Toward a Complexity Theory for Randomized Search Heuristics: Black-Box Models

Dissertation

zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

vorgelegt von

Carola Winzen

Saarbrücken
2011

Tag des Kolloquiums:
16. Dezember 2011

Dekan der Naturwissenschaftlich-Technischen Fakultät I:
Prof. Dr. Holger Hermanns
Universität des Saarlandes, Saarbrücken, Deutschland

Prüfungsausschuss:
Prof. Dr. Raimund Seidel (Vorsitzender des Prüfungsausschusses)
Universität des Saarlandes, Saarbrücken, Deutschland
Prof. Dr. Dr. h.c. mult. Kurt Mehlhorn
Max-Planck-Institut für Informatik, Saarbrücken, Deutschland
Prof. Dr. Markus Bläser
Universität des Saarlandes, Saarbrücken, Deutschland
Prof. Dr. Xin Yao
University of Birmingham, Birmingham, U.K.
Dr. Konstantinos Panagiotou (Akademischer Beisitzer)
Max-Planck-Institut für Informatik, Saarbrücken, Deutschland

Berichterstatter:
Prof. Dr. Dr. h.c. mult. Kurt Mehlhorn
Max-Planck-Institut für Informatik, Saarbrücken, Deutschland
Prof. Dr. Markus Bläser
Universität des Saarlandes, Saarbrücken, Deutschland
Prof. Dr. Xin Yao
University of Birmingham, Birmingham, U.K.

Abstract

Randomized search heuristics are a broadly used class of general-purpose algorithms. Analyzing them via classical methods of theoretical computer science is a growing field. While several strong runtime bounds exist, a powerful complexity theory for such algorithms is yet to be developed.

We contribute to this goal in several aspects. In a first step, we analyze existing black-box complexity models. Our results indicate that these models are not restrictive enough. This remains true if we restrict the memory of the algorithms under consideration.

These results motivate us to enrich the existing notions of black-box complexity by the additional restriction that not actual objective values, but only the relative quality of the previously evaluated solutions may be taken into account by the algorithms. Many heuristics belong to this class of algorithms. We show that our ranking-based model gives more realistic complexity estimates for some problems, while for others the low complexities of the previous models still hold.

Surprisingly, our results have an interesting game-theoretic aspect as well. We show that analyzing the black-box complexity of the $\textsc{OneMax}_n$ function class—a class often regarded to analyze how heuristics progress in easy parts of the search space—is the same as analyzing optimal winning strategies for the generalized Mastermind game with 2 colors and length-n codewords. This connection was seemingly overlooked so far in the search heuristics community.

Zusammenfassung

Randomisierte Suchheuristiken sind vielseitig einsetzbare Algorithmen, die aufgrund ihrer hohen Flexibilität nicht nur im industriellen Kontext weit verbreitet sind. Trotz zahlreicher erfolgreicher Anwendungsbeispiele steckt die Laufzeitanalyse solcher Heuristiken noch in ihren Kinderschuhen. Insbesondere fehlt es uns an einem guten Verständnis, in welchen Situationen problemunabhängige Heuristiken in kurzer Laufzeit gute Lösungen liefern können. Eine Komplexitätstheorie ähnlich wie es sie in der klassischen Algorithmik gibt, wäre wünschenswert.

Mit dieser Arbeit tragen wir zur Entwicklung einer solchen Komplexitätstheorie für Suchheuristiken bei. Wir zeigen anhand verschiedener Beispiele, dass existierende Modelle die Schwierigkeit eines Problems nicht immer zufriedenstellend erfassen. Wir schlagen daher ein weiteres Modell vor. In unserem Ranking-Based Black-Box Model lernen die Algorithmen keine exakten Funktionswerte, sondern bloß die Rangordnung der bislang angefragten Suchpunkte. Dieses Modell gibt für manche Probleme eine bessere Einschätzung der Schwierigkeit. Wir zeigen jedoch auch, dass auch im neuen Modell Probleme existieren, deren Komplexität als zu gering einzuschätzen ist.

Unsere Ergebnisse haben auch einen spieltheoretischen Aspekt. Optimale Gewinnstrategien für den Rater im Mastermindspiel (auch SuperHirn) mit n Positionen entsprechen genau optimalen Algorithmen zur Maximierung von $\textsc{OneMax}_n$-Funktionen. Dieser Zusammenhang wurde scheinbar bislang übersehen.

Diese Arbeit ist in englischer Sprache verfasst.

Acknowledgments

It is a great honor to work in such a lively, inspiring, and outstanding place as the Algorithms and Complexity group at the Max-Planck-Institut für Informatik. I am very grateful to Kurt Mehlhorn for admitting me to this great research environment.

I am greatly indebted to Benjamin Doerr, who advised most of the work contained in this thesis.

During my PhD research, I had the great pleasure of working with many people, among others my co-authors Benjamin Doerr, Michael Gnewuch, Thomas Jansen, Daniel Johannsen, Timo Kötzing, Per Kristian Lehre, Johannes Lengler, Dirk Sudholt, Markus Wagner, Magnus Wahlström, and Christine Zarges. I would like to thank all of them. I also had many interesting discussions with Konstantinos Panagiotou. He taught me many useful facts about randomized algorithms.

My work is supported by a Google Europe Fellowship in Randomized Algorithms, which I gratefully acknowledge. I also received financial support from the ACM special interest group SIGEVO for attending GECCO 2010 and from the Alfréd Rényi Institute of Mathematics for attending the conference in honors of Endre Szemerédi's seventieth birthday. Both conferences had a significant impact on my PhD studies.

I also thank my employer, McKinsey&Company, Inc. The educational leave offered by the Fellow program of the German office allowed me to pursue my PhD after two very insightful and inspiring years as a business consultant.

Parts of Section 6 have been done during the Dagstuhl seminar 10361 "Theory of Evolutionary Algorithms". I would like to thank the organizers for inviting me in a very early stage of my PhD studies.

I would also like to express my gratitude to several anonymous reviewers. Their comments significantly helped to improve the presentation of the results presented in the papers underlying this dissertation.

I am most grateful to my father for his steady support.

Contents

1 Introduction

Randomized search heuristics are general purpose algorithms for optimization problems. They include bio-inspired approaches such as evolutionary algorithms and ant colony optimization, but also classical approaches like random search or randomized hill-climbers.

In practice, randomized search heuristics often are surprisingly successful (and thus extensively used). They have the additional advantage that not too much understanding of the optimization problem at hand is needed, and that once implemented, they can easily be re-used for similar problems.

One of the difficulties in using such heuristics is that it is very hard to predict which problems are easy for a suitable heuristic and which are generally intractable for randomized search heuristics. In contrast to a large body of empirical work on this problem, there has been much less theoretical work.

This work mostly lead to results for particular problems and particular heuristics. Droste, Jansen, and Wegener [DJW02] determined the runtime of the $(1+1)$ evolutionary algorithm (EA) for several important test function classes. Another example is the work by Neumann and Wegener [NW07], which shows that the $(1+1)$ EA finds a minimum spanning tree using $O(m^2 \log m)$ function evaluations in connected graphs having m edges and polynomially bounded edge weights.

Still, for a broader understanding of what are easy and difficult problems, a complexity theory similar to what exists in classical algorithmics would be highly desirable also for randomized search heuristics. The seminal paper by Droste, Jansen, and Wegener [DJW06], introducing the so-called *unrestricted black-box model*, appears to be the first attempt to start such a complexity theory in the randomized search heuristics community.

The paradigm that randomized search heuristics should ideally be problem-independent implies that the only way such a heuristic can obtain problem-specific information is by evaluating a solution candidate. This evaluation is done by an oracle that returns the objective value, but reveals no further information about the objective function. An algorithm that has no access to the objective function (and thus has no access to the optimization problem to be solved) other than by querying objective values from such an oracle, is called a *black-box algorithm*.

For a class of functions $\mathcal{F}$, Droste et al. define the unrestricted black-box complexity of $\mathcal{F}$ to be the minimum (taken over all black-box algorithms) expected number of function evaluations needed to optimize every function $f \in \mathcal{F}$ (minimum worst-case runtime). This number, naturally, is a lower bound on the runtime of any randomized search heuristic for the class $\mathcal{F}$, including evolutionary algorithms, ant colony approaches, simulated annealing, et cetera.

Unrestricted black-box complexity is also studied under the notion of *(randomized) query complexity*. Many results for a variety of problems exist. Out of the many examples let us mention the problem of finding a local minimum of an unknown pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$. This problem has been studied intensively in the computer science literature, for deterministic algorithms (deterministic black-box complexity; confer, e.g., the work by Llewellyn, Tovey, and Trick [LTT89]) and for randomized algorithms, for example by Aldous [Ald83], Aaronson [Aar04], and by Zhang [Zha06]. Zhang gives a tight $\Theta(2^{n/2}\sqrt{n})$ bound for the randomized query complexity of finding a local minimum.

Originally motivated by the coin-weighing problem, a much earlier work studying the unrestricted black-box complexity of the generalized OneMax function class (definition follows) is the one by Erdős and Rényi [ER63], cf. Sections 2.3 and 7. This OneMax function class is also strongly related to the well-known Mastermind game, a game that has gained much attention from the computer science and mathematics community. For example, Chvátal [Chv83] studies a generalized version of this game with k colors and n positions. That is, the secret code is a length-n string $z \in \{0,1,\ldots,k-1\}^n$. Chvátal shows that for any constant number k of colors the codebreaker has a strategy revealing the secret code using only $\Theta(n/\log n)$ guesses. In our notation this result is equivalent to saying that the unrestricted black-box complexity of the generalized OneMax function class is $\Theta(n/\log n)$. The connection between unrestricted black-box complexity and randomized query complexity was seemingly overlooked so far in the randomized search heuristics community. Note, however, that the main focus of the latter works typically is not in understanding the performance of randomized search heuristics.

Unfortunately, it turned out that regarding all black-box algorithms leads to sometimes unexpectedly small complexities, which are obtained by not very sensible algorithms. As a trivial example, note that the unrestricted black-box complexity of any class of functions $\mathcal{F} = \{f\}$ consisting of a single objective function, is one. This is certified by the black-box algorithm that simply queries the optimum of f as first action.

This and further examples suggest that a restriction of the class of algorithms regarded might lead to more meaningful results. A major step in this direction is the work by Lehre and Witt [LW10a] (see also [LW10b]). They introduce a so-called *unbiased black-box model*, which, among other restrictions to the class of algorithms regarded, requires that all search points queried by the algorithm must be obtained from previous or random search points by so-called *unbiased variation operators* (see Section 3.2 for the full details). When only unary operators are allowed, this leads to a lower bound of $\Omega(n \log n)$ for the complexity of any single-element class $\mathcal{F} = \{f\}$ with f having a unique global optimum. This is, indeed, the typical runtime of simple search heuristics like randomized hill-climbers on simple function classes like monotone functions.

To allow comparisons between the unrestricted black-box model and the unbiased black-box model, it is often of interest to study the unrestricted black-box complexity of a class $\mathcal{F}$ of functions that contains generalized versions of the original test function. This is a standard approach in black-box optimization. As an example, let us consider the function $\textsc{Om} : \mathbb{R} \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} x_i$ which counts the number of ones in a bit string ($\textsc{Om}$ abbreviates $\textsc{OneMax}$). As pointed out above, this function clearly has an unrestricted black-box complexity of 1. However, since the string $(1 \dots 1)$ is the unique global optimum of $\textsc{Om}$, the result by Lehre and Witt implies a $\Omega(n \log n)$ lower bound for the unary unbiased black-box complexity of $\textsc{Om}$. This bound is tight. Indeed, a matching upper bound is provided by many search heuristics, e.g., Random Local Search or the (1+1) evolutionary algorithm (cf. Section 2.2 for the definitions). Due to a coupon collector effect they need, on average, $\Theta(n \log n)$ function evaluations to optimize $\textsc{Om}$.

As a moment of thought reveals, the $\Theta(n \log n)$ bound remains correct if we do not ask for the complexity of the single function $\textsc{Om}$ but if we consider the whole class $\textsc{OneMax}_n$ of generalized $\textsc{Om}$ functions: For all $z \in \{0,1\}^n$ let

$$\textsc{Om}_z : \{0,1\}^n \to [0..n], x \mapsto |\{i \in [n] \mid x_i = z_i\}| \,.$$

That is, $\textsc{Om}_z(x)$ counts the number of bit positions in which x and z coincide. Let $\textsc{OneMax}_n := \{\textsc{Om}_z \mid z \in \{0,1\}^n\}$ be the collection of all such $\textsc{OneMax}$-type functions.

Then the unary black-box complexity of $\textsc{OneMax}_n$ remains $\Theta(n \log n)$. However, the unrestricted black-box complexity of $\textsc{OneMax}_n$ increases to $\Theta(n / \log n)$ when augmenting $\mathcal{F} = \{\textsc{Om}\}$ to $\mathcal{F} = \textsc{OneMax}_n$.

As we did in this example, in the following, we shall always consider not one single objective function but instead we consider classes of generalized functions with isomorphic fitness landscapes.

We are now ready to give an overview over the main contributions of our work. This thesis has three parts. In the first part we study the unrestricted and the unbiased black-box models and we present several new results, cf. Section 1.1. The results indicate that even the unbiased model is still not restrictive enough, in the sense that it allows for black-box complexities that are much smaller than one would expect. We study two alternative black-box notions in the second part, a *memory-restricted* version of the unrestricted model and *ranking-based versions* of both the unrestricted as well as the unbiased black-box models. Section 1.2 provides an overview of the main results achieved in Part II. We conclude this thesis in Part III with a study of black-box complexities for combinatorial problems.

Table 1.1 summarizes the black-box models and the problems studied in this thesis.

Model	Basic		Memory-Restricted		Ranking-Based	
	Problem	Sec.	Problem	Sec.	Problem	Sec.
unrestricted					$\textsc{OneMax}_n$	8
	$\textsc{LeadingOnes}^*_n$	5	$\textsc{OneMax}_n$	7	$\textsc{BinaryValue}^*_n$	8
	MST	9	Mastermind	7	$\textsc{LeadingOnes}^*_n$	8
	SSSP	10			MST	9
					SSSP	10
unbiased	$\textsc{OneMax}_n$	4			$\textsc{OneMax}_n$	8
	$\textsc{LeadingOnes}^*_n$	4,5			$\textsc{BinaryValue}^*_n$	8
	$\textsc{Jump}_{n,k}$	6			$\textsc{LeadingOnes}^*_n$	8
	$\textsc{Partition}$	6			MST	9
	MST	9			SSSP	10
	SSSP	10				

Table 1.1. Black-box models and problems studied in this thesis. Abbreviations: Sec. = Section, MST = minimum spanning tree problem, SSSP = single-source shortest paths problem

1.1. New Results for the Basic Black-Box Models

As mentioned above, we shall first argue that the unbiased model of Lehre and Witt is still not restrictive enough. To this end, we provide several examples for which the unbiased black-box complexities are much smaller than what problem-independent search heuristics suggest.

In what follows, we give a rough overview over the three sections covered in this first part of the thesis.

Section 4: Faster Black-Box Algorithms Through Higher Arity Operators

Lehre and Witt [LW10a] analyze the unbiased black-box model for the *unary case* where algorithms may use only mutation-type operators, i.e., may only use the information from at most one previously queried search point to generate a new sample.

In Section 4 we extend their work by analyzing the unbiased black-box model for arities greater than ones. A variation operator is said to be of arity k if it creates new search points by recombining up to k previously queried search points. An algorithm invoking only variation operators of arity at most k is called a k-ary algorithm. We analyze the higher arity black-box complexities of the two standard function classes $\textsc{OneMax}_n$ and $\textsc{LeadingOnes}^*_n$ (definition follows).

We show that, surprisingly, by allowing binary (i.e., crossover-type) variation operators, the unbiased black-box complexity of $\textsc{OneMax}_n$ drops from $\Theta(n \log n)$ to $O(n)$.

We shall also prove that the black-box complexity drops further if we increase the arity of the variation operators. More precisely, we show that for every $k \leq n$, the unbiased k-ary black-box complexity of $\textsc{OneMax}_n$ can be bounded by $O(n/\log k)$. For $k = n^{\Omega(1)}$ this statement is asymptotically optimal since already for the unrestricted black-box complexity a lower bound of $\Omega(n/\log n)$ has been shown by Erdős and Rényi [ER63] and independently by Chvátal [Chv83] and again later also by Droste et al. [DJW06].

The other class of test functions that we shall consider in Section 6 is $\textsc{LeadingOnes}_n^*$. This class contains the generalizations of the standard LeadingOnes function $\textsc{Lo} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\}$, which counts the longest prefix of ones in a bit string. We consider the closure of Lo under all permutations $\sigma \in S_n$ and under all exchanges of the bit values 0 and 1. To be more precise, we define for any bit string $z \in \{0,1\}^n$ and any permutation σ of $[n]$ the function

$$\textsc{Lo}_{z,\sigma} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : z_{\sigma(j)} = x_{\sigma(j)}\}\,.$$

We let $\textsc{LeadingOnes}_n^*$ be the set $\{\textsc{Lo}_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\}$ of all such functions. Each of these functions has a fitness landscape that is isomorphic to the one induced by Lo.

We show that the $\Theta(n^2)$ unary unbiased black-box complexity of $\textsc{LeadingOnes}_n^*$ proven in [LW10a] drops to $O(n \log n)$ in the binary unbiased black-box model.

The results presented in this section are the first results for unbiased black-box models of arity larger than one.

Indication of source. The content of Section 4 has been previously published in the *Proceedings of FOGA '11* (cf. [DJK+11]).

Section 5: Breaking the $O(n \log n)$ Barrier of LeadingOnes

Building on the results of the Section 4 we show that the unrestricted black-box complexity of $\textsc{LeadingOnes}_n^*$ is $O(n \log n/\log\log n)$. This shows that the natural looking $O(n \log n)$ bound proven in Section 4 is not tight in the unrestricted model.

Moreover, we show that the black-box optimization algorithm leading to this bound can be implemented in a way that only 3-ary unbiased variation operators are used. Hence our bound implies that the unbiased black-box complexity of $\textsc{LeadingOnes}_n^*$ is $O(n \log n/\log\log n)$ for all arities larger than two.

The upper bound remains valid if we do not allow the algorithm to exploit knowledge of the *exact* function values but only their rankings. That is, we show that even the 3-ary unbiased ranking-based black-box complexity is $O(n \log n/\log\log n)$. This disproves a conjecture by Droste, Jansen, Tinnefeld, and Wegener made in the conference version [DJTW03] of [DJW06].

Indication of source. The results presented in Section 5 have been accepted for presentation at the conference *Artificial Evolution '11* (cf. [DW11a]).

Section 6: Too Fast Unary Unbiased Black-Box Algorithms

The results presented in the previous two sections indicate that the unbiased black-box complexities remain artificially small for several test functions once we increase the arity of the variation operators to values larger than one. In this section we show that the same effect occurs already in the unary unbiased model, where only mutation-type variation operators are allowed. More precisely, for both the classical $\textsc{Jump}_{n,k}$ test function class and for a subclass of the well-known $\textsc{Partition}$ problem we give mutation-only unbiased black-box algorithms with an expected optimization time of $O(n \log n)$ function evaluations. Since the first problem usually needs $\Theta(n^k)$ function evaluations to be optimized by standard heuristics and the second is even NP-hard, these black-box complexities do not seem to indicate the true difficulty of the two problems for randomized search heuristics.

Indication of source. Parts of the content of Section 6 have been previously published in the *Proceedings of GECCO '11* (cf. [DKW11]).

1.2. Alternative Black-Box Models

In the second part we analyze two alternative black-box notions.

Section 7: Memory-Restricted Black-Box Models

The first alternative model, the *memory-restricted black-box model*, restricts the memory of the algorithm to some fixed size $\mu \in \mathbb{N}$. Given such a memory of size μ, the algorithm can store up to μ search points and their corresponding objective values. Based *only* on this information, it decides on its next sample. After receiving that sample's objective value, based only on the content of the memory, the current sample, and its objective value, it decides which μ out of the $\mu+1$ samples and objective values to keep in the memory. Note that our memory restriction means that the algorithm truly has no other memory, in particular, no iteration or program counters. So, formally, a black-box algorithm with memory of size μ consists of a guessing strategy, which can be fully described by a mapping from μ-sets of samples and objective values to search points $x \in \{0,1\}^n$, and a forgetting strategy, which maps $(\mu+1)$-sets of samples and objective values to μ-subsets thereof.

The memory-restriction is natural in the sense that many heuristics, e.g., evolutionary algorithms, only store a bounded size *population* of search points. Simple hill-climbers or the Metropolis algorithm even store only one single search point.

The memory-restricted black-box model was suggested by Droste, Jansen, and Wegener. They conjecture in [DJW06, Section 6] that a memory restriction of size one leads to a black-box complexity of order $\Theta(n \log n)$ for the $\textsc{OneMax}_n$ function class. However, as we shall prove in Section 7, this is not correct. In fact, we show that the black-box complexity of $\textsc{OneMax}_n$ even with the memory restricted to one is $\Theta(n/\log n)$, disproving the conjecture of Droste, Jansen, and Wegener.

Indication of source. The content of Chapter 7 is currently under submission. Minor parts of it are available as technical report [DW11b].

Section 8: Ranking-Based Black-Box Models

In Section 8 we enrich the existing black-box complexity notions by the restriction that not actual objective values, but only the relative quality of the previously evaluated solutions may be taken into account by the algorithm. In other words, throughout the optimization, the algorithm knows for any two already queried search points x and y no more than whether $f(x) < f(y)$, $f(x) = f(y)$, or $f(x) > f(y)$. In particular, it does not know the true values of $f(x)$ and $f(y)$. Still, this *ranking-based black-box model* captures many commonly used randomized search heuristics.

We show that our proposed model removes some drawbacks of the previous models. For example, for the class of generalized binary-value functions $\textsc{BinaryValue}^*_n$ (cf. Section 2.3 for the definition), the ranking-based black-box complexity is of order $\Theta(n)$ instead of only $O(\log n)$ without the ranking restriction. That is, we see that for this function class, the ranking-based black-box complexity seems to give us a more useful complexity measure than the previous approaches.

However, we shall also analyze the ranking-based black-box complexity of the $\textsc{OneMax}_n$ function class for which all previous black-box models showed a complexity of $\Theta(n/\log n)$. The proofs of these results heavily exploit the fact that in these models, the oracle returns the precise fitness value. In spite of this, we still find a black-box algorithm in our ranking-based model that solves the problem with $\Theta(n/\log n)$ queries. This might be an indicator that for some problem classes the ranking-based model still allows too powerful algorithms.

Indication of source. Parts of the content of Chapter 8 have been previously published in the *Proceedings of CSR '11* (cf. [DW11c]). A journal version is currently in preparation.

1.3. Black-Box Complexities of Combinatorial Problems

In the third part of this thesis, we shall leave the world of the artificial test functions OneMax, BinaryValue, and LeadingOnes, and we analyze the different black-box complexities of two combinatorial problems, the minimum spanning tree problem (Section 9) and the single-source shortest paths problem (Section 10).

Besides giving interesting bounds for their black-box complexities, our work reveals that the choice of how to model the optimization problem is non-trivial here. This in particular comes true for the shortest paths problem where the search space does not consist of bit strings, and where a reasonable definition of unbiasedness has to be agreed on.

Indication of source. Parts of the content of Section 9 and Section 10 have been previously published in the *Proceedings of GECCO '11* (cf. [DKLW11]). A journal version is currently in preparation.

1.4. Further Contributions

This thesis summarizes my contribution to the development of a complexity theory for randomized search heuristics. However, during my PhD research, I have not only worked with black-box models, but I achieved also other results which are summarized in Appendix A.

In addition to three different results in the field of evolutionary computation (cf. references [DJS$^+$10], [DJW10b], and [DJW10a]), these contributions include

- [GWW11]: a new search heuristic for the NP-hard problem [GSW09] of computing the star discrepancy of a given point set [GWW11], and
- [Win11]: the definition and analysis of a direction-reversing quasi-random rumor spreading protocol, which—given the same dose of randomness—outperforms the recent hybrid protocol by Doerr and Fouz [DF11].

Journal versions are under submission for all five results mentioned. Note, however, that these topics and the results are not further discussed in this thesis.

2 Preliminaries

We give an overview of the most intensively studied general-purpose search heuristics (Section 2.2) and we introduce some terminology frequently used in the randomized search heuristics community.

Furthermore, we define several standard test function classes in Section 2.3. Typically, these function classes are generalizations of toy problems which are used to gather some basic knowledge on how randomized search heuristics proceed on very elementary fitness landscapes.

We conclude this part in Section 2.4 with some standard tools that will be used later in this thesis.

2.1. Basic Notation

Throughout this work, we use the following notations. We denote the positive integers by $\mathbb{N}$ and the positive reals by $\mathbb{R}^+$. For any $k \in \mathbb{N}$, we abbreviate $[k] := \{1, \ldots, k\}$. Analogously, we define $[0..k] := [k] \cup \{0\}$. For $k, \ell \in \mathbb{N}$ we write $[k \pm \ell] := [k-\ell, k+\ell] \cap \mathbb{Z}$. For $x = x_1 \cdots x_n \in \{0,1\}^n$ we denote by $\bar{x}$ the bitwise complement of x (i.e., for all $i \in [n]$ we have $\bar{x}_i = 1 - x_i$).

If $x, y \in \{0,1\}^n$, we obtain the bit string $x \oplus y$ by setting, for each $j \in [n]$, $(x \oplus y)_i := 1$ if $x_i \neq y_i$ and $(x \oplus y)_i := 0$ if $x_i = y_i$. That is, $\oplus$ denotes the bitwise exclusive-or.

By e_k^n we denote the k-th unit vector $(0, \ldots, 0, 1, 0, \ldots, 0)$ of length n. For a set $I \subseteq [n]$ we abbreviate $e_I^n := \sum_{i \in I} e_i^n = \oplus_{i \in I} e_i^n$.

We shall sometimes use the shorthand $|x|_1$ for the number of ones in the bit string x, i.e., $|x|_1 := \sum_{i=1}^n x_i$. If f is a function and S a set, we write $f(S) := \{f(s) \mid s \in S\}$ and we denote by 2^S the power set of S, i.e., the set of all subsets of S. We write id_S for the identity function of S, that is, we set $\mathrm{id}_S(s) := s$ for all $s \in S$.

For $n \in \mathbb{N}$, by S_n we denote the set of all permutations of $[n]$. For $\sigma \in S_n$ and $x \in \{0,1\}^n$ we abbreviate $\sigma(x) := x_{\sigma(1)} \ldots x_{\sigma(n)}$. Lastly, with $\log_a$ we denote the logarithm to base a, and with log we denote the natural logarithm to base $e := \exp(1)$.

If not otherwise indicated, all asymptotic notation (Landau symbols, big-Oh notation) will be with respect to n, which in almost all sections denotes the dimension of the search space $\{0,1\}^n$.

2.2. Some Common Randomized Search Heuristics

In this section we present some of the most prominent randomized search heuristics. Throughout this thesis we shall refer to these algorithms, for example, to illustrate the differences between different black-box models. The reader only interested in black-box complexity may skip this section without loss. However, we shall also introduce in this section some standard terms frequently used in the randomized search heuristics community—like *population, offspring, mutation, selection, fitness* et cetera. In the subsequent sections we will assume that the reader is familiar with these notions.

All algorithms in this section are presented for maximizing pseudo-Boolean functions $f : \{0,1\}^n \to \mathbb{R}$. The minimization versions are equivalent. Note also that for most algorithms the generalization to finite search spaces other than the hypercube $\{0,1\}^n$ is straightforward. As we are mainly interested in pseudo-Boolean functions, we omit the details. The main focus of this thesis is not a thorough discussion of particular randomized search heuristics but rather the development of a complexity theory for reasonably large classes of such algorithms. Therefore, we shall only provide the basic definitions here. Where more details are needed, they will be provided in the subsequent sections. A good survey for both classical and recent results on the algorithms presented in this sections is the book edited by Auger and Doerr [AD11] and, for results on combinatorial optimization problems, the book by Neumann and Witt [NW10].

Note that we do not specify a stopping criterion for any of the algorithms listed below, that is, all algorithms run forever. This is justified by the fact that we typically take as performance measure the expected number of function evaluations needed until an optimal search point is queried for the first time. This is the standard performance measure in the randomized search heuristics community, because in typical applications, evaluating the fitness of a search point is much more costly than the generation of a new search point.

