementary Markov chain arguments to estimate runtimes. We profit here from the fact that the optimization time when starting in an arbitrary solution in the negative drift regime is very close to the optimization time when starting in a solution that is a Hamming neighbor of the optimum. This runtime behavior, counter-intuitive at first sight, is caused by the fact that the apparent advantage of starting with a Hamming neighbor is diminished by that fact that (at least for α not too large) it is much easier to generate and accept an inferior solution than to flip the unique missing bit towards the optimum. We make this precise in the following theorem. Since it does not take additional effort, we formulate and prove this result for a range of starting points k that extends also in the regime of positive drift. In this section, we shall use it only for $\ell = k = \lceil k^* \rceil$.

Theorem 7.2. *For all* $1 \leq \ell \leq \frac{2.5}{1+2.5/\alpha}\frac{n}{\alpha}$, *we have*

$$E_1 \leq E_\ell^0 \leq (1 + O(\tfrac{\alpha}{n}))E_1.$$

Proof. By equation (7.2) and the values for p_i^+, p_i^- computed earlier, we see that

$$E_i = \frac{n}{i} + \frac{n-i}{\alpha i} E_{i+1}, \tag{7.4}$$

for all $i \in [1..n-1]$. By omitting the first summand, we obtain $E_{i+1} \leq \frac{\alpha i}{n-i} E_i$, and an elementary induction yields $E_{j+1} \leq \alpha^j \frac{1 \cdot 2 \dots j}{(n-j)\dots(n-1)} E_1$ for all $j \in [1..n-1]$. Using the estimate $j! \leq 3\sqrt{j}(\frac{j}{e})^j$ stemming from a sharp version of Stirling's formula due to Robbins [Rob95] (also stated as Theorem 1.4.10 in [Doe20b]), we estimate, for $j \in [1..\ell]$,

$$\begin{aligned}
\alpha^j \frac{1 \cdot 2 \dots j}{(n-j)\dots(n-1)} &\leq 3\frac{\alpha}{n-1} j^{3/2} \left(\frac{\alpha(j-1)}{e(n-j)}\right)^{j-1} \\
&\leq 3\frac{\alpha}{n-1} j^{3/2} \left(\frac{\alpha\ell}{e(n-\ell)}\right)^{j-1} \\
&\leq 3\frac{\alpha}{n-1} j^{3/2} \left(\tfrac{2.5}{e}\right)^{j-1},
\end{aligned}$$

where we note that our assumption $\ell \leq \frac{2.5}{1+2.5/\alpha}\frac{n}{\alpha}$ is equivalent to $\frac{\alpha\ell}{e(n-\ell)} \leq \frac{2.5}{e}$.

By (7.1), we have

$$\begin{aligned} E_\ell^0 &= E_1 + \sum_{j=1}^{\ell-1} E_{j+1} \\ &\leq E_1 + \sum_{j=1}^{\ell-1} \alpha^j \frac{1 \cdot 2 \dots j}{(n-j)\dots(n-1)} E_1 \\ &\leq E_1 + \sum_{j=1}^{\ell-1} 3\frac{\alpha}{n-1} j^{3/2} \left(\tfrac{2.5}{e}\right)^{j-1} E_1 \\ &= (1 + O(\tfrac{\alpha}{n}))E_1, \end{aligned}$$

where we exploited that the series $\sum_{j=1}^{\infty} j^A B^j$ converges for all constants $A \in \mathbb{R}$ and $b \in (0,1)$. Note that the first line in this set of equations also shows our elementary lower bound $E_\ell^0 \geq E_1$. □

From Theorems 7.1 and 7.2, both applied with $k = \lceil k^* \rceil$, we know the runtime of the Metropolis algorithm on OneMax except that we do not yet understand E_1. This is what we do now.

7.4.4 Estimating E_1

To estimate E_1, we use again elementary Markov arguments, but this time to derive an expression for E_1 in terms of E_ℓ for some ℓ sufficiently far in the regime with positive drift (Theorem 7.3). Being in the positive drift regime, E_ℓ then can be easily bounded via drift arguments, which gives the final estimate for E_1 (Corollary 7.4).

Theorem 7.3. *Let $\alpha \geq 1$ and $\ell = o(\sqrt{n})$. Let*

$$E_1^+ = n\left(\sum_{i=0}^{\ell-1} \left(\frac{n}{\alpha}\right)^i \frac{1}{(i+1)!}\right) + \left(\frac{n}{\alpha}\right)^\ell \frac{1}{\ell!} E_{\ell+1}.$$

Then $(1-o(1))E_1^+ \leq E_1 \leq E_1^+$.

Proof. We first show that for all $\ell \in [0..n-1]$, we have

$$E_1 = \sum_{i=0}^{\ell-1} \frac{n \cdot (n-1)\dots(n-i)}{\alpha^i (i+1)!} + \frac{(n-1)\dots(n-\ell)}{\alpha^\ell \ell!} E_{\ell+1}. \tag{7.5}$$

This is trivially true for $\ell = 0$ (when using the convention that an empty sum evaluates to zero and an empty product evaluates to one). Assume now that equation (7.5) is true for some $\ell \in [0..n-2]$. Together with equation (7.4), we compute

$$\begin{aligned} E_1 &= \sum_{i=0}^{\ell-1} \frac{n \cdot (n-1) \dots (n-i)}{\alpha^i (i+1)!} + \frac{(n-1)\dots(n-\ell)}{\alpha^\ell \ell!} E_{\ell+1} \\ &= \sum_{i=0}^{\ell-1} \frac{n \cdot (n-1) \dots (n-i)}{\alpha^i (i+1)!} + \frac{(n-1)\dots(n-\ell)}{\alpha^\ell \ell!} \left(\frac{n}{\ell+1} + \frac{n-(\ell+1)}{\alpha(\ell+1)} E_{\ell+2} \right) \\ &= \sum_{i=0}^{\ell} \frac{n \cdot (n-1) \dots (n-i)}{\alpha^i (i+1)!} + \frac{(n-1)\dots(n-(\ell+1))}{\alpha^{\ell+1} (\ell+1)!} E_{\ell+2}, \end{aligned}$$

which shows the equation also for $\ell + 1$. By induction, the equation holds for all $\ell \in [0..n-1]$.

From equation (7.5), we immediately obtain $E_1 \leq E_1^+$. For $i = o(\sqrt{n})$, we estimate $(n-1)\dots(n-i) = n^i \prod_{j=1}^{i}(1 - \frac{i}{n}) \geq n^i(1 - \frac{1}{n}\sum_{j=1}^{i} j) = n^i(1 - \frac{i(i-1)}{2n}) = n^i(1-o(1))$, where the inequality uses an elementary generalization of Bernoulli's inequality sometimes called Weierstrass inequality (see, e.g., Lemma 1.4.8 in [Doe20b]). This shows the lower bound $E_1 \geq (1-o(1))E_1^+$. □

By estimating E_ℓ for ℓ in the positive drift regime, we obtain the following estimate for E_1, which is tight apart from lower order terms when $\alpha = o(n)$.

Corollary 7.4. *Let $\alpha = \omega(\sqrt{n})$. Then*

$$(1 - 2\exp(-\tfrac{2}{3}\tfrac{n}{\alpha}) - o(1))\alpha e^{n/\alpha} \leq E_1 \leq \alpha e^{n/\alpha}.$$

If $\alpha \geq 2n$, then $E_1 \leq 2n$.

Proof. Let $\ell = \lceil 3\frac{n}{\alpha} \rceil$. Since $\ell = o(\sqrt{n})$, we can use Theorem 7.3 with this ℓ to estimate E_1. Let X denote a random variable following a Poisson distribution with parameter $\lambda := \frac{n}{\alpha}$. Then

$$\begin{aligned} n\left(\sum_{i=0}^{\ell-1} \left(\frac{n}{\alpha}\right)^i \frac{1}{(i+1)!}\right) &= \alpha \left(\sum_{i=0}^{\ell-1} \frac{\lambda^{i+1}}{(i+1)!}\right) \\ &= \alpha e^\lambda \left(\sum_{i=0}^{\ell} \frac{\lambda^i e^{-\lambda}}{i!} - e^{-\lambda}\right) \\ &= \alpha e^\lambda \left(\Pr[X \leq \ell] - e^{-\lambda}\right). \end{aligned}$$

To estimate the second summand in Theorem 7.3, we first note that $(\frac{n}{\alpha})^\ell \frac{1}{\ell!} \leq (\frac{ne}{\ell\alpha})^\ell \leq (\frac{e}{3})^{3\lambda}$ follows from the estimate $\ell! \geq (\frac{\ell}{e})^\ell$. To bound $E_{\ell+1}$, we observe from equation (7.3) that the drift of the fitness distance D_t, whenever $D_t \geq \ell+1$, satisfies $E[D_t - D_{t+1}] = D_t(\frac{1}{n} + \frac{1}{\alpha n}) - \frac{1}{\alpha} \geq \frac{\ell+1}{n} - \frac{1}{\alpha} \geq \frac{2}{\alpha}$, using $\ell \geq 3\frac{n}{\alpha}$ in the last estimate. Consequently, we have an additive drift of at least $\frac{2}{\alpha}$ for $D_t \in [\ell+1..n]$, and thus the additive drift theorem of He and Yao [HY01] (also found as Theorem 2.3.1 in [Len20]) yields that the expected time it takes to reach a D_t value of ℓ when starting in $\ell+1$ is at most $E_{\ell+1} \leq \frac{\alpha}{2}$.

Putting these three estimates together, we obtain an upper bound of

$$\begin{aligned} E_1 &\leq n \left(\sum_{i=0}^{\ell-1} \left(\frac{n}{\alpha}\right)^i \frac{1}{(i+1)!} \right) + \left(\frac{n}{\alpha}\right)^\ell \frac{1}{\ell!} E_{\ell+1} \\ &\leq \alpha e^\lambda \left(\Pr[X \leq \ell] - e^{-\lambda} \right) + \left(\frac{e}{3}\right)^{3\lambda} \cdot \frac{\alpha}{2} \leq \alpha e^\lambda. \end{aligned}$$

For the lower bound, we note that a Poisson random variable Z with parameter λ satisfies the Chernoff-type bound $\Pr[Z \geq \lambda+\gamma] \leq \exp(-\frac{\gamma^2}{2(\lambda+\gamma)})$ for all $\gamma \geq 0$, see [BLM13, Section 2.2]. Consequently, with $\gamma = 2\lambda$, we obtain $\Pr[X \leq \ell] \geq 1 - \exp(-\frac{2}{3}\lambda)$. This gives a lower bound of

$$\begin{aligned} E_1 &\geq (1-o(1))n \left(\sum_{i=0}^{\ell-1} \left(\frac{n}{\alpha}\right)^i \frac{1}{(i+1)!} \right) \\ &\geq (1-o(1))\alpha e^\lambda \left(\Pr[X \leq \ell] - e^{-\lambda} \right) \\ &\geq (1 - \exp(-\tfrac{2}{3}\lambda) - \exp(-\lambda) - o(1))\alpha e^\lambda. \end{aligned}$$

For the case $\alpha \geq 2n$, we use again the additive drift argument. Whenever $D_t \geq 1$, we have $E[D_t - D_{t+1} \mid D_t] = D_t(\frac{1}{n} + \frac{1}{\alpha n}) - \frac{1}{\alpha} \geq \frac{1}{n} - \frac{1}{\alpha} \geq \frac{1}{2n}$. Hence the additive drift theorem bounds the expected time E_1 to reach $D_t = 0$ starting from $D_t = 1$ by $1/\frac{1}{2n} = 2n$. □

7.4.5 A Tight Estimate for the Total Optimization Time

From the partial results proven so far, we now obtain an estimate for the total runtime that is tight apart from lower order terms.

Theorem 7.5. *Let T be the runtime of the Metropolis algorithm with parameter α on the* OneMax *function defined on bit strings of length n. Let $\alpha = \omega(\sqrt{n})$. Then*

$$E[T] = (1 \pm o(1))n\ln(n) + \mathbb{1}_{\alpha \leq n}(1 \pm o(1))\alpha e^{n/\alpha}.$$

Proof. Let $k = \lceil \frac{n}{\alpha+1} \rceil$. Let T_k be the first time that a solution x with $d(x) \leq k$ is found. By Theorem 7.1, we have $E[T_k] \leq (1+o(1))n\ln(n)$. Since $\alpha = \omega(1)$ and thus $k = o(n)$, Theorem 7.1 also gives the lower bound $E[T_k] \geq (1-o(1))n\ln(\frac{n}{k})$.

When $\alpha \geq n-1$, that is, $k = 1$, then by Corollary 7.4 the remaining expected runtime is $E_1 = O(n)$. Together with our estimates on T_k, this shows the claim $E[T] = (1 \pm o(1))n\ln(n)$ for this case.

Hence let $\alpha < n-1$ and thus $k \geq 2$. Since $\alpha = \omega(\sqrt{n})$ and we aim at an asymptotic result, we can assume that n, and thus α, are sufficiently large. Then $k \leq 2\frac{n}{\alpha+1} \leq \frac{2.5}{1+2.5/\alpha}\frac{n}{\alpha}$, that is, k satisfies the assumptions of Theorem 7.2. By this theorem, the expectation of the remaining runtime satisfies $E_k^0 = (1 + O(\frac{\alpha}{n}))E_1$. By Corollary 7.4, $E_1 \leq \alpha e^{n/\alpha}$. This shows an upper bound of $E[T] \leq (1+o(1))n\ln(n) + (1+O(\frac{\alpha}{n}))\alpha e^{n/\alpha}$. For $\alpha \geq \frac{n}{\ln\ln n}$, this is the claimed upper bound $(1+o(1))n\ln(n)$, for $\alpha < \frac{n}{\ln\ln n}$, this is the claimed upper bound $(1 \pm o(1))n\ln(n) + (1 \pm o(1))\alpha e^{n/\alpha}$.

If remains to show the lower bound for $\alpha < n-1$. If $\alpha \geq \frac{n}{\ln\ln n}$ and thus $k = O(\log\log n)$, the lower bound $E[T_k] \geq (1-o(1))n\ln(\frac{n}{k}) = (1-o(1))n\ln(n)$ suffices. For $\alpha < \frac{n}{\ln\ln n}$, we estimate $E[T] \geq E[T_k] + E_k^0 \geq E[T_k] + E_1 = (1-o(1))n\ln(\frac{n}{k}) + (1 - 2\exp(-\frac{2}{3}\frac{n}{\alpha}) - o(1))\alpha e^{n/\alpha} = (1-o(1))n\ln(\frac{n}{k}) + (1 - o(1))\alpha e^{n/\alpha}$, again using Theorem 7.2 and Corollary 7.4. For $\alpha \geq \frac{n}{\ln(n)}$, we have $\ln(\frac{n}{k}) = (1-o(1))\ln(n)$ and thus $E[T] \geq (1-o(1))n\ln(n) + (1-o(1))\alpha e^{n/\alpha}$. For $\alpha \leq \frac{n}{\ln(n)}$, we have $n\ln(n) = o(\alpha e^{n/\alpha})$, hence our claimed lower bound is $E[T] \geq (1-o(1))\alpha e^{n/\alpha}$, which follows trivially from the estimate $E[T] \geq (1-o(1))n\ln(\frac{n}{k}) + (1-o(1))\alpha e^{n/\alpha}$ just shown. □

7.5 Analysis of Cliff

We now examine how the Metropolis algorithm behaves when optimizing a function with a local optimum. We recall some definitions used in Subsection 7.4.1. Let L_i be the set of search points with i zero-bits, i. e., $L_i := \{x \in \{0,1\}^n \mid n - \|x\|_1 = i\}$. For $j < i$, let E_i^j be the expected number of iterations the Metropolis algorithm spends to find a solution in L_j when started with a solution in L_i. We write $E_i = E_i^{i-1}$. By the *distance* $d(x)$ of a search point x we understand the Hamming distance to the global optimum 1^n, i. e., $d(x) = n - \|x\|_1$.

Hereinafter, by writing Cliff, we mean the function $\text{Cliff}_{d,m}$ defined in the beginning of the section for parameters d, m clear from the context. There are two

slopes in CLIFF on which the algorithm has the same behavior as on ONEMAX. More precisely, for $i \notin \{m-1, m\}$, the expected time E_i the Metropolis algorithm takes to find a solution with distance $i-1$ when started with a solution with distance i follows Equation (7.4), that is, we have

$$E_i = \frac{n}{i} + \frac{n-i}{\alpha i} E_{i+1}. \tag{7.6}$$

However, in the local optimum, i. e., solutions with $n-m$ one-bits, increasing the number of ones does not increase the fitness. In this case, p_m^-, denoting the probability of accepting a search point with distance $m-1$ (i. e., a search point with $n-m+1$ one-bits), equals $\alpha^{-d} m/n$, and p_m^+, denoting the probability of accepting a search point with distance $m+1$, equals $\alpha^{-1}(n-m)/n$. Using Equation (7.2), we obtain

$$E_m = \alpha^d \frac{n}{m} + \alpha^{d-1} \cdot \frac{n-m}{m} E_{m+1}. \tag{7.7}$$

Finally, for the search points with distance $m-1$, we have $p_{m-1}^- = \frac{m-1}{n}$ and $p_{m-1}^+ = \frac{n-m+1}{n}$, resulting in

$$E_{m-1} = \frac{n}{m-1} + \frac{n-m+1}{m-1} E_m. \tag{7.8}$$

To ease our analysis of the optimization time T, we shall assume that $m = o(n)$. Then a simple Chernoff bound argument shows that the initial search point is at a distance greater than m from the global optimum with high probability. Thus, we have

$$(1-o(1))\left(E_m + E_{m-1} + E_{m-2}^0\right) \leq \mathrm{E}(T) \leq E_n^m + E_m + E_{m-1} + E_{m-2}^0.$$

Intuitively, the term E_{m-1} is one of the most influential terms on the total optimization time. The reason is that with the search points in this state, the algorithm goes back to the local optimum with $n-m$ one-bits $(1+o(1))n/m = \omega(1)$ times in expectation, where it has to again try many steps to accept a worse solution with fitness difference d. Besides aforementioned event, the term E_1, which is hidden in E_{m-2}^0, also plays an important role in the optimization time as their corresponding search points have the least drift value or basically, the most negative drift value among all search points except the ones near the drop for some α.

We will even observe that E_1 might impact on the total optimization time more significantly than E_{m-1} if all search points within the distance m have the

negative drift, that is, the drop is at a distance less than the equilibrium $k^* := \frac{n}{\alpha+1}$. In this case, if the algorithm is only one improvement away from the global optimum, it might get back to the local optimum with high probability. That is why we are not really interested in E_{m-1}, which is basically captured by E_1 in this case. Contrariwise, when the equilibrium point $k^* := \frac{n}{\alpha+1}$ is on the second slope, it becomes essential to consider and analyze the role of E_{m-1} in the total optimization time, and this term cannot be ignored in the total optimization time without additional assumptions.

The following theorem is our main result in this section. This theorem analyzes the optimization time of Metropolis algorithm on $\textsc{Cliff}_{d,m}$ in two parts: where the search points with distance $m+1$ are in the regime with positive drift (Part 1) or negative drift (Part 2).

In Part 1, if the equilibrium point is far from the drop (i. e., $m-2 > \beta \approx 2.5k^*$), we have to use some additional arguments (Lemma 7.8) as Theorem 7.2 is only valid for $\ell \leq \beta$. That is why there are two cases for the upper bound. We discuss this issue more comprehensively in the Subsection 7.5.1.

Theorem 7.6. *Let $k^* := \frac{n}{\alpha+1}$ and $\beta := \frac{2.5}{1+2.5/\alpha}(n/\alpha)$. Let T denote the first time Metropolis with $\alpha = \omega(\sqrt{n})$ on $\textsc{Cliff}_{d,m}$ with $m = o(\sqrt{n})$ and $d \geq 1$ finds the optimum point 1^n.*

1. *If $k^* < m+1$, then*

$$\mathrm{E}(T) \leq \begin{cases} \left((1+O(\frac{\alpha}{n}))\frac{\left(\frac{n}{\alpha}\right)^{m-2}}{(m-2)!} + 1 + o(\alpha/n)\right) E_{m-1} & \text{if } m-2 \leq \beta, \\ \left((1+O(\frac{\alpha}{n}))\frac{\left(\frac{n}{\alpha}\right)^{m-2}}{(m-2)!} + 5/3 + o(\alpha/n)\right) E_{m-1} & \text{if } m-2 > \beta, \end{cases}$$

   and

$$\mathrm{E}(T) \geq (1-o(1))\left(\frac{\left(\frac{n}{\alpha}\right)^{m-2}}{(m-2)!} + 1\right) E_{m-1},$$

   where

$$(1-o(1))\frac{n^2\alpha^{d-1}}{m(m-1)}\left(\alpha + \frac{n}{m+1}\right) \leq E_{m-1}$$
$$\leq (1+o(1))\frac{n^2\alpha^{d-1}}{m(m-1)}\left(\alpha + \frac{n}{(m+1)\frac{\alpha+1}{\alpha} - n/\alpha}\right).$$

2. *If $m+1 \leq k^*$, then*

$$\left(\frac{1}{\sqrt{2\pi}e^{\alpha/(12n)}} - o(1)\right)\frac{\alpha^{d+2}e^{n/\alpha}}{\sqrt{n/\alpha}} \leq \mathrm{E}(T) \leq (1-o(1))\alpha^{d+2}e^{n/\alpha}.$$

In Subsection 7.5.1, we discuss the optimization time for the part 1 while the part 2 is investigated in Subsection 7.5.2.

7.5.1 Progress When the Cliff is Below the Equilibrium Point

In this subsection, we investigate the case that the drift at the search points with distance $m+1$ is positive, that is, we prove part 1 of Theorem 7.6.

In this case, the algorithm easily climbs up to the local optimum, that is, a search point with $n-m$ one-bits, in $\Theta(n \log n)$ steps as shown in Subsection 7.4.2. Leaving the local optimum to a search point closer to the global optimum is difficult, that is, the time E_m is large, but since the algorithm from a search point in distance $m-1$ often moves back to the local optimum, we have this fact $E_m = o(E_{m-1})$.

Regarding E_{m-1}, using recurrence relations obtained at the beginning of the section, we obtain a relation between E_{m-1} and E_{m+1}. Also, since the drift at the search points with distance $m+1$ is positive, we have a closed form for E_{m+1} via the drift theorem, resulting in the following closed form for E_{m-1}.

Lemma 7.7. *Let $k^* = \frac{n}{\alpha+1}$. For $k^* < m+1 = o(n)$, we have*

$$\begin{aligned}&(1-o(1))\frac{n^2\alpha^{d-1}}{m(m-1)}\left(\alpha+\frac{n}{m+1}\right) \leq E_{m-1}\\ &\leq (1+o(1))\frac{n^2\alpha^{d-1}}{m(m-1)}\left(\alpha+\frac{n}{(m+1)\frac{\alpha+1}{\alpha}-n/\alpha}\right).\end{aligned}$$

Proof. Note that we have $\alpha > \frac{n}{m+1} - 1 = \omega(1)$ from our assumptions.

By Equation (7.7) and (7.8), we have

$$\begin{aligned}E_{m-1} &= \frac{n}{m-1} + \frac{n-m+1}{m-1}E_m\\ &= \frac{n}{m-1} + \frac{n-m+1}{m-1}\left(\alpha^{d-1}\left(\frac{\alpha n}{m} + \frac{n-m}{m}E_{m+1}\right)\right).\end{aligned}$$

For the lower bound, since $m = o(n)$, we have

$$E_{m-1} \geq \frac{n-m+1}{m-1}\left(\alpha^{d-1}\left(\frac{\alpha n}{m} + \frac{n-m}{m}E_{m+1}\right)\right)$$

$$= (1 - o(1))\frac{n}{m-1}\left(\alpha^{d-1}\left(\frac{\alpha n}{m} + (1 - o(1))\frac{n}{m}E_{m+1}\right)\right)$$
$$= (1 - o(1))\frac{n^2}{m(m-1)}\left(\alpha^{d-1}\left(\alpha + E_{m+1}\right)\right), \tag{7.9}$$

and for the upper bound,

$$E_{m-1} \leq \frac{n}{m-1} + \frac{n}{m-1}\left(\alpha^{d-1}\left(\frac{\alpha n}{m} + \frac{n}{m}E_{m+1}\right)\right)$$
$$= \frac{n}{m-1} + \frac{n^2}{m(m-1)}\left(\alpha^{d-1}\left(\alpha + E_{m+1}\right)\right)$$
$$= (1 + o(1))\frac{n^2}{m(m-1)}\left(\alpha^{d-1}\left(\alpha + E_{m+1}\right)\right). \tag{7.10}$$

It remains to estimate E_{m+1}. Using Equation (7.3), the drift at distance $m+1$ is positive and equals $\Delta := \frac{(m+1)(\alpha+1)-n}{\alpha n}$. Since the drift at larger distances is at least Δ, by the additive drift theorem of He and Yao [HY01] (also found as Theorem 2.3.1 in [Len20]), the expected time to reach the distance m starting from the distance $m+1$ is at most $1/\Delta = \frac{n}{(m+1)\frac{\alpha+1}{\alpha} - n/\alpha}$. For the lower bound on E_{m+1}, a necessary condition to reach the distance m from the distance $m+1$ is that one of $m+1$ zero-bits flips, happening with probability $(m+1)/n$. This upper bound on reaching the distance m holds in every step. Using the geometric distribution, we need at least $n/(m+1)$ steps in expectation. Altogether, we have

$$\frac{n}{m+1} \leq E_{m+1} \leq \frac{n}{(m+1)\frac{\alpha+1}{\alpha} - n/\alpha}. \tag{7.11}$$

Replacing E_{m+1} in Equation (7.9) and (7.10) with Equation (7.11), we get the following bounds.

$$(1 - o(1))\frac{n^2\alpha^{d-1}}{m(m-1)}\left(\alpha + \frac{n}{m+1}\right) \leq E_{m-1}$$
$$\leq (1 + o(1))\frac{n^2\alpha^{d-1}}{m(m-1)}\left(\alpha + \frac{n}{(m+1)\frac{\alpha+1}{\alpha} - n/\alpha}\right). \qquad \square$$

We note that the above estimate for E_{m+1} would be exactly the same for the optimization process on OneMax since the time to go from distance $m+1$ to m is not affected by the cliff. The reason why we could not use results from Section 7.4 here is that there we did not analyze the E_i separately, but used a simpler argument to analyze the sum of the first E_i.

Now, we discuss how we estimate E^0_{m-2}, i. e., the expected time to reach the global optimum from a search point located in the second position after the drop in the valley. Since in the valley, we have the same recurrence relation between E_i and E_{i+1} as for OneMax, we can use similar arguments as in Theorem 7.2. If we have $m-2 \leq \beta \coloneqq \frac{2.5}{1+2.5/\alpha}\frac{n}{\alpha}$, the term E^0_{m-2} asymptotically equals E_1. Otherwise, if $m-2 > \beta$, we only have the estimation $E^0_\beta = (1 + o(\alpha/n))E_1$, so we also need to analyze E^β_{m-2}.

In the following lemma, we prove that the expected time to reach the distance β starting from $m-2$, i. e., E^β_{m-2}, is at most by a constant factor larger than E_{m-1}.

Lemma 7.8. *Let $\beta \coloneqq \frac{2.5}{1+2.5/\alpha}(n/\alpha)$. For $\beta < m-2 = o(\sqrt{n})$, we have*

$$E^\beta_{m-2} \leq (2/3 + o(1))E_{m-1}.$$

For the proof of the previous lemma, we need a classical inequality from probability theory called Wald's inequality.

Lemma 7.9 (Wald's inequality from [DK15])**.** *Let T be a random variable with a finite expectation, and let X_1, X_2, ... be non negative random variables with $\mathrm{E}(X_i \mid T \geq i) \leq C$. Then*

$$\mathrm{E}\left(\sum_{i=1}^{T} X_i\right) \leq \mathrm{E}(T) \cdot C.$$

Proof of Lemma 7.8. For $\beta \leq i \leq m-1$, let S_i be the event of that the algorithm starts from a search point with distance i and reaches a search point with distance β before it reaches distance $m-1$. Reusing the notation $x^{(t)}$ from Subsection 7.4.2, we let $X_t = n - \|x^{(t)}\|_1$. If we define $U^b_a = \min\{t \mid X_t = b \text{ and } X_0 = a\}$, S_i is defined as the event that $U^\beta_i < U^{m-1}_i$.

In the first part of the proof, we aim at bounding $\Pr(S_{m-2})$ from below. According to the definition, we have $\Pr(S_{m-1}) = 0$, $\Pr(S_\beta) = 1$, and for $\beta + 1 \leq i \leq m-2$, using the law of total probability,

$$\Pr(S_i) = p^-_i \Pr(S_{i-1}) + p^+_i \Pr(S_{i+1}) + (1 - p^-_i - p^+_i)\Pr(S_i),$$

which can be rewritten as

$$\Pr(S_i) - \Pr(S_{i-1}) = \frac{p^+_i}{p^-_i}(\Pr(S_{i+1}) - \Pr(S_i)).$$

By denoting $w_i \coloneqq \prod_{k=i}^{m-2} \frac{p_k^+}{p_k^-}$ and carrying out a simple induction, for all $\beta+1 \leq i \leq m-1$, we have

$$\Pr(S_i) - \Pr(S_{i-1}) = w_i(\Pr(S_{m-1}) - \Pr(S_{m-2})).$$

Through a telescoping sum of the equations, we get

$$\Pr(S_{m-1}) - \Pr(S_\beta) = (\Pr(S_{m-1}) - \Pr(S_{m-2})) \sum_{i=\beta+1}^{m-1} w_i.$$

Hence, using the fact that $\Pr(S_{m-1}) = 0$, $\Pr(S_\beta) = 1$, we have

$$\Pr(S_{m-2}) = \frac{1}{\sum_{i=\beta+1}^{m-1} w_i}.$$

Furthermore, for $\beta + 1 \leq i \leq m-2$, since $\frac{p_i^+}{p_i^-} = \frac{n-i}{\alpha i} \leq \frac{n}{\alpha \cdot \beta}$, we obtain $w_i \leq \left(\frac{n}{\alpha\beta}\right)^{m-i-1}$. Since $n/(\alpha\beta) < 1$, using the geometric series sum formula, we get

$$\sum_{i=\beta+1}^{m-1} w_i \leq \sum_{i=0}^{\infty} \left(\frac{n}{\alpha\beta}\right)^k = \left(1 - \frac{n}{\alpha\beta}\right)^{-1},$$

resulting in $\Pr(S_{m-2}) \geq \left(1 - \frac{n}{\alpha\beta}\right) = \frac{3+5/\alpha}{5} \geq 3/5$ through the geometric distribution.

Now, in the second part of the proof, we estimate the time τ for the process starting from a search point with distance $m-2$ to reach a search point with distance either $m-1$ or β, that is,

$$\tau \coloneqq \min\left\{U_{m-2}^{m-1}, U_{m-2}^{\beta}\right\}.$$

To compute an upper bound on $E[\tau]$, we introduce a stopped process, $y^{(t)}$ defined as follows. We let $y^{(0)} = x^{(0)}$, and for $t \geq 1$,

$$y^{(t)} = \begin{cases} x^{(t)} & \text{if } X_t \neq m-1, \\ y^{(t-1)} & \text{otherwise,} \end{cases}$$

and let $Y_t \coloneqq n - \|y^{(t)}\|_1$, and $\tau' \coloneqq U_{m-2}^{\beta}$. The process $y^{(t)}$ follows the same movements as $x^{(t)}$, except if $x^{(t)}$ gets to a point with distance $m-1$, where it is stopped. With our definition, we see immediately that $\forall t \geq 0, Y_t \leq m-2$, and that $\tau \leq \tau'$.

To compute $E[\tau']$, we compute the drift associated to Y_t, and we notice that this process has the same drift as the process associated to the optimization of

OneMax by the MA ($y^{(t)}$ is not affected by the cliff because it can never reach it). We then use Theorem 7.1 to deduce that

$$E[\tau] \leq E[\tau'] \leq \frac{\alpha}{\alpha+1} n(\ln n + 1),$$

resulting in $E[\tau] = o(E_{m-1})$ through Lemma 7.7 with $m = o(\sqrt{n})$.

Now, in the final part of the proof, we estimate E^{β}_{m-2} by using $\Pr(S_{m-2})$ and $\mathrm{E}(\tau)$ which were bounded in the previous paragraphs. Let I_t be the random variable denoting the number of iterations starting from a search point with distance $m-2$ to reach a search point with distance $m-1$, and thereafter again to reach a search point with distance $m-2$. Then we have

$$E^{\beta}_{m-2} \leq \mathrm{E}\left(\sum_{t=1}^{\ell} I_t\right) + \mathrm{E}(\tau \mid S_{m-2}), \tag{7.12}$$

where ℓ is the number of times the algorithm reaches a search point with distance $m-1$ before β starting with a search point with distance $m-2$. Since the assumptions of the Wald's inequality in Lemma 7.9 are satisfied, the first term in the right-hand side of the inequality equals $\mathrm{E}(\ell)\, C$, where based on the definition, we have $\mathrm{E}(I_i \mid i \leq \ell) = \mathrm{E}(E_{m-1}) + \mathrm{E}\left(\tau \mid \overline{S_{m-2}}\right) =: C$. Also, using the geometric distribution, we have $\mathrm{E}(\ell) + 1 = \Pr(S_{m-2})^{-1}$. Altogether, the right-hand side of Inequality (7.12) is bounded from above by

$$\begin{aligned}
&\left(\Pr(S_{m-2})^{-1} - 1\right)\left(E_{m-1} + \mathrm{E}\left(\tau \mid \overline{S_{m-2}}\right)\right) + \mathrm{E}(\tau \mid S_{m-2}) \\
&\quad = \left(\Pr(S_{m-2})^{-1} - 1\right) E_{m-1} + \left(\frac{1 - \Pr(S_{m-2})}{\Pr(S_{m-2})}\right) \mathrm{E}\left(\tau \mid \overline{S_{m-2}}\right) + \mathrm{E}(\tau \mid S_{m-2}) \\
&\quad = \left(\Pr(S_{m-2})^{-1} - 1\right) E_{m-1} \\
&+ \left(\frac{\Pr\left(\overline{S_{m-2}}\right) \mathrm{E}\left(\tau \mid \overline{S_{m-2}}\right) + \Pr(S_{m-2})\, \mathrm{E}(\tau \mid S_{m-2})}{\Pr(S_{m-2})}\right) \\
&\quad = \left(\Pr(S_{m-2})^{-1} - 1\right) E_{m-1} + \left(\frac{\mathrm{E}(\tau)}{\Pr(S_{m-2})}\right).
\end{aligned}$$

Using the bounds obtained on $\Pr(S_{m-2})$ and $\mathrm{E}(\tau)$, we can finally conclude

$$E^{\beta}_{m-2} \leq (2/3 + o(1))\, E_{m-1}. \qquad \square$$

Finally, by working out the informal arguments for the overall proof idea given at the beginning of this subsection, we rigorously prove the optimization time in Part 1 of Theorem 7.6 as follows.

Proof of Theorem 7.6 part 1. Let $z_0 := \|x^{(0)}\|_1$ be the number of one-bits in the initial random search point. Then, using Chernoff's bound, for $m = o(n)$, $\Pr(z_0 \leq n - m)$, i. e., the probability that the initial search point is at a distance at least m from the optimum, is exponentially close to 1, more precisely $1 - 2^{-\Omega(n)}$. Therefore, we have

$$\left(1-2^{-\Omega(n)}\right)\left(E^m_{z_0}+E_m+E_{m-1}+E^0_{m-2}\right) \leq \mathrm{E}(T) \leq E^m_{z_0}+E_m+E_{m-1}+E^0_{m-2}. \quad (7.13)$$

In the following paragraphs, we estimate each of the terms in the last expression and then finally bound $\mathrm{E}(T)$.

$E^m_{z_0}$: Since $m+1 > \frac{n}{\alpha+1}$ and $\alpha = \omega(\sqrt{n})$, the drift as defined is positive, and by Theorem 7.1,

$$(1-o(1))n\ln n \leq E^m_{z_0} \leq \frac{\alpha}{\alpha+1}n(\ln(n)+1).$$

Using Lemma 7.7 and the fact that $d \geq 1$, we have $E_{m-1} = \Omega(n^3/m^3)$, resulting in $E^m_{z_0} = o(E_{m-1})$ for $m = o(\sqrt{n})$.

E_m, E_{m-1}: By Equation (7.8), we have $E_m = o(E_{m-1})$, and using Lemma 7.7, we have E_{m-1} as defined in the statement of the lemma.

E^0_{m-2}: We consider the following two cases. If $m-2 \leq \beta$, we use Theorem 7.2 since for all $i \in [1..m-2]$, the equation $E_i = \frac{n}{i} + \frac{n-i}{\alpha i}E_{i+1}$ holds due to Equation (7.6), so all arguments in the proof of Theorem 7.2 are still valid. This results in $E^0_{m-2} = (1+O(\alpha/n))E_1$. Otherwise, if $m-2 > \beta$, we have $E^0_{m-2} = E^\beta_{m-2} + E^0_\beta$. By Lemma 7.8 we get $E^\beta_{m-2} \leq (2/3+o(1))E_{m-1}$ and by Theorem 7.2, we have $E^0_\beta = (1+O(\alpha/n))E_1$ for the same reason as in the previous case. Therefore, we have $E^0_{m-2} \geq E_1$ and

$$E^0_{m-2} \leq \begin{cases} (1+O(\alpha/n))E_1 & \text{if } m-2 \leq \beta, \\ (2/3+o(1))E_{m-1} + (1+O(\alpha/n))E_1 & \text{if } m-2 > \beta. \end{cases}$$

To compute E_1, since $m = o(\sqrt{n})$ and the equation $E_i = \frac{n}{i} + \frac{n-i}{\alpha i}E_{i+1}$ holds for $1 \leq i \leq m-2$, we can use Theorem 7.3. By this theorem for $\ell = m-2$, we get $(1-o(1))E^+_1 \leq E_1 \leq E^+_1$ such that

$$\begin{aligned} E^+_1 &= n\left(\sum_{i=0}^{m-3}\left(\frac{n}{\alpha}\right)^i\frac{1}{(i+1)!}\right) + \left(\frac{n}{\alpha}\right)^{m-2}\frac{1}{(m-2)!}E_{m-1} \\ &= \alpha\left(\sum_{i=0}^{m-3}\left(\frac{n}{\alpha}\right)^{i+1}\frac{1}{(i+1)!}\right) + 5\left(\frac{n}{\alpha}\right)^{m-2}\frac{1}{(m-2)!}E_{m-1}. \end{aligned} \quad (7.14)$$

We now compute the first summand in two cases according to α: $\alpha < n$ and $\alpha \geq n$. In the first case, which is $n/\alpha > 1$, let us denote $f(k) := \left(\frac{n}{\alpha}\right)^k \frac{1}{k!}$. Since $f(k)/f(k-1) = \frac{n/\alpha}{k}$, $f(k)$ is increasing for $k < n/\alpha$. Thus, since we have $m+1 > \frac{n}{\alpha+1}$, the summation in the last expression (Equation (7.14)) can be

bounded based on the largest term, which is the term of $i = m-3$. Therefore, we get

$$\begin{aligned} E_1^+ &\leq \alpha(m-3)\left(\frac{n}{\alpha}\right)^{m-2}\frac{1}{(m-2)!} + \left(\frac{n}{\alpha}\right)^{m-2}\frac{1}{(m-2)!}E_{m-1} \\ &\leq \left(\frac{n}{\alpha}\right)^{m-2}\frac{1}{(m-2)!}\left(\alpha m + E_{m-1}\right) \\ &= (1+o(1))\left(\frac{n}{\alpha}\right)^{m-2}\frac{1}{(m-2)!}E_{m-1}, \end{aligned}$$

where we have used $m = o(\sqrt{n})$ and $E_{m-1} = \Omega(\alpha n^2/m^2)$ using Lemma 7.7 with $d \geq 1$.