2.2.1. Randomized Local Search

Randomized Local Search (RLS) is the most basic randomized search heuristic. It is a simple hill-climber. In the *mutation step*, it creates from the current best solution x a new search point ("*offspring*") y by flipping exactly one bit in x. It accepts the candidate solution y as new search point if and only if $f(y) \geq f(x)$, i.e., if and only if the objective value ("*fitness*") of y is at least as large as the fitness of x. This is called the *selection step* of the algorithm (line 5).

Algorithm 1: Randomized Local Search for maximizing $f\colon \{0,1\}^n \to \mathbb{R}$.

```
1 Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and query $f(x)$;
2 Optimization: for $t = 1, 2, 3, \dots$ do
3     Choose $j \in [n]$ uniformly at random;
4     Set $y \leftarrow x \oplus e_j^n$ and query $f(y)$ ; //mutation step
5     if $f(y) \geq f(x)$ then $x \leftarrow y$ ; //selection step
```

2.2.2. Elitist Evolutionary Algorithms

The $(1+1)$ evolutionary algorithm ($(1+1)$ EA, Algorithm 2) creates new search points by independently flipping each bit in the current best search point with a fixed *mutation probability* $1/n$. That is, on average, exactly one bit is being flipped in the mutation step. However, in contrast to the RLS algorithm, it is possible for the $(1+1)$ EA to move from one search point to another one at Hamming distance strictly larger than one. In fact, for any $x, y \in \{0,1\}^n$, the probability to move from x to y in one single iteration is strictly positive.

As does the RLS algorithm, the $(1+1)$ EA accepts a candidate solution y if and only if $f(y) \geq f(x)$. This selection scheme is often referred to as *elitist selection*. It is indicated by the "+" in the name $(1+1)$ EA.

Algorithm 2: The (1+1) EA for maximizing $f\colon \{0,1\}^n \to \mathbb{R}$.

```
1 Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and query $f(x)$;
2 Optimization: for $t = 1, 2, 3, \dots$ do
3     Sample $y \in \{0,1\}^n$ by flipping each bit in $x$ with probability $1/n$ and query
      $f(y)$ ; //mutation step
4     if $f(y) \geq f(x)$ then $x \leftarrow y$ ; //selection step
```

The $(1+1)$ EA can easily be generalized to a *population*-based algorithm which—instead of keeping only the single best individual—stores up to μ of the best search points queried so far. Further, there is no need to restrict ourselves to generating only one offspring per iteration. This yields the scheme of Algorithm 3, the $(\mu+\lambda)$ evolutionary algorithm.

Most theoretical work focuses on $(\mu+1)$ EAs and $(1+\lambda)$ EAs and only few results on the $(\mu+\lambda)$ EA for $\mu > 1$ and $\lambda > 1$ exist.

Algorithm 3: The $(\mu + \lambda)$ EA for maximizing $f\colon \{0,1\}^n \to \mathbb{R}$.

```
1 Initialization: Sample $x^{(1)}, \dots x^{(\mu)} \in \{0,1\}^n$ uniformly at random and query
  $f(x^{(1)}), \dots, f(x^{(\mu)})$;
2 Set $P \leftarrow \{x^{(1)}, \dots x^{(\mu)}\}$ ; //initialization of the population
3 Optimization: for $t = 1, 2, 3, \dots$ do
4     for $i = 1, \dots, \lambda$ do
5         Choose $x^{(i)} \in P$ uniformly at random ; //selection of the parent
6         Sample $y^{(i)} \in \{0,1\}^n$ by flipping each bit in $x^{(i)}$ with probability $1/n$
          and query $f(y^{(i)})$ ; //mutation
7     From $P \cup \{y^{(1)}, \dots, y^{(\lambda)}\}$ select the $\mu$ search points $x^{(t,1)}, \dots, x^{(t,\mu)}$ with
      largest fitness values (ties broken arbitrarily) ; //selection step
8     for $j = 1, \dots, \mu$ do $x^{(j)} \leftarrow x^{(t,j)}$;
9     Update $P \leftarrow \{x^{(1)}, \dots, x^{(\mu)}\}$;
```

2.2.3. Non-Elitist Algorithms

In this section we present two well-known heuristics that do not select according to an elitist selection rule. The first one, Algorithm 4 is Simulated Annealing, a generalization of the Metropolis algorithm. In Simulated Annealing, the offspring y is always accepted if it is as least as good as its parent (that is, if $f(y) \geq f(x)$ holds) but in contrast to the elitist selection schemes, the offspring is accepted with a positive probability if its fitness is worse than its parent's one. This probability depends on the absolute distance $f(x) - f(y)$. To prevent the algorithm from simply taking a random walk over the fitness landscape, the probability to accept a candidate solution, depending on $f(x) - f(y)$, is lowered during the run of the algorithm. This is done via some *cooling parameter* $0 < \alpha < 1$ and a constant *temperature* T.

Algorithm 4: Simulated Annealing for maximizing $f\colon \{0,1\}^n \to \mathbb{R}$.

```
1 Input: Temperature $T \in \mathbb{R}_{>0}$, cooling parameter $0 < \alpha < 1$;
2 Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and query $f(x)$;
3 Optimization: for $t = 1, 2, 3, \dots$ do
4     Choose $j \in [n]$ uniformly at random;
5     Set $y \leftarrow x \oplus e_j^n$ and query $f(y)$ ; //mutation step
6     With probability $p_t := \min\{1, \exp(\frac{f(y)-f(x)}{\alpha^t T})\}$ set $x \leftarrow y$ ; //selection step
```

Threshold Accepting (Algorithm 5) is a randomized local search algorithm based on a similar idea as Simulated Annealing. In Threshold Accepting, the selection step is derandomized.

Algorithm 5: Threshold Accepting for maximizing $f\colon \{0,1\}^n \to \mathbb{R}$.

```
1 Input: Threshold sequence $(T(t))_{t\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}_{<0}$;
2 Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and query $f(x)$;
3 Optimization: for $t = 1,2,3,\dots$ do
4     Choose $j \in [n]$ uniformly at random;
5     Set $y \leftarrow x \oplus e^n_j$ and query $f(y)$ ; //mutation step
6     if $f(y) - f(x) \geq T(t)$ then $x \leftarrow y$ ; //selection step
```

2.3. Standard Test Function Classes

Let us now introduce some function classes which shall be considered several times in this thesis. All of them are standard test function classes which are often regarded to understand how a particular search heuristic progresses on different fitness landscapes.

Note that in most classical runtime results for randomized search heuristics, the behavior of one particular algorithm for one particular function is analyzed. For example, the first theoretical work on the $(1+1)$ EA [Müh92] studies the runtime of this algorithm on the so-called OneMax function Om, which simply counts the number of 1-bits, $\textsc{Om}(x) = \sum_{i=1}^n x_i$. Here in this thesis, however, we are interested in the difficulty ("complexity") of this function for a whole class of algorithms. Though we have not yet formally defined what we mean by the complexity of a function class, it is easily seen that considering one single objective function is not very useful in this context. The complexity of such a single-element function class $\mathcal{F} = \{f\}$ is always one. This is verified by the algorithm which queries as first action a search point $x \in \arg\max f$. For the Om function, the algorithm asking for the bit string $(1\dots1)$ in the first step does exactly this: it optimizes the function in just one step. Thus, we need to consider in the context of black-box complexity some generalizations of the standard test functions. This is what we present in the following.

2.3.1. OneMax

As mentioned above, a classical easy test function in the theory of randomized search heuristics is the function Om. The natural generalization of this particular function to a non-trivial class of functions is as follows.

Definition 2.1 (OneMax function class). *Let $n \in \mathbb{N}$. For $z \in \{0,1\}^n$ let*

$$\textsc{Om}_z : \{0,1\}^n \to [0..n], x \mapsto \textsc{Om}_z(x) = |\{i \in [n] \mid x_i = z_i\}| \,.$$

*The string $z = \arg\max \textsc{Om}_z$ is called the **target string** of $\textsc{Om}_z$. Let*

$$\textsc{OneMax}_n := \{\textsc{Om}_z \mid z \in \{0,1\}^n\}$$

be the set of all generalized OneMax *functions.*

For all $z, x \in \{0,1\}^n$ the objective value $\textsc{Om}_z(x) = \textsc{Om}_x(z)$ is just the number of bit positions in which x and z coincide.

The $\textsc{OneMax}_n$ function class has been studied in several context. For example, Erdős and Rényi [ER63] studied it in the context of coin-weighing problems, cf. Section 4 for more details.

The class $\textsc{OneMax}_n$ gained a lot of attention after Meirowitz invented in the seventies the *Mastermind* game (confer, e.g., the works by Knuth [Knu77] and Chvátal [Chv83]). Mastermind is a board game for two players. The goal of the first player is to find a secret color combination made up by the second player. He does so by guessing color combinations and receiving information on how close this guess is to the secret code. The first player's goal is to use as few questions as possible. As we shall see in section 7, this game is closely related to optimizing $\textsc{OneMax}_n$.

2.3.2. BinaryValue

Another intensively studied linear function is the binary-value function $\textsc{Bv}(x) := \sum_{i=1}^{n} 2^{i-1}x_i$. That is, $\textsc{Bv}$ assigns to each bit string the value of the binary number it represents. As $2^i > \sum_{j=1}^{i} 2^{j-1}$, the bit value of some bit $i+1$ dominates the effect of all bits $1, \ldots, i$ on the function value.

As before, we generalize this single function to a function class $\textsc{BinaryValue}_n$, which is the $\oplus$-invariant closure of the standard $\textsc{Bv}$ function. We also consider the $\oplus$- and permutation-invariant function class $\textsc{BinaryValue}^*_n$.

In the following we denote by δ the Kronecker symbol, i.e., for any two numbers $k, \ell \in \mathbb{N}_0$ we have $\delta(k,\ell) = 1$ if $k = \ell$ and we have $\delta(k,\ell) = 0$ otherwise.

Definition 2.2 (BinaryValue function classes). *Let $n \in \mathbb{N}$. For $z \in \{0,1\}^n$ and $\sigma \in S_n$, we define the function*

$$\textsc{Bv}_{z,\sigma} : \{0,1\}^n \to \mathbb{N}_0, x \mapsto \textsc{Bv}(\sigma(x \oplus \bar{z})) = \sum_{i=1}^{n} 2^{i-1}\delta(x_{\sigma(i)}, z_{\sigma(i)})\,.$$

We set $\textsc{Bv}_z := \textsc{Bv}_{z,\mathrm{id}_{[n]}}$. We further define the classes

$$\textsc{BinaryValue}_n := \{\textsc{Bv}_z \mid z \in \{0,1\}^n\}$$

and

$$\textsc{BinaryValue}^*_n := \{\textsc{Bv}_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\}\ .$$

*If $f \in \textsc{BinaryValue}_n$ ($f \in \textsc{BinaryValue}^*_n$), there exist exactly one $z \in \{0,1\}^n$ (exactly one $z \in \{0,1\}^n$ and exactly one $\sigma \in S_n$) such that $f = \textsc{Bv}_z$ ($f = \textsc{Bv}_{z,\sigma}$). Since $z = \arg\max \textsc{Bv}_z$ ($z = \arg\max \textsc{Bv}_{z,\sigma}$), we call z the **target string** of f. Similarly, we call σ the **target permutation** of $\textsc{Bv}_{z,\sigma}$.*

2.3.3. LeadingOnes

For every bit string x, the leading-ones function $\textsc{Lo}$ assigns to each bit string x the length of the longest prefix of ones, $\textsc{Lo}(x) := \max\{i \in [0..n] \mid \forall j \in [i] : x_j = 1\}$. Again we define two generalized classes of this particular function.

Definition 2.3 (LeadingOnes function classes). *Let $n \in \mathbb{N}$. For any $z \in \{0,1\}^n$ let*

$$\textsc{Lo}_z : \{0,1\}^n \to \mathbb{N}, x \mapsto \max\{i \in [0..n] \mid \forall j \in [i] : x_j = z_j\}\,,$$

the length of the maximal joint prefix of x and z. Let $\textsc{LeadingOnes}_n$ be the collection of all such functions, i.e.,

$$\textsc{LeadingOnes}_n := \{\textsc{Lo}_z \mid z \in \{0,1\}^n\}\,.$$

For $z \in \{0,1\}^n$ and $\sigma \in S_n$ we set

$$\textsc{Lo}_{z,\sigma} : \{0,1\}^n \to \mathbb{N}, x \mapsto \max\{i \in [0..n] \mid \forall j \in [i] : x_{\sigma(j)} = z_{\sigma(j)}\}\,,$$

the maximal joint prefix of x and z with respect to σ. The set $\textsc{LeadingOnes}^*_n$ contains all such functions, i.e.,

$$\textsc{LeadingOnes}^*_n := \{\textsc{Lo}_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\}\,.$$

The function $\textsc{Lo} = \textsc{Lo}_{(1\ldots1)}$ is well-studied. It was introduced in [Rud97] to disprove a previous conjecture by Mühlenbein [Müh92] that any unimodal function can be optimized by the $(1+1)$ evolutionary algorithm (Algorithm 2) in $O(n \log n)$ iterations. Rudolph [Rud97] proves only an upper bound of $O(n^2)$ for the expected optimization time of the $(1+1)$ EA on $\textsc{Lo}$ and concludes from experimental studies a lower bound of $\Omega(n^2)$—a bound which was rigorously proven in 2002 by Droste, Jansen, and Wegener [DJW02]. This $\Theta(n^2)$ expected optimization time of the simple $(1+1)$ EA seems optimal among the commonly studied evolutionary algorithms.

2.3.4. Generalized Strictly Monotone Functions

Lastly, let us introduce the class of generalized monotone functions. Following the standard notation, we write $x < y$ if for all $i \in [n]$ we have $x_i \leq y_i$ and if there exists at least one $i \in [n]$ such that $x_i < y_i$. A function $f : \{0,1\}^n \to \mathbb{R}$ is said to be *strictly monotone* if for all $x, y \in \{0,1\}^n$ the relation $x < y$ implies $f(x) < f(y)$.

We extend this notation as follows.

Definition 2.4. *For all $z \in \{0,1\}^n$ we call $f : \{0,1\}^n \to \mathbb{R}$* ***strictly monotone with respect to*** *z if the function $f_z : \{0,1\}^n \to \mathbb{R}, x \mapsto f(x \oplus \bar{z})$ is strictly monotone.*

As is easy to verify, if f is monotone with respect to z, then $\arg\max f = z$. For two bit strings $x, y \in \{0,1\}^n$ the event $\forall i \in [n] : x_i = z_i \Rightarrow y_i = z_i$ implies $f(x) \leq f(z)$ and if, in addition, there exists a $i \in [n]$ such that $x_i \neq z_i = y_i$, then $f(x) < f(y)$ must hold.

Definition 2.5 (Generalized monotone functions). *Let $n \in \mathbb{N}$ and let $\mathcal{F}$ be a class of real-valued functions defined on $\{0,1\}^n$. We call $\mathcal{F}$ a class of* ***generalized strictly monotone functions*** *if for all $f \in \mathcal{F}$ there exists a $z \in \{0,1\}^n$ such that f is strictly monotone with respect to z.*

When we talk about ***"the"*** *class of strictly monotone functions, we refer to the class*

$$\begin{aligned}&\textsc{Monotone}_n := \\ &\quad \{f : \{0,1\}^n \to \mathbb{R} \mid \exists z \in \{0,1\}^n : f \text{ strictly monotone with respect to } z\}\,.\end{aligned}$$

2.4. Useful Tools

In this section we present both tools that are useful for analyzing randomized algorithms as well as tools that we shall apply to derive lower bounds on the black-box complexities of several function classes.

This section is meant as a useful reference for later parts of this thesis. For most of the statements we do not present a proof. Note, however, that these techniques are standard tools and inequalities that can be easily found in most textbooks on (randomized) algorithms. We shall give at least one reference for any of the tools present here.

2.4.1. Linearity of Expectations

A very elementary property of the function which assigns every random variable its expected value is the fact that this is a linear function (cf. [AD11, Section 1] for an introduction to basic probability theory).

Lemma 2.6 (Linearity of expectations). *Let $k \in \mathbb{N}$ and let $X_1, \ldots, X_k$ be random variables taking values in $\mathbb{R}$. Let $\alpha_1, \ldots, \alpha_k \in \mathbb{R}$. Then*

$$\mathrm{E}\Big[\sum_{i=1}^{k} \alpha_i X_i\Big] = \sum_{i=1}^{k} \alpha_i \,\mathrm{E}[X_i]\,.$$

2.4.2. A Few Elementary Bounds

The following bounds can be found, e.g., in [AD11].

Lemma 2.7 (Union bound). *Let $k \in \mathbb{N}$ and let $A_1, \ldots, A_k$ be events in some probability space. Then*

$$\Pr\Big[\bigcup_{i=1}^{k} A_i\Big] \le \sum_{i=1}^{k} \Pr[A_i]\,.$$

Lemma 2.8 (Approximating the inverse of Euler's number). *For all $n \in \mathbb{N}$ and for all $x \in \mathbb{R}$,*

$$\left(1 - \frac{1}{n}\right)^{n} \le \frac{1}{e} \le \left(1 - \frac{1}{n}\right)^{n-1}$$

and

$$1 + x \le e^{x}\,.$$

Lemma 2.9 (Bernoulli's inequality). *For all $n \in \mathbb{N}$ and for all $x \in \mathbb{R}_{\ge -1}$,*

$$\big(1 + x\big)^{n} \ge 1 + xn\,.$$

2.4.3. Factorials and Central Binomial Coefficient

The following estimate on factorials is a direct consequence of Stirling's formula. The version presented below is due to Robbins [Rob55].

Lemma 2.10 (Bounding factorials). *For all $n \in \mathbb{N}$,*

$$\sqrt{2\pi}n^{n+1/2}e^{-n}e^{1/(12n+1)} \leq n! \leq \sqrt{2\pi}n^{n+1/2}e^{-n}e^{1/(12n)}\,.$$

From this we easily get the following estimate for the central binomial coefficient.

Lemma 2.11 (Central binomial coefficient). *For all $n \in \mathbb{N}$,*

$$\frac{4^n}{\sqrt{2\pi n}} \leq \binom{2n}{n} \leq \frac{4^n}{\sqrt{\pi n}}\,.$$

Proof. Let $n \in \mathbb{N}$. By Lemma 2.10 we have that $n! = \sqrt{2\pi n}(n/e)^n \exp(\lambda_n)$ with $\frac{1}{12n+1} \leq \lambda_n \leq \frac{1}{12n}$. Thus,

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \geq \frac{\sqrt{4\pi n}(2n/e)^{2n}\exp(\frac{1}{24n+1})}{(2\pi n)(n/e)^{2n}\exp(\frac{1}{6n})} \geq (\sqrt{\pi n})^{-1}4^n\exp(-\tfrac{1}{6n}) \geq (\sqrt{2\pi n})^{-1}4^n\,,$$

where the last inequality follows from the fact that $\exp(-\frac{1}{6n}) \geq \exp(-\frac{1}{6}) > 0.8 > (\sqrt{2})^{-1}$.

And since $e^{1/24n}e^{-2/(12n+1)} \leq 1$ we also have

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \leq \frac{\sqrt{4\pi n}(2n/e)^{2n}\exp(\frac{1}{24n})}{(2\pi n)(n/e)^{2n}\exp(\frac{2}{12n+1})} \leq \frac{4^n}{\sqrt{\pi n}}\,.$$

□

2.4.4. Chernoff's Bounds

Throughout the thesis we shall apply several versions of Chernoff's bound. The following can be found, for example, in [DP09].

Lemma 2.12 (Chernoff's bounds). *Let $X = \sum_{i=1}^{n} X_i$ be the sum of n independently distributed random variables X_i, where each variable X_i takes values in $[0,1]$. Then the following statements hold.*

$$\begin{aligned} \forall t > 0 : &\Pr[X > \mathrm{E}[X] + t] \leq \exp(-2t^2/n) \text{ and} &(2.1)\\ &\Pr[X < \mathrm{E}[X] - t] \leq \exp(-2t^2/n)\,. \\ \forall \varepsilon > 0 : &\Pr\left[X < (1-\varepsilon)\mathrm{E}[X]\right] \leq \exp\left(-\varepsilon^2\mathrm{E}[X]/2\right) \text{ and} &(2.2)\\ &\Pr\left[X > (1+\varepsilon)\mathrm{E}[X]\right] \leq \exp\left(-\varepsilon^2\mathrm{E}[X]/3\right)\,. \\ \forall t > 2e\,\mathrm{E}[X] : &\Pr[X > t] \leq 2^{-t}\,. &(2.3)\end{aligned}$$

2.4.5. Coupon Collector

Another useful concept is the coupon collector's problem which we present in this section. It shall find several applications in this thesis. The following bounds can again be found, e.g., in [AD11] or, likewise, in [MR95].

In the coupon collector's problem there are n different types of coupons. In each round, one of the n coupons is chosen independently and uniformly at random. That is, for any $i \in [n]$ and in any round t, the probability that the i-th coupon is chosen in the t-th round is $1/n$. We are interested in the expected number of rounds needed until each coupon has been chosen at least once.

Lemma 2.13 (Coupon collector). *Let $n \in \mathbb{N}$ and let $H(n) := \sum_{i=1}^{n} 1/i$ be the n-th harmonic number. On average, it takes $nH(n)$ rounds until each coupon has been chosen at least once. Since $\log n < H(n) < 1 + \log n$ for all n, it takes $(1 - o(1))n \log n$ rounds on average until each coupon has been chosen at least once.*

For any constant $\beta > 1$, the probability that more than $\beta n \log n$ rounds are needed until each coupon has appeared at least once can be bounded from above by $n^{-(\beta-1)}$.

2.4.6. Upper Bounds for Black-Box Complexities via Algorithms with Constant Success Probability

Rather than bounding the expected runtime of an algorithm, it is sometimes easier to show that it solves the problem at hand with good probability in some number s of iterations. For example, if we are only interested in asymptotic runtimes, the following lemma allows us to use such statements for upper bounds.

Lemma 2.14. *Suppose for an optimization problem P there exists a (randomized) algorithm A that, with constant success probability, optimizes P in s iterations. Then there exists a (randomized) algorithm B that optimizes P in an expected number of $O(s)$ iterations.*

Proof. Let c be an upper bound for the failure probability of algorithm A after s iterations. We call the s iterations of A a *run* of A. If X_i denotes the indicator variable for the event that the i-th independent run of A is successful (i.e., computes an optimum), then $\Pr[X_i = 1] \geq 1 - c$. Clearly, $Y := \min\{k \in \mathbb{N} \mid X_k = 1\}$ is a geometric random variable with success probability at least $1 - c$. Hence, $\mathrm{E}[Y] \leq (1 - c)^{-1}$, i.e., the expected number of independent runs of A until success is at most $(1-c)^{-1}$. Thus, we can optimize P in an expected number of at most $(1 - c)^{-1}s$ iterations. Since c is at least constant, the claim follows. □

Since here in this thesis we are mainly concerned with black-box complexities, we reformulate Lemma 2.14 to the following statement.

Corollary 2.15. *Suppose for an optimization problem P there exists a black-box algorithm A that, with constant success probability, optimizes P in s iterations. Then the black-box complexity of P is at most $O(s)$.*

2.4.7. Yao's Minimax Principle

For deriving lower bounds, the minimax principle by Yao [Yao77] has proven to be very helpful. The following presentation is taken from the book by Motwani and Raghavan [MR95].

Theorem 2.16 (Yao's minimax principle). *Let Π be a problem with a finite set $\mathcal{I}$ of input instances (of a fixed size) permitting a finite set $\mathcal{A}$ of deterministic algorithms. Let p be a probability distribution over $\mathcal{I}$ and q be a probability distribution over $\mathcal{A}$. Then,*

$$\min_{A \in \mathcal{A}} \mathrm{E}[T(I_p, A)] \leq \max_{I \in \mathcal{I}} \mathrm{E}[T(I, A_q)]\,,$$

where I_p denotes a random input chosen from $\mathcal{I}$ according to p, A_q a random algorithm chosen from $\mathcal{A}$ according to q, and $T(I, A)$ denotes the running time of algorithm A on input I.

2.4.8. A Drift Theorem

Furthermore, we will use the following *drift theorem* in our proofs. It is useful for both proving upper and lower runtime bounds.

Theorem 2.17 (Additive Drift [HY04]). *Let $(X_t)_{t\geq 0}$ be random variables describing a Markov process over a finite state space $S \subseteq \mathbb{R}$. Let T be the random variable that denotes the earliest point in time $t \geq 0$ such that $X_t = 0$. If there exists a value $c > 0$ such that*

$$E[X_t - X_{t+1} | T > t] \leq c,$$

then

$$E[T | X_0] \geq \frac{X_0}{c}.$$

If there exists a value $d > 0$ such that

$$E[X_t - X_{t+1} | T > t] \geq d,$$

then

$$E[T | X_0] \leq \frac{X_0}{d}.$$

3 Two Basic Black-Box Models

We introduce the *unrestricted black-box model* by Droste, Jansen, and Wegener [DJW06] and the *unbiased black-box model* by Lehre and Witt [LW10a].

The latter one has been defined only for optimization problems where the search space is the n-dimensional hypercube $\{0,1\}^n$. Most of the work presented in this thesis fits this framework. Therefore, we present here the black-box models for optimization problems that can be modeled via functions $\{0,1\}^n \to \mathcal{L}$ where $\mathcal{L}$ is an arbitrary set. Typically we have $\mathcal{L} = \mathbb{R}$ but as we shall see in Section 10 this must not always be the case. In Fact, in Section 10.2 we analyze the black-box complexity of a multi-criteria fitness function.

Concerning more general search spaces let us mention that the generalization of the unrestricted black-box model to arbitrary search spaces is straightforward. For the unbiased black-box model this is not the case. We shall present and discuss three different generalization of the unbiased model for the single-source shortest paths problem in Section 10, cf. Definitions 10.7, 10.8, and 10.9.

Note, however, that many problems can be modeled as pseudo-Boolean functions. For example, we shall see in Section 6 that the class of pseudo-Boolean functions also includes the Partition problem and in Section 9 we shall see how to model minimum spanning tree problems as pseudo-Boolean functions.

3.1. The Unrestricted Black-Box Model

Complexity theory aims at determining the difficulty of computational problems. In theoretical computer science, the fruitful interplay between complexity theory, typically proving that a certain effort is necessary to solve a problem, and theory of algorithms, giving an algorithmic solution for a problem and thus showing that it can be solved with a certain computational effort, was a driving force to develop the field. Usually, the complexity of a problem is measured by the performance of the best algorithm out of some class of algorithms (e.g., all those algorithms which can be implemented on a Turing machine [GJ90], [Hro01]).

With this thesis we aim at continuing the development of a complexity theory for randomized search heuristics. What distinguishes randomized search heuristics from classical algorithms is that they are problem-independent. As such, the only way they obtain information about the problem to be solved is by learning the objective value of possible solutions ("*search points*"). To ensure this problem-independence, one usually assumes that the objective function is given by an oracle, or, equivalently, as a *black-box*. Using this oracle, the algorithm may query the objective value ("*fitness*") of all possible solutions, but any such query does only return this search point's objective value and no other information about the objective function.

Naturally, we do allow that the algorithms use random decisions. From the black-box concept, it follows that the only type of action the algorithm may perform is, based on the objective values learned so far, deciding on a probability distribution over $\{0,1\}^n$, sampling a search point $x \in \{0,1\}^n$ according to this distribution, and querying its objective value from the oracle. This leads to the scheme of Algorithm 6, which we call an *unrestricted black-box algorithm*.

Algorithm 6: Scheme of an unrestricted black-box algorithm

```
1 Initialization: Sample $x^{(0)}$ according to some probability distribution $p^{(0)}$ over
  $\{0,1\}^n$ and query $f(x^{(0)})$;
2 Optimization: for $t = 1, 2, 3, \ldots$ do
3     Depending on $\big((x^{(0)}, f(x^{(0)})), \ldots, (x^{(t-1)}, f(x^{(t-1)}))\big)$ choose a probability
      distribution $p^{(t)}$ over $\{0,1\}^n$ and sample $x^{(t)}$ according to $p^{(t)}$;
4     Query $f(x^{(t)})$;
```

Note again that Algorithm 6 runs forever. As argued in Section 2.2 this is justified by the fact that as performance measure of a black-box algorithm we take the number of queries to the oracle performed by the algorithm until it first queries an optimal solution. This is the standard performance measure for randomized search heuristics because in typical applications of such heuristics, evaluating the fitness of the search points is more costly than the generation of new ones. Since we are mainly talking about randomized algorithms, we regard the expected number of queries.

Formally, for an unrestricted algorithm A and a function $f : \{0,1\}^n \to \mathbb{R}$, let $T(A,f) \in \mathbb{R} \cup \{\infty\}$ be the expected number of fitness evaluations until A queries for the first time some $x \in \arg\max f$. We call $T(A,f)$ the *runtime of A for f* or, likewise, the *optimization time of A for f*.

We can now follow the usual approach in complexity theory. For a class $\mathcal{F}$ of functions $\{0,1\}^n \to \mathbb{R}$, the *A-black-box complexity of $\mathcal{F}$* is $T(A,\mathcal{F}) := \sup_{f \in \mathcal{F}} T(A,f)$, the worst-case runtime of A on $\mathcal{F}$. Let $\mathcal{A}$ be a class of black-box algorithms for functions $\mathcal{F}$. Then the *$\mathcal{A}$-black-box complexity of $\mathcal{F}$* is $T(\mathcal{A},\mathcal{F}) := \inf_{A \in \mathcal{A}} T(A,\mathcal{F})$, the minimum ("best") complexity among all $A \in \mathcal{A}$ for $\mathcal{F}$. If $\mathcal{A}$ is the class of all black-box algorithms, we also call $T(\mathcal{A},\mathcal{F})$ the *unrestricted black-box complexity* of $\mathcal{F}$. This is the black-box complexity as introduced by Droste, Jansen, and Wegener [DJW06].

3.2. The Unbiased Black-Box Model

As mentioned in the introduction, the unrestricted black-box complexity is a lower bound for the efficiency of randomized search heuristics optimizing $\mathcal{F}$. Unfortunately, often this lower bound is not very useful. That is, the unrestricted black-box model often gives unrealistically small complexity values—unrealistically as compared with runtimes exhibited by standard randomized search heuristics. For example, Droste, Jansen, and Wegener [DJW06] observe that the NP-complete MaxClique problem on graphs of n vertices has an unrestricted black-box complexity of only $O(n^2)$. This is certified by the algorithm that uses the first $\binom{n}{2}$ queries to learn for each edge individually whether or not it is present in the graph, computes from this information an optimal solution offline, and queries this optimal solution in the $(\binom{n}{2}+1)$st query. Furthermore, it is easily seen that for any function class $\mathcal{F} = \{f\}$ consisting of one single function, the unrestricted black-box complexity of $\mathcal{F}$ is 1—the algorithm that simply queries an optimal solution of f as first action shows this bound.

This might be a reason why the quest for a complexity theory for randomized search heuristics seemed to have come to an early end (apart from the again unrealistically low $O(n/\log n)$ bound for the $\textsc{OneMax}_n$ function class due to Anil and Wiegand [AW09]).

Black-box complexity was revived by Lehre and Witt in their best-paper awarded GECCO 2010 paper [LW10a]. To overcome the drawbacks of the previous unrestricted black-box model, they restrict the class of admitted black-box optimization algorithms in a natural way, still admitting a large class of classically used algorithms. In their *unbiased black-box complexity model*, they require that all solution candidates must be obtained by variation operators. In addition, these variation operators must be unbiased, that is, they must treat the bit positions and the bit entries 0 and 1 in an unbiased way. This model, in addition to excluding some highly artificial algorithms, also admits a notion of arity. A k-ary unbiased black-box algorithm is one that employs only such variation operators that take up to k arguments. This allows, for example, to talk about mutation-only algorithms (arity one).

For several function classes, the unbiased model leads to more realistic complexities. Still it captures most of the commonly studied search heuristics, such as many $(\mu+\lambda)$ and (μ,λ) evolutionary algorithms, Simulated Annealing, the Metropolis algorithm, and Randomized Local Search. In the following, we give a formal definition of the unbiased black-box model.

Definition 3.1 (k-ary unbiased variation operator). *Let $k \in \mathbb{N}$. A **k-ary unbiased distribution** $(D(. \mid y^{(1)},\dots,y^{(k)}))_{y^{(1)},\dots,y^{(k)}\in\{0,1\}^n}$ is a family of probability distributions over $\{0,1\}^n$ such that for all **inputs** $y^{(1)},\dots,y^{(k)} \in \{0,1\}^n$ the following two conditions hold.*

$$(i)\ \forall x,z \in \{0,1\}^n : D(x \mid y^{(1)},\dots,y^{(k)}) = D(x \oplus z \mid y^{(1)} \oplus z,\dots,y^{(k)} \oplus z)\,,$$
$$(ii)\ \forall x \in \{0,1\}^n\ \forall \sigma \in S_n : D(x \mid y^{(1)},\dots,y^{(k)}) = D(\sigma(x) \mid \sigma(y^{(1)}),\dots,\sigma(y^{(k)}))\,.$$

*We refer to the first condition as **$\oplus$-invariance** and we refer to the second as **permutation invariance**. A variation operator creating an offspring by sampling from a k-ary unbiased distribution is called a **k-ary unbiased variation operator**.*

Note that the combination of $\oplus$- and permutation invariance can be characterized as invariance under Hamming-automorphisms: $D(\cdot \mid x^1, \ldots, x^k)$ is unbiased if and only if, for all $\alpha : \{0,1\}^n \to \{0,1\}^n$ preserving the Hamming distance and all bit strings $y \in \{0,1\}^n$ we have $D(y \mid x^1, \ldots, x^k) = D(\alpha(y) \mid \alpha(x^1), \ldots, \alpha(x^k))$.

Note also that the only 0-ary unbiased distribution over $\{0,1\}^n$ is the uniform one.

1-ary operators, also called *unary* operators, are sometimes referred to as *mutation operators,* in particular in the field of evolutionary computation. In fact, the standard bitwise mutation operator (as used, e.g., by evolutionary algorithms, cf. Section 2.2) is a unary unbiased variation operator.

2-ary operators, also called *binary* operators, are often referred to as *crossover operators.* The *uniform crossover operator* is an unbiased binary one. Given two search points x and y, the uniform crossover operator creates an offspring z from x and y by choosing independently for each index $i \in [n]$ the entry $z_i \in \{x_i, y_i\}$ uniformly at random. However, the standard *one-point crossover operator*—which, given two search points $x, y \in \{0,1\}^n$ picks uniformly at random an index $k \in [n]$ and creates from x and y the two offsprings $x' := x_1 \ldots x_k y_{k+1} \ldots y_n$ and $y' := y_1 \ldots y_k x_{k+1} \ldots x_n$—is not permutation-invariant, and therefore not an unbiased operator.

If we allow arbitrary arities, we call the corresponding black-box model the $*$-ary unbiased black-box model.

k-ary unbiased black-box algorithms can now be described via the scheme of Algorithm 7. The *k-ary unbiased black-box complexity* of some class of functions $\mathcal{F}$ is the complexity of $\mathcal{F}$ with respect to all k-ary unbiased black-box algorithms.

Algorithm 7: Scheme of a k-ary unbiased black-box algorithm

```
1 Initialization: Sample $x^{(0)} \in \{0,1\}^n$ uniformly at random and query $f(x^{(0)})$;
2 Optimization: for $t = 1, 2, 3, \ldots$ do
3     Depending on $\big(f(x^{(0)}), \ldots, f(x^{(t-1)})\big)$ choose $k$ indices $i_1, \ldots, i_k \in [0..t-1]$
      and a $k$-ary unbiased distribution $(D(. \mid y^{(1)}, \ldots, y^{(k)}))_{y^{(1)}, \ldots, y^{(k)} \in \{0,1\}^n}$;
4     Sample $x^{(t)}$ according to $D(. \mid x^{(i_1)}, \ldots, x^{(i_k)})$ and query $f(x^{(t)})$;
```

It is important to note, that in line 3 of Algorithm 7, $x^{(i_1)}, \ldots, x^{(i_k)}$ do not necessarily have to be the k *immediately previously* queried search points. That is, the algorithm is allowed to choose from *all* previously sampled search points.

Note further that for all $k \leq \ell$, each k-ary unbiased black-box algorithm is contained in the ℓ-ary unbiased black-box model. This is due to the fact that we do not require the indices to be pairwise distinct.

Lehre and Witt [LW10a, Theorem 6] proved, among other results, that all functions with a single global optimum have a unary unbiased black-box complexity of $\Omega(n \log n)$. For several standard test problems this bound is met by different unary randomized search heuristics, such as the $(1+1)$ EA or by Randomized Local Search. Recall that, as pointed out above, the unrestricted black-box complexity of any such function is one.

Part I

New Results for the Basic Black-Box Models

4 Faster Black-Box Algorithms Through Higher Arity Operators

We extend the work of Lehre and Witt [LW10a] on the unbiased black-box model by considering higher arity variation operators. In particular, we show that already for binary operators the black-box complexity of $\textsc{LeadingOnes}^*_n$ drops from $\Theta(n^2)$ for unary operators to $O(n \log n)$. For $\textsc{OneMax}_n$, the $\Theta(n \log n)$ unary black-box complexity drops to $O(n)$ in the binary case. For k-ary operators, $2 \leq k \leq n$, the $\textsc{OneMax}_n$-complexity further decreases to $O(n/\log k)$.

These are the first bounds for the k-ary unbiased black-box model with k greater than one.

The results presented in this section are based on the conference publication [DJK+11]. They are joint work with Benjamin Doerr (MPI Saarbrücken), Daniel Johannsen (MPI Saarbrücken, now Tel Aviv University), Timo Kötzing (MPI Saarbrücken), Per Kristian Lehre (Technical University of Denmark), and Markus Wagner (MPI Saarbrücken, now University of Adelaide).

4.1. Introduction

We presented in Section 3.2 the *unbiased* black-box model by Lehre and Witt [LW10a]. Recall that in the unbiased model, intuitively, the unbiasedness condition implies that the variation operator is symmetric with respect to the bit values and bit positions. Or, to be more precise, it must be invariant under Hamming-automorphisms.

Among other problem instances, Lehre and Witt analyze the unbiased black-box complexity of the two function classes $\textsc{OneMax}_n$ and $\textsc{LeadingOnes}^*_n$. They show that the unary unbiased black-box complexity of $\textsc{OneMax}_n$ is $\Omega(n \log n)$ and they show that the unary unbiased black-box complexity of $\textsc{LeadingOnes}_n$ is $\Omega(n^2)$. These bounds match the runtimes of many common search heuristics, e.g., they match the runtime of the $(1+1)$ EA and the one of Randomized Local Search.

In their work, Lehre and Witt give no results on the black-box complexity of *higher arity* models. Recall that a variation operator is said to be of arity k if it creates new search points by recombining up to k previously queried search points (cf. Section 3.2). We are interested in higher arity black-box models because they include commonly used search heuristics, which are not covered by the unary unbiased model. Among such heuristics are evolutionary algorithms that employ uniform crossover, particle swarm optimization [KE01], ant colony optimization [DS04], and estimation of distribution algorithms [LL02].

Although search heuristics that employ higher arity operators are poorly understood from a theoretical point of view, there are some results proving that situations exist where higher arity is helpful. For example, Doerr, Klein, and Happ [DHK08] show that a concatenation operator reduces the runtime on the all-pairs shortest paths problem. Before their work, previous runtime analyses of higher arity variation operators had mostly considered the crossover operator, showing that it can be helpful on some constructed pseudo-Boolean functions [JW02, SW04, DNHW03], on energy-minimization of Ising-models [FW05, Sud05], and on some specific instances of the vertex cover problem [OHY08] to name a few examples. A very recent result is the paper by Kötzing, Sudholt, and Theile [KST11]. They analyze the runtime of genetic algorithms on the JUMP function in dependence of the interplay between mutation and crossover operators. There are also some runtime analyses concerning variation operators of arity higher than two, in particular ant colony optimization [NW09, Gut07], and estimation of distribution algorithms [CLTY09, CTCY10]. However, in spite of these results for particular crossover and higher arity operators for particular problems, a general understanding as of when crossover helps is yet to be developed.

In this section we show that for two standard function classes the binary unbiased black-box complexity greatly differs from the unary one. This is another indication that crossover indeed can help. Although most algorithms presented in this thesis are far from being "natural" bio-inspired search heuristics, we are in good hope that the results presented here will help to design better search heuristics.

In Section 4.2 we analyze the higher arity black-box complexities of ONEMAX_n and in Section 4.3 we analyze the binary black-box complexity of LEADINGONES_n^*. In particular, we show that, surprisingly, the unbiased black-box complexity drops from $\Theta(n^2)$ in the unary case to $O(n \log n)$ for LEADINGONES_n^* and from $\Theta(n \log n)$ to an at most linear complexity for ONEMAX_n. As the bounds for unbiased unary black-box complexities immediately carry over to all higher arity unbiased black-box complexities, we see that increasing the arity of the variation operators provably helps to decrease the complexity. We are optimistic that the ideas developed to prove the bounds can be further exploited to achieve reduced black-box complexities also for other function classes.

We also prove that increasing the arity further does again help. In particular, we show that for every $k \leq n$, the unbiased k-ary black-box complexity of ONEMAX_n can be bounded by $O(n/\log k)$. This bound is optimal for $k = n^{\Omega(1)}$, because the unbiased black-box complexity can always be bounded below by the unrestricted black-box complexity, which is known to be $\Omega(n/\log n)$ for ONEMAX_n. This follows easily from an information theoretic argument, cf. [ER63], [Chv83], or [DJW06] for a more detailed description.

Model	Arity	OneMax$_n$		LeadingOnes$^*_n$	
unbiased	1	$\Omega(n \log n)$	[LW10a]	$\Omega(n^2)$	[LW10a]
		$O(n \log n)$		$O(n^2)$	[Rud97]
unbiased	$2 \leq k \leq n$	$O(n/\log k)$	(here)	$O(n \log n)$	(here)
unbiased	$3 \leq k$			$O(n \log n/\log\log n)$	Section 5
unrestricted	n/a	$\Theta(n/\log n)$	[ER63]	$\Omega(n)$	[DJW06]

Table 4.1. Black-Box Complexity of OneMax$_n$ and LeadingOnes$^*_n$. Note that upper bounds for the unbiased unary black-box complexity immediately carry over to higher arities. Similarly, lower bounds for the unrestricted black-box model also hold for the unbiased model.

For LeadingOnes$^*_n$ we currently have no matching lower bound. In fact, the best known lower bound is a linear one by Droste, Jansen, and Wegener [DJW06]. This linear bound already holds for the much smaller subclass LeadingOnes$_n$ of LeadingOnes$^*_n$. Note that it is tempting to believe that the upper bound $O(n \log n)$ for LeadingOnes$^*_n$ is tight. However, as we shall prove in Section 5, this is not correct. In fact, we show that already the 3-ary unbiased black-box complexity of LeadingOnes$^*_n$ is $O(n \log n/\log\log n)$. Note that this result does not improve the one presented in this section as it only holds for arities greater than or equal to 3, whereas here in this section we consider binary variation operators. We do currently not know whether the binary and the 3-ary unbiased black-box complexities of LeadingOnes$^*_n$ truly differ.

Table 4.1 summarizes the results obtained in this section, and it provides a comparison with known results on black-box complexity of OneMax$_n$ and LeadingOnes$^*_n$.

4.2. The Unbiased Black-Box Complexities of OneMax$_n$

We show that the $*$-ary unbiased black-box complexity of OneMax$_n$ is $\Theta(n/\log n)$ and that the leading constant hidden in the big-Oh-notation is two.

We shall also argue that we gain a factor of $\log n$ already when switching from unary to binary variation operators and that we gain the other $\log n$ factor by allowing arities $k \in n^{\Omega(1)}$.

For the unrestricted black-box model, we have a tight asymptotic bound of $\Theta(n/\log n)$ by Erdős and Rényi [ER63]. They also prove that the leading constant is at least two. A matching upper bound of $(1 + o(1))2n/\log_2 n$ was given by Chvátal [Chv83] in his studies on the Mastermind problem (cf. Section 7).

As mentioned in the introduction, these results seem to have been overlooked in the randomized search heuristics community. For this reason, the lower bound of $\Omega(n/\log n)$ was rediscovered by Droste, Jansen, and Wegener in [DJW06] and an upper bound of $(1 + o(1))2n/\log_2 n$ was given by Anil and Wiegand in [AW09].

Theorem 4.1 (Unrestricted black-box complexity of OneMax$_n$ [ER63]). *The unrestricted black-box complexity of* OneMax$_n$ *is* $\Theta(n/\log n)$*. Moreover, the leading* *constant is at least 2.*

Clearly, the lower bound from Theorem 4.1 directly carries over to the unbiased black-box model. This is due to the fact that all unbiased black-box algorithms are also unrestricted ones.

Corollary 4.2. *The unbiased $*$-ary black-box complexity of* $\textsc{OneMax}_n$ *is* $\Omega(n/\log n)$ *with a leading constant* ≥ 2.

The main results of this section are the following.

Theorem 4.3 (Unbiased $*$-ary black-box complexity of OneMax$_n$). *The unbiased $*$-ary black-box complexity of* $\textsc{OneMax}_n$ *is at most* $(1+o(1))\frac{2n}{\log_2 n}$.

Theorem 4.4 (Unbiased $*$-ary black-box complexity of OneMax$_n$). *For every* $k \in [n]$ *with* $k \geq 2$, *the unbiased k-ary black-box complexity of* $\textsc{OneMax}_n$ *is at most linear in n. Moreover, for* $2 \leq k \leq n$, *it is at most* $O(n/\log k)$.

Both the results of Theorem 4.3 and Theorem 4.4 will be generalized in Section 8, where we study ranking-based versions of the unrestricted and the unbiased black-box models. As we shall see there, Theorem 4.4 remains correct even if we allow the algorithm to access only the ranking of the search points queried so far and not their precise function values.

We present here the main underlying ideas and we give a proof for Theorem 4.3, sparing only a few technical details that will be carried out in detail in Section 8. For the proof of Theorem 4.4, we refer the reader to Section 8.2.

Our proof for Theorem 4.3 is based on the same algorithmic approach as the one implicit in the work of Erdős and Rényi [ER63]. The same method has also been applied by Chvátal [Chv83] and by Anil and Wiegand [AW09], respectively. It is as follows. Assume that we want to optimize a function $\textsc{Om}_z \in \textsc{OneMax}_n$, where z, of course, is unknown. The rough description of the algorithm certifying Theorem 4.3, Algorithm 8, is fairly easy. It first samples $t \in O(n/\log n)$ search points $x^1, \ldots, x^t$ from $\{0,1\}^n$ mutually independent and uniformly at random and it queries their function values $\textsc{Om}_z(x^1), \ldots, \textsc{Om}_z(x^t)$. We show that, with high probability, there exists only one search point $y \in \{0,1\}^n$ such that the potential fitness values $\textsc{Om}_y(x^1), \ldots, \textsc{Om}_y(x^t)$ coincide with the answers $\textsc{Om}_z(x^1), \ldots, \textsc{Om}_z(x^t)$ obtained from the oracle. If there exists only one such y, then clearly $y = z$ must hold. We need to show that based only on $\textsc{Om}_z(x^1), \ldots, \textsc{Om}_z(x^t)$ we can sample z from an unbiased distribution of arity at most n.

To this end we define a t-ary variation operator `chooseConsistent`$(\cdot, \ldots, \cdot)$ as follows. For all search points $x^1, \ldots, x^t \in \{0,1\}^n$ let

$$F(x^1, \ldots, x^t) := \{y \in \{0,1\}^n \mid \forall i \in [t] : \textsc{Om}_y(x^i) = \textsc{Om}_z(x^i)\}$$

be the set of bit strings y that are *consistent* with the observed function values $\textsc{Om}_z(x^1), \ldots, \textsc{Om}_z(x^t)$.

The operator `chooseConsistent` is now based on the distribution $D(\cdot \mid$

Algorithm 8: Optimizing $\textsc{OneMax}_n$ with unbiased variation operators.

```
1 Initialization $t \leftarrow \lceil (1 + \frac{4\log_2\log_2 n}{\log_2 n}) \frac{2n}{\log_2 n} \rceil$;
2 repeat
3     for $i, \dots, t$ do
4         Choose $x^i$ from $\{0,1\}^n$ uniformly at random and query $\textsc{Om}_z(x^i)$;
5     $y \leftarrow$ chooseConsistent($x^1, \dots, x^t$);
6     Query $\textsc{Om}_z(y)$;
7 until $\textsc{Om}_z(y) = n$ ;
```

$x^1, \dots, x^t)_{x^1,\dots,x^t \in \{0,1\}^n}$, which, given some $x^1, \dots, x^t$, assigns to each y the probability

$$D(y \mid x^1, \dots, x^t) := \begin{cases} |F(x^1, \dots, x^t)|^{-1} & \text{if } y \in F(x^1, \dots, x^t)\,, \\ 0 & \text{if } F(x^1, \dots, x^t) \neq \emptyset \text{ and } y \notin F(x^1, \dots, x^t)\,, \\ 2^{-n} & \text{otherwise.} \end{cases}$$

Proving that this is indeed an unbiased distribution is not difficult, cf. Section 8 for the details.

An upper bound of $(1 + o(1))2n/\log_2 n$ for the expected runtime of Algorithm 8 follows directly from the following theorem which implies that the number of repetitions of steps 4 to 6 follows a geometric distribution with success probability $1 - o(1)$. This proves Theorem 4.3.

Lemma 4.5. *Let n be sufficiently large (i. e., let $n \geq N_0$ for some fixed constant $N_0 \in \mathbb{N}$). Let $z \in \{0,1\}^n$ and let X be a set of $t \geq \big(1 + \frac{4\log_2\log_2 n}{\log_2 n}\big)\frac{2n}{\log_2 n}$ samples chosen from $\{0,1\}^n$ uniformly at random and mutually independent. Then the probability that there exists an element $y \in \{0,1\}^n$ such that $y \neq z$ and $\textsc{Om}_y(x) = \textsc{Om}_z(x)$ for all $x \in X$ is bounded from above by $2^{-t/2}$.*

The previous lemma is a refinement of Theorem 1 in [AW09], and its proof follows the proof of Theorem 1 in [AW09], clarifying some inconsistencies[1] in that proof. Note that Lemma 4.5 has been proven implicitly also by Chvátal [Chv83]. To derive Lemma 4.5 we need to bound the following combinatorial quantity (compare Lemma 1 in [AW09]).

Proposition 4.6. *For sufficiently large n, for*

$$t \geq \left(1 + \frac{4\log_2\log_2 n}{\log_2 n}\right)\frac{2n}{\log_2 n}\,,$$

and for even $d \in \{2, \dots, n\}$, it holds that

$$\binom{n}{d}\left(\binom{d}{d/2}2^{-d}\right)^t \leq 2^{-3t/4}. \tag{4.1}$$

[1]For example, in the proof of Lemma 1 in [AW09] the following claim is made. Let $d(n)$ be a monotone increasing sequence that tends to infinity. Then for sufficient large n the sequence $h_d(n) = (\frac{\pi d(n)}{8})^{1/(2\log_2 n)}$ is bounded away from 1 by a constant $b > 1$. Clearly, this is not the case. For example, for $d(n) = \log_2 n$, the sequence $h_{\log_2}(n)$ converges to 1.

Proof. By Lemma 2.11 (bound on the central binomial coefficient using Stirling's formula) we have $\binom{d}{d/2} \leq \left(\frac{\pi d}{2}\right)^{-1/2} 2^d$. Therefore,

$$\binom{n}{d}\left(\binom{d}{d/2}2^{-d}\right)^t \leq \binom{n}{d}\left(\frac{\pi d}{2}\right)^{-t/2}. \tag{4.2}$$

We distinguish two cases. First, we consider the case $2 \leq d < n/\log_2^3 n$. By Stirling's formula, it holds that $\binom{n}{d} \leq \left(\frac{en}{d}\right)^d$. Thus, we get from (4.2) that

$$\begin{aligned}\binom{n}{d}\left(\binom{d}{d/2}2^{-d}\right)^t &\leq \left(\frac{en}{d}\right)^d\left(\frac{\pi d}{2}\right)^{-t/2} \\ &= 2^{\left(\frac{2d}{t}\log_2\left(\frac{en}{d}\right)-\log_2\left(\frac{\pi d}{2}\right)\right)\frac{t}{2}}.\end{aligned} \tag{4.3}$$

We bound d by its minimal value 2 and maximal value $n/\log_2^3 n$, and t by $2n/\log_2 n$ to obtain

$$\frac{2d}{t}\log_2\left(\frac{en}{d}\right) - \log_2\left(\frac{\pi d}{2}\right) \leq \frac{1}{\log_2^2 n}\log_2\left(\frac{en}{2}\right) - \log_2(\pi).$$

Since the first term on the right hand side converges to 0 and since $\log_2 \pi > 3/2$, the exponent in (4.3) can be bounded from above by -3t/4, if n is sufficiently large. Thus, we obtain inequality (4.1) for $2 \leq d < n/\log_2^3 n$.

Next, we consider the case $n/\log_2^3 n \leq d \leq n$. By the binomial formula, it holds that $\binom{n}{d} \leq 2^n$. Plugging this into inequality (4.2), we obtain

$$\binom{n}{d}\left(\binom{d}{d/2}2^{-d}\right)^t \leq 2^n\left(\frac{\pi d}{2}\right)^{-t/2} = 2^{\left(\frac{2n}{t}-\log_2\frac{\pi d}{2}\right)\frac{t}{2}}. \tag{4.4}$$

We have $\pi d/2 \geq n/\log_2^3 n$ and, by definition, $t \geq \left(1 + \frac{4\log_2\log_2 n}{\log_2 n}\right)\frac{2n}{\log_2 n}$. Hence,

$$\begin{aligned}\frac{2n}{t} - \log_2\frac{\pi d}{2} &\leq \frac{\log_2 n}{1+\frac{4\log_2\log_2 n}{\log_2 n}} - \log_2\left(\frac{n}{\log_2^3 n}\right. \\ &= \frac{\log_2 n}{1+\frac{4\log_2\log_2 n}{\log_2 n}} - \log_2 n + 3\log_2\log_2 n \\ &= \frac{3\log_2\log_2 n + \frac{4\log_2\log_2 n}{\log_2 n}(-\log_2 n + 3\log_2\log_2 n)}{1+\frac{4\log_2\log_2 n}{\log_2 n}} \\ &= -\frac{\log_2 n - 12\log_2\log_2 n}{\log_2 n + 4\log_2\log_2 n}\log_2\log_2 n.\end{aligned}$$

Again, for sufficiently large n the right hand side becomes smaller than $-3/2$. Plugging this into equation (4.4), we obtain inequality (4.1) for $n/\log_2^3 n \leq d \leq n$. □

With the previous proposition at hand, we finally prove Lemma 4.5.

Proof of Lemma 4.5. Let n be sufficiently large, let $z \in \{0,1\}^n$, and let X be a set of $t \geq \left(1 + \frac{4\log_2\log_2 n}{\log_2 n}\right)\frac{2n}{\log_2 n}$ samples chosen from $\{0,1\}^n$ uniformly at random and mutually independent.

For $d \in [n]$, let $A_d := \{y \in \{0,1\}^n \mid n - \textsc{Om}_z(y) = d\}$ be the set of all points with Hamming distance d from z. Let $d \in [n]$ and $y \in A_d$. We say the point y is *consistent* with x if $\textsc{Om}_y(x) = \textsc{Om}_z(x)$ holds. Intuitively, this means that $\textsc{Om}_y$ is a possible target function, given the fitness of x. It is easy to see that y is consistent with x if and only if x and y (and therefore x and z) differ in exactly half of the d bits that differ between y and z. Therefore, y is never consistent with x if d is odd and the probability that y is consistent with x is $\binom{d}{d/2}2^{-d}$ if d is even.

Let p be the probability that there exists a point $y \in \{0,1\}^n \setminus \{z\}$ such that y is consistent with all $x \in X$. Then,

$$p = \Pr\Big(\bigcup_{y \in \{0,1\}^n \setminus \{z\}} \ \bigcap_{x \in X} \text{“}y \text{ is consistent with } x\text{”}\Big).$$

Thus, by the union bound, we have

$$p \le \sum_{y \in \{0,1\}^n \setminus \{z\}} \Pr\Big(\bigcap_{x \in X} \text{“}y \text{ is consistent with } x\text{”}\Big).$$

Since, for a fixed y, the events “y is consistent with x” are mutually independent for all $x \in X$, it holds that

$$p \le \sum_{d=1}^{n} \sum_{y \in A_d} \prod_{x \in X} \Pr(\text{“}y \text{ is consistent with } x\text{”}).$$

We substitute the probability that a fixed $y \in \{0,1\}^n$ is consistent with a randomly chosen $x \in \{0,1\}^n$ as given above. Using $|A_d| = \binom{n}{d}$, we obtain

$$p \le \sum_{d \in \{1,\ldots,n\}:\, d \text{ even}} \binom{n}{d} \left(\binom{d}{d/2} 2^{-d}\right)^t$$

Finally, we apply Proposition 4.6 and have $p \le n2^{-3t/4}$ which concludes the proof since $n \le 2^{t/4}$ for sufficiently large n (as t in $\Omega(n/\log n)$). □

4.3. The Complexity of LeadingOnes

In this section, we show that allowing k-ary variation operators, for $k > 1$, greatly reduces the black-box complexity of the $\textsc{LeadingOnes}^*_n$ functions class, namely from $\Theta(n^2)$ (proven in [LW10a]) down to $O(n \log n)$.