For $n/\alpha \leq 1$, the first summand in Equation (7.14) is $O(\alpha)$, so it is again asymptotically dominated by $E_{m-1} = \Omega(\alpha n^2/m^2)$ using Lemma 7.7 with $d \geq 1$. Thus, for $n/\alpha \leq 1$, we have

$$E_1^+ = o(E_{m-1}) + \left(\frac{n}{\alpha}\right)^{m-2}\frac{1}{(m-2)!}E_{m-1}.$$

For both cases $n/\alpha > 1$ and $n/\alpha \leq 1$, we can conclude the upper bound

$$E_1^+ \leq o(E_{m-1}) + (1+o(1))\left(\frac{n}{\alpha}\right)^{m-2}\frac{1}{(m-2)!}E_{m-1}.$$

Altogether, by Equation (7.13), we can conclude $\mathrm{E}(T) \geq (1-o(1))E_{m-1}$, and

$$\mathrm{E}(T) \leq \begin{cases} E_{z_0}^m + (1+o(1))E_{m-1} + (1+O(\alpha/n))E_1^+ & \text{if } m-2 \leq \beta, \\ E_{z_0}^m + (1+o(1))E_{m-1} + (2/3+o(1))E_{m-1} + (1+O(\frac{\alpha}{n}))E_1^+ & \text{if } m-2 > \beta, \end{cases}$$

which gives us

$$\mathrm{E}(T) \leq \begin{cases} \left((1+O(\frac{\alpha}{n}))\frac{\left(\frac{n}{\alpha}\right)^{m-2}}{(m-2)!} + 1 + o(\alpha/n)\right)E_{m-1} & \text{if } m-2 \leq \beta, \\ \left((1+O(\frac{\alpha}{n}))\frac{\left(\frac{n}{\alpha}\right)^{m-2}}{(m-2)!} + 5/3 + o(\alpha/n)\right)E_{m-1} & \text{if } m-2 > \beta, \end{cases}$$

and for the lower bound, we get

$$\mathrm{E}(T) \geq (1-o(1))\left(\left(\frac{n}{\alpha}\right)^{m-2}\frac{1}{(m-2)!} + 1\right)E_{m-1}. \quad \square$$

7.5.2 Progress When the Cliff is in Negative Drift Region

In this subsection, we analyze the optimization time of Metropolis on Cliff, when the drift at the search points with distance $m+1$ is not positive. In other words, the algorithm reaches local optima no sooner than a search point with negative drift.

In this case, we shall argue (in the proof of Theorem 7.6, Part 2) that

$$\left(1-2^{-n}\right)E_1 < \mathrm{E}\left(T\right) \leq (1+O(\alpha/n))E_1,$$

that is, the optimization time is well described by the time taken to find the optimum from one of its Hamming neighbors. Here the lower bound stems from the fact that the algorithm has to visit a Hamming neighbor before finding the optimum except when the random initial solution is already the optimum.

For the upper bound, by elementary properties of Markov processes, we have

$$\mathrm{E}\left(T\right) \leq E_n^{\lceil \frac{n}{\alpha+1} \rceil} + E^m_{\lceil \frac{n}{\alpha+1} \rceil} + E_m + E^0_{m-1}.$$

Using Equation (7.6), (7.7), and (7.8), we will show that E_i is asymptotically larger than E_{i-1} for the last three terms representing the search points in the negative drift region.

Therefore, the following lemma bounding E_1 plays an important role to estimate the optimization time. The $\alpha^{d+2}e^{n/\alpha}$ factor appearing in both bounds comes from the fact that the gap is reached and has to be overcome a repeated number of times due to the negative drift.

Lemma 7.10. *If $m \leq \frac{n}{\alpha+1} - 1$ and $\alpha = \omega(\sqrt{n})$, we have*

$$\left(\frac{1}{\sqrt{2\pi}e^{\alpha/(12n)}} - o(1)\right)\frac{\alpha^{d+2}e^{\lfloor n/\alpha \rfloor}}{\sqrt{\lfloor n/\alpha \rfloor}} \leq E_1 \leq \alpha^{d+2}e^{n/\alpha} + o(n).$$

Proof. Regarding the lower bound, using Equation (7.6), we have $E_i = \frac{n}{i} + \frac{n-i}{\alpha i}E_{i+1} \geq \frac{n}{i} + \frac{n}{c\alpha i}E_{i+1}$ for $i \in [1..m-2] \cup [m+1..\lfloor \frac{n}{\alpha} \rfloor]$, where $c = 1 + \lfloor \frac{n}{\alpha} \rfloor / (n - \lfloor \frac{n}{\alpha} \rfloor)$ because

$$\begin{aligned} E_i &= \frac{n}{i} + \frac{n-i}{\alpha i}E_{i+1} = \frac{n}{i} + \frac{n}{(1+i/(n-i))\alpha i}E_{i+1} \\ &\geq \frac{n}{i} + \frac{n}{(1+\lfloor \frac{n}{\alpha} \rfloor/(n-\lfloor \frac{n}{\alpha} \rfloor))\alpha i}E_{i+1} = \frac{n}{i} + \frac{n}{c\alpha i}E_{i+1}. \end{aligned}$$

Then, using the recursive formulas for $i \in [1..m-2]$, we achieve

$$E_1 \geq \sum_{i=0}^{m-3} \frac{n}{(i+1)!}\left(\frac{n}{c\alpha}\right)^i + \frac{1}{(m-2)!}\left(\frac{n}{c\alpha}\right)^{m-2} E_{m-1}.$$

In the drop region, i.e., $i = m$ and $i = m-1$, using Equation (7.7) and (7.8), we have

$$E_{m-1} \geq \frac{n}{m-1} + \frac{n}{c(m-1)}E_m,$$

$$E_m \geq \alpha^d \frac{n}{m} + \alpha^d \frac{n}{c\alpha m} E_{m+1},$$

which results in

$$E_1 \geq \sum_{i=0}^{m-2} \frac{n}{(i+1)!} \left(\frac{n}{c\alpha}\right)^i + \frac{\alpha}{(m-1)!} \left(\frac{n}{c\alpha}\right)^{m-1} E_m,$$

and furthermore

$$E_1 \geq \sum_{i=0}^{m-2} \frac{n}{(i+1)!} \left(\frac{n}{c\alpha}\right)^i + \frac{n \cdot \alpha^{d+1}}{m!} \left(\frac{n}{c\alpha}\right)^{m-1} + \frac{\alpha^{d+1}}{m!} \left(\frac{n}{c\alpha}\right)^m E_{m+1}.$$

By using Equation (7.6) for $m+1 \leq i \leq \lfloor \frac{n}{\alpha} \rfloor$, we achieve

$$E_1 \geq \sum_{i=0}^{m-2} \frac{n}{(i+1)!} \left(\frac{n}{c\alpha}\right)^i + \alpha^{d+1} \sum_{i=m-1}^{\lfloor \frac{n}{\alpha} \rfloor - 1} \frac{n}{(i+1)!} \left(\frac{n}{c\alpha}\right)^i + \frac{\alpha^{d+1}}{\lfloor \frac{n}{\alpha} \rfloor!} \left(\frac{n}{c\alpha}\right)^{\lfloor \frac{n}{\alpha} \rfloor} E_{\lfloor \frac{n}{\alpha} \rfloor + 1}.$$

The last expression is bounded from below by

$$\begin{aligned} \alpha^{d+1} \sum_{i=m-1}^{\lfloor \frac{n}{\alpha} \rfloor - 1} \frac{n}{(i+1)!} \left(\frac{n}{c\alpha}\right)^i &\geq c\alpha^{d+2} \sum_{i=m-1}^{\lfloor \frac{n}{\alpha} \rfloor - 1} \frac{1}{(i+1)!} \left(\frac{n}{c\alpha}\right)^{i+1} \\ &\geq c\alpha^{d+2} \frac{1}{\lfloor n/\alpha \rfloor!} \left(\frac{n}{c\alpha}\right)^{\lfloor n/\alpha \rfloor}. \end{aligned}$$

Since $\frac{n}{\alpha+1} - 1 > m$ and $m > 0$, we have $n/\alpha > 2$. Thus, using Stirling's formula (Theorem 1.4.10 in [Doe20b]), the last term is bounded from below by

$$c\alpha^{d+2} \frac{1}{\sqrt{2\pi \lfloor n/\alpha \rfloor} e^{\alpha/(12n)}} \left(\frac{en}{c\alpha \lfloor \frac{n}{\alpha} \rfloor}\right)^{\lfloor n/\alpha \rfloor} \geq \frac{\alpha^{d+2}}{c^{n/\alpha - 1} \cdot \sqrt{2\pi \lfloor n/\alpha \rfloor} e^{\alpha/(12n)}} e^{\lfloor n/\alpha \rfloor}.$$

Since $n/\alpha = o(\sqrt{n})$, we have

$$c^{n/\alpha - 1} = \left(1 + \frac{\lfloor \frac{n}{\alpha+1} \rfloor}{(n - \lfloor \frac{n}{\alpha+1} \rfloor)}\right)^{n/\alpha - 1} \leq \left(1 + \frac{n/\alpha}{n - n/\alpha}\right)^{n/\alpha} \leq e^{\frac{n}{\alpha(\alpha-1)}} = 1 + o(1),$$

where $\alpha = \omega(\sqrt{n})$. Then, we have

$$E_1 \geq \left(\frac{1}{\sqrt{2\pi} e^{\alpha/(12n)}} - o(1)\right) \frac{\alpha^{d+2} e^{\lfloor n/\alpha \rfloor}}{\sqrt{\lfloor n/\alpha \rfloor}}.$$

Regarding the upper bound, using Equation (7.6), for $i \in [1..m-2] \cup [m+1..\lceil 3n/\alpha \rceil]$, we have $E_i = \frac{n}{i} + \frac{n-i}{\alpha i} E_{i+1} \leq \frac{n}{i} + \frac{n}{\alpha i} E_{i+1}$.

Then, using the recursive formulas for $i \in [1..m-2]$, we achieve

$$E_1 \leq \sum_{i=0}^{m-3} \frac{n}{(i+1)!} \left(\frac{n}{\alpha}\right)^i + \frac{1}{(m-2)!} \left(\frac{n}{\alpha}\right)^{m-2} E_{m-1}.$$

In the drop region, i.e., $i = m$ and $i = m-1$, using Equation (7.7) and (7.8) , we can compute

$$E_{m-1} \leq \frac{n}{m-1} + \frac{n}{m-1} E_m,$$

and

$$E_m \leq \alpha^d \frac{n}{m} + \alpha^d \frac{n}{\alpha m} E_{m+1},$$

which results in

$$E_1 \leq \sum_{i=0}^{m-2} \frac{n}{(i+1)!} \left(\frac{n}{\alpha}\right)^i + \frac{\alpha}{(m-1)!} \left(\frac{n}{\alpha}\right)^{m-1} E_m,$$

and

$$E_1 \leq \sum_{i=0}^{m-2} \frac{n}{(i+1)!} \left(\frac{n}{\alpha}\right)^i + \frac{n \cdot \alpha^{d+1}}{m!} \left(\frac{n}{\alpha}\right)^{m-1} + \frac{\alpha^{d+1}}{m!} \left(\frac{n}{\alpha}\right)^m E_{m+1}.$$

For $i \in [m+1..\lceil 3n/\alpha \rceil]$, using Equation (7.6), we have

$$\begin{aligned} E_1 &\leq \sum_{i=0}^{m-2} \frac{n}{(i+1)!} \left(\frac{n}{\alpha}\right)^i + \alpha^{d+1} \sum_{i=m-1}^{\lceil 3n/\alpha \rceil - 1} \frac{n}{(i+1)!} \left(\frac{n}{\alpha}\right)^i \\ &\quad + \frac{\alpha^{d+1}}{(\lceil 3n/\alpha \rceil)!} \left(\frac{n}{\alpha}\right)^{\lceil 3n/\alpha \rceil} E_{\lceil 3n/\alpha \rceil + 1} \\ &\leq \alpha^{d+2} \sum_{i=0}^{\infty} \frac{1}{i!} \left(\frac{n}{\alpha}\right)^i + \alpha^{d+1} \left(\frac{e}{3}\right)^{\lceil 3n/\alpha \rceil} E_{\lceil 3n/\alpha \rceil + 1} \leq \alpha^{d+2} e^{n/\alpha} + o\left(\alpha^{d+1}\right) \\ &= (1 + o(1)) \alpha^{d+2} e^{n/\alpha}, \end{aligned}$$

where we have $E_{\lceil 3n/\alpha \rceil + 1} = O(n)$ since the search points in the distance $\lceil 3n/\alpha \rceil + 1$ have positive drift (see Equation (7.3)). □

Finally, by working out the informal arguments for the overall proof idea given at the beginning of this subsection, we rigorously prove the optimization time for the part 2 in Theorem 7.6 as follows.

Proof of Theorem 7.6 part 2. We first prove that

$$(1-o(1))\,(n\ln n+E_1)\le \mathrm{E}\,(T)\le \frac{\alpha}{\alpha+1}n(\ln n+1)+E_1(1+O(\alpha/n)).$$

Regarding the lower bound, using Theorem 7.1, the running time to reach a search point with a negative drift (at distance $\lceil\frac{n}{\alpha+1}\rceil\ge m+1$ where $m>1$) is at least $(1-o(1))n\ln n$ for $\alpha=\omega(\sqrt{n})$. Since the algorithm flips at most one bit each iteration, at a point of time, one search point at distance 1 is reached if the initial search point is not the global optimum with probability 2^{-n}. Therefore, for the lower bound, we have $\mathrm{E}\,(T)\ge(1-o(1))\,(n\ln n+E_1)$.

Regarding the upper bound, we have the following inequalities

$$\mathrm{E}\,(T)\le E_n^{\lceil\frac{n}{\alpha+1}\rceil}+E^m_{\lceil\frac{n}{\alpha+1}\rceil}+E_m+E^0_{m-1}.$$

In the following paragraphs, we aim at estimating the terms in the last expression.

E^0_{m-1}: Via Equation (7.6), we use the recurrence relation

$$E_{i+1}=\frac{\alpha i}{n-i}E_i-\frac{\alpha n}{n-i}\le\frac{\alpha i}{n-i}E_i. \tag{7.15}$$

For $i\le n/(\alpha+1)$, we have

$$\frac{\alpha i}{n-i}\le\alpha\cdot\frac{n/(\alpha+1)}{n-(n/(\alpha+1))}=\frac{\alpha}{\alpha+1}\cdot\frac{n}{n(1-1/(\alpha+1)}=1, \tag{7.16}$$

resulting in $E_{i+1}\le E_i$. For $m=2$, $E^0_{m-1}=E_1$ is immediately obtained from the definitions and for $m\ge3$, we have

$$\begin{aligned}E^0_{m-1}=\sum_{i=1}^{m-1}E_i&\le E_1+E_2+E_3(m-3)\\&\le E_1+O\,(\alpha/n)\,E_1+O\,(\alpha/n)\,E_2(m-3)\\&\le E_1+O\,(\alpha/n)\,E_1+O\left(\alpha^2/n^2\right)E_1(m-3),\end{aligned}$$

where we used Equation (7.15) for E_2 and E_3 and for $i\ge4$, $E_i\le E_3$. Since $m<\frac{n}{\alpha+1}-1=O(n/\alpha)$, the last expression is bounded from above by

$$E_1+O\,(\alpha/n)\,E_1+O\left(\alpha^2/n^2\right)E_1\cdot O(n/\alpha)=E_1(1+O(\alpha/n)).$$

Therefore, we have $E^0_{m-1}=E_1(1+O(\alpha/n))$.

E_m: Via Equation (7.8), we have

$$E_m = \frac{m-1}{n-m+1}E_{m-1} - \frac{n}{n-m+1} \leq \frac{m-1}{n-m+1}E_{m-1}.$$

Thus, $E_m = \Theta(m/n)E_{m-1}$. Since $E_{m-1} \leq E^0_{m-1} = E_1(1+O(\alpha/n))$, we get

$$E_m = \Theta(m/n)E_1(1+O(\alpha/n)). \tag{7.17}$$

$E^m_{\lceil \frac{n}{\alpha+1} \rceil}$: We have $E_{i+1} \leq E_i$ for $i \leq n/(\alpha+1)$ similarly to the paragraph corresponding to E^0_{m-1} (Equation (7.16)). We compute

$$E^m_{\lceil \frac{n}{\alpha+1} \rceil} = \sum_{i=m+1}^{\lceil \frac{n}{\alpha+1} \rceil} E_i \leq \lceil \frac{n}{\alpha+1} \rceil E_{m+1}.$$

Using Equation (7.7) and (7.17), for $m = o(n)$, we have

$$\lceil \frac{n}{\alpha+1} \rceil E_{m+1} = \lceil \frac{n}{\alpha+1} \rceil \Theta(m/n)E_m = \lceil \frac{n}{\alpha+1} \rceil \Theta(m^2/n^2)E_1(1+O(\alpha/n)).$$

Since $m < \lceil \frac{n}{\alpha+1} \rceil = o(\sqrt{n})$, we have $E^m_{\lceil \frac{n}{\alpha+1} \rceil} = o\left(E_1\right)(1+O(\alpha/n))$.

$E_n^{\lceil \frac{n}{\alpha+1} \rceil}$: Since Equation (7.6) denoting $E_i = \frac{n}{i} + \frac{n-i}{\alpha i}E_{i+1}$ for $i \in [\lceil \frac{n}{\alpha+1} \rceil..n]$ is the same as the corresponding recursive equation for OneMax, the drift equation is also the same as Equation (7.3), so $E_n^{\lceil \frac{n}{\alpha+1} \rceil} = \mathrm{E}\left(T'\right)$ can be estimated by Theorem 7.1, where T' is the first time that the algorithm finds a solution with distance $\lceil \frac{n}{\alpha+1} \rceil$, resulting in $E_n^{\lceil \frac{n}{\alpha+1} \rceil} \leq \frac{\alpha}{\alpha+1}n(\ln n+1)$.

Altogether, we have

$$\mathrm{E}\left(T\right) \leq E_n^{\lceil \frac{n}{\alpha+1} \rceil} + E^m_{\lceil \frac{n}{\alpha+1} \rceil} + E_m + E^0_{m-1} \leq \frac{\alpha}{\alpha+1}n(\ln n+1) + E_1(1+O(\alpha/n)),$$

and using Lemma 7.10, we obtain

$$\begin{aligned}&(1-o(1))n\ln n + \left(\frac{1}{\sqrt{2\pi}e^{\alpha/(12n)}} - o(1)\right)\frac{\alpha^{d+2}e^{n/\alpha}}{\sqrt{n/\alpha}} \leq \mathrm{E}\left(T\right)\\ &\leq \frac{\alpha}{\alpha+1}n(\ln n+1) + \alpha^{d+2}e^{n/\alpha} + o(n).\end{aligned}$$

Since $d \geq 1$ and $\alpha = \omega(\sqrt{n})$, $\alpha^{d+2} = \omega(n^{1.5})$, so we have

$$\left(\frac{1}{\sqrt{2\pi}e^{\alpha/(12n)}} - o(1)\right)\frac{\alpha^{d+2}e^{n/\alpha}}{\sqrt{n/\alpha}} \leq \mathrm{E}\left(T\right) \leq (1-o(1))\alpha^{d+2}e^{n/\alpha}. \qquad \square$$

7.6 Comparison of MA and $(1+1)$ EA

In order to compare the Metropolis algorithm with evolutionary algorithms, we estimate the optimization time of the (1+1) EA on CLIFF functions. An expected runtime of $\Theta(n^m)$ is has already been proven in [PHST17] for the classic case $d = m - 3/2$ and mutation rate $p = \frac{1}{n}$.

In the following theorem, we prove an upper bound on the optimization time of the (1+1) EA with general mutation rate p on $\text{CLIFF}_{d,m}$. Since our main aim is showing that the $(1+1)$ EA in many situations is faster than the MA, we prove no lower bounds. We note that for k or p not too large, one could show matching lower bounds with the methods developed in [DLMN17, BBD21a].