Theorem 4.7. *The unbiased binary black-box complexity of* $\textsc{LeadingOnes}^*_n$ *is* $O(n \log n)$.

The key ingredient of the black-box algorithm which yields the upper bound is an emulation of a binary search which determines the unique bit that increases the fitness *and* does flip this bit. Surprisingly, this can be done already with binary operators. The main idea is to keep two individuals x and y such that for all bit positions $i \in [n]$ in which x and y agree, the corresponding bit value x_i equals the one of the optimal solution.

We will use the two unbiased binary variation operators `randomWhereDifferent`$(\cdot,\cdot)$ and `switchIfDistanceOne`$(\cdot,\cdot)$. The operator `randomWhereDifferent`(y,y') generates a search point w out of y and y' such that $w_i = y_i$, if $y_i = y'_i$, and $w_i \in \{0,1\}$ is chosen uniformly at random for all other bit positions. The operator `switchIfDistanceOne`(y,y') returns y' if y and y' differ in exactly one bit, and it returns y otherwise. It is easy to see that both `randomWhereDifferent`$(\cdot,\cdot)$ and `switchIfDistanceOne`$(\cdot,\cdot)$ are unbiased variation operators: For `randomWhereDifferent`$(\cdot,\cdot)$ this follows essentially from the fact that for all $v,w,y,y' \in \{0,1\}^n$ and for all $\sigma \in S_n$ we have $w_i = y_i = y'_i$ if and only if $(w\oplus v)_i = (y\oplus v)_i = (y'\oplus v)_i$ and if and only if $\sigma(x)_{\sigma^{-1}(i)} = \sigma(y)_{\sigma^{-1}(i)} = \sigma(y')_{\sigma^{-1}(i)}$. Therefore the probability distribution $D(\cdot \mid y,y')$ that assigns to each x the probability

$$D(x \mid y,y') = \begin{cases} 2^{n-|\{i\in[n] \mid y_i=y'_i\}|} & \text{if } \forall i\in[n] : y_i = y'_i \Rightarrow x_i = y_i\,, \\ 0 & \text{otherwise} \end{cases}$$

is an unbiased distribution. The unbiasedness of the variation operator `switchIfDistanceOne`$(\cdot,\cdot)$ can be seen as follows. For all x,y,y' let

$$D'(x \mid y,y') = \begin{cases} 1 & \text{if } x = y' \text{ and } \exists i\in[n] : (y_i = y'_i) \wedge (\forall j\neq i : y_j \neq y'_j)\,, \\ 1 & \text{if } x = y \text{ and } \Big(\forall i\in[n] : y_i \neq y'_i \text{ or} \\ & \qquad\qquad \exists i_1 \neq i_2 \in [n] : (y_{i_1} = y'_{i_1}) \wedge (y_{i_2} = y'_{i_2})\Big)\,, \\ 0 & \text{otherwise.} \end{cases}$$

Then it is easy to verify that $D'(x \mid y,y') = D'(\sigma(x\oplus v) \mid \sigma(y\oplus v), \sigma(y'\oplus v))$ for all $v\in\{0,1\}^n$ and all $\sigma\in S_n$.

Further we apply the unary variation operator `complement`$(\cdot)$, which, given some $x\in\{0,1\}^n$ returns $\bar{x}$, the bitwise complement of x. This is clearly an unbiased operator as $\overline{x\oplus w} = \bar{x}\oplus w$ for all $x,w\in\{0,1\}^n$ and $\overline{\sigma(x)} = \sigma(\bar{x})$ for all $\sigma\in S_n$.

We call a pair (x,y) of search points *critical* for a function f, if the following two conditions are satisfied. (i) $f(x)\geq f(y)$. (ii) There are exactly $f(y)$ bit positions $i\in[n]$ such that $x_i = y_i$. The following is a simple observation.

Lemma 4.8. *Let $f \in \textsc{LeadingOnes}^*_n$. If (x,y) is a critical pair for f, then either $f(x) = n = f(y)$ or $f(x) > f(y)$.*

For $f\in\textsc{LeadingOnes}^*_n$ $f(x) > f(y)$, then the unique bit position k such that flipping the k-th bit in x reduces its fitness to $f(y)$—or equivalently, the unique bit position such that flipping this bit in y increases y's fitness—shall be called the *critical bit position*. We also call $f(y)$ the *value* of the pair (x,y).

If $f = \textsc{Lo}_{\sigma,z}$, then (x,y) is a critical pair for f if x and y coincide on the bit positions $\sigma(1),\ldots,\sigma(f(y))$ and they disagree on all other bit positions. Also, the critical bit position is $\sigma(f(y)+1)$, and the only way to improve the fitness of y is to flip this particular bit position while keeping the positions $\sigma(1),\ldots,\sigma(f(y))$ unchanged. The central part of Algorithm 9, which is contained in lines 3 to 9, manages to transform a critical pair of value $v<n$ into one of value $v+1$ in $O(\log n)$ time. This is analyzed in the following lemma.

Algorithm 9: Optimizing $\textsc{LeadingOnes}^*_n$ with unbiased binary variation operators.

```
1  Initialization Choose $x$ uniformly at random and query $f(x)$;
2  Set $y \leftarrow$ complement($x$) and query $f(y)$;
3  repeat
4      if $f(y) > f(x)$ then $(x, y) \leftarrow (y, x)$;
5      $y' \leftarrow x$;
6      repeat
7          Set $y'' \leftarrow$ randomWhereDifferent($y, y'$) and query $f(y'')$;
8          if $f(y'') > f(y)$ then $y' \leftarrow y''$;
9          Set $y \leftarrow$ switchIfDistanceOne($y, y'$) and query $f(y)$;
10     until $f(y) = f(y')$ ;
11 until $f(x) = f(y)$ ;
```

Lemma 4.9. *Assume that the execution of Algorithm 9 is before line 4, and that the current value of (x, y) is a critical pair of value $v < n$. Then after an expected number of $O(\log n)$ iterations, the loop in lines 5-9 is left and (x, y) or (y, x) is a critical pair of value $v + 1$.*

Proof. Let k be the critical bit position of the pair (x, y). Let $y' = x$ be a copy of x. Let $J := \{i \in [n] \mid y_i \neq y'_i\}$. Our aim is to flip all bits of y' with index in $J \setminus \{k\}$.

We define y'' by flipping each bit of y' with index in J with probability 1/2. Equivalently, we can say that y''_i equals y'_i for all i such that $y'_i = y_i$, and is random for all other i (thus, we obtain such y'' by applying `randomWhereDifferent`(y, y')).

With probability exactly 1/2, the critical bit was not flipped ("success"), and consequently, $f(y'') > f(y)$. In this case (due to independence), each other bit with index in J has a chance of 1/2 of being flipped. So with constant probability at least 1/2, $\{i \in [n] \mid y_i \neq y''_i\} \setminus \{k\}$ is at most half the size of $J \setminus \{k\}$. In this success case, we take y'' as new value for y'.

In consequence, the cardinality of $J \setminus \{k\}$ does never increase, and with probability at least 1/4, it decreases by at least 50%. Consequently, after an expected number of $O(\log n)$ iterations, we have $|J| = 1$, namely $J = \{k\}$. We check this via an application of `switchIfDistanceOne`. □

We are now ready to prove the main result of this section.

Proof Theorem 4.7. We regard the following invariant: (x, y) or (y, x) is a critical pair. This is clearly satisfied after execution of line 1. From Lemma 4.9, we see that a single execution of the outer loop does not dissatisfy our invariant. Hence by Lemma 4.8, our algorithm is correct (provided it terminates). The algorithm does indeed terminate, namely in $O(n \log n)$ time, because, again by Lemma 4.9, each iteration of the outer loop increases the value of the critical pair by one. □

4.4. Conclusion and Future Work

We continue the study of the unbiased black-box model introduced in [LW10a]. For the first time we analyzed black-box models of arities larger than one. Our results show that already two-ary operators can allow significantly faster algorithms than unary ones.

The problem $\textsc{OneMax}_n$ cannot be solved in shorter time than $\Omega(n \log n)$ with unary variation operators [LW10a]. However, the runtime can be reduced to $O(n)$ with binary operators. The runtime can be decreased even further with higher arities than two. For k-ary variation operators, $2 \leq k \leq n$, the runtime can be reduced to $O(n/\log k)$, which for $k = n^{\Omega(1)}$ matches the lower bound in the unrestricted black-box model.

A similar positive effect of higher arity variation operators can be observed for the function class $\textsc{LeadingOnes}_n^*$. While this function class cannot be optimized faster than $\Omega(n^2)$ with unary variation operators [LW10a], we show that the runtime can be reduced to $O(n \log n)$ with binary variation operators.

Despite the restrictions imposed by the unbiasedness conditions, our analysis demonstrates that black-box algorithms can employ new and more efficient search heuristics with higher arity variation operators. In particular, binary variation operators allow a memory mechanism that can be used to implement binary search on the positions in the bit string. The algorithm can thereby focus on parts of the bit string that has not investigated previously.

An important open problem arising from this work is to provide lower bounds in the unbiased black-box model for higher arities than one. Due to the greatly enlarged computational power of black-box algorithms using higher arity operators, proving lower bounds in this model seems significantly harder than in the unary model. Currently all known lower bounds for unbiased models of arities larger than one are lower bounds that hold even in the much stronger unrestricted black-box model.

5 Breaking the $O(n \log n)$ Barrier of LeadingOnes

We show that the unrestricted black-box complexity of $\textsc{LeadingOnes}_n^*$ is $O(n \log n / \log\log n)$. This shows that the natural looking $O(n \log n)$ bound proven in Section 4 is not tight. The black-box optimization algorithm leading to this bound can be implemented in a way that only 3-ary unbiased variation operators are used. The bound also remains valid if we impose the additional restriction that the black-box algorithm does not have access to the objective values but only to their relative order (ranking-based black-box complexity, cf. Section 8).

Our results disproves a previous conjecture by Droste, Jansen, Tinnefeld, and Wegener made in the conference version [DJTW03] of [DJW06]. They conjectured that the ranking-based black-box complexity of $\textsc{LeadingOnes}_n^*$ is $\Theta(n \log n)$, cf. [DJTW03, Section 7].

The results presented in this section are based on the conference publication [DW11a]. They are joint work with Benjamin Doerr.

5.1. Introduction

We continue our study of how difficult it is to optimize an unknown function $\textsc{Lo}_{z,\sigma} \in \textsc{LeadingOnes}_n^*$. In the previous section we have shown that already in the binary unbiased black-box model, assuming knowledge on $\sigma(1), \ldots, \sigma(\ell)$, one can perform a binary search to determine $\sigma(\ell+1)$ and its corresponding bit value. Since this has to be done at most n times, an upper bound of $O(n \log n)$ for the binary unbiased black-box complexity of $\textsc{LeadingOnes}_n^*$ follows.

In this section we show that both in the unrestricted black-box model (Section 5.2) and in the unbiased black-box model for arities at least three (Section 5.3), one can do better. More precisely, we show that the corresponding black-box complexities are $O(n \log n / \log\log n)$. That is, for each of these models, there exists a black-box algorithm which needs, on average, only $O(n \log n / \log\log n)$ queries to optimize any function in $\textsc{LeadingOnes}^*_n$. This breaks the previous $O(n \log n)$ barrier. This result also shows why previous attempts to prove an $\Omega(n \log n)$ lower bound must fail.

Unfortunately, also the ranking-based model (cf. Section 8) does not help to overcome this unnatural low black-box complexity. We shall comment in Section 5.4 that the 3-ary unbiased ranking-based black-box complexity of $\textsc{LeadingOnes}^*_n$, too, is $O(n \log n / \log\log n)$.

As for the memory-restricted model (cf. Section 7) we note without proof that a memory of size $O(\sqrt{\log n})$ suffices to achieve the same bound.

5.2. On LeadingOnes$^*_n$ in the Unrestricted Model

The main contribution of this section is the following statement.

Theorem 5.1. *The unrestricted black-box complexity of* $\textsc{LeadingOnes}^*_n$ *is* $O(n \log n / \log\log n)$.

The proof of Theorem 5.1 is technical. For this reason, we split it into several lemmata. The main proof can be found at the end of this section. We remark already here that the algorithm certifying Theorem 5.1 will make use of unbiased variation operators only. Hence, it also proves that the $*$-ary unbiased black-box complexity of $\textsc{LeadingOnes}^*_n$ is $O(n \log n / \log\log n)$. This will be improved in Section 5.3.

The main idea of both the $*$-ary and the 3-ary algorithm is the following. Given a bit string x of fitness $\textsc{Lo}_{z,\sigma}(x) = \ell$, we iteratively first learn $k := \lceil \sqrt{\log_2 n} \rceil$ bit positions $\sigma(\ell+1), \ldots, \sigma(\ell+k)$ and their corresponding bit values, which we fix for all further iterations of the algorithm. Learning such a block of size k will require $O(k^3/\log k^2)$ queries. Since we have to optimize $\lceil n/k \rceil$ such blocks, an overall expected number of $O(nk^2/\log k^2) = O(n \log n / \log\log n)$ queries are needed to determine the target string z. In fact, by this approach, we also determine the target permutation σ. Note that this is not necessarily needed for optimizing $\textsc{Lo}_{z,\sigma}$. That is, in principle, it could be possible to determine z without learning σ.

Conventions: For any two strings $x, y \in \{0,1\}^n$ let $\mathcal{B}(x,y) := \{i \in [n] \mid x_i = y_i\}$, the set of positions in which x and y coincide.

For $r \in \mathbb{R}_{\geq 0}$, let $\lceil r \rceil := \min\{n \in \mathbb{N}_0 \mid n \geq r\}$ and $\lfloor r \rfloor := \max\{n \in \mathbb{N}_0 \mid n \leq r\}$. To ease reading we sometime omit the $\lceil \cdot \rceil$ signs. That is, whenever in this section we write r where an integer is required, we implicitly mean $\lceil r \rceil$.

For all following statements let us fix a positive integer n, a bit string $z \in \{0,1\}^n$ and a permutation $\sigma \in S_n$.

Let us now present a refined definition of critical pairs that we introduced in Section 4.3.

Definition 5.2 (Encoding pairs). *Let* $\ell \in [0..n]$ *and let* $y \in \{0,1\}^n$ *with* $\textsc{Lo}_{z,\sigma}(y) = \ell$. *If* $x \in \{0,1\}^n$ *satisfies* $\textsc{Lo}_{z,\sigma}(x) \geq \textsc{Lo}_{z,\sigma}(y)$ *and* $\ell = |\{i \in [n] \mid x_i = y_i\}|$, *we call* (x,y) *an* ***ℓ-encoding pair for $\textsc{Lo}_{z,\sigma}$***.

If (x,y) *is an* ℓ*-encoding pair for* $\textsc{Lo}_{z,\sigma}$, *the bit positions* $\mathcal{B}(x,y)$ *are called the* ***ℓ-encoding bit positions of $\textsc{Lo}_{z,\sigma}$*** *and the bit positions* $j \in [n] \backslash \mathcal{B}(x,y)$ *are called*

non-encoding.

If (x,y) is an ℓ-encoding pair for $\textsc{Lo}_{z,\sigma}$ we clearly either have have $\textsc{Lo}_{z,\sigma}(x) > \ell$ or $\textsc{Lo}_{z,\sigma}(x) = \textsc{Lo}_{z,\sigma}(y) = n$. For each non-optimal $y \in \{0,1\}^n$ we call the unique bit position which needs to be flipped in y in order to increase the objective value of y the *ℓ-critical bit position of* $\textsc{Lo}_{z,\sigma}$. Clearly, the ℓ-critical bit position of $\textsc{Lo}_{z,\sigma}$ equals $\sigma(\ell+1)$, but since σ is unknown to the algorithm we shall make use of this knowledge only in the analysis, and not in the definition of our algorithms. In the same spirit, we call the k bit positions $\sigma(\ell+1),\ldots,\sigma(\ell+k)$ the *k ℓ-critical bit positions of* $\textsc{Lo}_{z,\sigma}$.

Lemma 5.3. *Let $\ell \in [0..n-1]$ and let $(x,y) \in \{0,1\}^n \times \{0,1\}^n$ be an ℓ-encoding pair for $\textsc{Lo}_{z,\sigma}$. Furthermore, let $k \in [n - \textsc{Lo}_{z,\sigma}(y)]$ and let $y' \in \{0,1\}^n$ with $\ell \leq \textsc{Lo}_{z,\sigma}(y') < \ell + k$.*

If we create y'' from y' by flipping each non-encoding bit position $j \in [n]\backslash\mathcal{B}(x,y)$ with probability $1/k$, then

$$\Pr[\textsc{Lo}_{z,\sigma}(y'') > \textsc{Lo}_{z,\sigma}(y')] \geq (ek)^{-1}\,.$$

Proof. First note that due to $\textsc{Lo}_{z,\sigma}(y') \geq \textsc{Lo}_{z,\sigma}(y)$, we clearly have $y'_j = x_j$ for all ℓ-encoding bit positions $j \in \mathcal{B}(x,y)$. Since we do not allow these bit positions $\mathcal{B}(x,y)$ to be changed, we necessarily also have $\textsc{Lo}_{z,\sigma}(y'') \geq \ell$.

Let $c \in [0..\sqrt{\log_2 n}]$ such that $\textsc{Lo}_{z,\sigma}(y') = \ell + c$. Then $\textsc{Lo}_{z,\sigma}(y'') > \textsc{Lo}_{z,\sigma}(y')$ if and only if both

(i) none of the c bit positions $\sigma(\textsc{Lo}_{z,\sigma}(y)+1),\ldots,\sigma(\textsc{Lo}_{z,\sigma}(y)+c)$ is flipped, and

(ii) the $\ell+c$-critical bit position $\sigma(\ell+c+1)$ is flipped.

This yields

$$\Pr[\textsc{Lo}_{z,\sigma}(y'') > \textsc{Lo}_{z,\sigma}(y')] = (1-\tfrac{1}{k})^c\tfrac{1}{k} \geq (1-\tfrac{1}{k})^{k-1}\tfrac{1}{k} \geq (ek)^{-1}\,,$$

where we make use of the well-known fact that for all positive integers k it holds that $(1-\frac{1}{k})^{k-1} \geq e^{-1}$ (cf. Lemma 2.8). □

Lemma 5.3 motivates us to formulate Algorithm 10 which can be seen as a variant of the standard (1+1) EA, in which we fix some bit positions and where we apply a non-standard mutation probability. The variation operator $\texttt{random}(y', x, y, 1/k)$ samples a bit string y'' from y' by flipping each non-encoding bit position $j \in [n]\backslash\mathcal{B}(x,y)$ with probability $1/k$. This is easily seen to be an unbiased variation operator of arity three (compare the proof of the unbiasedness of operator `randomWhereDifferent` in Section 4.3).

The following statement follows easily from Lemma 5.3 and the linearity of expectation (cf. Lemma 2.6).

Corollary 5.4. *Let (x,y), ℓ, and k be as in Lemma 5.3. Then the (x,y)-encoded $(1+1)$ EA with mutation probability $1/k$, after an expected number of $O(k^2)$ queries, outputs a bit string $y' \in \{0,1\}^n$ with $\textsc{Lo}_{z,\sigma}(y') \geq \ell + k$.*

Algorithm 10: The (x,y)-encoded (1+1) evolutionary algorithm with mutation probability $1/k$.

```
1 Input: $\ell$-encoding pair $(x,y) \in \{0,1\}^n \times \{0,1\}^n$.
2 $y' \leftarrow y$;
3 while $\textsc{Lo}_{z,\sigma}(y') < \textsc{Lo}_{z,\sigma}(y) + k$ do
4     Set $y'' \leftarrow$ random($y', x, y, 1/k$) and query $\textsc{Lo}_{z,\sigma}(y'')$;
5     if $\textsc{Lo}_{z,\sigma}(y'') > \textsc{Lo}_{z,\sigma}(y')$ then $y' \leftarrow y''$;
6 Output $y'$;
```

A second key argument in the proof of Theorem 5.1 is the following. Given an ℓ-encoding pair (x,y) and a bit string y' with $\textsc{Lo}_{z,\sigma}(y') \geq \ell + \sqrt{\log_2 n}$, we are able to learn the $\sqrt{\log_2 n}$ ℓ-critical bit positions $\sigma(\ell+1), \ldots, \sigma(\ell+\sqrt{\log_2 n})$ in an expected number of $O(\log^{3/2} n/\log\log n)$ queries. This will be formalized in the following statements.

Lemma 5.5. *Let $\ell \in [0..n - \lceil\sqrt{\log_2 n}\rceil]$ and let (x,y) be an ℓ-encoding pair for $\textsc{Lo}_{z,\sigma}$. Furthermore, let y' be a bit string with $\textsc{Lo}_{z,\sigma}(y') \geq \ell + \sqrt{\log_2 n}$.*

For each $i \in [8e\log_2^{3/2} n/\log_2\log_2 n]$ let y^i be sampled from y' by independently flipping each non-encoding bit position $j \in [n]\backslash\mathcal{B}(x,y)$ with probability $1/\sqrt{\log_2 n}$.

For each $c \in [\sqrt{\log_2 n}]$ let

$$X_c := \{y^i \mid i \in [8e\log_2^{3/2} n/\log_2\log_2 n] \text{ and } \textsc{Lo}_{z,\sigma}(y^i) = \ell + c - 1\},$$

the set of all samples y^i with $\textsc{Lo}_{z,\sigma}(y^i) = \ell + c - 1$.

Then

$$\Pr\left[\forall c \in [\sqrt{\log_2 n}] : |X_c| \geq 4\log_2 n/\log_2\log_2 n\right] \geq 1 - o(1).$$

Proof. For readability purposes, let us abbreviate $k := \sqrt{\log_2 n}$.

First, let us consider the size $|X_c|$ for some fixed $c \in [k]$. By the same arguments as used in the proof of Lemma 5.3 we have for any $i \in [8e\log_2^{3/2} n/\log_2\log_2 n]$ that

$$\Pr[\textsc{Lo}_{z,\sigma}(y^i) = \ell + c - 1] = (1 - \tfrac{1}{k})^{c-1}\tfrac{1}{k} \geq (1 - \tfrac{1}{k})^{k-1}\tfrac{1}{k} \geq \tfrac{1}{ek}.$$

Thus, $\mathrm{E}[|X_c|] \geq 8\log_2 n/\log_2\log_2 n$ by the linearity of expectation.

If we set $Y_{i,c} = 1$ if $\textsc{Lo}_{z,\sigma}(y^i) = \ell + c - 1$ and $Y_{i,c} = 0$ otherwise, we have $|X_c| := \sum_{i=1}^{8e\log_2^{3/2} n/\log_2\log_2 n} Y_{i,c}$. In particular, $|X_c|$ is the sum of independent random variables. We can thus apply Chernoff's bounds (cf. Lemma 2.12 , (2.2)) to bound the deviation of $|X_c|$ from its expectation and obtain

$$\Pr\left[|X_c| < \tfrac{1}{2}\mathrm{E}[|X_c|]\right] \leq \exp\left(-\tfrac{1}{8}\mathrm{E}[|X_c|]\right) \leq \exp\left(-\log_2 n/\log_2\log_2 n\right) \leq 1/\log_2 n,$$

where the last inequality follows from $\log_2 n/\log_2\log_2 n > \log\log_2 n$ for large enough n. By a simple union bound (cf. Lemma 2.7) we conclude

$$\Pr\left[\forall c \in [\sqrt{\log_2 n}] : |X_c| \geq 4\log_2 n/\log_2\log_2 n\right] \geq 1 - 1/\sqrt{\log_2 n} = 1 - o(1).$$

□

These sets X_c are large enough to identify $\sigma(\ell+c)$.

Lemma 5.6. *Let ℓ, (x,y), and y' be as in Lemma 5.5 and let $t := 4\log_2 n/\log_2\log_2 n$.*

For any $c \in [\sqrt{\log_2 n}]$ let X_c be a set of at least t bit strings $y^1(c),\ldots,y^{|X_c|}(c)$ with fitness $\text{Lo}_{z,\sigma}(y^i(c)) = \ell+c-1$, which are sampled from y' by independently flipping each non-encoding bit position $j \in [n]\backslash\mathcal{B}(x,y)$ with probability $1/\sqrt{\log_2 n}$.

Then we have, with probability at least $1-o(1)$, that for all $c \in [\sqrt{\log_2 n}]$ there exists only one non-encoding $j := j_{\ell+c} \in [n]\backslash\mathcal{B}(x,y)$ with $y'_j = 1 - y^i_j(c)$ for all $i \in [|X_c|]$. Clearly, $j = \sigma(\ell+c)$.

Proof. Let us first consider some fix value $c \in [\sqrt{\log_2 n}]$. For any $i \in [|X_c|]$ we have, by definition, that $\text{Lo}_{z,\sigma}(y^i(c)) = \ell+c-1$. Thus, it must hold that $y^i_{\sigma(\ell+c)}(c) = 1-y'_{\sigma(\ell+c)}$ and $y^i_{\sigma(j)}(c) = y'_{\sigma(j)}$ for all $j < \ell+c$. Let

$$\begin{aligned}\mathcal{I}_{\ell+c} &:= [n]\backslash(\mathcal{B}(x,y)\cup\{\sigma(\ell+1),\ldots,\sigma(\ell+c)\})\\ &= [n]\backslash\{\sigma(1),\ldots,\sigma(\ell+c)\}\,.\end{aligned}$$

Since all bit flips are mutually independent, we have for any $i \in [|X_c|]$ and any fixed $j \in \mathcal{I}_{\ell+c}$ that the entry $y^i_j(c)$ in the j-th bit position equals $1-y'_j$ with probability $1/\sqrt{\log_2 n}$. Note that this remains true despite the fact that we condition on $\text{Lo}_{z,\sigma}(y^i) = \ell+c-1$.

Thus, for any $j \in \mathcal{I}_{\ell+c}$ we have

$$\begin{aligned}\Pr[\forall i \in [|X_c|] : y^i_j(c) = 1-y'_j] &\le (1/\sqrt{\log_2 n})^{|X_c|}\\ &\le (1/\sqrt{\log_2 n})^{4\log_2 n/\log_2\log_2 n}\\ &= 2^{-2\log_2 n} = 1/n^2\,.\end{aligned}$$

By the union bound, the probability that there exists a $j \in \mathcal{I}_{\ell+c}$ with $y^i_j(c) = 1-y'_j$ for all $i \in [|X_c|]$ is bounded from above by $1/n$.

And by again applying a union bound we find that

$$\Pr[\exists c \in [\sqrt{\log_2 n}]\exists j \in \mathcal{I}_{\ell+c}\forall i \in [|X_c|] : y^i_j(c) = 1-y'_j] \le \sqrt{\log_2 n}/n = o(1)\,.$$

□

Combining Lemma 5.5 with Lemma 5.6 we immediately gain the following.

Corollary 5.7. *Let $\ell, (x,y), y'$, and $y^i, i = 1,\ldots,8e\log_2^{3/2} n/\log_2\log_2 n$, be as in Lemma 5.5.*

With probability at least $1-o(1)$ we have that for all $c \in [\sqrt{\log_2 n}]$ there exists only one non-encoding $j := j_{\ell+c} \in [n]\backslash\mathcal{B}(x,y)$ with $y'_j = 1-y^i_j$ for all $i \in [8e\log_2^{3/2} n/\log_2\log_2 n]$ with $\text{Lo}_{z,\sigma}(y^i) = \ell+c-1$. Clearly, $j = \sigma(\ell+c)$.

We are now ready to prove Theorem 5.1. As mentioned above, the proof also shows that the statement remains correct if we consider the unbiased black-box model of arbitrary arity ($*$-ary unbiased black-box model).

Proof of Theorem 5.1. We need to show that there exists an algorithm which maximizes any (a priori unknown) function $\textsc{Lo}_{z,\sigma} \in \textsc{LeadingOnes}_n^*$ using, on average, $O(n \log n / \log\log n)$ queries.