Theorem 7.11. *Consider the (1+1) EA with general mutation rate $0 < p < \frac{1}{2}$ optimizing $\text{CLIFF}_{d,m}$ with arbitrary m and $1 \leq d < m - 1$. Then the expected optimization time is at most*

$$E[T] \leq p^{-1}(1-p)^{-n+1}(1+\ln n) + \binom{m}{\lfloor d \rfloor + 2}^{-1} p^{-\lfloor d \rfloor - 2}(1-p)^{-n+\lfloor d \rfloor + 2}.$$

Any p minimizing this bound satisfies $p \leq \frac{\lfloor d \rfloor + 2}{n}$. If $m = O(n^{1/2}/\log n)$ and $p = \frac{\lambda}{n}$ for some $0 < \lambda \leq \lfloor d \rfloor + 2$, then this bound is

$$E[T] \leq (1+o(1))\frac{e^\lambda}{\lambda^{\lfloor d \rfloor + 2}} \binom{m}{\lfloor d \rfloor + 2}^{-1} n^{\lfloor d \rfloor + 2}.$$

This latter bound is minimized for $\lambda = \lfloor d \rfloor + 2$, which yields

$$E[T] \leq (1+o(1)) \binom{m}{\lfloor d \rfloor + 2}^{-1} \left(\frac{ne}{\lfloor d \rfloor + 2}\right)^{\lfloor d \rfloor + 2}.$$

Since we analyze an elitist algorithm, we can use Wegener's [Weg01] fitness level argument, which estimates the expected runtime by the sum of the expected times to leave each fitness level (apart from the optimal one).

In our proof, we will need the following elementary estimate.

Lemma 7.12. *Let $m = O(n^{1/2}/\log n)$, $2 \leq D \leq m$, and $0 < p \leq \frac{D}{n}$. Then*

$$p^{-1}(1-p)^{-n+1}\ln(n) = o\left(p^{-D}(1-p)^{-n+D}\binom{m}{D}^{-1}\right).$$

Proof. It suffices to show that $p^{-D+1}(1-p)^{D-1}\binom{m}{D}^{-1} = \omega(\log n)$. Assuming n to be sufficiently large, we have $p \leq \frac{D}{n} \leq \frac{m}{n} \leq \frac{1}{2}$ and thus $p^{-D+1}(1-p)^{D-1}\binom{m}{D}^{-1} \geq (\frac{2D}{n})^{-D+1}(\frac{D}{em})^D = \frac{D}{em}(\frac{n}{2em})^{D-1} \geq \frac{2}{em}\frac{n}{2em} = \Omega((\log n)^2) = \omega(\log n)$. □

Proof of Theorem 7.11. Let us first assume that $d \notin \mathbb{N}$ as in this case each set $L_i = \{x \in \{0,1\}^n \mid \|x\|_1 = i\}$ is a separate fitness level of CLIFF. Let s_i be the probability that a single iteration of the $(1+1)$ EA starting with a solution in L_i ends with a solution of better fitness (note that this is independent of the particular solution from L_i). Then Wegener's [Weg01] fitness level theorem gives the bound $E[T] \leq \sum_{i=0}^{n-1} \frac{1}{s_i}$ for the runtime T of the algorithm.

For $i \neq n-m$, that is, a level different from the local optimum, we have $s_i \geq (n-i)p(1-p)^{n-1}$ simply by regarding the event that the mutation operator flips a single zero-bit. Consequently, $\sum_{i \neq n-m} \frac{1}{s_i} \leq p^{-1}(1-p)^{-n+1}(1+\ln n) =: T'$.

To estimate s_{n-m}, we note that if the current search point is on the local optimum, then flipping any $\lfloor d \rfloor + 2$ of the zero-bits and no other bits leads to a better solution. Hence

$$s_{n-m} \geq \binom{m}{\lfloor d \rfloor + 2} p^{\lfloor d \rfloor + 2}(1-p)^{n-\lfloor d \rfloor - 2}.$$

Consequently, by the fitness level argument,

$$E[T] \leq \sum_{i=0}^{n-1} \tfrac{1}{s_i} \leq T' + \binom{m}{\lfloor d \rfloor + 2}^{-1} p^{-\lfloor d \rfloor - 2}(1-p)^{-n+\lfloor d \rfloor + 2},$$

which proves our claim for arbitrary mutation rate p. We note that the expression $p^x(1-p)^{n-x}$ is maximal exactly for $p = \frac{x}{n}$. Consequently, both T' and our estimate for $\frac{1}{s_{n-m}}$ are strictly increasing for $p \geq \frac{\lfloor d \rfloor + 2}{n}$. Hence any mutation rate minimizing our estimate for the expected runtime cannot be larger than $\frac{\lfloor d \rfloor + 2}{n}$.

If $p \leq \frac{\lfloor d \rfloor + 2}{n}$, then by our assumption $m = O(n^{1/2}/\log n)$ and Lemma 7.12, we have $E[T] \leq (1+o(1))\binom{m}{\lfloor d \rfloor + 2}^{-1} p^{-\lfloor d \rfloor - 2}(1-p)^{-n+\lfloor d \rfloor + 2}$, which yields the remaining small claims.

When d is an integer, then only the L_i with $i \in [0..n-1] \setminus \{n-m, n-m+d+1\} =: I$ form a complete fitness level of non-optimal solutions. The solutions on the remaining two Hamming levels have equal fitness. We estimate the probability to leave this level by the probability to generate a search point in

$L_{n-m+d+2}$. By [Wit13, Lemma 6.1], since $p \leq \frac{1}{2}$, this probability is at least the probability of finding an improvement from the (farther) level L_{n-m}. Hence $s^* = \binom{m}{d+2} p^{d+2}(1-p)^{n-d-2}$ is a lower bound for the probability to leave this fitness level, independent of the current search point. The resulting runtime estimate $E[T] \leq \sum_{i \in I} \frac{1}{s_i} + \frac{1}{s^*}$ is identical to our above estimate for the d-value $d + 0.5$, which concludes this proof. □

Still considering the CLIFF function, we shall now compare the bounds we have obtained for the expected runtime of the MA in Theorem 7.6 with the bounds on the runtime of the $(1+1)$ EA from Theorem 7.11. To this end, we will first investigate optimal parameter choices for α depending on the CLIFF parameters m and d and compare it with the bound for the $(1+1)$ EA, both for the standard mutation probability $1/n$ and the optimized one $(\lfloor d \rfloor + 2)/n$.

Our bounds on the runtime of MA and $(1+1)$ EA are rather precise, but arithmetically complicated and not necessarily tight. Since we want to analyze how much faster the MA can be compared to the $(1+1)$ EA, we compute parameter settings for α that make the lower bounds for the MA as small as possible. These minimized lower bounds will be contrasted with the upper bounds for the $(1+1)$ EA. Again, we have to distinguish between the two main cases for m in relation to n and α that appear in Theorem 7.6.

Case $m+1 > \lceil \frac{n}{\alpha+1} \rceil$: In this case, corresponding to Part 1 of Theorem 7.6, we have a lower bound on the runtime of the MA of

$$(1 - o(1)) \left(1 + \frac{\left(\frac{n}{\alpha}\right)^{m-2}}{(m-2)!}\right) \frac{n^2 \alpha^d}{m(m-1)}.$$

By computing the derivative of $\alpha^d \left(1 + \frac{\left(\frac{n}{\alpha}\right)^{m-2}}{(m-2)!}\right)$, we find that the expression is first decreasing and then increasing in α if $m - 2 \geq d$. We assume this condition on d now without analyzing the border case $d \in (m-2, m-1)$. Then the bound (up to a factor $1 \pm o(1)$ is minimized for

$$\alpha^* = n \left(\frac{m-d-2}{d(m-2)!}\right)^{1/(m-2)}.$$

For convenience, we assume $m = \omega(1)$ hereinafter and obtain $\alpha^* = (1 \pm o(1)) \frac{en}{m-2}$. Plugging this in our lower bound, we have an expected runtime for the MA of at least

$$(1 - o(1)) \frac{n^2}{m^2} \alpha^d = (1 - o(1)) e^d \left(\frac{n}{m}\right)^{d+2}.$$

By comparison, the bounds for the $(1+1)$ EA with the two mutation probabilities are no larger than

$$(1+o(1))e\left(\frac{n(\lfloor d\rfloor+2)}{m}\right)^{\lfloor d\rfloor+2} \text{ and } (1+o(1))\left(\frac{ne}{m}\right)^{\lfloor d\rfloor+2},$$

respectively, where we used $\binom{a}{b} \geq (a/b)^b$. Hence, if d is not an integer, the bound for the optimized $(1+1)$ EA is by a factor $\Theta((n/m)^{\lceil d\rceil-d})$ smaller than for the MA; in the border case of integral d, the bound for the MA is by a factor no larger than $(1+o(1))e^2$ smaller. The bound for the standard $(1+1)$ EA loses at most a factor of order $O(d^d)$. Note also that d must be a small constant for efficient (polynomial) optimization times anyway. Hence, the optimized MA is not much faster than the standard $(1+1)$ EA, while typically the optimized $(1+1)$ EA is even faster than the optimized MA.

Intuitively, the value $\alpha^* \approx \frac{en}{m}$ says that the equilibrium point $\frac{n}{\alpha+1}$ is around $\frac{m}{e}$, i. e., the cliff is clearly in the positive drift region. Hence, it seems plausible that the true minimal expected runtime of the MA is obtained for α falling into the present case. However, since we do not have a sufficiently precise, global expression for the runtime, we still have to consider the case with the cliff in the negative drift region.

Case $m+1 \leq \lceil \frac{n}{\alpha+1} \rceil$: In this case, corresponding to Part 2, the bound on the expected runtime of the MA is at least

$$\left(\frac{1}{\sqrt{2\pi}e^{1/12}} - o(1)\right)\frac{\alpha^{d+2}e^{n/\alpha}}{\sqrt{n/\alpha}}, \tag{7.18}$$

where we have used $\alpha \leq n$. For given n and d, the latter expression is first decreasing and then increasing in α, with the minimum taken at $\alpha^* = \frac{n}{d+5/2}$. Depending on the integrality of $\frac{n}{\alpha+1}$ appearing in the case condition, this choice of α may be slightly too big and violate the general assumption $d < m-1$; however, then $\alpha^* \approx \frac{n}{d+3}$ can be chosen to minimize the bound while meeting the condition. This will not essentially change the following reasoning. Plugging in $\alpha = \frac{n}{d+5/2}$ in (7.18) gives a lower bound for the MA of

$$\left(\frac{1}{\sqrt{2\pi}e^{1/12}} - o(1)\right)\left(\frac{ne}{d+5/2}\right)^{d+2}$$

then. The upper bounds for the $(1+1)$ EA in Theorem 7.11 for mutation probabilities $1/n$ and $(\lfloor d\rfloor+2)/n$ are no larger than

$$(1+o(1))en^{\lceil d\rceil+1} \text{ and } (1+o(1))\left(\frac{ne}{\lfloor d\rfloor+2}\right)^{\lfloor d\rfloor+2},$$

respectively, where we simply estimated $\binom{m}{\lfloor d \rfloor + 2}^{-1} \leq 1$. If d is not an integer, the bound for the optimized $(1+1)$ EA turns out lower than for the MA by a factor $\Theta((n/d)^{\lceil d \rceil - d})$; if it is an integer, the bound for MA is at most by a constant factor $\sqrt{2\pi} e^{1/12} \left(\frac{d+2.5}{d+2}\right)^{d+2}$ smaller. Moreover, the bound for the $(1+1)$ EA with standard mutation rate is at most by a factor of order $O(d^d)$ bigger. Altogether, we have arrived at the same conclusions as in the previous case.

7.7 Experiments

To supplement our theoretical results, we have run the MA, the $(1+1)$ EA and a few related algorithms on different instances of the CLIFF problem. More precisely, besides the MA with $\alpha \in \{20, 30, 40\}$ (good values in a preliminary experiment with broader range of α), we used the $(1+1)$ EA both with the standard mutation probability $1/n$ and the higher mutation probability $\lceil d+1 \rceil / n$, the Fast-$(1+1)$ EA using heavy-tailed mutation from [DLMN17] (with parameter $\beta = 1.5$), and the SD-$(1+1)$ EA, using stagnation detection, from [RW20b] (with parameters $R = n^3$ and the threshold value $(n/r)^r (n/(n-r))^{n-r} \ln(enR)$ for strength r). We ran these algorithms on CLIFF functions with problem size $n = 150$ and problem parameters $d = 3$ and growing $m \in \{8, 12, 16, \ldots, 32\}$.

The runtimes depicted in Figure 7.2 show that for all three value of α, the MA is clearly slower than the other algorithms (among which the $(1+1)$ EA with high mutation rate and Fast $(1+1)$ EA performed best).

Since the EAs using global mutation apparently coped well with the valley of low fitness, we also tried the MA with these mutation operators instead of one-bit flips. In this set of experiments, we ran the standard MA, the MA using standard bit mutation with mutation rate $1/n$, and the MA using the heavy-tailed mutation of the Fast-$(1+1)$ EA, all for $\alpha = 20$ (the most promising value for the smaller problem size $n = 100$ which we used here). To investigate whether the ability of MA to accept worse search points was relevant, we included the $(1+1)$ EA with mutation rate $1/n$ in this comparison. In the runtime results presented in Figure 7.3, the two global-mutation MAs overall perform better than the standard MA. The $(1+1)$ EA might be the overall best choice, only occasionally mildly beaten by the heavy-tailed MA. While these preliminary experiments do not suffice to draw final conclusions, they motivate as future work a closer analysis of MA with global mutation operators.

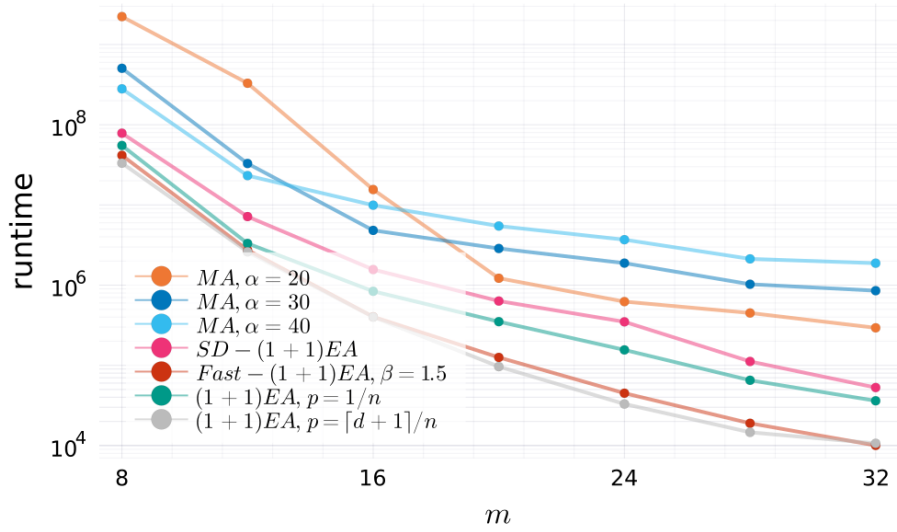


Figure 7.2: Comparison of MA with $(1+1)$ EA and its variants on $\textsc{Cliff}_{m,d}$ for $n = 150$, $d = 3$ and increasing m; averaged over 100 runs.

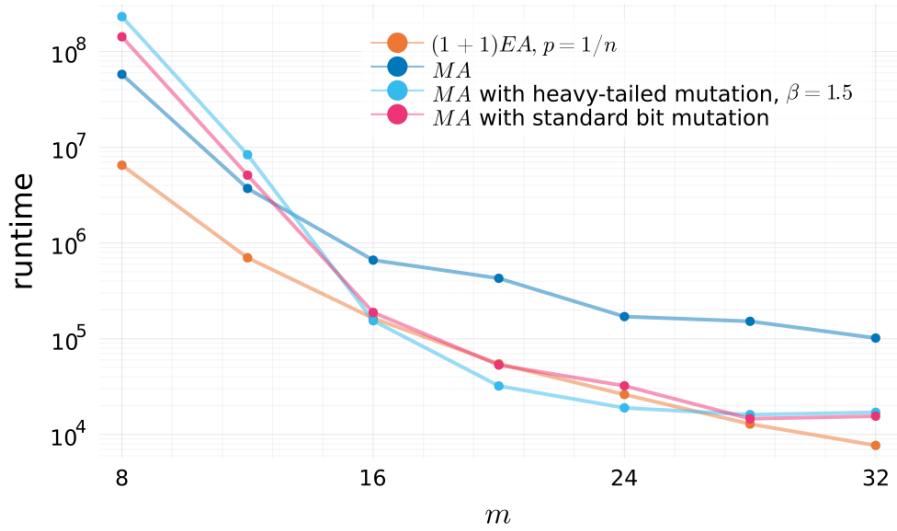


Figure 7.3: Comparison of $(1+1)$ EA and different variants of MA, including global mutation, on $\textsc{Cliff}_{m,d}$ for $n = 100$, $\alpha = 20$, $d = 3$ and increasing m; averaged over 50 runs.

7.8 Conclusions

We have conducted a mathematical runtime analysis of the Metropolis Algorithm (MA), a local search algorithm that with positive probability may accept worse solutions, on the multimodal CLIFF benchmark problem. This problem, which always has a gradient towards the global optimum except for one Hamming level, seems like a canonical candidate where MA can profit from its ability to accept inferior solutions. However, our mathematical runtime analysis has revealed that this intuition is not correct. The simple elitist $(1 + 1)$ EA is provably at least as fast and for many parameters faster than the MA, even for an optimal choice of the temperature parameter of the MA. This failed attempt to explain the effectiveness of the MA raises the question of what is the real reason for the success of the MA in many practical applications.

The comparably good performance of algorithms using global mutation operators suggest to also use the MA with such operators. Our preliminary experimental results show that this can indeed be an interesting idea – backing up these findings with a mathematical runtime analysis is an interesting problem for future research.

A second question of interest is to what extent simulated annealing, that is, the MA with a temperature decreasing over time, can improve the relatively weak performance of the classic MA on CLIFF even with optimal choice of the temperature. From our understanding gained in this work, we are slightly pessimistic, mostly because again a relatively small temperature is necessary to reach the local optimum and then diving into the fitness valley is difficult, but definitely a rigorous analysis of this question is necessary to understand this question.

Chapter 8

Paper F: Simulated Annealing is a Polynomial-Time Approximation Scheme for the Minimum Spanning Tree Problem

We prove that Simulated Annealing with an appropriate cooling schedule computes arbitrarily tight constant-factor approximations to the minimum spanning tree problem in polynomial time. This result was conjectured by Wegener (2005). More precisely, denoting by $n, m, w_{\max}$, and $w_{\min}$ the number of vertices and edges as well as the maximum and minimum edge weight of the MST instance, we prove that simulated annealing with initial temperature $T_0 \geq w_{\max}$ and multiplicative cooling schedule with factor $1-1/\ell$, where $\ell = \omega(mn\ln(m))$, with probability at least $1-1/m$ computes in time $O(\ell(\ln\ln(\ell)+\ln(T_0/w_{\min})))$ a spanning tree with weight at most $1+\kappa$ times the optimum weight, where $1+\kappa = \frac{(1+o(1))\ln(\ell m)}{\ln(\ell)-\ln(mn\ln(m))}$. Consequently, for any $\varepsilon > 0$, we can choose ℓ in such a way that a $(1+\varepsilon)$-approximation is found in time $O((mn\ln(n))^{1+1/\varepsilon+o(1)}(\ln\ln n + \ln(T_0/w_{\min})))$ with probability at least $1-1/m$. In the special case of so-called $(1+\varepsilon)$-separated weights, this algorithm computes an optimal solution (again in time $O((mn\ln(n))^{1+1/\varepsilon+o(1)}(\ln\ln n + \ln(T_0/w_{\min}))))$, which is a significant speed-up over Wegener's runtime guarantee of $O(m^{8+8/\varepsilon})$.

8.1 Introduction

The theory of randomized search heuristics, mostly in the last 25 years, has considerably increased our understanding of this class of algorithms. A closer look at this field shows that in the early years, significant efforts were devoted also to simulated annealing (SA) [SH88, JS98, Weg05, JW07], whereas more recently these algorithms at most appear in side results of works focused on other heuristics. Due to this decline in attention, the gap between theory and practice, at least as wide in heuristics as in classic algorithms, is even wider for SA.

Since we do not see a reducing interest in SA in practice [FS19], with this first theoretical work solely devoted to SA after a longer time, we aim at reviving the theoretical analysis of this famous heuristic. To this aim, we revisit a classic problem, namely how SA computes minimum spanning trees (MSTs) [Weg05]. We are, of course, not finally interested in using SA for this purpose – for this several very efficient near-linear time algorithms are known –, but we use this problem to try to understand the working principles of SA.

Wegener's seminal work [Weg05] is well-known for the construction of an instance of the MST problem where the Metropolis algorithm with any fixed temperature fails badly, but SA with a simple multiplicative cooling schedule computes an optimal solution efficiently. Much less known, but equally interesting is another result in this work, namely that SA with a suitable multiplicative cooling schedule can efficiently find optimal solutions to the MST problem when the edge weights are $(1+\varepsilon)$-separated.

Theorem 8.1 ([Weg05])**.** *Let $G = (V, E)$ with $w : E \to \mathbb{Z}_{>0}$ be an instance of the MST problem. Let $\varepsilon > 0$ be such that for all edges $e_1, e_2 \in E$, we have that $w(e_1) > w(e_2)$ implies $w(e_1) \geq (1+\varepsilon)w(e_2)$. Assume further that $w(e) \leq 2^m$ for all $e \in E$. Then SA with initial temperature $T_0 = 2^m$ and cooling factor $\beta = (1+\varepsilon/2)^{-m^{-7-8/\varepsilon}}$ with probability $1 - O(1/m)$ finds an optimal solution in at most $2\log_2(1+\varepsilon/2)^{-1}m^{8+8/\varepsilon}$ iterations.*

Wegener [Weg05] conjectured that his SA algorithm for general weights instead of $(1+\varepsilon)$-separated ones computes $(1+\varepsilon)$-approximate minimum spanning trees, that is, trees with weight at most $(1+\varepsilon)$ times the weight of a true minimum spanning tree. While this conjecture is very natural, it was never proven.

Our main result is that Wegener's conjecture is indeed true, even though our proof does not confirm his statement that "it is easy to generalize our result to prove that SA is always highly successful if one is interested in $(1+\varepsilon)$-optimal spanning trees." More precisely, we show the following result (see Theorem 8.4

for a slightly stronger, but more complicated version of this result). We note that SA cannot compute $(1+\varepsilon)$-approximations for sub-constant ε, see again [Weg05], so in this sense our result is as good as possible.

Let $\varepsilon > 0$ be a constant. Consider a run of SA with cooling factor $\beta = 1 - 1/\ell$, where $\ell = (mn\ln(m))^{1+1/\varepsilon+o(1)}$, and $T_0 \geq w_{\max}$ on an instance of the MST problem. Then there is a time $T^* = O((mn\ln(n))^{1+1/\varepsilon+o(1)}(\ln\ln n + \ln(T_0/w_{\min})))$ such that with probability at least $1 - 1/m$, at all times $t \geq T^*$ the current solution is a $(1+\varepsilon)$-approximation.

Due to the use of proof methods not available at that time, our time bound is significantly better than Wegener's. To compute a $(1+\varepsilon)$-approximation, or to compute an optimal solution when the edge weights are $(1+\varepsilon)$-separated (see Theorem 8.11), our runtime guarantee is roughly $O((mn\log n)^{1+1/\varepsilon}\log\frac{w_{\max}}{w_{\min}})$ as opposed to $O(m^{8+8/\varepsilon})$ in Theorem 8.1.

Mostly because of a different organization of the proof, our result gives more insights into the influence of the algorithm parameters. Our result only applies to initial temperatures T_0 that are at least the maximum edge weight. This is very natural since with substantially smaller temperatures, the heaviest edge cannot be included in the solution with reasonable probability (this follows right from the definition of the algorithm). It is also not difficult to prove that once the temperature is somewhat below the smallest edge weight, then no new edges will ever enter the solution (see Lemma 8.6 for the precise statement of this result). This implies that there is no reason to run the algorithm longer than roughly for time $\log_{1/\beta}(T_0/w_{\min}) = O(\ell\log(T_0/w_{\min}))$, see Theorem 8.9 for the details. From the perspective of the algorithm user, this is an interesting insight since it gives an easy termination criterion. Also without understanding the precise influence of the cooling factor β on the approximation quality, this insight motivates to use the algorithm for decreasing values of β, say $\beta_i = 2^{-i}$, always until the above-determined time is reached, and follow this procedure until a sufficiently good MST approximation is found.

The remainder of this paper is organized as follows. In Section 8.2, we describe the most relevant previous works. We define SA and the minimum spanning tree problem in Section 8.3. The core of this work is our mathematical runtime analysis in Section 8.4. Afterwards, in Section 8.5, we give the result carried out for the MST problem with $(1+\varepsilon)$-separated weights. The paper ends with a conclusion and a discussion of possible future works.

8.2 Previous Work

As mentioned in the introduction, there are relatively few runtime analyses for SA as discrete optimization algorithm, see also the survey [Jan11].

The first such result [SH88] proves that SA can compute good approximations to the maximum matching problem. A closer look at the result reveals that a constant temperature is used, that is, the SA algorithm is in fact the special case of the Metropolis algorithm. It has to be noted that to obtain a particular approximation quality, the temperature has to be set suitably. In this light, the following result from [GW03] shows a light advantage for evolutionary algorithms: When running the $(1+1)$ EA with standard mutation rate on this problem, then the expected first time to find a $(1+\varepsilon)$-approximation is $O(m^{2\lceil 1/\varepsilon \rceil})$. Note that in this result, the parameters of the algorithm do not need to be adjusted to the desired approximation rate.

For a different problem, namely the bisection problem, it was shown in [JS98] that SA, again with constant temperature, can solve certain random instances in quadratic time.

Wegener's above mentioned work [Weg05] on the MST problem was the first to show that for some non-artificial problem, a non-trivial cooling schedule is necessary.

A runtime analysis of the Metropolis algorithm on the classic benchmark OneMax was conducted in [JW07]. Not surprisingly, the ability to accept inferior solutions is not helpful when optimizing this unimodal function. The interesting side of this result, though, is that the Metropolis algorithm is efficient on OneMax only for very small temperatures of asymptotic order $O(\log(n)/n)$.

A recent study [WZD21] on the deceiving-leading-blocks (DLB) problem shows that here the Metropolis algorithm with a constant temperature has a good performance, beating the known runtime results for evolutionary algorithms by a factor of $\Theta(n)$. We note that the DLB problem, just as the MST problem, has many local optima which all can be left by flipping two bits.

As side results of a fundamental analysis of hyper-heuristics, two easy lower bounds on the runtime of the Metropolis algorithm (that is, SA with constant temperature) are proven in [LOW19]: (i) The Metropolis algorithm needs time $\tilde{\Omega}(n^{d-1/2})$ on cliff functions with constant cliff width d and super-polynomial time when the cliff width is super-polynomial. (ii) The Metropolis algorithm with a temperature small enough to allow efficient hill-climbing needs exponential time to optimize jump functions.

As part of a broader analysis of single-trajectory search heuristics, it was found that the Metropolis algorithm can optimize all weakly monotonic pseudo-Boolean functions in at most exponential time [Doe21].

Some more results exist on problems designed for demonstrating a particular phenomenon. In [DJW00], a problem called VALLEY is designed that has the property the Metropolis algorithm with any temperature needs at least exponential expected time, whereas SA with a suitable cooling schedule only needs time $O(n^5 \log n)$. In [JW07], examples are constructed where one of $(1+1)$ EA and SA has a small polynomial runtime and the other has an exponential runtime. Also, a class of functions is constructed where both algorithms have a similar performance despite dealing with the local optimum in a very different manner. In [OPH+18], a class of problems with tunable width and depths of a valley of low fitness is proposed. It is proven that the performance of the elitist $(1+1)$ EA is mostly influenced by the width of the valley, whereas the performance of the Metropolis algorithm and a similar non-elitist algorithm inspired from population genetics is mostly influenced by the depths of the valley.

For evolutionary algorithms, for which the theory is more developed than for SA, there are a larger number of results showing that they can serve as approximation algorithms for optimization problems, including NP-hard problems [NW10]. However, results describing an approximation scheme where the user can provide a parameter ε to the evolutionary algorithm to compute a $(1+\varepsilon)$-approximation are rare; apart from the maximum matching problem mentioned above, we are only aware of related results for parallel (1+1) EAs, (1+1) EAs with ageing and simple artificial immune systems on the number partitioning problem [Wit05, COY19] and for an evolutionary algorithm on the multi-objective shortest path problem [Hor10]. Evolutionary algorithms that approximate the optimum are also known in the subfield of fixed-parameter tractability. While most of these results prove an approximation within a constant factor or growing slowly with the problem dimension, there are also statements similar to approximation schemes for the vertex cover problem [NS20]. However, in general it is safe to say that there are only few results in the literature that characterize very simple randomized search heuristics like the $(1+1)$ EA and SA as polynomial-time approximation schemes for classical (non-noisy) combinatorial optimization problems.

Finally, we remark that the classical (1+1) EA and a variant of randomized local search can solve the MST problem in expected pseudo-polynomial time $O(m^2 \log(nw_{\max}))$ [NW07]. While SA in general does not solve the problem in expected polynomial time, its time bound to achieve a $(1+\epsilon)$-approximation (see Theorem 8.4 below) can be smaller than the time bound for the (1+1) EA in certain cases where $m = \omega(n)$ and ϵ is a constant.

8.3 Preliminaries

We now define the SA algorithm and the MST problem. Also, we state a technical tool our main proof builds on.

Algorithm 19: Simulated Annealing (SA) with starting temperature T_0 and cooling factor $\beta \leq 1$ for the minimization of $f\colon \{0,1\}^n \to \mathbb{R}$

Select $x^{(0)}$ from $\{0,1\}^n$;
for $t \leftarrow 0, 1, \ldots$ **do**
  Create y by flipping a bit of $x^{(t)}$ chosen uniformly at random;
  if $f(y) \leq f(x^{(t)})$ **then**
    $x^{(t+1)} \leftarrow y$;
  else
    $x^{(t+1)} \leftarrow y$ with probability $e^{(f(x^{(t)})-f(y))/T_t}$ and
    $x^{(t+1)} \leftarrow x^{(t)}$ otherwise;
  $T_{t+1} \coloneqq T_t \cdot \beta$;

Simulated annealing (SA) is a simple stochastic hill-climber first proposed as optimization algorithm in [KGJV83]. Different from a true hill-climber it may, with small probability, also accept inferior solutions. Working with bit-string representations, we use the classic *bit-flip neighborhoods*, that is, the neighbors of a solution are all other solutions that differ from it in a single bit value. For the acceptance of inferior solutions, we use the widely accepted *Metropolis condition*, that is, a solution with fitness loss δ over the current solution is accepted with probability $e^{-\delta/T}$, where T is the current *temperature*. The temperature is usually not taken as constant, but is reduced during the run of the algorithm. This allows the algorithm to accept worsening moves easy in the early stages of the run, whereas later worsening moves are accepted with smaller probability, bringing the algorithm closer to a true hill-climber. The choice of the *cooling schedule* is a critical decision in the design of a SA algorithm. A popular choice, already proposed in [KGJV83], is a multiplicative cooling schedule (also called geometric cooling scheme). Here we start with a given temperature T_0 and reduce the temperature by some factor β in each iteration. This common variant of SA, see Algorithm 19 for the pseudocode, was regarded also in the predecessor work of Wegener [Weg05].

The *minimum spanning tree (MST) problem* is defined as follows. We are given an undirected, connected, weighted graph $G = (V, E)$. We denote by n its number of vertices and by m its number of edges. Let the set of edges be $E = \{e_1, \ldots, e_m\}$. The weight of edge e_i, where $i \in \{1, \ldots, m\}$, is a positive

number w_i. We write $w_{\min} \coloneqq \min\{w_i \mid i \in \{1, \dots, m\}\}$ and $w_{\max} \coloneqq \max\{w_i \mid i \in \{1, \dots, m\}\}$ for the minimum and maximum edge weight.

The task in the MST problem is to find a subset $E' \subseteq E$ such that (V, E') is a spanning tree of G having minimal total weight $w(E') = \sum_{e_i \in E'} w_i$. We use the natural bit-string representation for sets E' of edges, that is, a bit string $x = (x_1, \dots, x_m) \in \{0,1\}^m$ represents the set $E(x) = \{e_i \mid x_i = 1\}$. As objective function, we use the sum of the weights of the selected edges when these form a connected graph on V and ∞ otherwise:

$$f(x) = \begin{cases} w_1 x_1 + \cdots + w_m x_m & \text{if } (V, E(x)) \text{ is connected,} \\ \infty & \text{otherwise.} \end{cases}$$

Here ∞ can be replaced by an extremely large value without essentially changing the result. To ensure that we start with a feasible solution (one that has finite objective value), we assume that SA is initialized with the all-ones string $x^{(0)} = (1, \dots, 1)$. From this initial string, SA can move to solutions having fewer edges by flipping one-bits; however, it will never accept solutions that are not connected due to their infinitely high f-value. We note that, similarly to the analysis of the $(1+1)$ EA on the MST problem [NW07], one could use a more involved fitness function to penalize connected components and thus lead the algorithm towards connected subgraphs when the current solution is not connected. However, since we assumme SA to start from a connected solution and connected solutions will not be replaced with disconnected solutions with the present definition of f, this would not provide new insights. Overall, our setup is the same as the one used by Wegener [Weg05].

When the temperature has become sufficiently low, it is likely that SA has reached a solution describing a spanning tree. If this spanning tree is suboptimal, improvements require a change of at least 2 bits. Since SA only flips one bit per iteration, this is only possible by temporarily including one more edge, i. e., closing a cycle, and then removing another edge from the cycle in the next iteration. This requires a temperature still being sufficiently high for the temporary inclusion to be accepted.

Our measure of complexity is the first hitting time T^* for a certain set of solutions S^*, e. g., globally optimal solutions or solutions satisfying a certain approximation guarantee with respect to the set of global optima. That is, we give bounds on the smallest t such that SA has found a solution in S^*. Due to the probabilistic nature of the algorithm, we will usually give bounds that hold with high probability, e. g., with probability $1 - 1/n$. The expected value of T^* may be undefined since the cooling schedule may make it less and less likely to hit the set S^* when the algorithm has been unsuccessful during the steps where a promising temperature held. This is different from the analysis of, e. g.,

simple evolutionary algorithms, where one often considers the so-called *runtime* as the first hitting time of the set of optimal solutions and bounds the expected runtime. However, as described in detail by Wegener [Weg05], there are simple restart schemes for SA that guarantee expected polynomial optimization times if there is a sufficiently high probability of a single run being successful in polynomial time.

The proof of our main result uses multiplicative drift analysis as state-of-the-art technical tool, which was not available to Wegener [Weg05]. The multiplicative drift theorem in Theorem 8.2 below goes back to [DJW12] and was enhanced with tail bounds in [DG13]. We give a slightly generalized presentation that can be found in [LW21].

Theorem 8.2 (Multiplicative Drift, cf. [DJW12, DG13, LW21])**.** *Let $(X_t)_{t\geq 0}$, be a stochastic process, adapted to a filtration $\mathcal{F}_t$, over a state space $S \subseteq \{0\} \cup [s_{\min}, s_{\max}]$, where $s_{\min} > 0$ and $\{0\} \in S$. Suppose that there exists a $\delta > 0$ such that for all $t \geq 0$, we have*

$$\mathrm{E}\left(X_t - X_{t+1} \mid \mathcal{F}_t\right) \geq \delta X_t.$$

Then the first hitting time $T := \min\{t \mid X_t = 0\}$ satisfies

$$\mathrm{E}\left(T \mid \mathcal{F}_0\right) \leq \frac{\ln(X_0/s_{\min}) + 1}{\delta}.$$

Moreover, $\Pr(T > (\ln(X_0/s_{\min}) + r)/\delta) \leq e^{-r}$ for any $r > 0$.

8.4 SA as Approximation Scheme for the Minimum Spanning Tree Problem

In this section, we prove our main results on how well SA computes approximate solutions for the MST problem. These results easily imply improved bounds for the previously regarded special case of $(1+\varepsilon)$-separated instances, see Section 8.5.

8.4.1 Main Results and Proof Outline

As outlined above in the introduction, this paper revisits Wegener's [Weg05] analysis of SA on the MST problem. Our main result is Theorem 8.3 below,

proving that SA is a polynomial-time approximation scheme for the MST problem as originally conjectured by Wegener. The statement of our main theorem describes the approximation quality and the required time to reach it as a function of the cooling factor, the desired success probability and of course the instance parameters. Theorem 8.4 takes the dual perspective of computing cooling schedules and running times that allow SA to find a $(1+\varepsilon)$-approximation for a given ε with high probability.

We now present the main theorem and a variant of it, corresponding to the two perspectives mentioned above for analyzing the approximation quality.

Theorem 8.3. *Let $\delta < 1$. Consider a run of SA with multiplicative cooling schedule with $\beta = 1 - 1/\ell$ for some $\ell = \omega(mn \ln(m/\delta))$ and $T_0 \geq w_{\max}$ on an instance of the MST problem. With probability at least $1-\delta$, at all times $t \geq (\ell/2) \ln\left(\frac{\ln(4(\ell-1)/\delta)T_0}{w_{\min}}\right)$ the current solution is a $(1+\kappa)$-approximation, where*

$$1 + \kappa \leq (1 + o(1)) \frac{\ln(\ell/\delta)}{\ln(\ell) - \ln(mn \ln(m/\delta))}.$$

Theorem 8.4. *Let $\delta = \omega(1/(mn \ln n))$ and $\delta < 1$, $\varepsilon > 0$. Consider a run of SA with $\beta = 1 - 1/\ell$ for $\ell = (mn \ln(m/\delta))^{1+1/\varepsilon}$ and $T_0 \geq w_{\max}$ on an instance of the MST problem. With probability at least $1-\delta$, at all times $t \geq (\ell/2) \ln\left(\frac{\ln(4(\ell-1)/\delta)T_0}{w_{\min}}\right)$ the current solution is a $(1+o(1))(1+\varepsilon)$-approximation.*

The last theorem is stated in somewhat weaker, but simpler form in the following corollary. In particular, it gives a concrete time bound until SA has computed a $(1+\varepsilon)$-approximation with probability at least $1-\delta$, where δ and ε are chosen by the user.

Corollary 8.5. *Let $\varepsilon > 0$ be a constant and $\delta = \omega(1/(mn \ln n))$. Consider a run of SA with $\beta = 1 - 1/\ell$, where*

$$\ell = \left(mn \ln\left(\frac{m}{\delta}\right)\right)^{1+1/\varepsilon+o(1)},$$

and $T_0 \geq w_{\max}$ on an instance of the MST problem. With probability at least $1-\delta$, at all times $t \geq T^* := (\ell/2) \ln\left(\frac{\ln(4(\ell-1)/\delta)T_0}{w_{\min}}\right)$ the current solution is a $(1+\varepsilon)$-approximation. Moreover,

$$T^* = O\left((mn \ln(n))^{1+1/\varepsilon+o(1)} \left(\ln \ln n + \ln\left(\frac{T_0}{w_{\min}}\right)\right)\right).$$

The idea of the proof of all results formulated above is to consider phases in the optimization process, concentrating on different intervals for the edge weights,

with the size and center of the intervals decreasing over time. In each phase, the number of edges chosen from such an interval will achieve some close-to-optimal value with high probability. After the end of the phase, the temperature of SA is so low that basically no more changes occur to the edges with weights in the interval.

In more detail, the proofs of Theorem 8.3 and its variant are composed of several lemmas. We are now going to outline the main ideas of these lemmas and how they relate to each other in the roadmap of the final proof.