To ease reading, let us fix some function $f = \textsc{Lo}_{z,\sigma} \in \textsc{LeadingOnes}_n^*$ to be maximized by the algorithm.

First, let us give a rough idea of our algorithm, Algorithm 11. A detailed analysis can be found below.

The main idea is the following. We maximize f block-wise, where each block has a length of $\sqrt{\log_2 n}$ bits. Due to the influence of the permutation σ on f, these bit positions are a priori unknown. Assume for the moment that we have an ℓ-encoding pair (x, y), where $\ell \in [0..n - \lceil\sqrt{\log_2 n}\,\rceil]$. In the beginning we have $\ell = 0$ and $y = x \oplus (1, \ldots, 1)$, the bitwise complement of x. To find an $(\ell + \sqrt{\log_2 n})$-encoding pair, we first create a string y' with objective value $f(y') \geq \ell + \sqrt{\log_2 n}$. By Corollary 5.4, this requires on average $O(\log n)$ queries. Next, we need to identify the $\sqrt{\log_2 n}$ $f(y)$-critical bit positions $\sigma(\ell+1), \ldots, \sigma(\ell + \sqrt{\log_2 n})$. To this end, we sample enough bit strings such that we can unambiguously identify these bit positions. As we shall see, this requires on average $O(\log^{3/2} n / \log\log n)$ queries. After identifying the critical bits, we update (x, y) to a $(\ell + \sqrt{\log_2 n})$-encoding pair. Since we need to optimize $n / \sqrt{\log_2 n}$ such blocks of size $\sqrt{\log_2 n}$, the overall expected optimization time is $O(n \log n / \log\log n)$.

Let us now present a more detailed analysis. We start by querying two complementary bit strings x, y. By swapping x with y in case $f(y) \geq f(x)$, we ensure that $f(x) > f(y) = 0$. This gives us a 0-encoding pair.

Let an ℓ-encoding pair (x, y), for some fixed value $\ell \in [0..n - \lceil\sqrt{\log_2 n}\,\rceil]$, be given. We show how from this we find an $(\ell + \sqrt{\log_2 n})$-encoding pair in an expected number of $O(\log^{3/2} n / \log\log n)$ queries.

As mentioned above, we first find a bit string y' with objective value $f(y') \geq \ell + \sqrt{\log_2 n}$. We do this by running Algorithm 10, the (x, y)-encoded $(1+1)$ EA with mutation probability $1/\sqrt{\log_2 n}$ until we obtain such a bit string y'. By Corollary 5.4 this takes, on average, $O(\log n)$ queries.

Next we want to identify the $\sqrt{\log_2 n}$ ℓ-critical bit positions $\sigma(\ell+1), \ldots, \sigma(\ell + \sqrt{\log_2 n})$. To this end, we query in the i-th iteration of the second phase, a bit string y^i which has been created from y' by flipping each non-encoding bit $y'_j, j \in [n] \backslash \mathcal{B}(x, y)$ independently with probability $1/\sqrt{\log_2 n}$. If $f(y^i) = \ell + c - 1$ for some $c \in [\sqrt{\log_2 n}]$, we update $X_{\ell+c} \leftarrow X_{\ell+c} \cup \{y^i\}$, the set of all queries with objective value $\ell + c - 1$, and we compute $\mathcal{J}_{\ell+c} := \{j \in [n] \backslash \mathcal{B}(x, y) \mid \forall w \in X_{\ell+c} : w_j = 1 - y'_j\}$, the set of candidates for $\sigma(\ell + c)$. We do so until we find $|\mathcal{J}_{\ell+c}| = 1$ for all $c \in [\sqrt{\log_2 n}]$. By Corollary 5.7 this takes, on average, at most $8e \log_2^{3/2} n / \log_2 \log_2 n$ queries.

Thus, all we need to do in the third step is to update (x, y) to an $(\ell + \sqrt{\log_2 n})$-encoding pair by exploiting the information gathered in the second phase. For any $c \in [\sqrt{\log_2 n}]$ let us denote the element in $\mathcal{J}_{\ell+c}$ by $j_{\ell+c}$. We go through the positions $\sigma(\ell+1), \ldots, \sigma(\ell + \sqrt{\log_2 n})$ one after the other and either we update $y \leftarrow y \oplus e^n_{j_{\ell+c}}$ (if $f(y) < f(x)$), and we update $x \leftarrow x \oplus e^n_{j_{\ell+c}}$ otherwise. It is easy to verify that after $\sqrt{\log_2 n}$ such steps we have $f(x) \geq \ell + \sqrt{\log_2 n}$ and $f(y) \geq \ell + \sqrt{\log_2 n}$. It remains to swap $(x, y) \leftarrow (y, x)$ in case $f(y) > f(x)$ in order to obtain an $(\ell + \sqrt{\log_2 n})$-encoding

Algorithm 11: A $*$-ary unbiased black-box algorithm for maximizing $f \in$ LeadingOnes$_n^*$. Recall that we have defined $\mathcal{J}_{\ell+c} := \{j \in [n] \backslash \mathcal{B}(x,y) \mid \forall w \in X_{\ell+c} : w_j = 1 - y'_j\}$.

```
Initialization:
    for $i = 1, \dots, n$ do $X_i \leftarrow \emptyset$;
    Sample $x \in \{0,1\}^n$ uniformly at random and query $f(x)$;
    Set $y \leftarrow x \oplus (1, \dots, 1)$ and query $f(y)$;
    if $f(y) \geq f(x)$ then $(x,y) \leftarrow (y,x)$;
Optimization:
    while $|\mathcal{B}(x,y)| \leq \lfloor \frac{n}{\lceil \sqrt{\log_2 n} \rceil} \rfloor \lceil \sqrt{\log_2 n} \rceil$ do
        Set $\ell \leftarrow |\mathcal{B}(x,y)|$;
        Apply Algorithm 10 with input $(x,y)$ and mutation probability
        $1/\sqrt{\log_2 n}$ until it outputs a bit string $y'$ with $f(y') \geq \ell + \sqrt{\log_2 n}$;
        Initialize $i \leftarrow 1$;
        while $\exists c \in [\sqrt{\log_2 n}] : |\mathcal{J}_{\ell+c}| > 1$ do
            Sample $y^i \leftarrow$ random$(y', x, y, 1/\sqrt{\log_2 n})$ and query $f(y^i)$;
            if $f(y^i) \in [\ell, \dots, \ell + \sqrt{\log_2 n} - 1]$ then
                Update $X_{f(y^i)+1} \leftarrow X_{f(y^i)+1} \cup \{y^i\}$;
                Update $\mathcal{J}_{f(y^i)}$;
            $i \leftarrow i + 1$;
        for $c = 1, \dots, \sqrt{\log_2 n}$ do update$(x, y, y', X_{\ell+c})$;
        if $f(y) > f(x)$ then $(x,y) \leftarrow (y,x)$;
    Apply Algorithm 10 with input $(x,y)$ and mutation probability $1/\sqrt{\log_2 n}$
    until it queries for the first time a string $y'$ with $f(y') = n$;
```

pair (x,y).

This shows how, given a ℓ-encoding pair (x,y), we find an $(\ell + \sqrt{\log_2 n})$-encoding pair in $O(\log n) + O(\log^{3/2} n / \log\log n) + O(\sqrt{\log n}) = O(\log^{3/2} n / \log\log n)$ queries.

By definition of Algorithm 11, all bit positions in $\mathcal{B}(x,y)$ remain untouched in all further iterations of the algorithm. Thus, in total, we need to optimize $\lfloor \frac{n}{\lceil \sqrt{\log_2 n} \rceil} \rfloor$ blocks of size $\lceil \sqrt{\log_2 n} \rceil$ until we have a $(\lfloor \frac{n}{\lceil \sqrt{\log_2 n} \rceil} \rfloor \lceil \sqrt{\log_2 n} \rceil)$-encoding pair (x,y). For each block, the expected number of queries needed to fix the corresponding bit positions is $O(\log^{3/2} n / \log\log n)$. By linearity of expectation this yields a total expected optimization time of $O(n/\sqrt{\log n}) O(\log^{3/2} n / \log\log n) = O(n \log n / \log\log n)$ for optimizing the first $k := \lfloor \frac{n}{\lceil \sqrt{\log_2 n} \rceil} \rfloor \lceil \sqrt{\log_2 n} \rceil$ bit positions $\sigma(1), \dots, \sigma(k)$.

The remaining $n - k \leq \lfloor \sqrt{\log_2 n} \rfloor$ bit positions can be found by Algorithm 10 in an expected number of $O(\log n)$ queries (Corollary 5.4). This does not change the asymptotic number of queries needed to identify z.

Putting everything together, we have shown that Algorithm 11 optimizes any function Lo$_{z,\sigma} \in$ LeadingOnes$_n^*$ in an expected number of $O(n \log n / \log\log n)$ queries. It is not difficult to verify that all variation operators are unbiased. □

Algorithm 12: Subroutine update$(x, y, y', X_{\ell+c})$

```
1 Input: An $\ell$-encoding pair $(x,y)$, a bit string $y'$ with $f(y') \geq \ell + \sqrt{\log_2 n}$, and,
  a set $X_{\ell+c}$ of samples $w$ with $f(w) = \ell + c - 1$ such that
  $|\mathcal{J}_{\ell+c}| = |\{j \in [n] \backslash \mathcal{B}(x,y) \mid \forall w \in X_c : w_j = 1 - y'_j\}| = 1$;
2 if $f(y) \leq f(x)$ then
3     Set $y \leftarrow y \oplus e^n_{\mathcal{J}(\ell+c)}$ and query $f(y)$;
4 else
5     Set $x \leftarrow x \oplus e^n_{\mathcal{J}(\ell+c)}$ and query $f(x)$;
```

5.3. The Unbiased Black-Box Complexity of LeadingOnes$^*_n$

Next we show how a slight modification of Algorithm 11 yields a 3-ary unbiased black-box algorithm with the same asymptotic expected optimization time.

Theorem 5.8. *The 3-ary unbiased black-box complexity of* LeadingOnes$^*_n$ *is* $O(n \log n / \log \log n)$.

Proof. Key for this result is the fact that, instead of storing for any $c \in [\sqrt{\log_2 n}]$ the whole query history $X_{\ell+c}$, we need to store only one additional bit string $x^{\ell+c}$ to keep all the information needed to determine $\sigma(\ell + c)$.

Algorithm 13 gives the full algorithm. Here, the bit string update2$(w, y', x^{\ell+c})$ is defined via $\left(\texttt{update2}(w, y', x^{\ell+c})\right)_i = w_i$ if $i \in [n] \backslash \mathcal{B}(y', x^{\ell+c})$ and $\left(\texttt{update2}(w, y', x^{\ell+c})\right)_i = 1 - w_i$ for $i \in \mathcal{B}(y', x^{\ell+c})$.

Note that, throughout the run of the algorithm, the pair $(y', x^{\ell+c})$, or more precisely, the set $\mathcal{B}(y', x^{\ell+c})$ encodes which bit positions j are still possible to equal $\sigma(\ell + c)$. Expressing the latter in the notation used in the proof of Theorem 5.1, we have in any iteration of the first **while**-loop that for all $i \in [n]$ it holds $y'_i = x^{\ell+c}_i$ if and only if $i \in \mathcal{J}_{\ell+c}$. This can be seen as follows. In the beginning, we only know that $\sigma(\ell + c) \neq \mathcal{B}(x, y)$. Thus, we initialize $x^{\ell+c}_i \leftarrow 1 - y'_i$ if $i \in \mathcal{B}(x, y)$ and $x^{\ell+c}_i \leftarrow y'_i$ for $i \in [n] \backslash \mathcal{B}(x, y)$. In each iteration of the second **while**-loop, we update $x^{\ell+c}_i \leftarrow 1 - y'_i$ if $\sigma(\ell + c) = i$ can no longer hold, i.e., if we have sampled a bit string w with $f(w) = \ell + c - 1$ and $w_i = y'_i$. □

5.4. LeadingOnes$^*_n$ in the Ranking-Based Models

In Section 8 we shall introduce two ranking-based versions of the black-box complexity notion: the *unbiased ranking-based* and the *unrestricted ranking-based black-box complexity*. Instead of querying the absolute fitness values $f(x)$, in the ranking-based model, the algorithms may only query the ranking of y among all previously queried search points, cf. Section 8 for the motivation and formal definitions. We briefly remark the following.

Algorithm 13: A 3-ary unbiased black-box algorithm for maximizing $f \in \textsc{LeadingOnes}^*_n$.

```
1  Initialization:
2    Sample $x \in \{0,1\}^n$ uniformly at random and query $f(x)$;
3    Set $y \leftarrow x \oplus (1,\ldots,1)$ and query $f(y)$;
4    if $f(y) \geq f(x)$ then $(x,y) \leftarrow (y,x)$;
5  Optimization:
6    while $|\mathcal{B}(x,y)| \leq \lfloor \frac{n}{\lceil\sqrt{\log_2 n}\rceil} \rfloor \lceil\sqrt{\log_2 n}\rceil$ do
7      $\ell \leftarrow |\mathcal{B}(x,y)|$;
8      Apply Algorithm 10 with input $(x,y)$ and mutation probability
       $1/\sqrt{\log_2 n}$ until it outputs a bit string $y'$ with $f(y') \geq \ell + \sqrt{\log_2 n}$;
9      for $c = 1,\ldots,\sqrt{\log_2 n}$ do
10       for $i = 1,\ldots,n$ do
11         if $i \in \mathcal{B}(x,y)$ then $x_i^{\ell+c} \leftarrow 1 - y'_i$ else $x_i^{\ell+c} \leftarrow y'_i$;
12     while $\exists c \in [\sqrt{\log_2 n}] : |\mathcal{B}(x^{\ell+c}, y')| > 1$ do
13       Sample $w \leftarrow$ random$(y', x, y, 1/\sqrt{\log_2 n})$ and query $f(w)$;
14       if $\exists c \in [\sqrt{\log_2 n}] : f(w) = \ell + c - 1$ then
15         for $i = 1,\ldots,n$ do if $x_i^{\ell+c} = y'_i = w_i$ then $x_i^{\ell+c} \leftarrow 1 - y'_i$;
16     for $c = 1,\ldots,\sqrt{\log_2 n}$ do
17       if $f(y) \leq f(x)$ then update2$(y, y', x^{\ell+c})$ else update2$(x, y', x^{\ell+c})$;
18     if $f(y) > f(x)$ then $(x,y) \leftarrow (y,x)$;
19   Apply Algorithm 10 with input $(x,y)$ and mutation probability $1/\sqrt{\log_2 n}$
     until it queries for the first time a string $y'$ with $f(y') = n$;
```

Theorem 5.9. *The 3-ary unbiased ranking-based black-box complexity of* $\textsc{LeadingOnes}^*_n$ *is* $O(n \log n / \log\log n)$.

This theorem immediately implies that the unrestricted ranking-based black-box complexity of $\textsc{LeadingOnes}^*_n$ is $O(n \log n / \log\log n)$ as well.

Theorem 5.9 can be proven by combining the Algorithm 13 presented in the proof of Theorem 5.8 with a sampling strategy as used in Lemma 5.5. Although the latter is not optimal, it suffices to show that after sampling $O(\log^{3/2} n / \log\log n)$ such samples, we can identify the ranks of $f(\ell+1),\ldots,f(\ell+\sqrt{\log_2 n})$, with probability at least $1-o(1)$. We do the sampling right after line 8 of Algorithm 13. After having identified the ranks of $f(\ell+1),\ldots,f(\ell+\sqrt{\log_2 n})$, we can continue as in Algorithm 13.

5.5. Conclusions

We have shown that there exists a 3-ary unbiased ranking-based black-box algorithm, which optimizes any function $\textsc{Lo}_{z,\sigma} \in \textsc{LeadingOnes}^*_n$ in an expected number of $O(n \log n / \log\log n)$ queries. This establishes a new upper bound on the unrestricted and the 3-ary unbiased black-box complexity of $\textsc{LeadingOnes}^*_n$.

Our result raises several questions for future research. The obvious one is to close the gap between the current best lower bound of $\Omega(n)$ (cf. [DJW06, Theorem 6] and Theorem 8.16 in Section 8) and our upper bound of $O(n \log n / \log \log n)$. Currently, we cannot even prove an $\omega(n)$ lower bound. Secondly, it would also be interesting to know whether the gap between the 2-ary and the 3-ary unbiased black-box model is an artifact of our analysis or whether 3- and higher arity operators are truly more powerful than binary ones.

6 Too Fast Unary Unbiased Black-Box Algorithms

The results presented in the previous two sections indicate that the unbiased black-box model by Lehre and Witt does not seem to reflect the tractability of a problem class once we increase the arities of the variation operators to be called by the black-box algorithms to some value strictly larger than one. However, we have not yet seen an example with a strong mismatch between the *unary* unbiased black-box complexity of a problem class and the complexity that one would expect from previous runtime analyses. The aim of this section is to provide two such examples for which the unary unbiased black-box complexities seem artificially small.

For the $\textsc{Jump}_{n,k}$ function class, typically needing $\Omega(n^k)$ function evaluations to be optimized via a randomized search heuristic, we present a unary unbiased black-box algorithm that maximizes every such function using only $O(n \log n)$ function evaluations.

Similarly, for an NP-hard subclass of the well-known $\textsc{Partition}$ problem, we show that it can be solved by a unary unbiased black-box algorithm needing only $O(n \log n)$ oracle queries.

These results indicate that, in addition to the progress made in [LW10a], more effort is needed to define a complexity model that gives realistic complexity statements for a broader class of problems.

The results presented in this section are based on the conference publication [DKW11]. They are joint work with Benjamin Doerr and Timo Kötzing.

6.1. Jump Functions

In this section we analyze the unbiased black-box complexity of the so-called *jump* functions: for all $k < n/2$, let $\textsc{Jump}_{n,k}$ denote the fitness function defined by

$$\textsc{Jump}_{n,k} : \{0,1\}^n \to [0..n], x \mapsto \begin{cases} n, & \text{if } |x|_1 = n\,, \\ |x|_1, & \text{if } k < |x|_1 < n-k\,, \\ 0, & \text{otherwise.} \end{cases}$$

The functions $\textsc{Jump}_{n,k}$ are a slight variant of similar jump-type functions that were introduced by Droste, Jansen, and Wegener [DJW02] in one of the first papers that rigorously analyzes the runtime of the (1+1) EA on several function classes. The definition given above can be found in [LW10a]. Lehre and Witt argue that such situations, in which the algorithm needs to "jump" a Hamming distance of $k \geq 2$ for an improvement, can be frequently observed in combinatorial optimization problems. This was observed, for example, in the work by Neumann and Wegener [NW07] on the Minimum Spanning Tree (MST) problem (cf. Section 9). The $\textsc{Jump}_{n,k}$ functions are interesting in that they provide examples for which the (1+1) EA needs to jump a Hamming distance of $k+1$. This requires, on average, $\Theta(n^{k+1})$ queries. Lehre and Witt call this a "hierarchy of difficult problems". A similar term was used in [DJW02]. However, here in this work we show that for every constant k, the unbiased black-box complexity of $\textsc{Jump}_{n,k}$ is surprisingly low. Already for the unary unbiased black-box model it is of order $O(n \log n)$. This shows that the hierarchy of the $\textsc{Jump}_{n,k}$ does not carry over to the unary unbiased black-box model. Of course, this does not rule out the fact that such a hierarchy exists and in fact we strongly believe that for any $k \in [n]$ such function classes of complexity $\Theta(n^k)$ exist.

Key to our analysis is the following lemma. It shows that one can compute, with high probability, the Hamming-weight of any search point x (i.e., the number of ones in x, or, equivalently, the $\textsc{OneMax}$-value $\textsc{OneMax}(x) = |x|_1$) with few black-box calls to $\textsc{Jump}_{n,k}$. With this, we can orient ourselves on the large plateau surrounding the optimum and thus revert to the problem of optimizing $\textsc{OneMax}$.

We collect these computations in a subroutine, to be called by the black-box algorithm certifying our claim that for optimizing any $\textsc{Jump}_{n,k}$ function $O(n \log n)$ function evaluations suffice. This is Theorem 6.2.

Lemma 6.1. *For all constants k and c, there is a unary unbiased subroutine s using $c+1$ queries to $\textsc{Jump}_{n,k}$ such that, for all bit strings x, $s(x) = \textsc{OneMax}(x)$ with probability $1 - O(n^{-c})$.*

Proof. We use a unary unbiased variation operator $\texttt{flip}_k$, which samples uniformly a k-neighbor (a bit string which differs in exactly k positions) of the argument. This is unbiased as can be easily verified using the fact that for all $x, y, z \in \{0,1\}^n$ the Hamming distance $|(z \oplus y) \oplus (x \oplus y)|_1$ of $z \oplus y$ and $x \oplus y$ equals the Hamming distance $|z \oplus x|_1$ of z and x and that for all $\sigma \in S_n$ the Hamming distance of $\sigma(x)$ and $\sigma(z)$ equals the one of x and z.

Next we give the subroutine s, which uses $\textsc{Jump}_{n,k}$ to approximate $\textsc{OneMax}$ as desired, see Algorithm 14. Intuitively, the subroutine samples c bit strings in the k-

Algorithm 14: Simulation of OneMax using the jump function.

```
1 subroutine $s(x)$ is
2     if $\textsc{Jump}_{n,k}(x) \neq 0$ then Output $\textsc{Jump}_{n,k}(x)$;
3     $M \leftarrow \{\textsc{Jump}_{n,k}(\texttt{flip}_k(x)) \mid i \in [c]\}$;
4     if $\max(M) < n/2$ then $m \leftarrow \max(M) - k$;
5     else $m \leftarrow \min(M \setminus \{0\}) + k$;
6     Output $m$;
```

neighborhood of x. If $|x|_1 \geq n-k$ then it is likely that at least once only one-entries of x have been flipped. This yields a $\textsc{Jump}_{n,k}$-value of $|x|_1 - k$. As no sample will have a lower $\textsc{Jump}_{n,k}$-value, adding k to the minimum non-0 fitness of one of the sampled bit strings gives the desired output. The case of x with $|x|_1 \leq k$ is analogous.

Clearly, the subroutine is correct with certainty on all x with $k < |x|_1 < n-k$. The other two cases are symmetric, so let x with $|x|_1 \geq n-k$ be given. Obviously, the output of the subroutine is correct if and only if at least one of the c samples flips only one-entries in x. We denote the probability of this event with p. We start by bounding the probability that a single sample flips only ones. Since we assume $|x|_1 \geq n-k$ this probability p_1 can be bounded as follows.

$$\begin{aligned} p_1 &\geq \left(\frac{n-k}{n}\right) \cdot \left(\frac{n-k-1}{n-1}\right) \cdots \left(\frac{n-k-(k-1)}{n-(k-1)}\right) = \prod_{i=0}^{k-1} \left(1 - \frac{k}{n-i}\right) \\ &\geq \left(1 - \frac{k}{n-k}\right)^k \geq \left(1 - \frac{k^2}{n-k}\right), \end{aligned}$$

where in the last step we use Bernoulli's inequality (cf. Lemma 2.9). Thus, we have

$$p = 1 - (1-p_1)^c \geq 1 - \left(\frac{k^2}{n-k}\right)^c .$$

□

Now we can use the subroutine from Lemma 6.1 to turn results on the unbiased black-box complexity for OneMax into results on $\textsc{Jump}_{n,k}$ for constant k.

Theorem 6.2. *For constant k, the unbiased black-box complexity of $\textsc{Jump}_{n,k}$ is*

- $O(n \log n)$*, for unary variation operators.*
- $O(n/\log m)$*, for m-ary variation operators with $2 \leq m \leq n$.*
- $O(n/\log n)$*, for $*$-ary variation operators.*

Proof. Note that the above black-box complexities claimed for $\textsc{Jump}_{n,k}$ are shown for OneMax in [LW10a] for the unary case and for arities larger than one we have shown them in Section 4, Theorem 4.4. We use Lemma 6.1 with $c = 4$ and run the unbiased black-box algorithms of the appropriate arity for OneMax; all sampled bit strings are evaluated using the subroutine s. Thus, this algorithm samples as if working on

OneMax and finds the all-ones bit string after the desired number of iterations. Note that, for up to $n \log n$ uses of s, we expect no more than $n \log n O(n^{-4}) \leq O(n^{-2})$ incorrect evaluations of s. Therefore, there is a small chance of failing, and the claim follows from Corollary 2.15. □

Note that the subroutine from Lemma 6.1 requires to know the parameter k; however, this subroutine can be modified to work without that knowledge as follows. The first time that the subroutine samples a search point with fitness 0 it will determine k; after knowing k, it will work as before (and before sampling a search point of fitness 0, it does not need to know). The parameter k is determined by sampling sufficiently many i-neighbors of the search point with fitness 0, starting with $i = 1$ and stopping when a search point with fitness $\neq 0$ is found. This search point will have maximum fitness among all non-optimal search points, equal to $n - k - 1$. From this fitness and n, the subroutine can infer k.

There may be cases when one of the algorithms implicit in the proof of Theorem 6.2 never samples a search point with fitness 0 and does not have to determine k. In this case, such an algorithm will optimize the target function without completely learning it. However, in a second phase after finding the optimum, an algorithm could determine k with a binary search, as k equals the largest distance from the optimum at which all and any search point has fitness 0. This phase requires $O(\log n)$ queries.

6.2. Partition

One objection concerning the unrestricted black-box model is the fact that there are NP-hard problems, which have a polynomial black-box complexity. As an example let us consider the well-known Clique problem. Although it is known to be NP-hard, Droste et al. [DJW06] show that its optimization version MaxClique has an unrestricted black-box complexity of at most $\binom{n}{2} + 1 = \Theta(n^2)$. That is, even the optimization problem can be solved by an unrestricted black-box algorithm using only $\Theta(n^2)$ queries.

In this section we prove that in the unbiased black-box model, too, there are NP-hard problems that have a polynomial black-box complexity. Even more, we show that this is already true for the unary unbiased black-box model.

To this end we consider an NP-hard subclass of the Partition problem. Whereas the decision version of the Partition problem asks the question "Given a multiset $\mathcal{I}$ of positive integers ("weights"), is it possible to split the set into two disjoint subsets $\mathcal{I} = \mathcal{I}_0 \dot\cup \mathcal{I}_1$ such that $\sum_{w \in \mathcal{I}_0} w = \sum_{w \in \mathcal{I}_1} w$?", the optimization version asks for a partition $(\mathcal{I}_0, \mathcal{I}_1)$ of $\mathcal{I}$ such that the difference $|\sum_{w \in \mathcal{I}_0} w - \sum_{w \in \mathcal{I}_1} w|$ is minimized.

It is known that Partition permits heuristics which solve many instances of the problem in a polynomial number of fitness evaluations. For example, Frenk and Kan [FK86] showed that the greedy approach converges to optimality almost surely for reasonably chosen random instances. Furthermore, greedy approaches are known to deliver in a polynomial number of queries solutions of good approximation quality. For example, Witt [Wit05] has shown that both Randomized Local Search (Algorithm 1) and the (1+1) EA (Algorithm 2) need at most $O(n^2)$ iterations until they reach for the first time a solution of approximation quality 4/3. For analyzing approximation ratios, clearly, one has to resort to a different objective function. The 4/3-approximation result by Witt holds for the objective "minimize $\max\{\sum_{w\in\mathcal{I}_0} w, \sum_{w\in\mathcal{I}_1} w\}$", where the minimum ranges over all partitions $(\mathcal{I}_0, \mathcal{I}_1)$ of $\mathcal{I}$.

The decision version of PARTITION has been shown to be NP-complete, cf. [Kar72, GJ90]. It is actually one of the most famous NP-complete problems. Thus, assuming $P \neq NP$, the decision problem does not allow a polynomial-time algorithm. Note that $P \neq NP$ would also imply that the optimization problem cannot be solved in polynomial time.

It is easily seen that PARTITION remains NP-hard if we restrict the problem to instances with all weights distinct.

Lemma 6.3. PARTITION *remains NP-complete when restricted to instances $\mathcal{I}$ with $v \neq w$ for all $v, w \in \mathcal{I}$.*

In the following we give a short reasoning for Lemma 6.3. Although it is probably well known, we were unable to find a formal proof in the computer science literature.

Proof. Let $\mathcal{I}$ be an instance of the general problem class PARTITION (i.e., $\mathcal{I}$ may contain multiples of the same integer). Let $n := |\mathcal{I}|$ We create from $\mathcal{I}$ a new instance $\mathcal{I}'$ of size n by the procedure described below. $\mathcal{I}'$ will have pairwise different weights only and we will prove that an optimal partition for $\mathcal{I}'$ immediately yields an optimal partition for the original input $\mathcal{I}$.

To construct $\mathcal{I}'$, we first (arbitrarily) enumerate the values in $\mathcal{I}$ by φ, i.e., $\varphi : \mathcal{I} \to [n]$ is a bijection. We then set $c := n^3$ and we let $\mathcal{I}' := \{cw + \varphi(w) \mid w \in \mathcal{I}\} \cup [n]$. By construction, all integers in $\mathcal{I}'$ do have different weights.

Let $(\mathcal{I}_0', \mathcal{I}_1')$ be an optimal partition for $\mathcal{I}'$. We set $\mathcal{I}_0 := \{(w - \varphi(w))/c \mid w \in \mathcal{I}_0' \backslash [n]\}$ and $\mathcal{I}_1 := \{(w - \varphi(w))/c \mid w \in \mathcal{I}_1' \backslash [n]\}$. We use contraposition to show that $(\mathcal{I}_0, \mathcal{I}_1)$ is an optimal solution for $\mathcal{I}$. More precisely, we show that any partition of $\mathcal{I}$ which is better than $(\mathcal{I}_0, \mathcal{I}_1)$ gives rise to a partition of $\mathcal{I}'$ which is better than $(\mathcal{I}_0', \mathcal{I}_1')$, contradicting the choice of $(\mathcal{I}_0', \mathcal{I}_1')$. Thus, if $(\mathcal{I}_0', \mathcal{I}_1')$ is optimal, so is $(\mathcal{I}_0, \mathcal{I}_1)$.

To prove the claim, assume that there exists a solution (O_0, O_1) for $\mathcal{I}$ with $|\sum_{w\in O_0} w - \sum_{w\in O_1} w| < |\sum_{w\in\mathcal{I}_0} w - \sum_{w\in\mathcal{I}_1} w|$. We set $O_0' := \{c \cdot w + \varphi(w) \mid w \in O_0\} \cup \{\varphi(w) \mid w \in O_1\}$ and $O_1' := \{c \cdot w + \varphi(w) \mid w \in O_1\} \cup \{\varphi(w) \mid w \in O_0\}$. Note that $c\left(\sum_{w\in O_0} w - \sum_{w\in O_1} w\right) = \sum_{w\in O_0'} w - \sum_{w\in O_1'} w$.

We assume without loss of generality that $\sum_{w\in\mathcal{I}_0} w - \sum_{w\in\mathcal{I}_1} w > 0$. Then $\sum_{w\in\mathcal{I}_0'} w - \sum_{w\in\mathcal{I}_1'} w > 0$ since $c\left(\sum_{w\in\mathcal{I}_0} w - \sum_{w\in\mathcal{I}_1} w\right) > 0$, $\sum_{w\in\mathcal{I}_0} \varphi(w) - \sum_{w\in\mathcal{I}_1} \varphi(w) + \sum_{w\in\mathcal{I}_0'\cap[n]} w - \sum_{w\in\mathcal{I}_1'\cap[n]} w > \sum_{w\in\mathcal{I}_0} \varphi(w) - \sum_{w\in\mathcal{I}_1} \varphi(w) - \sum_{i\in[n]} i > -(n^2 + n) > -c$ (by definition of c and assuming $n \geq 2$) and, thus,

$$\sum_{w\in\mathcal{I}_0'} w - \sum_{w\in\mathcal{I}_1'} w = c\left(\sum_{w\in\mathcal{I}_0} w - \sum_{w\in\mathcal{I}_1} w\right) + \sum_{w\in\mathcal{I}_0} \varphi(w) - \sum_{w\in\mathcal{I}_1} \varphi(w)$$
$$+ \sum_{w\in\mathcal{I}_0'\cap[n]} w - \sum_{w\in\mathcal{I}_1'\cap[n]} w > c - c = 0\,.$$

Similarly we obtain

$$\begin{aligned}|\sum_{w\in\mathcal{I}'_0} w - \sum_{w\in\mathcal{I}'_1} w| > c(\sum_{w\in\mathcal{I}_0} w - \sum_{w\in\mathcal{I}_1} w) - c = c(\sum_{w\in\mathcal{I}_0} w - \sum_{w\in\mathcal{I}_1} w - 1)\\ \geq c|\sum_{w\in O_0} w - \sum_{w\in O_1} w| = |\sum_{w\in O'_0} w - \sum_{w\in O'_1} w|\,,\end{aligned}$$

contradicting the optimality of $(\mathcal{I}'_0, \mathcal{I}'_1)$ for $\mathcal{I}'$ (which implies $|\sum_{w\in\mathcal{I}'_0} w - \sum_{w\in\mathcal{I}'_1} w| \leq |\sum_{w\in O'_0} w - \sum_{w\in O'_1} w|$). □

In the following, let Partition$_{n,\neq}$ be the subclass of Partition instances with n pairwise different weights. That is, for all $\mathcal{I} \in$ Partition$_{n,\neq}$ we have $|\mathcal{I}| = n$ and for all $w \in \mathcal{I}\backslash\{v\}$ we have $v \neq w$. There is no one best way of how to consider Partition$_{n,\neq}$ as an optimization problem. In the following we present two different fitness function and show that for both problems a polynomial unary unbiased black-box complexity can be achieved. In Section 6.2.1 we consider a signed fitness function, which allows to learn from the fitness values which bin is the heavier one. We show that the unary unbiased black-box complexity of Partition$_{n,\neq}$ under this fitness function is $O(n \log n)$.

In Section 6.2.2 we then consider an unsigned fitness function. A priori we do have less information than in the situation of Section 6.2.1. However, we are still able to prove the same asymptotic bound. This second fitness function is probably the more natural one but note that the key arguments for our upper bound are essentially the same.

6.2.1. The Signed Fitness Function

As mentioned above, we consider in this section a signed fitness function for Partition$_{n,\neq}$. Given an instance $\mathcal{I}$ of Partition$_{n,\neq}$, we set $\mathcal{F}_\mathcal{I} := \{(\mathcal{I}_0, \mathcal{I}_1) \in 2^\mathcal{I} \times 2^\mathcal{I} \,|\, \mathcal{I}_0 \dot{\cup} \mathcal{I}_1 = \mathcal{I}\}$, the set of all feasible solutions of $\mathcal{I}$. We define the (signed) fitness function to measures the quality of the queried solutions via

$$f^*_\mathcal{I} : \mathcal{F} \to \mathbb{Z}, (\mathcal{I}_0, \mathcal{I}_1) \mapsto \sum_{w\in\mathcal{I}_0} w - \sum_{w\in\mathcal{I}_1} w\,.$$

Note that we aim at minimizing $|f^*_\mathcal{I}|$.

Since unbiased black-box complexity typically requires the search space $\{0,1\}^n$, let us fix some enumeration $\sigma : \mathcal{I} \to [n]$ of the elements of $\mathcal{I}$. To ease reading, let σ be the ordering of the elements in $\mathcal{I}$, i.e., $\sigma(v) < \sigma(w)$ for all $v, w \in \mathcal{I}$ with $v < w$. For any $x \in \{0,1\}^n$ let $\mathcal{I}_0(x) := \{w \in \mathcal{I} \,|\, x_{\sigma(w)} = 0\}$ and, accordingly, $\mathcal{I}_1(x) := \{w \in \mathcal{I} \,|\, x_{\sigma(w)} = 1\}$. Note that $\{0,1\}^n \to \mathcal{F}_\mathcal{I}, x \mapsto (\mathcal{I}_0(x), \mathcal{I}_1(x))$ is a bijection between $\{0,1\}^n$ and the original search space $\mathcal{F}_\mathcal{I}$. Therefore, we let

$$f_\mathcal{I} : \{0,1\}^n \to \mathbb{Z}, x \mapsto \sum_{i\in[n],x_i=0} \sigma^{-1}(i) - \sum_{i\in[n],x_i=1} \sigma^{-1}(i)\,.$$

Theorem 6.4. *The unary unbiased black-box complexity of* Partition$_{n,\neq}$ *modeled via the signed fitness functions $f_\mathcal{I}$ is $O(n \log n)$.*

Algorithm 15: A unary unbiased black-box algorithm for $\textsc{Partition}_{n,\neq}$ with the signed fitness function

```
1  Initialization:
2  Sample $x^{(0)} \leftarrow$ uniform() and query $f(x^{(0)})$;
3  Initialize $t \leftarrow 0$ and $,\mathcal{I}'_0, \mathcal{I}'_1, \mathcal{W}_0 = \emptyset$;
4  Learning the integers:
5  while $|\mathcal{W}_t| < n$ do
6      $t \leftarrow t + 1$;
7      Sample $x^{(t)} \leftarrow$ RLS($x^{(0)}$) and query $f(x^{(t)})$;
8      Update $\mathcal{W}_t \leftarrow \mathcal{W}_{t-1} \cup \{|f(x^{(0)}) - f(x^{(t)})|/2\}$;
9      if $f(x^{(0)}) > f(x^{(s)})$ then
10         $\mathcal{I}'_0 \leftarrow \mathcal{I}'_0 \cup \{|f(x^{(0)}) - f(x^{(s)})|/2\}$;
11     else $\mathcal{I}'_1 \leftarrow \mathcal{I}'_1 \cup \{|f(x^{(0)}) - f(x^{(s)})|/2\}$;
12 Optimization:
13 Offline compute an optimal solution $(\mathcal{O}_0, \mathcal{O}_1)$ and set
   $\mathcal{M} \leftarrow \{w \in \mathcal{O}_0 \mid w \notin \mathcal{I}'_0\} \cup \{w \in \mathcal{O}_1 \mid w \notin \mathcal{I}'_1\}$, the set of integers that need to be
   moved;
14 Set $z \leftarrow x^{(0)}$;
15 while $|\mathcal{M}| > 0$ do
16     Sample $y \leftarrow$ RLS($z$) and query $f(y)$;
17     if $w := |f(y) - f(z)|/2 \in \mathcal{M}$ then
18         $z \leftarrow y$ and $\mathcal{M} \leftarrow \mathcal{M} \setminus \{w\}$;
```

In the proof of Theorem 6.4 we will apply only two variation operators, namely `uniform()`, which samples a bit string $x \in \{0,1\}^n$ uniformly at random and `RLS`$(\cdot)$ (randomized local search), which, given some $x \in \{0,1\}^n$, creates from x a new bit string $y \in \{0,1\}^n$ by flipping exactly one bit in x, the bit position being chosen uniformly at random. Note that `RLS`$(\cdot)$ equals $\texttt{flip}_1(\cdot)$ as defined in Section 6.1. We have already seen that these two variation operators are unbiased. In fact, `uniform()` is the only 0-ary unbiased variation operator and `RLS`$(\cdot)$ is a unary unbiased variation operator.

Proof Theorem 6.4. We need to show that there exists an algorithm which, for any instance $\mathcal{I}$ of $\textsc{Partition}_{n,\neq}$, needs an expected number of $O(n \log n)$ iterations to compute a partition $(\mathcal{O}_0, \mathcal{O}_1) \in 2^{\mathcal{I}} \times 2^{\mathcal{I}}$ such that $|\sum_{w \in \mathcal{O}_0} w - \sum_{w \in \mathcal{O}_1} w|$ is minimized. We shall show that Algorithm 15 satisfies this. As mentioned above, it only employs two different variation operators, `uniform()` and `RLS`$(\cdot)$, both unbiased and of arity at most 1.

Let us fix some instance $\mathcal{I}$ of $\textsc{Partition}_{n,\neq}$. Abbreviate $f := f_{\mathcal{I}}$.

Let us now comment on the different steps of the algorithm.

After an expected number of $(1 + o(1))n \log n$ iterations, we have learned the weights of the problem instance as follows. First note that in the t-th iteration of the algorithm, the weight of the flipped bit is $|f(x^{(0)}) - f(x^{(t)})|/2$. Therefore, let $\mathcal{W}_t :=$

$\{|f(x^{(0)}) - f(x^{(s)})|/2 \mid s \in [t]\}$. By a coupon collector argument (cf. Section 2.4.5) the expected number of queries until we have flipped each bit position of $x^{(0)}$ at least once is $(1 + o(1))n \log n$. Thus, we can expect that we need $t^* = (1 + o(1))n \log n$ queries until $\mathcal{W}_{t^*} = \mathcal{I}$. That is, we can assume to have learned all n different weights in $\mathcal{I}$ in $(1 + o(1))n \log n$ queries.

Knowing the problem instance $\mathcal{I}$ we can compute an optimal partition $(\mathcal{O}_0, \mathcal{O}_1)$ for $\mathcal{I}$ *offline*, i.e., we do not need to query any further search points for this step. The computation can be done, e.g., by applying the brute force algorithm which compares all 2^n possible solutions. All we need to do now is to create a representation of $(\mathcal{O}_0, \mathcal{O}_1)$ via unbiased variation operators of arity at most 1.

To this end let us define $\mathcal{I}'_0(x^{(0)}) := \{|f(x^{(0)}) - f(x^{(s)})|/2 \mid s \in [t^*], f(x^{(0)}) > f(x^{(s)})\}$ and, accordingly, $\mathcal{I}'_1(x^{(0)}) := \{|f(x^{(0)}) - f(x^{(s)})|/2 \mid s \in [t^*], f(x^{(0)}) < f(x^{(s)})\}$. It is easily verified that $x^{(0)}$ is a binary representation of the partition $(\mathcal{I}'_0(x^{(0)}), \mathcal{I}'_1(x^{(0)}))$.

To create $(\mathcal{O}_0, \mathcal{O}_1)$ we set $\mathcal{M} := \{w \in \mathcal{O}_0 \mid w \notin \mathcal{I}'_0(x^{(0)})\} \cup \{w \in \mathcal{O}_1 \mid w \notin \mathcal{I}'_1(x^{(0)})\}$, the set of all weights that, in order to generate the optimal solution $(\mathcal{O}_0, \mathcal{O}_1)$, need to be moved from one of the sets $\mathcal{I}'_0(x^{(0)}), \mathcal{I}'_1(x^{(0)})$ to the other one.

In the optimization phase we do the following. In each iteration we create a new solution y from the current solution z by flipping exactly one bit of z. If $w := |f(y) - f(z)|/2 \in \mathcal{M}$, we update $z \leftarrow y$ and $\mathcal{M} \leftarrow \mathcal{M} \backslash \{w\}$.

As discussed above we can expect that after $(1 + o(1))n \log n$ such one bit flips we have flipped each bit position $i \in [n]$ at least once. That is, after an expected number of $(1 + o(1))n \log n$ queries, we have $\mathcal{M} = \emptyset$ and that we created $(\mathcal{O}_0, \mathcal{O}_1)$.

Putting everything together, we see that, despite using only unary unbiased variation operators, we can optimize any instance $\mathcal{I}$ of the $\textsc{Partition}_{n,\neq}$ problem modeled via the signed fitness function in an expected number of $2(1+o(1))n \log n = O(n \log n)$ queries. □

6.2.2. The Unsigned Fitness Function

One might dislike the fact that in the proof of Theorem 6.4 we neither minimize nor maximize $f_{\mathcal{I}}$ itself but only its absolute value $|f_{\mathcal{I}}|$. However, we can achieve the same asymptotic optimization complexity as in the statement of Theorem 6.4 if we only allow the latter, unsigned fitness function. Although the algorithm itself does not become more difficult to define, proving its correctness is more technical. The difficulty for the analysis stems from the fact that, given two bit strings x and y which differ in only one bit position, we cannot unambiguously learn from the corresponding fitness values $|f_{\mathcal{I}}(x)|$ and $|f_{\mathcal{I}}(y)|$ the weight of the flipped bit, cf. Remark 6.6. This results in a more complex procedure to learn all the weights.

Theorem 6.5. *The unary unbiased black-box complexity* $\textsc{Partition}_{n,\neq}$ *with respect to* $|f_{\mathcal{I}}|$ *is* $O(n \log n)$.

For a clearer presentation of the proof, we defer some technical elements used in the proof of Theorem 6.5 to Remark 6.6 and to the Lemmata 6.7 and 6.8, which will be presented after the proof of the main Theorem.

Algorithm 16: Unary unbiased black-box algorithm for PARTITION$_{n,\neq}$ with the unsigned fitness function

```
1  Initialization:
2  Sample $x^{(1,0)} \leftarrow$ uniform() and query $f(x^{(1,0)})$;
3  Shifting all weights to one bin:
4  for $t = 1$ to $2n \log n$ do
5      Sample $x^{(1,t)} \leftarrow$ RLS($x^{(1,0)}$) and query $f(x^{(1,t)})$;
6  Let $\ell \in \arg\max_{0 \le t \le 2nlogn} f(x^{(1,t)})$
7  $x \leftarrow x^{(1,\ell)}$;
8  for $t = 2n \log n + 1$ to $4n \log n$ do
9      Sample $y \leftarrow$ RLS($x$) and query $f(y)$;
10     if $f(y) > f(x)$ then $x \leftarrow y$;
11 Learning the instance $\mathcal{I}$:
12 for $t = 1$ to $2n \log n$ do
13     Sample $x^{(2,t)} \leftarrow$ RLS($x$) and query $f(x^{(2,t)})$;
14 Optimization:
15 Compute an optimal solution $(\mathcal{O}_0, \mathcal{O}_1)$ such that $w_{\max} \in \mathcal{O}_1$ offline and set
   $\mathcal{M} \leftarrow \mathcal{O}_1$.
16 for $t = 1$ to $2n \log n$ do
17     Sample $x^{(3,t)} \leftarrow$ RLS($x$) and query $f(x^{(3,t)})$;
18     if $f(x) > 2w_{\max}$ and $f(x^{(3,t)}) < f(x)$ then
19         compute $w := \big(f(x) - f(x^{(3,t)})\big)/2$;
20         if $w \neq w_{\max}$ and $w \in \mathcal{M}$ then
21             $x \leftarrow x^{(3,t)}$; $\mathcal{M} \leftarrow \mathcal{M} \setminus \{w\}$;
22 for $t = 1$ to $n \log n$ do
23     Sample $x^{(4,t)} \leftarrow$ RLS($x$) and query $f(x^{(4,t)})$;
```

Proof Theorem 6.5. By Corollary 2.15 it suffices to show that there exists an algorithm that, for any instance $\mathcal{I}$ of PARTITION$_{n,\neq}$, with high probability (w.h.p.), needs only $O(n \log n)$ fitness queries until it queries an optimal search point.

Let us fix an instance $\mathcal{I}$ of PARTITION$_{n,\neq}$. We abbreviate $f := |f_{\mathcal{I}}|$, where $f_{\mathcal{I}}$ is defined as in Section 6.2.1.

For readability purposes let us introduce the following notation. Note, however, that these values are a priori not identifiable for the algorithm. First note that, using the notation from Section 6.2.1, each $x \in \{0,1\}^n$ corresponds to a solution $(\mathcal{I}_0(x), \mathcal{I}_1(x)) \in \mathcal{F}_{\mathcal{I}}$. We set $\mathcal{S}_0(x) := \sum_{w \in \mathcal{I}_0(x)} w$ and $\mathcal{S}_1(x) := \sum_{w \in \mathcal{I}_1(x)} w$, the corresponding sum of the weights and let $\mathcal{I}_{\max}(x)$ be the set of weights belonging to the bin of larger weight, that is $\mathcal{I}_{\max} := \mathcal{I}_0$ if $\mathcal{S}_0(x) \ge \mathcal{S}_1(x)$ and $\mathcal{I}_{\max}(x) = \mathcal{I}_1$ otherwise. We call $\mathcal{I}_{\max}$ the "heavier" bin and we call the other one the "lighter" bin.

Lastly, let $w_{\max} = \max \mathcal{I}$ be the maximum weight of instance $\mathcal{I}$.

The general approach of Algorithm 16 is the following. First, we produce a string which represents a solution where all weights are in the same class of the partition, i.e., at the end of this phase we have $\mathcal{I}_{\max}(x) = \mathcal{I}$. With high probability this can be achieved with $4n \log n$ queries. Next, we perform $2n \log n$ `RLS` steps (i.e., random one bit flips). Through this we learn all n different weights in $\mathcal{I}$ w.h.p. After that, we compute an optimal solution offline. A representation of this solution can be generated in another $3n \log n$ iterations w.h.p.

If in any iteration of Algorithm 16 we have constructed a solution s with $f(s) = 0$ we are obviously done and do not need to run the algorithm any further. Therefore, we assume in the following that for all search points s but the last one we have $f(s) \neq 0$.

Algorithm 16 employs only two different variation operators, `uniform`() and `RLS`$(\cdot)$. We have argued that these operators are unbiased. Clearly, they are of arity at most 1. It remains to show that w.h.p. Algorithm 16 queries an optimal solution after at most $O(n \log n)$ queries. We show correctness for the three phases. The high probability statement follows from a simple union bound over the failure probabilities.

Shifting all weights to one bin. It follows from the coupon collector argument (cf. Section 2.4.5) that, with probability at least $1 - n^{-1}$, there exists for each $i \in [n]$ at least one $t_i \leq 2n \log n$ such that $x^{(1,0)}$ and $x^{(1,t_i)}$ differ exactly in the i-th bit. Using this information, Lemma 6.7 yields that for each string $x^{(1,\ell)} \in \{x^{(1,t)} \mid t \in [0..2n \log n]\}$ with the largest fitness $f(x^{(1,\ell)}) = \max\{f(x^{(1,t)}) \mid t \in [0..2n \log n]\}$ it holds that $w_{\max} \in \mathcal{I}_{\max}(x^{(1,\ell)})$. Let us fix one such ℓ and set $x := x^{(1,\ell)}$.

Lemma 6.7 and Lemma 6.8 verify the following. If y is created from x by flipping the i-th bit of x, then $f(y) > f(x)$ if and only if the i-th heaviest weight is not in the heavier bin, i.e., $\sigma(i) \notin \mathcal{I}_{\max}(x)$.

In the second part of the first phase we aim at creating a string x' with $\mathcal{I}_{\max}(x') = \mathcal{I}$. We do that by querying $y =$ `RLS`(x) and updating $x \leftarrow y$ if and only if $f(y) > f(x)$. From the statement of the previous paragraph this is the case only if the bit flip has moved the corresponding weight from the lighter to the heavier bin. Again from the coupon collector argument it follows that after an additional $2n \log n$ iterations we have $\mathcal{I}_{\max}(x) = \mathcal{I}$, with probability at least $1 - n^{-1}$.

Hence, after a total number of $4n \log n + 1$ iterations, we have created a bit string x with $\mathcal{I}_{\max}(x) = \mathcal{I}$, with probability at least $1 - 2n^{-1}$.

Learning instance $\mathcal{I}$. For the correctness of the second phase observe that either we have $w_{\max} \geq \sum_{w \in \mathcal{I}} w/2$ in which case one of the strings $x^{(2,t)}$, $t \in [2n \log n]$ is optimal (i.e., $\{w_{\max}\} = \mathcal{I}_{\max}(x^{(2,t)})$ for some $t \in [2n \log n]$) with probability at least $1 - n^{-1}$. This is again the coupon collector argument. Note that we are done in this case. Otherwise we have that for any $t \leq 2n \log n$ it holds that $f(x^{(2,t)}) < f(x)$ (since we are always shifting exactly one weight from the bin containing all weights to the empty one) and that the corresponding weight which has been flipped from one bin to the other is of size $(f(x) - f(x^{(2,t)}))/2$. In this case we have, again by the coupon collector argument that $\mathcal{I}' := \{(f(x) - f(x^{(2,t)}))/2 \mid t \in [2n \log n]\}$ equals $\mathcal{I}$, with probability at least $1 - n^{-1}$.

Optimization phase. Knowing instance $\mathcal{I}$, we can now compute an optimal solution $(\mathcal{O}_0, \mathcal{O}_1)$ for $\mathcal{I}$ offline, e.g., by the brute force algorithm. Note that for each $y \in \{0,1\}^n$ and its bitwise complement $\bar{y}$ it holds that $f(\bar{y}) = f(y)$. Thus, we can assume without loss of generality that $(\mathcal{O}_0, \mathcal{O}_1)$ is chosen such that $w_{\max} \in \mathcal{O}_1$.

For creating the bit string which corresponds to $(\mathcal{O}_0, \mathcal{O}_1)$, we initialize $\mathcal{M} := \mathcal{O}_1$.

Throughout this phase $\mathcal{M}$ denotes the set of all weights that, in order to create the string corresponding to $(\mathcal{O}_0, \mathcal{O}_1)$, still need to be "moved" from one bin to the other. The key idea here is that we required $w_{\max} \in \mathcal{M}$ and that we do not accept the weight $w_{\max}$ to be flipped too early. This is important for the following reason.

Recall from Remark 6.6 that if y is created from x by flipping the i-th bit in x and if $f(y) < f(x)$ then the corresponding weight $\sigma^{-1}(i) \in \{((f(x) - f(y))/2), (f(x) + f(y))/2\}$. But, as long as $f(x) > 2w_{\max}$ we have $(f(x) + f(y))/2 > w_{\max}$ (unless $f(y) = 0$ in which case we are done). That is, as long as $f(x) > 2w_{\max}$ it holds in the situation above that $\sigma^{-1}(i) = (f(x) - f(y))/2$.

It is easy to verify that as soon as $f(x) \leq 2w_{\max}$ we have $\mathcal{M} = \{w_{\max}\}$. It again is the coupon collector argument which ensures with probability at least $1 - n^{-1}$ that after $2n \log n$ iterations of the third phase we are in this situation. Thus, all we need to do now is to put $w_{\max}$ from bin $\mathcal{I}_{\max}(x)$ to the other one, i.e., we need to flip $\sigma(w_{\max})$. As for each iteration the probability to flip this position is $1/n$, we can bound the probability that we have flipped it after an additional $n \log n$ iterations from below by $1 - (1 - 1/n)^{n \log n} \geq 1 - 1/n$. Here we have used that for all $r \in \mathbb{R}$ we have $1 + r \leq \exp(r)$, cf. Lemma 2.8. □

Let us now prove the statements omitted in the proof of Theorem 6.5. We use the same notation as above.

Remark 6.6. *Let $\mathcal{I}$ be an instance of* Partition$_{n,\neq}$ *equipped with the ordering $\sigma_{\mathcal{I}}$ and fitness function $f = |f_{\mathcal{I}}|$. If y has been created from x by flipping the i-th bit of x and $0 \neq f(y) \neq f(x) \neq 0$, we cannot uniquely identify the corresponding weight $w_i = \sigma^{-1}(i)$. More precisely, if we do not have further knowledge on the size of the weights, there are the two possibilities*

$$w_i \in \begin{cases} \{\frac{1}{2}\big(f(y) - f(x)\big), \frac{1}{2}\big(f(y) + f(x)\big)\}, & \text{if } f(y) > f(x), \\ \{\frac{1}{2}\big(f(x) - f(y)\big), \frac{1}{2}\big(f(y) + f(x)\big)\}, & \text{if } f(y) < f(x). \end{cases}$$

Proof. The first statement follows from the second. But for a simple example consider the following situation. Let $\mathcal{I} := \{1, 2, 3, 4, 6\}$, $\sigma_{\mathcal{I}}$ the ordering of $\mathcal{I}$ and $x := (10001)$, i.e., weights 1 and 6 are in one bin and the other weights are in the second bin. Then $f(x) = |7 - 9| = 2$. Now both bit strings $y := (00001)$ and $z := (10101)$ have Hamming distance 1 from x and both have fitness $f(y) = |6 - 10| = 4 = f(z)$. Hence, knowing x, knowing that $|x - z|_1 = 1$, and knowing the fitness values $f(x)$ and $f(z)$ does not suffice to compute the bit in which x and z differ.

For the second statement we distinguish whether $f(y) > f(x)$ or $f(x) > f(y)$.

Case 1, $f(y) > f(x)$. If $\mathcal{I}_{\max}(y) \cap \mathcal{I}_{\max}(x) \neq \{w_i\}$ then $w_i \notin \mathcal{I}_{\max}(x)$ for otherwise $f(y) = f(x) - 2w_i < f(x)$. Thus, $w_i \notin \mathcal{I}_{\max}(x)$ and $f(y) = f(x) + 2w_i$.

In case $\mathcal{I}_{\max}(y) \cap \mathcal{I}_{\max}(x) = \{w_i\}$ then $w_i \in \mathcal{I}_{\max}(x)$ and $f(y) = 2w_i - f(x)$. Furthermore, since $f(y) > f(x)$, this yields $w_i > f(x)$.

Case 2, $f(y) < f(x)$. In this case we must have $w_i \in \mathcal{I}_{\max}(x)$ for otherwise $f(y) = f(x) + 2w_i > f(x)$.