It is useful to formulate the main results in terms of a cooling factor $\beta = 1 - 1/\ell$ for some $\ell > 1$ since ℓ carries the intuition of a "half-life" for the temperature; more precisely, after ℓ iterations of SA the temperature has decreased by the constant factor of $(1 - 1/\ell)^\ell \approx e^{-1}$. Lemma 8.6 is (on top of the usual graph parameters and the starting temperature) based on ℓ, a weight w and some parameter a. Intuitively, it describes a point of time t_w after which edges of weight at least w are no longer flipped in with high probability and can be ignored for the rest of the analysis due to an exponential decay in the probability of accepting search points of higher f-value. This probability depends on the parameter a which will be optimized later in the composition of the main proof.

While Lemma 8.6 will be used to show that edges above a certain weight are no longer included in the current solution after the temperature has dropped sufficiently, Lemma 8.7, which is the main lemma in our analysis, deals with the structure of the current solution after edges of a certain weight w are no longer included. It considers connected components that can be spanned by cheaper edges and states that these connected components are essentially connected in an optimal way in the whole solution up to multiplicative deviations of a factor $(1 + \kappa)$ in the weights of the connecting edges. Lemma 8.7 uses careful edge exchange arguments in its proof and bounds the time to do these exchanges in a multiplicative drift analysis. Moreover, it features another parameter called γ that will be optimized later along with the above-mentioned a.

Lemma 8.8 puts together the previous two lemmas to consider the run of SA over up to n phases depending on the weight spectrum of the graph until the temperature has dropped to a value being so small that no more changes are accepted. This will be the final solution considered in the main proof. Essentially, having listed the weights of an MST decreasingly, the lemma will match the weights of the final solution to the weights of the MST and show for each element in the list that the final solution matches the weight of the element up to a factor $1 + \kappa$. Its proof uses a bijection argument proved by induction to apply Lemma 8.7 and is crucially different from Wegener's analysis.

The final lemma, Lemma 8.10, finds choices for the parameter γ to minimize the bound $1 + \kappa$ on the approximation ratio. Its proof uses several results from calculus. Afterwards, Theorem 8.3 also chooses the parameter a carefully and arrives at the first statement on the approximation ratio depending on ℓ, the desired success probability $1 - \delta$, and the graph parameters, only. The second main theorem, Theorem 8.4 then essentially translates parameters into each other to compute ℓ and to express time bounds based on the desired ε. A weaker but simpler formulation of that theorem is finally stated in Corollary 8.5.

8.4.2 Detailed Technical Analysis

In this subsection, we collect the technical lemmas and theorems outlined above.

Let $a > 1$ and t_w be the earliest point of time when $T(t_w) \leq w/a$. In the following lemma, we state that the probability that SA accepts edges of weight w after t_w is exponentially small with respect to a. It shows that after the temperature becomes less than w, the probability of accepting such an edge is sharply decreasing.

Lemma 8.6. *Consider a run of SA with multiplicative cooling schedule with $\beta = 1 - 1/\ell$ and $T_0 \geq w_{\max}$ on an instance of the MST problem. Let $\ell > 2$, $1 < a \leq \ell - 1$ and for any $w > 0$, t_w be the earliest point of time when $T(t_w) \leq w/a$. It holds that no new edge of weight at least w is included in the solutions after time t_w with probability at least*

$$1 - \frac{2(\ell - 1)}{ae^a},$$

which is at least $1 - \delta/2$ for $\delta < 1$, if we set $a \geq \ln(4(\ell - 1)/\delta)$.

Proof. Let s be an edge of weight at least w, which is not in the solution at the beginning of the step t_w. Let $t \in \mathbb{N}_{\geq 0}$ and $E_s^{(t_w+t)}$ be the event of accepting the edge s at step $t_w + t$. This event happens if the edge s is flipped with probability $1/m$ and the algorithm accepts this worse solution. Thus

$$\Pr\left(E_s^{(t_w)}\right) = m^{-1} \cdot \exp\left(\frac{-w}{T(t_w)}\right) \leq \frac{e^{-a}}{m}.$$

For all integers $t \geq 0$, we have $T(t_w + t) = T(t_w)(1 - 1/\ell)^t$. Then

$$\begin{aligned}\Pr\left(E_s^{(t_w+t)}\right) &= m^{-1} \exp\left(\frac{-w}{T(t_w)(1 - \frac{1}{\ell})^t}\right) \\ &\leq m^{-1} e^{-a(1+\frac{1}{\ell-1})^t}\end{aligned}$$

$$\leq m^{-1}e^{-a(1+\frac{t}{\ell-1})},$$

where we used the inequality $(1+x)^r \geq 1+rx$ for $x > -1$ and $r \in \mathbb{N}_{\geq 0}$.

Let $E_s^{\geq t_w}$ be the event of accepting the edge e of weight at least w after step t_w at least once. Then, using the geometric series sum formula, we get

$$\begin{aligned}\Pr\left(E_s^{\geq t_w}\right) &\leq \sum_{t=0}^{\infty} \Pr\left(E_e^{(t_w+t)}\right) \leq \sum_{t=0}^{\infty} m^{-1}e^{-a(1+\frac{t}{\ell-1})} \\ &= m^{-1}\frac{e^{-a}}{1-e^{-a/(\ell-1)}} \leq m^{-1}\frac{e^{-a}}{1-(1-\frac{a}{2(\ell-1)})} \\ &= m^{-1}\frac{2(\ell-1)}{ae^a},\end{aligned}$$

where we have $a \leq \ell - 1$ and use the inequality $e^{-x} \leq 1 - x/2$ for $0 \leq x \leq 1$.

Since there are m edges, with probability $1 - \frac{2(\ell-1)}{ae^a}$, there is no inclusion of edges after their corresponding steps t_w.

Moreover, if we set $a \geq \ln(4(\ell-1)/\delta)$, the probability is at least

$$1 - \frac{2(\ell-1)}{\ln(4(\ell-1)/\delta)\cdot 4(\ell-1)/\delta} = 1 - \frac{\delta/2}{\ln(4(\ell-1)/\delta)} \geq 1 - \frac{\delta}{2},$$

where we have $\ell > 2$ and $\delta < 1$. □

In Lemma 8.7, we consider a time interval of length $4.21\gamma mn\ln(2m^2/\delta)+1$ starting from t_w (for fixed a) and prove that at the end of this period, there are no edges of weight at least w left that could be replaced by an edge of weight at most $w/(1+\kappa)$, where κ depends on the algorithm parameter ℓ and parameters γ and a. We optimize these parameters later in this paper.

Lemma 8.7. *Let $\gamma > 1$, $\delta < 1$, $\ell > 2$, $a > 1$. Consider a run of SA with multiplicative cooling schedule with $\beta = 1 - 1/\ell$ and $T_0 \geq w_{\max}$ on an instance of the MST problem. Let t_w be the earliest point of time when $T(t_w) \leq w/a$, and assume that no further edges of weight at least w are added to the solution from time t_w. Let*

$$1 + \kappa = \frac{a\exp\left(\gamma\frac{4.21mn\ln(2m^2/\delta)}{\ell-1}\right)}{\ln\gamma}.$$

Let n_w be the number of connected components in the subgraph using only edges with weight at most $w/(1+\kappa)$ in G. After time $t_w + 4.21\gamma mn\ln(2m^2/\delta)$, the

number of edges in the current solution with weight at least w is at most $n_w - 1$ with probability at least $1 - \delta/(2m)$.

Proof. Let $T_{base} = 4.21mn \ln(2m^2/\delta)$. We analyze the steps $t_w, \ldots, t_w + \gamma T_{base}$. The temperature during this phase is at least

$$T(t_w)\left(1 - \tfrac{1}{\ell}\right)^{\gamma T_{base}} \geq T(t_w) e^{-\frac{\gamma T_{base}}{\ell - 1}},$$

so the probability to accept a chosen edge with weight at most $w/(1+\kappa)$ in one step is bounded from below by

$$\exp\left(\frac{-w/(1+\kappa)}{T(t_w)e^{-\frac{\gamma T_{base}}{\ell-1}}}\right) = \exp\left(-\frac{a e^{\frac{\gamma T_{base}}{\ell-1}}}{(1+\kappa)}\right) = \gamma^{-1}$$

during this phase. By our assumption in the statement, we do not include edges of weight at least w.

Let us partition the set of edges with weight at least w in the current solution x, that is, the graph $G_x = (V, E(x))$, into three disjoint subsets. An edge $e = \{u, v\}$ with weight at least w has one of the following three properties,

1. the edge e lies on a cycle in G_x;
2. the edge e does not lie on a cycle, but there is at least one edge $e' \in E \backslash E(x)$ with weight at most $w/(1+\kappa)$ such that e lies on a cycle in the graph $(V, E(x) \cup \{e'\})$;
3. the edge e has neither of the two properties. In this case, we call this edge essential for the current and forthcoming solutions.

As long as an edge with weight at least w is not essential, it can either be removed from the current solution or become an essential edge. When the edge disappears, since its weight is at least w, it will not appear again.

Also, when the edge becomes essential, it remains essential in the solution to the end, because in order to create a cycle containing this edge, an edge with weight at least w has to appear, which does not happen, and also removing this edge makes the graph unconnected.

We claim that the number of essential edges does not exceed $n_w - 1$. In order to prove this, we define the graph $H = (V_H, E_H)$ as follows. There is a vertex in V_H for each connected component of the induced subgraph on the edges of weight at most $w/(1+\kappa)$ in G, and there is an edge between two vertices $v_i, v_j \in V_H$

if there is an essential edge $e = \{u, v\}$ in the solution that u and v belong to the corresponding connected components C_i and C_j respectively. Formally, let $C = \{C_1, \ldots, C_{n_w}\}$ be the connected components of the induced subgraph on the edges of weight at most $w/(1+\kappa)$. Then, $V_H = \{v_1, \ldots, v_{n_w}\}$ and

$$E_H = \{\{i, j\} \mid \exists \text{ essential } e = \{u, v\}, u \in C_i, v \in C_j\}.$$

We claim that there is no essential edge with both endpoints in the same C_i. To prove this, we assume for contradiction that there is such an edge $e = \{u, v\}$. Then, since e is essential, it cannot be on a cycle in the current solution. Let S_u and S_v denote the sets of vertices connected to u and v respectively using edges in the solution but e. $S_u \cup S_v = V(G)$ because the solution is always connected. Since e is essential, there is no edge with weight at most $w/(1+\kappa)$ in G from S_u to S_v (see the property (2)), so there is no such cheap edges in G from $S_u \cap C_i$ to $S_v \cap C_i$, which results in that there is a partition of vertices of C_i that are disconnected in the subgraph using only edges with weight at most $w/(1+\kappa)$ in G, which contradicts the definition of C_i. Also, H has to be a forest since we also know that essential edges are not on a cycle. Therefore, since there are n_w connected components, there are at most $n_w - 1$ essential edges.

Now, in the next paragraphs, we state the number of steps needed to remove edges with weight at least w or to make them essential. We consider some epochs consisting of $2m$ iterations each and let X_t be the random variable denoting the number of non-essential edges with weight at least w whose exclusion is possible at epoch t. We claim that

$$\Delta_t(s) \coloneqq \mathrm{E}\left(X_t - X_{t+1} \mid X_t = s\right) \geq s \cdot \frac{(1 - e^{-3})n^{-1}}{2\gamma}.$$

If no cycle with a non-essential edge $e = \{u, v\}$ with weight at least w exists, the probability of creating such a cycle by adding the cheap edge considered in Case 2 between S_u and S_v in each step is at least $1/(\gamma m)$ and in m steps, is at least

$$1 - \left(1 - \frac{1}{\gamma m}\right)^m \geq 1 - e^{-1/\gamma} \geq \frac{1}{2\gamma},$$

where we have $1 + x \leq e^x$ for all $x \in R$ and the inequality $e^{-x} \leq 1 - x/2$ for $0 \leq x \leq 1$.

Then, after the cycle is created in the first m iterations, or the cycle already existed, the probability of the exclusion of such an edge in m steps of the second half of the epoch is only $(1 - e^{-3})n^{-1}$ because the probability of observing at least one edge from the cycle of length k in m steps is $1 - (1 - k/m)^m \geq 1 - (1 - 3/m)^m \geq 1 - e^{-3}$, and the probability that the edge selected is e equals

$1/n$. Altogether, the probability of excluding a non-essential edge with weight at least w is at least $(1-e^{-3})n^{-1}/(2\gamma)$, which results in decreasing X_t by at least one because removing e might also make some other edges essential. Since there are s non-essential edges, we have $\Delta_t(s) \geq s \cdot (1-e^{-3})n^{-1}/(2\gamma)$. Since there can be at most m essential edges at the beginning, we have $X_0 \leq m$. Assume Y denotes the number of epochs needed to have only essential edges with weight at least w. Using the upper tail bound of multiplicative drift in Theorem 8.2, we have

$$\Pr\left(Y > \frac{\ln(2m/\delta) + \ln X_0}{(1-e^{-3})n^{-1}/(2\gamma)}\right) \leq e^{-\ln(2m/\delta)} = \frac{\delta}{2m}.$$

Since each epoch consists of $2m$ iterations,

$$2m \cdot 2(1-e^{-3})^{-1} n\gamma \ln\left(2\tfrac{m^2}{\delta}\right) \leq 4.21\gamma mn \ln\left(2\tfrac{m^2}{\delta}\right)$$

is sufficient to arrive at a solution where all edges of weight at least w are essential. □

SA does with high probability not accept an inclusion of any edge via Lemma 8.6 when the temperature is colder than $w_{\min}/a$ for some a that is still a parameter chosen later. This is the time from when the solution is invariant. Let $t_{w_{\min}}$ be the earliest time when $T(w_{\min}) \leq w_{\min}/a$ and $t_{\text{end}} \coloneqq t_{w_{\min}}$.

In the following lemma, we show that there is a bijective relation between the edges of the solution at time t_{end} and a MST such that the ratio between the weights of corresponding edges is less than $(1+\kappa)$.

Lemma 8.8. *Let $\delta < 1$, $\gamma > 1$, $\ell = \omega(1)$ and $a \geq \ln(4(\ell-1)/\delta)$. Let*

$$1+\kappa = \frac{a \exp\left(\gamma \frac{4.21mn\ln(2m^2/\delta)}{\ell-1}\right)}{\ln \gamma}.$$

Consider a run of SA with multiplicative cooling schedule with $\beta = 1-1/\ell$ and $T_0 \geq w_{\max}$ on an instance of the MST problem. Assume that $\mathcal{T}^*$ is a minimum spanning tree and $\mathcal{T}'$ is the solution of SA at time t_{end} where $T(t_{\text{end}}) \leq w_{\min}/a$.

For an arbitrary spanning tree $\mathcal{T}$, let $w_{\mathcal{T}} = (w_{\mathcal{T}}(1), \ldots, w_{\mathcal{T}}(n-1))$ be a decreasingly sorted list of the weights on its edges, i. e., $w_{\mathcal{T}}(j) \geq w_{\mathcal{T}}(i)$ for all $1 \leq j \leq i \leq n-1$. With probability at least $1-\delta$, we have

$$w_{\mathcal{T}^*}(k) \leq w_{\mathcal{T}'}(k) < (1+\kappa) w_{\mathcal{T}^*}(k) \text{ for each } k \in [1..n-1].$$

Proof. We recall that t_w is the earliest point of time when $T(t_w) \leq w/a$. With probability $1-\delta/2$, edges of weight w are not included after their corresponding times t_w via Lemma 8.6. Thus conditional on this event, we can use Lemma 8.7 stating that with probability at least $1-\delta/(2m)$, the number of edges with weight at least w is at most $n_w - 1$. This condition must hold for at most m distinct values, happening with probability at least $1-\delta/2$ according to a union bound. Altogether, since the event in Lemma 8.6 must happen with probability $1-\delta/2$ and the condition in Lemma 8.7 must hold for all weights, with probability at least $1-\delta$, the statement in Lemma 8.7 is valid for all possible weights.

We use induction on the index k. The case $k = 0$ is trivial as the basic step. Regarding the inductive step, assume that for all $0 \leq k \leq i-1$, the inequality is valid. If $i = n$, the claim is proved. Otherwise, let $w_{\mathcal{T}^*}(i)$ be the next unique largest weight and j be the largest index that $w_{\mathcal{T}^*}(j) = w_{\mathcal{T}^*}(i)$. In fact, we have

$$w_{\mathcal{T}^*}(i-1) < w_{\mathcal{T}^*}(i) = \cdots = w_{\mathcal{T}^*}(j) < w_{\mathcal{T}^*}(j+1).$$

There are exactly $j-i+1$ edges with weight $w_{\mathcal{T}^*}(i)$ in the minimum spanning tree $\mathcal{T}^*$. The number of connected components in G using only edges at most $w_{\mathcal{T}^*}(i)$ is i since they are connected using $i-1$ edges in $\mathcal{T}^*$. Using Lemma 8.7 with $w = (1+\kappa)w_{\mathcal{T}^*}(i)$ and considering $n_w = i$, there are at most $i-1$ edges with weight at least $(1+\kappa)w_{\mathcal{T}^*}(i)$ in $\mathcal{T}'$, which means that the rest of the weight values in $\mathcal{T}'$ are less than $(1+\kappa)w_{\mathcal{T}^*}(i)$. Since we know that the graph cannot be connected using less than j edges with weight at least $w_{\mathcal{T}^*}(i)$, we can conclude that there are at least j edges with weight between $w_{\mathcal{T}^*}(i)$ and $(1+\kappa)w_{\mathcal{T}^*}(i)$. Therefore, for $i \leq k \leq j$, the inequality suggested above holds. □

With the above lemmas at hand, we can prove the first theorem. Given ℓ, Theorem 8.9 states the approximation ratio that the algorithm with cooling schedule $\beta = 1 - 1/\ell$ can obtain.

Theorem 8.9. *Let $\delta < 1$, $\gamma > 1$ and $\ell = \omega(1)$. Consider a run of SA with multiplicative cooling schedule with $\beta = 1-1/\ell$ and $T_0 \geq w_{\max}$ on an instance of the MST problem. For $a \geq \ln(4(\ell-1)/\delta)$, with probability at least $1-\delta$, at all times $t \geq (\ell/2)\ln(aT_0/w_{\min})$ the current solution is a $(1+\kappa)$-approximation where*

$$1+\kappa = \frac{a \exp\left(\gamma \frac{4.21mn\ln(2m^2/\delta)}{\ell-1}\right)}{\ln\gamma}.$$

Proof. We consider the time t_{end} when $T(t_{\text{end}}) \leq w_{\min}/a$ and show the approximation result for the current solution of SA at that time. Concretely, assume that $\mathcal{T}^*$ is a minimum spanning tree and $\mathcal{T}'$ is the solution of the algorithm at time t_{end}. Assume $w(\mathcal{T})$ is the total weight of edges in the tree $\mathcal{T}$. Using

Lemma 8.8, with probability $1-\delta$, we have $w_{\mathcal{T}'} < (1+\kappa)w_{\mathcal{T}^*}(k)$ for each $k \in [1..n-1]$. Thus, we have

$$w(\mathcal{T}') = \sum_{i=1}^{n-1} w_{\mathcal{T}'}(i) < \sum_{i=1}^{n-1} w_{\mathcal{T}^*}(i)(1+\kappa) = (1+\kappa)w(\mathcal{T}^*).$$

To complete the proof, we only have to find the time t_{end} from when the temperature is less than $w_{\min}/a$, so after that, no edges are included anymore via Lemma 8.6. Then t_{end} satisfies

$$T_0\left(1-\frac{1}{\ell}\right)^{t_{\text{end}}} = \frac{w_{\min}}{a}.$$

Then

$$t_{\text{end}} = \log_{1-1/\ell}\left(\frac{w_{\min}}{aT_0}\right) = \frac{\ln(w_{\min}/(aT_0))}{\ln(1-1/\ell)}.$$

Using the inequality $1 - x/2 \geq e^{-x}$ for $0 \leq x \leq 1$ with $x = 2/\ell$, we can bound t_{end} from above by

$$t_{\text{end}} \leq \frac{\ln(w_{\min}/(aT_0))}{-2/\ell} = \left(\frac{\ell}{2}\right)\ln\left(\frac{aT_0}{w_{\min}}\right). \qquad \square$$

The formula for κ, which we obtained in Theorem 8.9, holds for all $\gamma > 1$. In the following lemma, we suggest a value for γ, leading to the smallest value for $1+\kappa$. With the help of that, we give also some bounds on $1+\kappa$ considering different cases for ℓ.

Lemma 8.10. *Let κ be defined as in Theorem 8.9 and $T_{base} \coloneqq 4.21mn\ln(2m^2/\delta)$. Then the minimum value of κ is achieved by setting $\gamma = \exp\left(W\left(\frac{\ell-1}{T_{base}}\right)\right)$, where W is the Lambert W function. Moreover, if $\ell < eT_{base}+1$, $1+\kappa \geq e^{(1/e)-1}a$. Otherwise, if $\ell \geq eT_{base}+1$,*

$$1+\kappa \leq a\frac{\exp\left(\left(\ln\left(\frac{\ell-1}{T_{base}}\right)\right)^{\frac{e}{e-1}\ln^{-1}\left(\frac{\ell-1}{T_{base}}\right)-1}\right)}{\ln\left(\frac{\ell-1}{T_{base}}\right) - \ln\ln\left(\frac{\ell-1}{T_{base}}\right)}.$$

For $\ell = \omega(T_{base})$, the last fraction is $(1+o(1))\frac{a}{\ln(\ell-1)-\ln(T_{base})}$.

The proof of Lemma 8.10 uses the first derivative to find the minimum value for $1+\kappa$. This method gives us the minimum value

$$a\frac{e^{e^{W(b)}/b}}{W(b)}, \tag{8.1}$$

appearing at $\gamma = \exp(W(b))$, where $b = \frac{\ell-1}{T_{base}}$. By considering cases where $b \geq e$ and $b < e$ and using some inequalities on the Lambert W function, we obtain the results.

Proof of Lemma 8.10. From the definition of κ in Theorem 8.9, for $\gamma > 1$, we have

$$1 + \kappa = a\frac{e^{\gamma/b}}{\ln \gamma}, \tag{8.2}$$

where $b \coloneqq \frac{\ell-1}{T_{base}}$.

Let $f(x) = e^{x/b}/\ln x$ for $x > 1$. Then its derivative is $f'(x) = \frac{e^{x/b}}{b\ln(x)} - \frac{e^{x/b}}{x\ln^2(x)}$. For $x > 1$, we have the only root $x = e^{W(b)}$, where W is the Lambert W function. Therefore, Equation (8.2) with $\gamma = e^{W(b)}$ gives us the minimum value for $(1 + \kappa)$ and equals

$$a\frac{e^{e^{W(b)}/b}}{W(b)}. \tag{8.3}$$

Now, we aim at finding some bounds on $1 + \kappa$. We analyze Equation (8.3) for two cases of b.

For $b \geq e$, using the inequality

$$\ln b - \ln\ln b + \frac{\ln\ln b}{2\ln b} \leq W(b) \leq \ln b - \ln\ln b + \frac{e}{e-1}\frac{\ln\ln b}{\ln b},$$

from [HH08], we get

$$\begin{aligned} a\frac{e^{e^{W(b)}/b}}{W(b)} &\leq a\frac{\exp\left(b^{-1}e^{\ln(b)}e^{-\ln\ln b}e^{\frac{e}{e-1}\frac{\ln\ln b}{\ln b}}\right)}{\ln(b) - \ln\ln(b)} \\ &= a\frac{\exp\left(e^{-\ln\ln b}e^{\frac{e}{e-1}\frac{\ln\ln b}{\ln b}}\right)}{\ln(b) - \ln\ln(b)} \\ &= a\frac{\exp\left((\ln b)^{-1+\frac{e}{(e-1)\ln b}}\right)}{\ln(b) - \ln\ln(b)}. \end{aligned}$$

For $b = \omega(1)$, the last expression equals $\frac{a(1+o(1))}{\ln b - \ln\ln b} = (1 + o(1))\frac{a}{\ln b}$ since

$$(\ln b)^{-1+\frac{e}{(e-1)\ln b}} = \frac{e^{\frac{e\ln\ln b}{(e-1)\ln b}}}{\ln b} \leq \frac{1}{\ln b} = o(1).$$

Regarding the case $b < e$, using the definition $W(x)e^{W(x)} = x$, we have $e^{W(x)} = \frac{x}{W(x)}$. By applying these inequalities on Equation (8.3), we obtain

$$a\frac{e^{e^{W(b)}/b}}{W(b)} = a\frac{e^{\left(\frac{b}{bW(b)}\right)}}{W(b)} = a\frac{e^{1/W(b)}}{W(b)}.$$

From the definition again, we have $W(b)e^{W(b)} = b$. Since for $x \geq 0$, we have $e^x \geq 1$, we can conclude $W(b) \leq b$, resulting in $W(b) < e$. Thus the last expression can be bounded from below by

$$a\frac{e^{1/e}}{e} = e^{(1/e)-1}a. \qquad \square$$

Finally, we give the proofs of the two main theorems in this paper.

Proof of Theorem 8.3. Using Theorem 8.9, we have

$$1 + \kappa = \frac{a \exp\left(\gamma \frac{T_{base}}{\ell - 1}\right)}{\ln \gamma}.$$

By setting $a = \ln(4(\ell - 1)/\delta)$ and using the upper bound on $(1 + \kappa)$ obtained in Lemma 8.10 for $\ell = \omega(T_{base}) = \omega(mn \ln(m/\delta))$, we get

$$\begin{aligned} 1 + \kappa &\leq (1 + o(1))\frac{\ln(4(\ell - 1)/\delta)}{\ln(\ell - 1) - \ln(4.21mn \ln(2m^2/\delta))} \\ &= (1 + o(1)) \cdot (1 + o(1))\frac{\ln((\ell - 1)/\delta)}{\ln(\ell) - \ln(mn \ln(m/\delta))} \\ &\leq (1 + o(1))\frac{\ln(\ell/\delta)}{\ln(\ell) - \ln(mn \ln(m/\delta))}. \end{aligned} \qquad \square$$

In Theorem 8.3, we only consider the case $\ell = \omega(T_{base})$ since the other cases for ℓ cannot lead to constant approximation ratios and therefore are not interesting to study. More precisely, let us assume $\ell = \omega(1)$. In the case that $\ell < eT_{base} + 1$, we have the lower bound $\Omega(\ln(4(\ell - 1)/\delta)) = \omega(1)$ on $1 + \kappa$ from Lemma 8.10. Regarding the case that $\ell \geq eT_{base} + 1$ and $\ell = O(T_{base})$, it can be proved that $1 + \kappa = \Omega(a) = \omega(1)$, since $\ell/T_{base} = O(1)$ makes all terms constant except a in Equation (8.3). Then again for $a \geq \ln(4(\ell - 1)/\delta)$ and $\ell = \omega(1)$, the approximation ratio is $\omega(1)$.

Now, we give the proof of Theorem 8.4.

Proof of Theorem 8.4. Let $\ell = (mn\ln(m/\delta))^{1+1/\varepsilon}$. Via Theorem 8.3, we have

$$\begin{aligned}
1+\kappa &\leq (1+o(1))\frac{\ln(\ell/\delta)}{\ln\left(\frac{\ell}{mn\ln(m/\delta)}\right)} \\
&= (1+o(1))\frac{(1+1/\varepsilon)\ln\left(mn\ln(m/\delta)\right)+\ln(1/\delta)}{(1/\varepsilon)\ln\left(mn\ln(m/\delta)\right)} \\
&= (1+o(1))\left(\frac{1+1/\varepsilon}{1/\varepsilon}+\frac{\ln(1/\delta)}{(1/\varepsilon)\ln(mn\ln(m/\delta))}\right) \\
&\leq (1+o(1))\left(1+\frac{\ln(1/\delta)}{\ln(mn\ln(m/\delta))}\right)(1+\varepsilon)\,.
\end{aligned}$$

For $\delta^{-1} = o(mn\ln n)$, the last expression can be bounded from above by $(1 + o(1))\,(1+\varepsilon)$. □

A more straightforward result of Theorem 8.4 is stated in Corollary 8.5. In this corollary, we are aiming at expressing an asymptotic time for the algorithm to find the approximation, and we assume that ε is constant.

Proof of Corollary 8.5. Using Theorem 8.4, we will first prove the result for an approximation ratio of $(1 + o(1))(1 + \varepsilon')$ for some constant $\varepsilon' > 0$ and then bound this by a ratio of at most $1+\varepsilon$ such that $(1 + o(1))(1 + \varepsilon') \leq 1 + \varepsilon$ for n large enough.

Note that $\ell = (mn\ln(m/\delta))^{1+1/\varepsilon'}$ and $\delta = \omega(1/(mn\ln n))$ and invoke Theorem 8.4. The asymptotic bound on T^* is obtained in the following way: we note that $\ln(n/\delta) = O(\ln n)$ since $1/\delta = n^{O(1)}$ by assumption and $m \leq n^2$. Since $\varepsilon' > 0$ is constant, we have $\ln(\ell) = O((1+1/\varepsilon')\ln(mn\ln(\frac{m}{\delta}))) = O((1+1/\varepsilon')\ln n))) = O(\ln n)$. Moreover, $\ell = O((mn\ln(n/\delta))^{1+\varepsilon'}) = O((mn\ln(n))^{1+\varepsilon'})$. Putting this together, we have

$$T^* = O\left((mn\ln(n))^{1+1/\varepsilon'}\left(\ln\ln n + \ln\left(\frac{T_0}{w_{\min}}\right)\right)\right).$$

We have that $1/\varepsilon' = 1/\varepsilon + o(1)$ since ε and ε' are constants. Hence, we obtain the statement of the corollary. □

8.5 $(1+\varepsilon)$-separated weights

In this section, we revisit the case that the weights $w_1, \ldots, w_m$ are $(1+\varepsilon)$-*separated*, i. e., there is a constant $\varepsilon > 0$ such that $w_j \geq (1+\varepsilon)w_i$ if $w_j > w_i$

for all $i, j \in \{1, \ldots, n\}$. As mentioned in the introduction in Theorem 8.1, Wegener proves that SA with high probability finds an MST for any instance with $(1+\varepsilon)$-separated weights if $w_{\max} \leq 2^m$. More precisely, the proof of his theorem considers a time span of $O(m^{8+8/\varepsilon})$ steps and shows that SA constructs an MST within this time span with probability $1 - O(1/m)$.

In the following, we improve this result in two ways. As acknowledged by Wegener himself, he did not optimize the parameters in the final bound on the runtime. Therefore, we can give an improved time bound of

$$O((mn\ln(n))^{1+1/\varepsilon+o(1)}(\ln\ln n + \ln(T_0/w_{\min}))),$$

see Theorem 8.11 for the precise, more general result. Moreover, we replace the assumption on the largest edge weight by the parameter $w_{\max}$. Essentially, we have done all work necessary to show the following theorem already in the previous section, where we proved an approximation result. Now, the $(1+\varepsilon)$-separation implies that indeed an optimal solution is found with high probability.

Theorem 8.11. *Let $\delta = \omega(1/(mn\ln(m)))$ and $\delta < 1$, $\varepsilon > 0$ be a constant. Consider a run of SA with multiplicative cooling schedule with $\beta = 1 - 1/\ell$ for $\ell = (mn\ln(m/\delta))^{1+1/\varepsilon+o(1)}$ and $T_0 \geq w_{\max}$ on an instance of the MST problem with $(1+\varepsilon)$-separated weights. With probability at least $1-\delta$, at all times $t \geq T^* := (\ell/2)\ln\left(\frac{\ln(4(\ell-1)/\delta)T_0}{w_{\min}}\right)$ the current solution is optimal. Moreover,*

$$T^* = O\left((mn\ln(n))^{1+1/\varepsilon+o(1)}\left(\ln\ln n + \ln\left(\frac{T_0}{w_{\min}}\right)\right)\right).$$

Proof. We first prove the result for $(1+o(1))(1+\varepsilon')$-separated weights for some constant $\varepsilon' > 0$. Then we prove the result for $(1+\varepsilon)$-separated weights such that $(1+o(1))(1+\varepsilon') \leq (1+\varepsilon)$ for n large enough.

Using Lemma 8.8, with probability $1-\delta$, we have $w_{\mathcal{T}^*}(k) \leq w_{\mathcal{T}'}(k) < (1+\kappa)w_{\mathcal{T}^*}(k)$ for each $k \in [1..n-1]$. The $(1+\kappa)$-separated graphs do not have edge weight between $w_{\mathcal{T}^*}(k)$ and $(1+\kappa)w_{\mathcal{T}^*}(k)$ except $w_{\mathcal{T}^*}(k)$. Therefore, the algorithm finds an optimal solution.

We need to bound $1+\kappa$ using the assumptions in the statement. By setting $a = \ln(4(\ell-1)/\delta)$ and using Lemma 8.10 for $\ell = (mn\ln(m/\delta))^{1+1/\varepsilon'}$, we bound $1+\kappa$ from above by $(1+o(1))(1+\varepsilon')$ similarly to the proof of Theorem 8.4. Since ε and ε' are constants and we have $1/\varepsilon' = 1/\varepsilon + o(1)$, we obtain the claim for $(1+\varepsilon)$-separated weights.

Regarding T^*, since $\varepsilon > 0$ is constant, we have

$$\ln(\ell) = O((1+1/\varepsilon+o(1))\ln(mn\ln(m/\delta))) = O((1+1/\varepsilon+o(1))\ln n))) = O(\ln n).$$

Moreover, $\ell = O((mn\ln(n/\delta))^{1+\varepsilon+o(1)}) = O((mn\ln(n))^{1+\varepsilon+o(1)})$. Putting this together, we have

$$T^* = O\left((mn\ln(n))^{1+1/\varepsilon+o(1)}\left(\ln\ln n + \ln\left(\frac{T_0}{w_{\min}}\right)\right)\right). \qquad \square$$

8.6 Conclusions

We have shown that simulated annealing is a polynomial-time approximation scheme for the minimum spanning tree problem, thereby proving a conjecture by Wegener [Weg05]. Our analyses use state-of-the-art methods and have led to improved results in the case of $(1+\epsilon)$-separated weights, where simulated annealing yields an optimal solution with high probability. Our main result is one of the rare examples where simple randomized search heuristics, with a straightforward representation and objective function, serve as polynomial-time approximation scheme.

Since the runtime analysis of simulated annealing is still underrepresented in the theory of randomized search heuristics, our understanding of its working principles is still limited. In particular, we do not have a clear characterization of the fitness landscapes in which its non-elitism, along with a cooling schedule, is more efficient than global search. The study of the Metropolis Algorithm for the DLB problem in [WZD21] and our analysis on the minimum spanning tree problem might indicate that landscapes with many, but easy to leave local optima are beneficial; however, more research is needed to support this conjecture.

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 grant by the Independent Research Fund Denmark (DFF-FNU 8021-00260B).

Bibliography

[ABD20a] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. Fast mutation in crossover-based algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1268–1276. ACM, 2020.

[ABD20b] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. First steps towards a runtime analysis when starting with a good solution. In *Parallel Problem Solving From Nature, PPSN 2020, Part II*, pages 560–573. Springer, 2020.

[ABD21] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. Lazy parameter tuning and control: choosing all parameters randomly from a power-law distribution. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 1115–1123. ACM, 2021.

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

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

[AD21] Denis Antipov and Benjamin Doerr. A tight runtime analysis for the $(\mu+\lambda)$ EA. *Algorithmica*, 83:1054–1095, 2021.

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

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

[ADY19] Denis Antipov, Benjamin Doerr, and Quentin Yang. The efficiency threshold for the offspring population size of the (μ, λ) EA. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 1461–1469. ACM, 2019.

[AM16] Aldeida Aleti and Irene Moser. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys (CSUR)*, 49(3):1–35, 2016.

[AN21] Denis Antipov and Semen Naumov. The effect of non-symmetric fitness: the analysis of crossover-based algorithms on realjump functions. In *Foundations of Genetic Algorithms, FOGA 2021*, pages 1–15. ACM, 2021.

[Bäc93] Thomas Bäck. Optimal mutation rates in genetic search. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, Urbana-Champaign, IL, USA, June 1993*, pages 2–8. Morgan Kaufmann, 1993.

[BBD21a] Henry Bambury, Antoine Bultel, and Benjamin Doerr. Generalized jump functions. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 1124–1132. ACM, 2021.

[BBD21b] Riade Benbaki, Ziyad Benomar, and Benjamin Doerr. A rigorous runtime analysis of the 2-MMAS_{ib} on jump functions: ant colony optimizers can cope well with local optima. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 4–13. ACM, 2021.

[BBS21] AO Bassin, MV Buzdalov, and AA Shalyto. The "one-fifth rule" with rollbacks for self-adjustment of the population size in the $(1+(\lambda,\ \lambda))$ genetic algorithm. *Automatic Control and Computer Sciences*, 55(7):885–902, 2021.

[BDK16] Maxim Buzdalov, Benjamin Doerr, and Mikhail Kever. The unrestricted black-box complexity of jump functions. *Evolutionary Computation*, 24:719–744, 2016.

[BDN10] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Parallel Problem Solving from Nature, PPSN 2010*, pages 1–10. Springer, 2010.

[BGH+13] Edmund K. Burke, Michel Gendreau, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64:1695–1724, 2013.

[BLM13] Stéphane Boucheron, G'abor Lugosi, and Pascal Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. OUP Oxford, 2013.

[Che02] Shu-Heng Chen. *Evolutionary computation in economics and finance*, volume 100. Springer Science & Business Media, 2002.

[COY18] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. Fast artificial immune systems. In *Parallel Problem Solving from Nature, PPSN 2018, Part II*, pages 67–78. Springer, 2018.

[COY19] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. Artificial immune systems can find arbitrarily good approximations for the NP-hard number partitioning problem. *Artificial Intelligence*, 274:180–196, 2019.

[COY20] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. When hypermutations and ageing enable artificial immune systems to outperform evolutionary algorithms. *Theoretical Computer Science*, 832:166–185, 2020.

[COY21a] Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. Automatic adaptation of hypermutation rates for multimodal optimisation. In *Foundations of Genetic Algorithms, FOGA 2021*, pages 4:1–4:12. ACM, 2021.

[COY21b] Dogan Corus, Pietro S Oliveto, and Donya Yazdani. Fast immune system inspired hypermutation operators for combinatorial optimisation. *IEEE Transactions on Evolutionary Computation*, pages 956–970, 2021.

[DD16] Benjamin Doerr and Carola Doerr. The impact of random initialization on the runtime of randomized search heuristics. *Algorithmica*, 75:529–553, 2016.

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

[DD20] Benjamin Doerr and Carola Doerr. Theory of parameter control for discrete black-box optimization: provable performance gains through dynamic parameter choices. In Benjamin Doerr and Frank

Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 271–321. Springer, 2020. Also available at `https://arxiv.org/abs/1804.05650`.

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

[DDK18] Benjamin Doerr, Carola Doerr, and Timo Kötzing. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica*, 80:1732–1768, 2018.

[DDL21] Benjamin Doerr, Carola Doerr, and Johannes Lengler. Self-adjusting mutation rates with provably optimal success rules. *Algorithmica*, 83:3108–3147, 2021.

[DDY16a] Benjamin Doerr, Carola Doerr, and Jing Yang. k-bit mutation with self-adjusting k outperforms standard bit mutation. In *Parallel Problem Solving from Nature, PPSN 2016*, pages 824–834. Springer, 2016.

[DDY16b] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. In *Genetic and Evolutionary Computation Conference, GECCO 2016*, pages 1123–1130. ACM, 2016.

[DDY20] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. *Theoretical Computer Science*, 801:1–34, 2020.

[DEL21a] Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. Escaping local optima with non-elitist evolutionary algorithms. In *AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 12275–12283. AAAI Press, 2021.

[DEL21b] Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. Non-elitist evolutionary algorithms excel in fitness landscapes with sparse deceptive regions and dense valleys. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 1133–1141. ACM, 2021.

[DFK+16a] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S Krejca, Per Kristian Lehre, Pietro S Oliveto, Dirk Sudholt, and Andrew M Sutton. Emergence of diversity and its benefits for crossover in genetic algorithms. In *Parallel Problem Solving from Nature, PPSN 2016*, pages 890–900. Springer, 2016.