If $\mathcal{I}_{\max}(y) \subseteq \mathcal{I}_{\max}(x)$ then $f(y) = f(x) - 2w_i$ and if $\mathcal{I}_{\max}(y) \cap \mathcal{I}_{\max}(x) = \{w_i\}$ then $w_i > f(x)/2$ and $f(y) = 2w_i - f(x)$. Since $f(y) < f(x)$ we also have $w_i < f(x)$.

This enumerates all possible combinations and the claim follows. □

Lemma 6.7. *Let $\mathcal{I}$ be an instance of* $\textsc{Partition}_{n,\neq}$*, let $\sigma = \sigma_{\mathcal{I}}$ be its ordering, and let $f = |f_{\mathcal{I}}|$.*

Furthermore, let $x^{(0)} \in \{0,1\}^n$ and for each $i \in [n]$ let $x^{(i)}$ be created from $x^{(0)}$ by flipping the i-th bit.

If we choose $\ell \in [0..n]$ such that $f(x^{(\ell)}) = \max\{f(x^{(t)}) \mid t \in [0..n]\}$, then $w_{\max} \in \mathcal{I}_{\max}(x^{(\ell)})$.

Proof. We assume that $w_{\max} \notin \mathcal{I}_{\max}(x^{(\ell)})$ to show the contrapositive. If $\ell = 0$, we can flip the bit corresponding to $w_{\max}$ (by our assumption on σ this is the n-th bit) in $x^{(0)}$ to get $f(x^{(n)}) = f(x^{(0)}) + 2w_{\max} > f(x^{(0)})$. Similarly, if $\ell = n$ then $f(x^{(0)}) = f(x^{(n)}) + 2w_{\max}$. All other values of ℓ imply $w_{\max} \notin \mathcal{I}_{\max}(x^{(0)})$ and thus, $f(x^{(n)}) - f(x^{(0)}) = 2w_{\max}$. But since the weights are pairwise different, $\sigma^{-1}(\ell) < w_{\max}$ and thus $f(x^{(\ell)}) - f(x^{(0)}) < 2w_{\max}$. □

The previous lemma has shown that the largest weight $w_{\max}$ is in the larger of the two bins of $x^{(\ell)}$. The following lemma shows that if we have iteratively increased the value of $x^{(\ell)}$ through 1-bit flips, we only have shifted weights from the smaller bin to the larger one.

Lemma 6.8. *Let $\mathcal{I}$ be an instance of* $\textsc{Partition}_{n,\neq}$*, let $\sigma := \sigma_{\mathcal{I}}$ be the ordering of $\mathcal{I}$, and $f := |f_{\mathcal{I}}|$.*

(i) *If $x \in \{0,1\}^n$ with $f(x) \geq w_{\max}$, then for all $i \in [n]$ we have $f(x \oplus e_i) > f(x)$ if and only if $w_i := \sigma^{-1}(i) \notin \mathcal{I}_{\max}(x^{\ell})$.*

(ii) *For $x^{(0)}, \ldots, x^{(n)}$ and ℓ as in Lemma 6.7 we have $f(x^{(\ell)}) \geq w_{\max}$.*

Proof. **(i).** Let $x \in \{0,1\}^n$ with $f(x) \geq w_{\max}$ and let $i \in [n]$. Clearly, if $w_i \notin \mathcal{I}_{\max}(x)$ then $f(x \oplus e_i) = f(x) + 2w_i > f(x)$. On the other hand, if $w_i \in \mathcal{I}_{\max}(x)$ then either $f(x \oplus e_i) = f(x) - 2w_i < f(x)$ or $f(x \oplus e_i) = 2w_i - f(x) \leq 2w_i - w_{\max} \leq w_{\max} \leq f(x)$.

(ii). If $w_{\max} \notin \mathcal{I}_{\max}(x^{(0)})$, then $\ell = n$ since for all $i \in [n]$ we have

$$f(x^{(n)}) = f(x^{(0)}) + 2w_{\max} \geq f(x^{(0)}) + 2w_i \geq f(x^{(i)})\,.$$

The above calculation immediately yields $f(x^{(\ell)}) > w_{\max}$.

Therefore, we may thus assume that $w_{\max} \in \mathcal{I}_{\max}(x^{(0)})$. To show the contrapositive, let us also assume that $f(x^{(\ell)}) < w_{\max}$. Then $f(x^{(0)}) \leq f(x^{(\ell)}) < w_{\max}$ and thus $f(x^{(n)}) = 2w_{\max} - f(x^{(0)}) > w_{\max} > f(x^{(\ell)})$, contradicting the choice of ℓ. Thus, $f(x^{(\ell)}) \geq w_{\max}$. □

It is not difficult to see that already with 3-ary variation operations it is possible to access every bit position in a linear number of iterations. Hence, a small modification of Algorithm 15 solves $\textsc{Partition}_{n,\neq}$ and even all instances $\textsc{Partition}_n$ of the original $\textsc{Partition}$ problem of size n[1] in a linear number of queries, using only unbiased variation operators of arity at most three.

Remark 6.9. *The 3-ary unbiased black-box complexity of* $\textsc{Partition}_n$ *is at most linear in n.*

[1]That is, $\textsc{Partition}_n$ is the collection of instances $\mathcal{I}$ of the $\textsc{Partition}$ problem with $|\mathcal{I}| = n$.

In the unrestricted model we can learn the weights by querying first the all-zeros bit string and then the n different unit vectors $e_i^n = (0 \dots 010 \dots 0), i \in [n]$.

Remark 6.10. *The unrestricted black-box complexity of* $\textsc{Partition}_n$ *is at most* $n+2$.

6.3. Conclusions

We have shown that already the unary unbiased black-box model contains algorithms, which solve NP-hard problems in a polynomial number of queries. Furthermore, we could also prove that the unary unbiased black-box complexity of the $\textsc{Jump}_{n,k}$ function is of order $O(n \log n)$, a bound which is not achieved by standard search heuristics.

These results indicate that the unbiased black-box model, while clearly closer to the truth than the unrestricted one, still does not give a full picture of how difficult a problem is to be solved via randomized search heuristics. It seems that further restrictions to the power of the algorithms are needed to obtain meaningful results. We shall present two such approaches in the two following sections, Section 7 and Section 8. In the latter one we introduce a black-box model where the algorithm can only compare the quality of solutions, but has no access to the absolute value of a search point's fitness. We do not know yet whether the black-box complexity of $\textsc{Partition}_n$ in this new model is still polynomial in n and, in fact, we conjecture that it is not.

Part II

Alternative Black-Box Models

7 Memory-Restricted Black-Box Models

In the first part of this thesis we have studied the unrestricted and the unbiased black-box models and we have shown that for several test functions the respective black-box complexities do not resemble the typical behavior of randomized search heuristics. We therefore concluded that further restrictions to the models are needed to obtain more sensible bounds.

One such approach was suggested by Droste, Jansen, and Wegener in their seminal paper on black-box complexity [DJW06]. Instead of allowing the algorithm to access the full query history, they suggest to restrict the *memory* of the algorithms under consideration. Droste et al. conjecture [DJW06, Section 6] that restricting the memory of the algorithms to a capacity, that can store at most one previously queried search point and its corresponding fitness, increases the black-box complexity of $\textsc{OneMax}_n$ from $\Theta(n/\log n)$ in the unrestricted case to $\Omega(n \log n)$.

Here in this section we disprove this conjecture. In fact, the black-box complexity of $\textsc{OneMax}_n$ remains $\Theta(n/\log n)$ even if the memory is restricted to one. Moreover, we show that optimizing $\textsc{OneMax}_n$ in a very natural way relates to a generalization of the classical board game Mastermind with n holes and a constant number k of colors.

The results presented in this section are currently under submission. For $k = 2$ colors, Lemma 7.4 can also be found in the preprint [DW11b]. All results are joint work with Benjamin Doerr.

7.1. The Mastermind Game

The original *Mastermind* game is a board game for two players. It was invented in the seventies by Meirowitz. It has pegs of six different colors. The goal of the *codebreaker*, for brevity called *Paul* here, is to find a color combination made up by *codemaker* (called *Carole* in the following). He does so by guessing color combinations and receiving information on how close this guess is to Carole's secret code. Paul's aim is to use as few guesses as possible.

For a more precise description, let us call the colors 1 to 6. Carole's secret code is a length-4 string of colors, that is, a $z \in [6]^4$. In each iteration, Paul guesses a string $x \in [6]^4$ and Carole replies with a pair $(\mathrm{eq}(z,x), \pi(z,x))$ of numbers. The first number, $\mathrm{eq}(z,x)$, usually indicated via black answer-pegs, is the number of positions in which Paul's and Carole's string coincide. The other number, $\pi(z,x)$, which is usually indicated by white answer-pegs, is the number of additional pegs having the right color, but being in the wrong position. Formally $\mathrm{eq}(z,x) := |\{i \in [4] \mid z_i = x_i\}|$ and $\pi(z,x) := \max_{\rho \in S_4} |\{i \in [4] \mid z_i = x_{\rho(i)}\}| - \mathrm{eq}(z,x)$. Paul "wins" the game if he guesses Carole's string, that is, if Carole's answer is $(4,0)$.

We study a generalized version of the Mastermind game with k colors and n positions, that is, Carole's secret code is a length-n string $z \in [k]^n$. We are interested in strategies for Paul that *guarantee* him to find the secret code with few questions. We thus adopt a worst-case view with respect to Carole's secret code. This is equivalent to assuming that Carole may change her hidden string at any time as long as it remains consistent with all previous answers (*devil's strategy*).

Previous Results. Mathematics and computer science literature produces a plethora of results on the Mastermind problem. For the original game with 6 colors and 4 positions, Knuth [Knu77] showed that Paul needs at most four queries until being able to identify Carole's string (which he may query in the fifth iteration to win the game).

For the generalized game denote by $d(n,k)$ the minimum number of guesses that enable Paul to win the game for every secret code $z \in [k]^n$. Chvátal [Chv83] proves that for $k < n^{1-\varepsilon}$, $\varepsilon > 0$ an arbitrarily small constant, we have $d(n,k) = O(\frac{n \log k}{\log n - \log k})$. More precisely, he shows that for any $\varepsilon > 0$ and n sufficiently large, $(2+\varepsilon)\frac{n(1+2\log_2 k)}{\log_2 n - \log_2 k}$ guesses chosen from $[k]^n$ independently and uniformly at random, with high probability, suffice to distinguish between all possible codes (that is, each secret code leads to a different sequence of answers). Therefore, the secret code can be determined after that many guesses. This remains true if Carole replies only with black answer-pegs, that is, if for any of Paul's guesses x she reveals to him only $\mathrm{eq}(z,x)$, the number of bits in which her and Paul's string coincide.

For larger values of k, the following is known. For $n \leq k \leq n^2$, Chvátal proves $d(n,k) \leq 2n\log_2 k + 4n$ and for $k = \omega(n^2 \log n)$ he shows $(k-1)/n \leq d(n,k) \leq \lceil k/n \rceil + d(n,n^2)$. These results have subsequently been improved. Chen, Cunha, and Homer [CCH96] show that $d(n,k) \leq 2n\lceil \log_2 n \rceil + 2n + \lceil k/n \rceil + 2$ for $k \geq n$. Goodrich [Goo09] proves $d(n,k) \leq n\lceil \log_2 k \rceil + \lceil (2-1/k)n \rceil + k$ for arbitrary k.

Concerning the computational complexity, Stuckman and Zhang [SZ06] show that it is NP-hard to decide whether a given sequence $(x^{(i)}, (\mathrm{eq}^{(i)}, \pi^{(i)}))_{i=1}^t$ of queries $x^{(i)}$ and answers $(\mathrm{eq}^{(i)}, \pi^{(i)})$ of black and white pegs has a secret code leading to these answers, i.e., whether there exists a string $z \in [k]^n$ such that $\mathrm{eq}(z, x^{(i)}) = \mathrm{eq}^{(i)}$ and $\pi(z, x^{(i)}) = \pi^{(i)}$ for all $i \in [t]$. Goodrich [Goo09] proves that this is already NP-hard if we only ask for consistence with the black answer-peg replies $\mathrm{eq}^{(i)}$.

The Connection Between Mastermind and Black-Box Complexity. Consider the Mastermind problem with n holes and $k = 2$ colors. Abbreviate Carole's secret code by z and assume that she only replies with black answer-pegs $\mathrm{eq}(z,x)$. By definition, $\mathrm{eq}(z,x)$ is exactly the function $\textsc{Om}_z(\cdot)$ evaluated in x. Therefore, any optimal guessing strategy for Paul corresponds to an optimal unrestricted black-box algorithm for the $\textsc{OneMax}_n$ problem and vice versa.

We have mentioned already in the previous sections that, motivated by a coin-weighing problem, Erdős and Rényi [ER63] studied this problem already in 1963. They showed a sharp $\Theta(n/\log n)$ bound for the unrestricted black-box complexity of $\textsc{OneMax}_n$ and, thereby, for any of Paul's optimal winning strategies.

Our results. Motivated by the conjecture of Droste, Jansen, and Wegener [DJW06, Section 6] on the memory-restricted black-box complexity of $\textsc{OneMax}_n$, we study optimal memory-restricted winning strategies for Paul. Since this original motivation asks for the case of two colors only, we restrict ourselves to the number k of colors being constant, though the methods presented below can also be used to analyze larger numbers of colors.

The memory-restriction can be briefly described as follows. Given a memory of size $m \in \mathbb{N}$, Paul can store up to m guesses and Carole's corresponding replies. Based *only* on this information, Paul decides on his next guess. After receiving Carole's reply, based only on the content of the memory, the current guess, and the current answer, he decides which m out of the $m+1$ strings and answers he keeps in the memory. Note that our memory restriction means that Paul truly has no other memory, in particular, no iteration counters, no experience that certain colors are not used, and so one. So formally Paul's strategy consists of a guessing strategy, which can be fully described by a mapping from m-sets of guesses and answers to strings $x \in [k]^n$, and a forgetting strategy, which maps $(m+1)$-sets of guesses and answers to m-subsets thereof.

Clearly, a memory-restriction makes Paul's life not easier. The $O(n/\log n)$ strategies for the 2-color game by Erdős and Rényi [ER63] and by Anil and Wiegand [AW09] as well as the generalized k-color game winning strategy by Chvátal [Chv83] do use the full history of guesses and answers and thus only work with a memory of size $\Theta(n/\log n)$. Surprisingly, this amount of memory is not necessary. In fact, one single memory cell suffices.

Theorem 7.1. *Let $k \in \mathbb{N}$. For all $n \in \mathbb{N}$, Paul has a size-one memory strategy winning the Mastermind game with k colors and n positions in $O(n/\log n)$ guesses. This remains true if we allow Carole to play a devil's strategy and if Carole only reveals the number of fully correct pegs $\mathrm{eq}(x,z)$ ("black answer-peg version of Mastermind").*

By Theorem 4.1 the bound in Theorem 7.1 is asymptotically tight. Already without memory restrictions $\Omega(n/\log n)$ queries are needed to determine the secret code.

The proof of Theorem 7.1 is quite technical. For a clearer presentation of the ideas, we first consider the size-two memory-restricted model, cf. Section 7.3. The proof of Theorem 7.1 is given in Section 7.4. Before going into the proofs, we give a short introduction to the memory-restricted black-box models.

7.2. The Memory-Restricted Black-Box Model

As mentioned above, Droste, Jansen, and Wegener [DJW06] conjectured a lower bound of $\Omega(n \log n)$ for the black-box complexity of $\textsc{OneMax}_n$ when the memory of the black-box algorithms is restricted to fit only one search point and its fitness value. Forbidding the algorithm to exploit the whole history of search points evaluated so far is originally motivated by the fact that many heuristics, e.g., evolutionary algorithms, only store a bounded size population of search points. Simple hill-climbers or the Metropolis algorithm even store only one single search point. Therefore, we agree with Droste et al. that this is a very natural restriction.

Algorithm 17 is the scheme of a black-box algorithm with bounded memory of size μ. The *μ-memory-restricted black-box complexity* of a function class $\mathcal{F}$ is $\min_{A \in \mathcal{A}} \max_{f \in \mathcal{F}} T(A, f)$, the minimal worst-case runtime (i.e., expected number of function evaluations) of any such black-box algorithm A with bounded memory of size μ.

It is important to note that a black-box algorithm with bounded memory is not allowed to access any other information than the one stored in the μ pairs $(x^{(1)}, f(x^{(1)})), \ldots, (x^{(\mu)}, f(x^{(\mu)}))$ which are currently in the memory and, in the selection step, also the information provided by $(x^{(\mu+1)}, f(x^{(\mu+1)}))$. In particular, the algorithm does not have access to an iteration counter.

Algorithm 17: Scheme of a black-box algorithm with memory of size μ for optimizing a function $f : \mathcal{S} \to \mathbb{R}$

```
1 Initialization: $\mathcal{M} \leftarrow \emptyset$;
2 for $t = 1, 2, \ldots$ do
3     Depending (only) on $\mathcal{M}$ choose a probability distribution $p$ over $\mathcal{S}$ and
      sample $x^{(\mu+1)}$ according to $p$; //variation step
4     Query $f(x^{(\mu+1)})$;
5     Select $\mathcal{M} \subseteq \mathcal{M} \cup \{(x^{(\mu+1)}, f(x^{(\mu+1)}))\}$ of size $|\mathcal{M}| \leq \mu$; //selection step
```

Recall that the memory-restricted black-box complexity of ONEMAX_n and optimal strategies for Mastermind with two colors, black answer-pegs only, and a corresponding memory restriction are equivalent questions. Consequently, our result can be rephrased to saying that the 1-memory-restricted black-box complexity of ONEMAX_n is $\Theta(n/\log n)$. This disproves the previous conjecture of Droste, Jansen, and Wegener.

From the view-point of building a useful complexity theory for randomized search heuristics, Theorem 7.1 indicates that a memory restriction does not help to remove the drawbacks of the existing black-box models. For this reasons we decided to settle for this result on the memory-restricted black-box complexity of ONEMAX_n. That is, here in this thesis, we do not study unbiased versions of the memory-restricted black-box model and we do not study memory-restricted black-box complexities of other function classes.

7.3. The Mastermind Game with Memory of Size Two

We show that with a memory of size two Paul can win the n holes, k colors black answer-peg only version of the Mastermind game using only $O(n/\log n)$ guesses. Already this proof contains many ingredients needed to prove Theorem 7.1, e.g., the use of the random guessing strategy with limited memory, the block-wise determination of the secret code, and the simulation of iteration counters in the memory.

Let $k \geq 2$ be the number of colors used. In particular for $k = 2$, it will be convenient to label the colors from 0 to $k - 1$. Let us denote the set of colors by $\mathcal{C} := [0..k-1] := \{0, 1, \ldots, k-1\}$. We assume that k is constant and that the number n of positions in the string is large, that is, all asymptotic notation is with respect to n.

Theorem 7.2. *Paul has a size-two memory strategy winning the black answer-peg only Mastermind game with k colors and n positions in $O(n/\log n)$ guesses. This remains true if we allow Carole to play a devil's strategy.*

As many previous works on Mastermind and $\textsc{OneMax}_n$, the proof of Theorem 7.2 heavily relies on *random guessing*. For the case of $k = 2$ colors, already Erdős and Rényi [ER63] showed that there is a $t \in \Theta(n/\log n)$ such that t guesses $x^{(1)}, \ldots, x^{(t)}$ chosen from $\{0,1\}^n$ independently and uniformly at random, together with Carole's black answer-peg replies, uniquely define the hidden code. This was generalized by Chvátal [Chv83] to the following result.

Theorem 7.3 (from [Chv83]). *Let $\varepsilon > 0$, let $n > n(\varepsilon)$ be sufficiently large and let $k < n^{1-\varepsilon}$. Let $x^{(1)}, \ldots, x^{(t)}$ be $t \geq (2+\varepsilon)\frac{n(1+2\log_2 k)}{\log_2 n - \log_2 k}$ samples chosen from $\mathcal{C}^n$ independently and uniformly at random. Then for all $z \in \mathcal{C}^n$, the set*

$$\mathcal{S}^{consistent} := \{y \in \mathcal{C}^n \mid \forall i \in [t] : \mathrm{eq}(y, x^{(i)}) = \mathrm{eq}(z, x^{(i)})\}$$

satisfies $\mathrm{E}[|\mathcal{S}^{consistent}|] \leq 1 + 1/n$.

Since the strategy implicit in Theorem 7.3 needs a memory of size $\Theta(n/\log n)$, we cannot apply it directly in our setting. We can, however, adapt it to work on smaller portions ("blocks") of the secret code, and this with much less memory.

Let $y \in \mathcal{C}^n$ and let $B \subseteq [n]$ be a block (i.e., an interval) of size $s := \lceil \sqrt{n} \rceil$. As we shall see, by $t \in O(s/\log s)$ times guessing a string obtained from y by replacing the colors in B by randomly chosen ones (and guessing k additional *reference strings*), we can determine $z_{|B}$, the part of the secret code z in block B.

We can do so with a memory of size two only. We store the string obtained from y by altering it on B (*sampling string*) in one cell. Note that we do not need to remember y, as we only need to ensure that our guesses agree in the positions $[n] \setminus B$. We use the other memory cell for storing the random substrings of length s, which we substituted into y at B, and for storing Carole's corresponding replies. Note that each such answer can be encoded in binary using $\ell_n \in O(\log n)$ entries of the string. Hence the t guesses and answers can be memorized using a total number of $t(s + \ell_n) = O(n/\log n)$ positions. Therefore, all relevant information can be stored in one string of length n, the *storage string*, in the following typically denoted by x.

This approach allows us to determine any length-s block B of Carole's secret code z using $t = O(s/\log s)$ guesses. Since we need to determine $\lceil n/s \rceil$ such length-s blocks, we can determine the secret code z with $t\lceil n/s \rceil = O(n/\log n)$ guesses, as desired.

In Algorithm 18 (notation used will be introduced below) we make this strategy more precise by giving it in pseudo-code. Note, however, that this algorithm does not fully satisfy the size-two memory restriction. The reason is that the queries do not only depend on the current state of the memory, but also on iteration counters and,

Algorithm 18: An almost size-two memory-restricted algorithm winning the k-color black answer-peg only Mastermind game in $O(n/\log n)$ guesses. **Remark:** x denotes the unique string in $\mathcal{M}$ with $x_n = 1$ and y denotes the unique string in $\mathcal{M}$ with $y_n = 0$.

```
1  Initialization: $y \leftarrow [0 \dots 0]$;
2      Query $\text{eq}(z,y)$ and update $\mathcal{M} \leftarrow \{(y, \text{eq}(z,y))\}$;
3  for $i = 1$ to $\lceil (n-1)/s \rceil$ do
4      $x \leftarrow [0 \dots 0|1]$; //initialization of $x$
5      Query $\text{eq}(z,x)$ and update $\mathcal{M}$ by adding ($i = 1$) or replacing ($i > 1$)
       $(x, \text{eq}(z,x))$ in $\mathcal{M}$;
6      for $q = 0$ to $t + k - 1$ do
7          if $q < k$ then $y \leftarrow \texttt{substitute}(y, B_i, [q \dots q])$ ; //reference string
8          else $y \leftarrow \texttt{substitute}(y, B_i, r)$ where $r \in \mathcal{C}^{|B_i|}$ u.a.r.; //random guess
9          Query $\text{eq}(z,y)$ and update $\mathcal{M}$ by replacing $(y, \text{eq}(z,y))$;
10         $x \leftarrow [x_1 \dots x_{p_1(x)}|\texttt{BLOCK}_i(y)|\texttt{binary}_{\ell_n}(\text{eq}(z,y))|1|0 \dots 0|1]$ ; //add $y$'s info
           to $x$
11         Query $\text{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \text{eq}(z,x))$;
12     while $\Delta_i(y) < |B_i|$ do
13         $y \leftarrow \texttt{substitute}(y, B_i, w)$, where $w \in \mathcal{S}_i^{\text{consistent}}$ u.a.r.;
14         Query $\text{eq}(z,y)$ and update $\mathcal{M}$ by replacing $(y, \text{eq}(z,y))$;
15 while $\text{eq}(z,y) < n$ do $y \leftarrow \texttt{substitute}(y, \{n\}, c)$, where $c \in \mathcal{C}$ u.a.r., and query
   $\text{eq}(z,y)$;
```

for example, in lines 9 and 11, on a program counter. Further below, in Algorithm 19, we shall remove this shortcoming with a few additional technicalities, which we are happy to spare for the moment.

Before we argue for the correctness of Algorithm 18, let us fix the notation. For any string $x \in \mathcal{C}^n$ we also write $x = [x_1 \dots x_n]$. To ease reading, we allow ourselves to indicate different structural components of x by vertical bars, e.g., $x = [x_1 \dots x_p|x_{p+1} \dots x_n]$. For $i \in [\lceil (n-1)/s \rceil]$ let $B_i := \{(i-1)s+1, \dots, is\} \cap [n-1]$, the positions of the i-th block. Set

$$\texttt{BLOCK}_i(x) := x_{|B_i} := [x_{(i-1)s+1} \cdots x_{\min\{is,n-1\}}] ,$$

the i-th block of x. For any string $r \in \mathcal{C}^{|B_i|}$ we define

$$\texttt{substitute}(x, B_i, r) := [x_1 \dots x_{(i-1)s}|r|x_{\min\{is,n-1\}+1} \cdots x_n] ,$$

the string with the i-th block substituted by r. Similarly, let $\texttt{substitute}(y, \{n\}, c) := [y_1 \dots y_{n-1}|c]$. Note that we do not assign the n-th position to any of the blocks. We do so because in Algorithms 18 and 19 we shall use that position to indicate which one of the two strings in the memory $\mathcal{M}$ is the storage string (the unique string $x \in \mathcal{M}$ with $x_n = 1$) and which one is the sampling string (the unique string $y \in \mathcal{M}$ with $y_n = 0$).

Let $p_1(x) := \max\{i \in [n-1] \mid x_i = 1\}$, the largest position $i < n$ of x with entry "1". As mentioned above, we encode Carole's answers $\text{eq}(z,y) \in [0..n]$ in binary, using $\ell_n := \lceil \log_2 n \rceil + 1$ positions, and we denote this binary encoding of length ℓ_n by $\texttt{binary}_{\ell_n}(\text{eq}(z,y))$. By $\Delta_i(y)$ we denote the contribution of the i-th block to the value $\text{eq}(z,y)$. That is, $\Delta_i(y)$ is the number of positions in the i-th block in which Paul's guess y and Carole's secret code z coincide. Formally, $\Delta_i(y) := \text{eq}(z_{|B_i}, y_{|B_i})$. We shall see below how $\Delta_i(y)$ can be computed from $\text{eq}(z,y)$ and $\text{eq}(z,x^i), i = 0,1,\ldots,k-1$, where the x^i are suitably chosen reference strings.

Lastly, let $\mathcal{S}_i^{\text{consistent}}$ be the set of strings w of length $|B_i|$ such that $\texttt{substitute}(z, B_i, w)$ is consistent with all of Carole's replies, i.e., $\text{eq}(\texttt{substitute}(z, B_i, w), y) = \text{eq}(z,y)$ for all search points y queried so far (a formal definition will be given below). We shall see below that both $\Delta_i(y)$ and $\mathcal{S}_i^{\text{consistent}}$ can be computed solely from the content of the memory cells (lines 12–14).

We now argue for the correctness of Algorithm 18. First we show that, when Algorithm 18 leaves the random guessing phase in lines 6–11, based only on the information given in the memory, we can restore the full history of guesses for the i-th block. To this end, first note that for any guess y done in line 9, we used $s + \ell_n + 1$ positions in x for storing its information (line 10; we add the additional "1" at the end to ease determining via $p_1(x)$ the positions in x which have not yet been used for storing information). In lines 6–11 we first asked and stored k non-random guesses

$$x^c = \texttt{substitute}(y, B_i, [c \ldots c])$$

and we stored these *reference strings* together with Carole's replies

$$\text{eq}(z, x^c) = \sum_{h=1}^{\ell_n} 2^{h-1} x_{c(s+\ell_n+1)-h}\,,$$

$c \in [0..k-1]$. For $j \in [t]$, the j-th random sample is

$$r^{(j)} = [x_{(k+j-1)(s+\ell_n+1)+1} \cdots x_{(k+j-1)(s+\ell_n+1)+|B_i|}]$$

and the corresponding query was

$$y^{(j)} = \texttt{substitute}(y, B_i, r^{(j)})\,.$$

We have stored Carole's reply to this guess in binary, and we can infer

$$\text{eq}(z, y^{(j)}) = \sum_{h=1}^{\ell_n} 2^{h-1} x_{(k+j)(s+\ell_n+1)-h}\,.$$

This shows how to regain the full guessing history.