[DFK+16b] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima with diversity mechanisms and crossover. In *Genetic and Evolutionary Computation Conference, GECCO 2016*, pages 645–652. ACM, 2016.

[DFK+18] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro S. Oliveto, Dirk Sudholt, and Andrew M. Sutton. Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation*, 22:484–497, 2018.

[DFW10] Benjamin Doerr, Mahmoud Fouz, and Carsten Witt. Quasirandom evolutionary algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2010*, pages 1457–1464. ACM, 2010.

[DFW11] Benjamin Doerr, Mahmoud Fouz, and Carsten Witt. Sharp bounds by probability-generating functions and variable drift. In *Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 2083–2090. ACM, 2011.

[DG13] Benjamin Doerr and Leslie A. Goldberg. Adaptive drift analysis. *Algorithmica*, 65:224–250, 2013.

[DGWY19] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. The $(1 + \lambda)$ evolutionary algorithm with self-adjusting mutation rate. *Algorithmica*, 81:593–631, 2019.

[DHK12] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425:17–33, 2012.

[DHRW22] Benjamin Doerr, Taha El Ghazi El Houssaini, Amirhossein Rajabi, and Carsten Witt. How fast does the metropolis algorithm leave local optima? To be submitted to a conference, 2022.

[DJK08] Benjamin Doerr, Thomas Jansen, and Christian Klein. Comparing global and local mutations on bit strings. In *Genetic and Evolutionary Computation Conference, GECCO 2008*, pages 929–936. ACM, 2008.

[DJW00] Stefan Droste, Thomas Jansen, and Ingo Wegener. Dynamic parameter control in simple evolutionary algorithms. In *Foundations of Genetic Algorithms, FOGA 2000*, pages 275–294. Morgan Kaufmann, 2000.

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

[DJW12] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. *Algorithmica*, 64:673–697, 2012.

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

[DK18] Benjamin Doerr and Martin S. Krejca. Significance-based estimation-of-distribution algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 1483–1490. ACM, 2018.

[DK20] Benjamin Doerr and Martin S. Krejca. The univariate marginal distribution algorithm copes well with deception and epistasis. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP 2020*, pages 51–66. Springer, 2020.

[DKLL20] Benjamin Doerr, Timo Kötzing, J. A. Gregor Lagodzinski, and Johannes Lengler. The impact of lexicographic parsimony pressure for ORDER/MAJORITY on the run time. *Theoretical Computer Science*, 816:144–168, 2020.

[DL16a] Duc-Cuong Dang and Per Kristian Lehre. Runtime analysis of non-elitist populations: from classical optimisation to partial information. *Algorithmica*, 75:428–461, 2016.

[DL16b] Duc-Cuong Dang and Per Kristian Lehre. Self-adaptation of mutation rates in non-elitist populations. In *Parallel Problem Solving from Nature, PPSN 2016*, pages 803–813. Springer, 2016.

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

[DM13] Dipankar Dasgupta and Zbigniew Michalewicz. *Evolutionary algorithms in engineering applications*. Springer Science & Business Media, 2013.

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

[Doe19] Benjamin Doerr. A tight runtime analysis for the cGA on jump functions: EDAs can cross fitness valleys at no extra cost. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 1488–1496. ACM, 2019.

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

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

[Doe20c] Carola Doerr. Complexity theory for discrete black-box optimization heuristics. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 133–211. Springer, 2020. Also available at `https://arxiv.org/abs/1801.02037`.

[Doe21] Benjamin Doerr. Exponential upper bounds for the runtime of randomized search heuristics. *Theoretical Computer Science*, 851:24–38, 2021.

[DR22] Benjamin Doerr and Amirhossein Rajabi. Stagnation detection meets fast mutation. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pages 191–207. Springer, 2022.

[DRW22] Benjamin Doerr, Amirhossein Rajabi, and Carsten Witt. Simulated annealing is a polynomial-time approximation scheme for the minimum spanning tree problem. In *Genetic and Evolutionary Computation Conference, GECCO 2022*. ACM, 2022. To appear.

[DW18] Carola Doerr and Markus Wagner. Sensitivity of parameter control mechanisms with respect to their initialization. In *Parallel Problem Solving From Nature, PPSN 2018*, pages 360–372. Springer, 2018.

[DWY18] Benjamin Doerr, Carsten Witt, and Jing Yang. Runtime analysis for self-adaptive mutation rates. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 1475–1482. ACM, 2018.

[DYvR+18] Carola Doerr, Furong Ye, Sander van Rijn, Hao Wang, and Thomas Bäck. Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: Profiling $(1+\lambda)$ EA

variants on OneMax and LeadingOnes. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, page 951–958. ACM Press, 2018.

[DZ21] Benjamin Doerr and Weijie Zheng. Theoretical analyses of multi-objective evolutionary algorithms on multi-modal objectives. In *Conference on Artificial Intelligence, AAAI 2021*, pages 12293–12301. AAAI Press, 2021.

[EMV04] A. E. Eiben, Elena Marchiori, and V. A. Valkó. Evolutionary algorithms with on-the-fly population size adjustment. In *Parallel Problem Solving From Nature, PPSN 2004*, pages 41–50. Springer, 2004.

[ER60] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5(1):17–60, 1960.

[ER63] Paul Erdős and Alfréd Rényi. On two problems of information theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézet Közleményei*, 8:229–243, 1963.

[Faj19] Mario A. Hevia Fajardo. An empirical evaluation of success-based parameter control mechanisms for evolutionary algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 787–795. ACM Press, 2019.

[FC03] Gary B Fogel and David W Corne. *Evolutionary computation in bioinformatics*. Morgan Kaufmann, 2003.

[FGQW18a] Tobias Friedrich, Andreas Göbel, Francesco Quinzan, and Markus Wagner. Evolutionary algorithms and submodular functions: Benefits of heavy-tailed mutations. *CoRR*, abs/1805.10902, 2018.

[FGQW18b] Tobias Friedrich, Andreas Göbel, Francesco Quinzan, and Markus Wagner. Heavy-tailed mutation operators in single-objective combinatorial optimization. In *Parallel Problem Solving from Nature, PPSN 2018, Part I*, pages 134–145. Springer, 2018.

[FKK16a] Tobias Friedrich, Timo Kötzing, and Martin S. Krejca. EDAs cannot be balanced and stable. In *Genetic and Evolutionary Computation Conference, GECCO 2016*, pages 1139–1146. ACM, 2016.

[FKK+16b] Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Samadhi Nallaperuma, Frank Neumann, and Martin Schirneck. Fast building block assembly by majority vote crossover. In *Genetic and Evolutionary Computation Conference, GECCO 2016*, pages 661–668. ACM, 2016.

[FKKR22] Tobias Friedrich, Timo Kötzing, Martin S. Krejca, and Amirhossein Rajabi. Escaping local optima with local search: A theory-driven discussion. In *Parallel Problem Solving From Nature, PPSN 2022*. Springer, 2022. submitted to Parallel Problem Solving from Nature, PPSN 2022.

[FQW18] Tobias Friedrich, Francesco Quinzan, and Markus Wagner. Escaping large deceptive basins of attraction with heavy-tailed mutation operators. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 293–300. ACM, 2018.

[FS19] Alberto Franzin and Thomas Stützle. Revisiting simulated annealing: A component-based analysis. *Computers and Operations Research*, 104:191–206, 2019.

[FS21] Mario Alejandro Hevia Fajardo and Dirk Sudholt. Self-adjusting offspring population sizes outperform fixed parameters on the cliff function. In *Foundations of Genetic Algorithms, FOGA 2021*, pages 5:1–5:15. ACM, 2021.

[GKS99] Josselin Garnier, Leila Kallel, and Marc Schoenauer. Rigorous hitting times for binary mutations. *Evolutionary Computation*, 7:173–203, 1999.

[GW03] Oliver Giel and Ingo Wegener. Evolutionary algorithms and the maximum matching problem. In *Symposium on Theoretical Aspects of Computer Science, STACS 2003*, pages 415–426. Springer, 2003.

[Haj82] Bruce Hajek. Hitting and occupation time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 14:502–525, 1982.

[HFS21] Mario Alejandro Hevia Fajardo and Dirk Sudholt. Self-adjusting population sizes for non-elitist evolutionary algorithms: why success rates matter. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 1151–1159, 2021.

[HH08] Abdolhossein Hoorfar and Mehdi Hassani. Inequalities on the lambert w function and hyperpower function. *Journal of Inequalities in Pure & Applied Mathematics*, 9:5–9, 2008.

[HM18] Pierre Hansen and Nenad Mladenovic. Variable neighborhood search. In Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende, editors, *Handbook of Heuristics*, pages 759–787. Springer, 2018.

[Hor10] Christian Horoba. Exploring the runtime of an evolutionary algorithm for the multi-objective shortest path problem. *Evolutionary Computation*, 18(3):357–381, 2010.

[HS18] Václav Hasenöhrl and Andrew M. Sutton. On the runtime dynamics of the compact genetic algorithm on jump functions. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, pages 967–974. ACM, 2018.

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

[Jan11] Thomas Jansen. Simulated annealing. In Anne Auger and Benjamin Doerr, editors, *Theory of Randomized Search Heuristics*, pages 171–195. World Scientific Publishing, 2011.

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

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

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[JS98] Mark Jerrum and Gregory B. Sorkin. The metropolis algorithm for graph bisection. *Discrete Applied Mathematics*, 82:155–175, 1998.

[JS07] Jens Jägersküpper and Tobias Storch. When the plus strategy outperforms the comma strategy and when not. In *Foundations of Computational Intelligence, FOCI 2007*, pages 25–32. IEEE, 2007.

[JW00] Thomas Jansen and Ingo Wegener. On the choice of the mutation probability for the (1+1) EA. In *Parallel Problem Solving from Nature, PPSN 2000*, pages 89–98. Springer, 2000.

[JW02] Thomas Jansen and Ingo Wegener. The analysis of evolutionary algorithms – a proof that crossover really can help. *Algorithmica*, 34:47–66, 2002.

[JW04] Thomas Jansen and R. Paul Wiegand. The cooperative coevolutionary (1+1) EA. *Evolutionary Computation*, 12(4):405–434, 2004.

[JW07] Thomas Jansen and Ingo Wegener. A comparison of simulated annealing with a simple evolutionary algorithm on pseudo-Boolean functions of unitation. *Theoretical Computer Science*, 386:73–93, 2007.

[JZ14] Thomas Jansen and Christine Zarges. Performance analysis of randomised search heuristics operating with a fixed budget. *Theoretical Computer Science*, 545:39–58, 2014.

[KGJV83] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[KLLZ22] Marc Kaufmann, Maxime Larcher, Johannes Lengler, and Xun Zou. Self-adjusting population sizes for the (1,λ)-ea on monotone functions. *CoRR*, abs/2204.00531, 2022.

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

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

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

[Len20] Johannes Lengler. Drift analysis. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 89–131. Springer, 2020. Also available at `https://arxiv.org/abs/1712.00964`.

[LN19] Per Kristian Lehre and Phan Trung Hai Nguyen. On the limitations of the univariate marginal distribution algorithm to deception and where bivariate EDAs might help. In *Foundations of Genetic Algorithms, FOGA 2019*, pages 154–168. ACM, 2019.

[LO13] Per Kristian Lehre and Ender Özcan. A runtime analysis of simple hyper-heuristics: To mix or not to mix operators. In *Foundations of Genetic Algorithms, FOGA 2013*, pages 97–104. ACM, 2013.

[LOW19] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. On the time complexity of algorithm selection hyper-heuristics for multimodal optimisation. In *Conference on Artificial Intelligence, AAAI 2019*, pages 2322–2329. AAAI Press, 2019.

[LOW20] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. Simple hyper-heuristics control the neighbourhood size of randomised local search optimally for LeadingOnes. *Evolutionary Computation*, 28:437–461, 2020.

[LS11] Jörg Lässig and Dirk Sudholt. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *Foundations of Genetic Algorithms, FOGA 2011*, pages 181–192. ACM, 2011.

[LSW21] Johannes Lengler, Dirk Sudholt, and Carsten Witt. The complex parameter landscape of the compact genetic algorithm. *Algorithmica*, 83:1096–1137, 2021.

[Lug17] Michael Lugo. Sum of "the first k" binomial coefficients for fixed n. MathOverflow, 2017. (version: 2017-10-01), `https://mathoverflow.net/q/17236`.

[LW12] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. *Algorithmica*, 64:623–642, 2012.

[LW21] Per Kristian Lehre and Carsten Witt. Tail bounds on hitting times of randomized search heuristics using variable drift analysis. *Combinatorics, Probability & Computing*, 30(4):550–569, 2021.

[LY12] Per Kristian Lehre and Xin Yao. On the impact of mutation-selection balance on the runtime of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 16:225–241, 2012.

[MHF93] Melanie Mitchell, John Holland, and Stephanie Forrest. When will a genetic algorithm outperform hill climbing. *Advances in neural information processing systems*, 6:51–58, 1993.

[MRR+53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087–1092, 1953.

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

[NPW19] Frank Neumann, Mojgan Pourhassan, and Carsten Witt. Improved runtime results for simple randomised search heuristics on linear functions with a uniform constraint. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 1506–1514, 2019.

[NS20] Frank Neumann and Andrew M. Sutton. Parameterized complexity analysis of randomized search heuristics. In Benjamin Doerr and Frank Neumann, editors, *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, pages 213–248. Springer, 2020. Also available at `https://arxiv.org/abs/2001.05120`.

[NW07] Frank Neumann and Ingo Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378:32–40, 2007.

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

[OHY09] Pietro S. Oliveto, Jun He, and Xin Yao. Analysis of the (1+1)-EA for finding approximate solutions to vertex cover problems. *IEEE Transactions on Evolutionary Computation*, 13:1006–1029, 2009.

[OPH+18] Pietro S. Oliveto, Tiago Paixão, Jorge Pérez Heredia, Dirk Sudholt, and Barbora Trubenová. How to escape local optima in black box optimisation: when non-elitism outperforms elitism. *Algorithmica*, 80:1604–1633, 2018.

[PHST17] Tiago Paixão, Jorge Pérez Heredia, Dirk Sudholt, and Barbora Trubenová. Towards a runtime comparison of natural and artificial evolution. *Algorithmica*, 78:681–713, 2017.

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

[QGWF21] Francesco Quinzan, Andreas Göbel, Markus Wagner, and Tobias Friedrich. Evolutionary algorithms and submodular functions: benefits of heavy-tailed mutations. *Natural Computing*, 20:561–575, 2021.

[RA19] Jonathan E. Rowe and Aishwaryaprajna. The benefits and limitations of voting mechanisms in evolutionary optimisation. In *Foundations of Genetic Algorithms, FOGA 2019*, pages 34–42. ACM, 2019.

[RABD19] Anna Rodionova, Kirill Antonov, Arina Buzdalova, and Carola Doerr. Offspring population size matters when comparing evolutionary algorithms with self-adjusting mutation rates. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 855–863. ACM Press, 2019.

[RKJ06] Günther R. Raidl, Gabriele Koller, and Bryant A. Julstrom. Biased mutation operators for subgraph-selection problems. *IEEE Transaction on Evolutionary Computation*, 10(2):145–156, 2006.

[RLY09] Philipp Rohlfshagen, Per Kristian Lehre, and Xin Yao. Dynamic evolutionary optimisation: an analysis of frequency and magnitude of change. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 1713–1720. ACM Press, 2009.

[Rob95] Herbert Robbins. A remark on Stirling's formula. *The American Mathematical Monthly*, 62:26–29, 1995.

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

[RW20a] Amirhossein Rajabi and Carsten Witt. Evolutionary algorithms with self-adjusting asymmetric mutation. In *Parallel Problem Solving from Nature, PPSN 2020, Part I*, pages 664–677. Springer, 2020.

[RW20b] Amirhossein Rajabi and Carsten Witt. Self-adjusting evolutionary algorithms for multimodal optimization. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, pages 1314–1322. ACM, 2020.

[RW21a] Amirhossein Rajabi and Carsten Witt. Stagnation detection in highly multimodal fitness landscapes. In *Genetic and Evolutionary Computation Conference, GECCO 2021*, pages 1178–1186. ACM, 2021.

[RW21b] Amirhossein Rajabi and Carsten Witt. Stagnation detection with randomized local search. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP 2021*, pages 152–168. Springer, 2021.

[RW22a] Amirhossein Rajabi and Carsten Witt. Self-adjusting evolutionary algorithms for multimodal optimization. *Algorithmica*, pages 1–30, 2022. Preliminary version in GECCO 2020.

[RW22b] Amirhossein Rajabi and Carsten Witt. Stagnation detection in highly multimodal fitness landscapes. *Algorithmica*, 2022. Preliminary version in GECCO 2021, submitted to the Algorithmica journal.

[RW22c] Amirhossein Rajabi and Carsten Witt. Stagnation detection with randomized local search. *Evolutionary Computation*, 2022. Preliminary version in EvoCOP 2021, to appear in the EC journal.

[SH88] Galen H. Sasaki and Bruce Hajek. The time complexity of maximum matching by simulated annealing. *Journal of the ACM*, 35:387–403, 1988.

[Sud13] Dirk Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17:418–435, 2013.

[vLA87] Peter J. M. van Laarhoven and Emile H. L. Aarts. *Simulated Annealing: Theory and Applications*, volume 37 of *Mathematics and Its Applications*. Springer, 1987.

[War19] John Alasdair Warwicker. *On the runtime analysis of selection hyper-heuristics for pseudo-Boolean optimisation*. PhD thesis, University of Sheffield, UK, 2019.

[Weg01] Ingo Wegener. Theoretical aspects of evolutionary algorithms. In *Automata, Languages and Programming, ICALP 2001*, pages 64–78. Springer, 2001.

[Weg02] Ingo Wegener. Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In Ruhul Sarker, Masoud Mohammadian, and Xin Yao, editors, *Evolutionary Optimization*, pages 349–369. Kluwer, 2002.

[Weg05] Ingo Wegener. Simulated annealing beats Metropolis in combinatorial optimization. In *Automata, Languages and Programming, ICALP 2005*, pages 589–601. Springer, 2005.

[Wit03] Carsten Witt. Population size vs. runtime of a simple EA. In *Congress on Evolutionary Computation, CEC 2003*, volume 3, pages 1996–2003. IEEE Press, 2003.

[Wit05] Carsten Witt. Worst-case and average-case approximations by simple randomized search heuristics. In *Symposium on Theoretical Aspects of Computer Science, STACS 2005*, pages 44–56. Springer, 2005.

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

[Wit08] Carsten Witt. Population size versus runtime of a simple evolutionary algorithm. *Theoretical Computer Science*, 403:104–120, 2008.

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

[Wit21] Carsten Witt. On crossing fitness valleys with majority-vote crossover and estimation-of-distribution algorithms. In *Foundations of Genetic Algorithms, FOGA 2021*, pages 2:1–2:15. ACM, 2021.

[WQT18] Mengxi Wu, Chao Qian, and Ke Tang. Dynamic mutation based Pareto optimization for subset selection. In *Intelligent Computing Methodologies, ICIC 2018, Part III*, pages 25–35. Springer, 2018.

[WVHM18] Darrell Whitley, Swetha Varadarajan, Rachel Hirsch, and Anirban Mukhopadhyay. Exploration and exploitation without mutation: solving the jump function in $\Theta(n)$ time. In *Parallel Problem Solving from Nature, PPSN 2018, Part II*, pages 55–66. Springer, 2018.

[WW05] Ingo Wegener and Carsten Witt. On the optimization of monotone polynomials by simple randomized search heuristics. *Combinatorics, Probability & Computing*, 14:225–247, 2005.

[WZD21] Shouda Wang, Weijie Zheng, and Benjamin Doerr. Choosing the right algorithm with hints from complexity theory. In *International Joint Conference on Artificial Intelligence, IJCAI 2021*, pages 1697–1703. ijcai.org, 2021.

[YDB19] Furong Ye, Carola Doerr, and Thomas Bäck. Interpolating local and global search by controlling the variance of standard bit mutation. In *Congress on Evolutionary Computation, CEC 2019*, pages 2292–2299, 2019.