Next we show how to compute the contributions $\Delta_i(y^{(j)})$ of the entries in the i-th block. To this end, note that the constant substrings $[c \ldots c]$ in the reference strings x^c in total contribute exactly $|B_i|$ to the sum $\text{eq}(z,x^0) + \ldots + \text{eq}(z,x^k)$. Formally,

$$\sum_{c=0}^{k-1} \text{eq}([z_{(i-1)s+1} \cdots z_{\min\{is,n-1\}}], [c \ldots c]) = |B_i|\,.$$

Since all other positions of the sampling string y are not changed during the phase, in which we determine the i-th block, we infer that

$$\Delta_i(y^{(j)}) = \mathrm{eq}(z, y^{(j)}) - (\mathrm{eq}(z, x^0) + \ldots + \mathrm{eq}(z, x^k) - |B_i|)/k\,.$$

Consequently, in lines 12–14, the algorithm can compute $\Delta_i(y^{(j)})$ for all $j \in [t]$. From this it can infer

$$\begin{aligned}\mathcal{S}_i^{\text{consistent}} : &= \{\tilde{z} \in \mathcal{C}^{|B_i|} \mid \forall j \in [t] : \mathrm{eq}(\tilde{z}, \texttt{BLOCK}_i(y^{(j)})) = \Delta_i(y^{(j)})\} \\ &= \{\tilde{z} \in \mathcal{C}^{|B_i|} \mid \forall j \in [t] : \mathrm{eq}(\tilde{z}, r^{(j)}) = \mathrm{eq}(\texttt{BLOCK}_i(z), r^{(j)})\}\end{aligned}$$

the set of possible code segments in B_i. By Theorem 7.3, the expected size of $\mathcal{S}_i^{\text{consistent}}$ is bounded from above by $1+1/|B_i|$. Thus, in lines 12–14 we need an expected number of $1 + 1/|B_i|$ samples w chosen from $\mathcal{S}_i^{\text{consistent}}$ uniformly at random until we find a $y = \texttt{substitute}(y, B_i, w)$ with $\Delta_i(y) = s$ (which implies that the i-th block of y coincides with Carole's secret code). This shows how we determine the entries of the i-th block in an expected total number of $t = O(s/\log s)$ guesses.

When Algorithm 18 executes line 15, all but the last entry of y coincide with Carole's secret code. Hence trying random colors in the n-th position finds the hidden code z with an additional expected number of $k = \Theta(1)$ guesses.

To turn Algorithm 18 into a truly size-two memory-restricted one, we use the first ℓ_n entries of x to store in binary the index of the block currently being under consideration. This is iteration counter i in Algorithm 18. At the same time we move the storage space for the guesses and answers by ℓ_n positions to the right. Formally, we define

$$i(x) := \sum_{h=0}^{\ell_n - 1} 2^h x_{\ell_n - h}\,.$$

The inner **for**-loop of Algorithm 18 needs no additional memory to be simulated, because we can infer from $p_1(x)$ how many guesses $q(x)$ have been queried already: since storing each guess requires $s + \ell_n + 1$ positions and since the first ℓ_n positions are used for indicating the number of already determined entries, we have

$$q(x) := \frac{p_1(x) - \ell_n}{s + \ell_n + 1}\,.$$

Lastly, we need to replace the sequential queries in lines 9 and 11 of Algorithm 18 (as this exploits information stored in the program counter). Fortunately, again we can deduce from the memory the status of the algorithm. We define a function $\texttt{Part}(y, x)$, which equals 1 if the information of y has been added to the storage string x already and which equals 0 otherwise. That is, we set

$$\texttt{Part}(y,x) = \begin{cases} 1, \text{ if } \sum_{i=1}^{\ell_n} 2^{i-1} x_{p_1(x)-i} = \mathrm{eq}(z,y) \text{ and} \\ \qquad \texttt{BLOCK}_{i(x)}(y) = [x_{p_1(x)-\ell_n-|B_{i(x)}|} \cdots x_{p_1(x)-\ell_n-1}]\,, \\ 0, \text{ otherwise}\,. \end{cases}$$

Algorithm 19: A size-two memory-restricted algorithm winning the k-color black answer-peg only Mastermind game in $O(n/\log n)$ guesses. **Remark:** x denotes the unique string in $\mathcal{M}$ with $x_n = 1$ and y denotes the unique string in $\mathcal{M}$ with $y_n = 0$.

```
1  Initialization: Let $\mathcal{M} \leftarrow \emptyset$ ; // clear memory
2  if $\mathcal{M} = \emptyset$ then
3      $y \leftarrow [0...0]$ ; //first reference string
4      Query $\mathrm{eq}(z, y)$ and update $\mathcal{M} \leftarrow \{(y, \mathrm{eq}(z, y))\}$;
5  else if $|\mathcal{M}| = 1$ then
6      $x \leftarrow [0...0|1]$ ; //initialization of storage string
7      Query $\mathrm{eq}(z, x)$ and update $\mathcal{M} \leftarrow \mathcal{M} \cup \{(x, \mathrm{eq}(z, x))\}$;
8  else if $i(x) < \lceil (n-1)/s \rceil$ then
9      if $x = [0 \dots 0|1]$ or $\Delta_{i(x)}(y) = |B_{i(x)}|$ then
10         $x \leftarrow [\mathtt{binary}_{\ell_n}(i(x)+1)|\mathtt{BLOCK}_{i(x)+1}(y)|\mathtt{binary}_{\ell_n}(\mathrm{eq}(z,y))|1|0 \dots 0|1]$ ;
           //clear storage string and add first reference string
11         Query $\mathrm{eq}(z, x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z, x))$;
12     else if $\mathtt{Part}(y, x) = 1$ and $q(x) < t + k$ then
13         if $q(x) < k$ then $y \leftarrow \mathtt{substitute}(y, B_{i(x)}, [q(x) \dots q(x)])$ ; //reference
           string
14         else $y \leftarrow \mathtt{substitute}(y, B_{i(x)}, r)$ where $r \in \mathcal{C}^{|B_{i(x)}|}$ u.a.r.; //random
           guess
15         Query $\mathrm{eq}(z, y)$ and update $\mathcal{M}$ by replacing $(y, \mathrm{eq}(z, y))$;
16     else if $\mathtt{Part}(y, x) = 0$ and $\Delta_{i(x)}(y) < |B_{i(x)}|$ then
17         $x \leftarrow [x_1 \dots x_{p_1(x)}|\mathtt{BLOCK}_{i(x)}(y)|\mathtt{binary}_{\ell_n}(\mathrm{eq}(z,y))|1|0 \dots 0|1]$; //add $y$'s
           info to $x$
18         Query $\mathrm{eq}(z, x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z, x))$;
19     else if $\mathtt{Part}(y, x) = 1$ and $q(x) = t + k$ then
20         $y \leftarrow \mathtt{substitute}(y, B_{i(x)}, w)$ where $w \in \mathcal{S}^{\text{consistent}}_{i(x)}$ chosen u.a.r.;
21         Query $\mathrm{eq}(z, y)$;
22         if $\Delta_{i(x)}(y) = |B_{i(x)}|$ then Update $\mathcal{M}$ by replacing $(y, \mathrm{eq}(z, y))$;
23 else if $i(x) = \lceil (n-1)/s \rceil$ then
24     $y \leftarrow \mathtt{substitute}(y, \{n\}, c)$ where $c \in \mathcal{C} \backslash \{y_n\}$ u.a.r.;
25     Query $\mathrm{eq}(z, y)$ ;
26 Go to line 2;
```

Note that $\mathtt{Part}(y, x) = 1$ indicates that the information of y has been stored in x also in the case that our current random sample equals the previous one. However, this does not cause any problems as in this case the current guess does not give any new information. The use of $\mathtt{Part}(\cdot, \cdot)$ modifies the algorithm to sample t random guesses without immediate repetition. Note that the probability to sample the same string $r \in \mathcal{C}^{|B_{i(x)}|}$ twice in a row is at most $1/2$ (if the last block consists only of one position and $k = 2$) and is typically much smaller. Hence, occurrences of this event have no influence on the asymptotic number of guesses needed to win the game.

With these modifications, Algorithm 18 becomes the truly size-two memory-restricted Algorithm 19.

7.4. Memory of Size One: Proof of Theorem 7.1

Compared to the situation in Section 7.3, Paul faces two additional challenges in the size-one memory-restricted setting. The obvious one is that he has less memory available, in particular, after a large part of the code has been determined and needs to be stored. The more subtle one is that he cannot any longer query a search point and then store whatever is worth storing in the second memory cell. With one memory cell, all he can do is to guess a new string and keep or forget it.

7.4.1. Linear Query Time Strategies

Before we prove Theorem 7.1, let us discuss a linear query time winning strategy, i.e., a strategy that allows Paul to find Carole's secret code in a linear expected number of guesses, using one memory cell only. This linear query time strategy will be used in the proof of Theorem 7.1 to determine the last $\Theta(n/\log n)$ entries of the secret code and it will be used to determine the first $\Theta(\log n)$ entries, which shall be used for storing reference information.

The basic idea of the linear query time strategy is to test each position one by one, from left to right. Since we have just one memory cell, we need to indicate in this one string which entries have been determined already. We do so by keeping all not yet determined entries at one identical value different from the one of the entry determined last. To this end, let us for all $x \in \mathcal{C}^n$ define

$$\operatorname{tn}(x) := \min\{i \in [n] \mid \forall j \in \{i, \ldots, n\} : x_j = x_i\}\,,$$

the *tail number of* x. The following lemma describes the linear query time strategy.

Lemma 7.4. *Let $x \in \mathcal{C}^n$. Furthermore, let us denote Carole's secret code by $z \in \mathcal{C}^n$. Let us assume that the first $\operatorname{tn}(x) - 1$ entries of z have been determined (i.e., Carole can no longer change the entries of $[z_1 \ldots z_{\operatorname{tn}(x)-1}]$). Further assume that $x_i = z_i$ for all $i < \operatorname{tn}(x)$ and that $\mathcal{M} = \{(x, \operatorname{eq}(z, x))\}$ is the current content of the memory cell.*

There is a size-one memory-restricted guessing procedure `LinAlg` that—even if Carole plays a devil's strategy—after an expected constant number of successive calls modifies the memory such that the string y now in the memory satisfies $y_i = z_i$ for all $i \leq \operatorname{tn}(x)$ and $\operatorname{tn}(y) = \operatorname{tn}(x) + 1$. Every call of `LinAlg` requires only one guess.

Interestingly, for the definition of `LinAlg`, we need to distinguish between the cases of $k = 2$ and $k \geq 3$ colors.

The case of $k = 2$ colors $\mathcal{C} = \{0, 1\}$

In this section we prove that for $k = 2$ colors $\mathcal{C} = \{0, 1\}$, `LinAlg` is a procedure that requires, in expectation, three calls to modify the memory content by replacing the current string x that is assumed to satisfy the conditions of Lemma 7.4, by a string y with $\mathrm{tn}(y) = \mathrm{tn}(x) + 1$ and $y_i = z_i$ for all $i \leq \mathrm{tn}(x)$.

Recall that for all $i \in [n]$ we defined e_i^n to be the i-th unit vector of length n.

Algorithm 20: Routine `LinAlg` for $k = 2$ colors

```
1 Assumption: The string $x \in \{0,1\}^n$ in the memory satisfies $\mathrm{tn}(x) < n$ and
    $x_i = z_i$ for all $i < \mathrm{tn}(x)$;
2 Sample $y \in \{x \oplus e^n_{\mathrm{tn}(x)}, x \oplus \sum_{i=\mathrm{tn}(x)+1}^n e_i^n\}$ uniformly at random;
3 Query $\mathrm{eq}(z, y)$;
4 if $y = x \oplus e^n_{\mathrm{tn}(x)}$ then
5     if $\mathrm{eq}(z, y) > \mathrm{eq}(z, x)$ then $\mathcal{M} \leftarrow \{(y, \mathrm{eq}(z, y))\}$;
6 else
7     if $\mathrm{eq}(z, x) + \mathrm{eq}(z, y) = n + \mathrm{tn}(x)$ then $\mathcal{M} \leftarrow \{(y, \mathrm{eq}(z, y))\}$;
```

Proposition 7.5. *For $k = 2$ colors, Algorithm 20 satisfies the conditions of Lemma 7.4. In expectation, three calls to routine `LinAlg` suffice.*

Proof. Let $x \in \{0,1\}^n$ be a bit string with $\mathrm{tn}(x) < n$ and $x_i = z_i$ for all $i < \mathrm{tn}(x)$.

Algorithm 20 samples with probability 1/2 the string $y = x \oplus e^n_{\mathrm{tn}(x)}$, and with probability 1/2 it samples $y = x \oplus \sum_{i=\mathrm{tn}(x)+1}^n e_i^n$. That is, either it flips only the $\mathrm{tn}(x)$-th bit of x or it flips all "tail bits" but the tail numbered one.

If $y = x \oplus e^n_{\mathrm{tn}(x)}$, clearly we have $z_{\mathrm{tn}(x)} = y_{\mathrm{tn}(x)}$ if and only if $\mathrm{eq}(z, y) > \mathrm{eq}(z, x)$.

Therefore, let us assume that Algorithm 20 samples $y = x \oplus \sum_{i=\mathrm{tn}(x)+1}^n e_i^n$. We show that $z_{\mathrm{tn}(x)} = y_{\mathrm{tn}(x)} (= x_{\mathrm{tn}(x)})$ holds if and only if $\mathrm{eq}(z, x) + \mathrm{eq}(z, y) = n + \mathrm{tn}(x)$. By definition we have $y_i = x_i = z_i$ for all $i < \mathrm{tn}(x)$. Thus, the first $\mathrm{tn}(x) - 1$ bits of x and y contribute $2(\mathrm{tn}(x) - 1)$ to the sum $\mathrm{eq}(z, x) + \mathrm{eq}(z, y)$; formally,

$$\mathrm{eq}([z_1 \dots z_{\mathrm{tn}(x)-1}], [x_1 \dots x_{\mathrm{tn}(x)-1}]) + \mathrm{eq}([z_1 \dots z_{\mathrm{tn}(x)-1}], [y_1 \dots y_{\mathrm{tn}(x)-1}]) = 2(\mathrm{tn}(x) - 1) \,.$$

On the other hand, for all $i > \mathrm{tn}(x)$ either have $z_i = x_i$ or $z_i = 1 - x_i = y_i$. Thus, the last last $n - \mathrm{tn}(x)$ bits of x and y contribute exactly $n - \mathrm{tn}(x)$ to the sum $\mathrm{eq}(z, x) + \mathrm{eq}(z, y)$; formally,

$$\mathrm{eq}([z_{\mathrm{tn}(x)+1} \dots z_n], [x_{\mathrm{tn}(x)+1} \dots x_n]) + \mathrm{eq}([z_{\mathrm{tn}(x)+1} \dots z_n], [y_{\mathrm{tn}(x)+1} \dots y_n]) = n - \mathrm{tn}(x) \,.$$

By definition we also have $y_{\mathrm{tn}(x)} = x_{\mathrm{tn}(x)}$ and, thus,

$$\begin{aligned}
\mathrm{eq}(z, x) + \mathrm{eq}(z, y) &= \mathrm{eq}([z_1 \dots z_{\mathrm{tn}(x)-1}], [x_1 \dots x_{\mathrm{tn}(x)-1}]) + \mathrm{eq}(z_{\mathrm{tn}(x)}, x_{\mathrm{tn}(x)}) \\
&\quad + \mathrm{eq}([z_{\mathrm{tn}(x)+1} \dots z_n], [x_{\mathrm{tn}(x)+1} \dots x_n]) \\
&\quad + \mathrm{eq}([z_1 \dots z_{\mathrm{tn}(x)-1}], [y_1 \dots y_{\mathrm{tn}(x)-1}]) + \mathrm{eq}(z_{\mathrm{tn}(x)}, x_{\mathrm{tn}(x)}) \\
&\quad + \mathrm{eq}([z_{\mathrm{tn}(x)+1} \dots z_n], [y_{\mathrm{tn}(x)+1} \dots y_n]) \\
&= 2(\mathrm{tn}(x) - 1) + n - \mathrm{tn}(x) + 2\,\mathrm{eq}(z_{\mathrm{tn}(x)}, x_{\mathrm{tn}(x)}) \\
&= n + \mathrm{tn}(x) + 2\,\mathrm{eq}(z_{\mathrm{tn}(x)}, x_{\mathrm{tn}(x)}) - 2
\end{aligned}$$

This shows that $\mathrm{eq}(z,x)+\mathrm{eq}(z,y)=n+\mathrm{tn}(x)$ if and only if $\mathrm{eq}(z_{\mathrm{tn}(x)},x_{\mathrm{tn}(x)})=1$, i.e., if and only if $z_{\mathrm{tn}(x)}=x_{\mathrm{tn}(x)}(=y_{\mathrm{tn}(x)})$.

It is immediate that for a secret code z taken from $\{0,1\}^n$ uniformly at random, the probability to obtain, in one call of `LinAlg`, a string y with $\mathrm{tn}(y)=\mathrm{tn}(x)+1$ and $y_i=z_i$ for all $i<\mathrm{tn}(y)$ is 1/2. This shows that, if Carole does not play a devil's strategy and if her string is taken from $\{0,1\}^n$ uniformly at random, we need, on average, two successive calls to procedure `LinAlg` until we obtain a string y as desired.

Proposition 7.5 follows from the easy observation that it takes, on average, three iterations until both $y=x\oplus e^n_{\mathrm{tn}(x)}$, and $y=x\oplus\sum_{i=\mathrm{tn}(x)+1}^{n}e_i^n$ have been sampled. That is, even if Carole plays a devil's strategy, three calls of Algorithm 20 force her to accept one entry $z_{\mathrm{tn}(x)}\in\{0,1\}$. □

The case of $k\geq 3$ colors $\mathcal{C}=[0..k-1]$

The main argument of Proposition 7.5, namely that $\sum_{c=0}^{k-1}\mathrm{eq}(z,[c\dots c])=n$, seems hard to extend to more than two colors with no additional memory. However, having more than two colors can be exploited in a different way as it gives more than one way to mark the *tail* $[x_{\mathrm{tn}(x)}\dots x_n]$ of a search point x.

Proposition 7.6. *For $k\geq 3$ colors, Algorithm 21 satisfies the claims of Lemma 7.4.*

```
Algorithm 21: Routine LinAlg for $k \geq 3$ colors
1 Assumption: The string $x \in \mathcal{C}^n$ in the memory satisfies $\mathrm{tn}(x) < n$ and $x_i = z_i$
  for all $i < \mathrm{tn}(x)$;
2 With probability $(k-1)/k$ sample
  $y \in \{[x_1 \dots x_{\mathrm{tn}(x)-1}|j|x_{\mathrm{tn}(x)+1} \dots x_n] \mid j \in \mathcal{C}\backslash\{x_{\mathrm{tn}(x)}\}\}$ uniformly at random
  and with probability $1/k$ sample
  $y \in \{[x_1 \dots x_{\mathrm{tn}(x)-1}|j \dots j] \mid j \in \mathcal{C}\backslash\{x_{\mathrm{tn}(x)-1}\}\}$ uniformly at random;
3 Query $\mathrm{eq}(z,y)$;
4 if $y = [x_1 \dots x_{\mathrm{tn}(x)-1}|j|x_{\mathrm{tn}(x)+1} \dots x_n]$, $j \neq x_{\mathrm{tn}(x)}$ then
5     if $\mathrm{eq}(z,y) > \mathrm{eq}(z,x)$ then $\mathcal{M} \leftarrow \{(y,\mathrm{eq}(z,y))\}$ ; //$z_{\mathrm{tn}(x)} = j$
6 else
7     $\mathcal{M} \leftarrow \{(y,\mathrm{eq}(z,y))\}$;
```

Proof. Let $x\in\mathcal{C}^n$ with $\mathrm{tn}(x)<n$ and $x_i=z_i$ for all $i<\mathrm{tn}(x)$. If $y=[x_1\dots x_{\mathrm{tn}(x)-1}|j|x_{\mathrm{tn}(x)+1}\dots x_n]$, then clearly we have $\mathrm{eq}(z,y)>\mathrm{eq}(z,x)$ if and only if $y_{\mathrm{tn}(x)}=j=z_{\mathrm{tn}(x)}$. Therefore, all we need to show is that, using the strategy of Algorithm 21, it takes a constant number of guesses until for each $j\in\mathcal{C}$ there exists an $i_j\in\mathcal{C}\backslash\{j\}$ such that we have queried both $x=[z_1\dots z_{\mathrm{tn}(x)-1}|i_j\dots i_j]$ and $y=[z_1\dots z_{\mathrm{tn}(x)-1}|j|i_j\dots i_j]$ in two subsequent guesses. This follows essentially from the fact that k is constant.

More precisely—regardless of the current search point x—for any bitstring $y=[z_1\dots z_{\mathrm{tn}(x)-1}|j|i_j\dots i_j]$ the probability to sample y in the second of two subsequent calls to Algorithm 21 is constant. Therefore, the expected number of calls to Algorithm 21 until y is sampled is constant. The claim follows by the linearity of expectation. $\square$

7.4.2. Proof of Theorem 7.1

Building on `LinAlg` and the block-wise random guessing strategy introduced in Section 7.3, we can now present Paul's winning strategy for the single memory cell setting which proves Theorem 7.1.

Proof of Theorem 7.1. The structure of this proof is as follows. First we sketch the main ideas and give a high-level pseudo-code for the size-one memory-restricted strategy winning the black answer-peg only Mastermind game with k colors in $O(n/\log n)$ guesses. After fixing some notation, we then present more details for the different phases, in particular for the random guessing phase, which is the most critical part of this proof. We present here the details of Paul's strategy for the case of $k = 2$ colors. The generalization to $k \geq 3$ colors is pretty much straightforward. Some remarks on the differences between the case of $k = 2$ and $k \geq 3$ colors are given at the end of this proof.

Let us begin with a rough overview of Paul's strategy. He determines the first $n - \Theta(n/\log n)$ positions using random guessing, where he manages to store the random substrings and Carole's answers in the yet undetermined part of his one string in the memory. As in the proof of Theorem 7.2, he does so by iteratively determining blocks of length $s := \lceil \sqrt{n} \rceil$. This is the first phase of the algorithm. Then, using the linear query time strategy from Lemma 7.4, he determines the missing $\Theta(n/\log n)$ entries in $O(n/\log n)$ guesses (phase 2).

To distinguish between the sampling and the linear query time phase, Paul uses the last two entries $\texttt{suffix}(x) := [x_{n-1}x_n]$ of his string x. He has $\texttt{suffix}(x) = [01]$, when he is in the random guessing phase, and he uses $\texttt{suffix}(x) = [cc]$ for some $c \in \mathcal{C}$ to indicate that he applies calls to `LinAlg`. Once Paul has determined all but the last two entries (visible from $\text{tn}(x) = n - 1$), he simply needs to sample uniformly at random from the set of all $k^2 - 1$ remaining possible strings. This clearly determines z in a constant expected number of additional queries (phase 3). Therefore, the total expected number of guesses can be bounded by

$$\overbrace{\frac{n-2}{s}(1-\Theta(\log^{-1} n))}^{\text{number of blocks determined in phase 1}} \; \overbrace{O(\frac{s}{\log s})}^{\text{queries needed to determine any such block}} + \overbrace{O(\frac{n}{\log n})}^{\text{queries needed in phase 2}} + \overbrace{O(1)}^{\text{queries needed in phase 3}} = O(\frac{n}{\log n}).$$

A non-trivial part is the random guessing phase. As in the proof of Theorem 7.2, after guessing $t+k$ strings, we want to be able to regain the full guessing history. If we simply stored the random substring and Carole's reply in some unused part of x, then this changed memory would influence Carole's next answer and we would be unable to deduce the necessary information on the next guess from it. We solve this difficulty as follows. We store Carole's latest reply (i.e., value $\text{eq}(z,x)$ currently in the memory) and we sample new (random) substrings for the current block at the same time. Here we store the value $\text{eq}(z,x)$ in a part of x for which we know the entries of Carole's hidden code. By this, we can separate in Carole's next answer the influence of the just stored information from the one of the random guess. The precise description of this `Sampling` strategy will be presented below.

To gain the storage space for which we know the hidden code, we need to add another phase, phase 0, in which we apply $O(\log n)$ calls to the `LinAlg` procedure (cf. Lemma 7.4) until we have determined the first

$$\ell := \ell_n + 1 = \lceil \log_2 n \rceil + 2$$

positions of z. Here, $\ell_n = \lceil \log_2 n \rceil + 1$ denotes again the number of bits needed to encode in binary any value $v \in [0..n]$. This additional phase 0 does not change the overall asymptotic number of queries Paul needs to win the game.

The pseudo-code for this size-one memory-restricted strategy is given in Algorithm 22. Similar to the notation in the proof of Theorem 7.2, we denote for any $h \in [0..n]$ its binary encoding of length ℓ_n by $\texttt{binary}_{\ell_n}(h)$ and for $h \in [0..s]$ we denote its binary encoding of length $\ell_s := \lceil \log_2 s \rceil + 1$ by $\texttt{binary}_{\ell_s}(h)$. The current block of interest $i(x)$ is encoded in positions $\{n - \ell_s - 1, \ldots, n-2\}$, i.e., we have

$$i(x) := \sum_{h=0}^{\ell_s - 1} 2^h x_{n-2-h} \,,$$
$$B_{i(x)} := \{\ell + (i(x)-1)s + 1, \ldots, \ell + i(x)s\} \,,$$

and, consequently,

$$\texttt{BLOCK}_{i(x)}(x) := [x_{\ell+(i(x)-1)s+1} \cdots x_{\ell+i(x)s}] \,.$$

The total number of blocks which we determine via random guessing is

$$b := \lfloor \tfrac{n-2}{s}(1 - \tfrac{K}{\log_2 n}) \rfloor$$

for some suitable large constant K. The number of random guesses for each block is

$$t := \lceil (2+\varepsilon) \tfrac{s(1+2\log_2 k)}{\log_2 s - \log_2 k} \rceil \,,$$

where $\varepsilon > 0$ is an arbitrarily small constant. Lastly, the actual number of already sampled guesses for block $B_{i(x)}$ is denoted by $q(x)$. As in the proof of Theorem 7.2, $q(x)$ can be computed via

$$p_1(x) := \max\{i \in [n - \ell_s - 3] \mid x_i = 1\} \,,$$

the largest position $i < n - 2 - \ell_s$ with entry $x_i = 1$. The exact definition of $q(x)$ will be given in the description of the `OptimizeBlock` routine which, after t random samples have been sampled via the `Sampling` routine, determines $\texttt{BLOCK}_{i(x)}(z)$, stores it in $B_{i(x)}$ and increases the block counter $i(x)$ by one.

Let us now present a more detailed description of Algorithm 22.

As in the proof of Theorem 7.2 let us assume that Carole has chosen a fix code $z \in \mathcal{C}^n$ which she does not change during the game, i.e., to any of Paul's guesses x she

Algorithm 22: A size-one memory-restricted algorithm winning the k-color black answer-peg only Mastermind game in $O(n/\log n)$ guesses.

```
1  Initialization: Let $\mathcal{M} \leftarrow \emptyset$;
2  if $\mathcal{M} = \emptyset$ then
3      $x \leftarrow [c \dots c]$ for some $c \in \mathcal{C}$ chosen u.a.r.;
4      Query $\mathrm{eq}(z, x)$ and update $\mathcal{M} \leftarrow \{(x, \mathrm{eq}(z, x))\}$;
5  if $\exists c \in \mathcal{C}$ : $\mathtt{suffix}(x) = [cc] \wedge \mathrm{tn}(x) \le \ell$ then
6      LinAlg; //find the first $\ell$ entries $[z_1 \dots z_\ell]$
7  else if $\exists c \in \mathcal{C}$ : $\mathtt{suffix}(x) = [cc] \wedge \mathrm{tn}(x) = \ell + 1$ then
8      $x \leftarrow [\underbrace{0 \dots 0}_{\ell} | \underbrace{0 \dots 0}_{bs} | x_1 \dots x_\ell | \underbrace{0 \quad \dots \quad 0}_{n-(2\ell+bs+\ell_s+2)} \; |\mathtt{binary}_{\ell_s}(1)|01]$; //copy prefix
       (which coincides with the hidden code)
9      Query $\mathrm{eq}(z, x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z, x))$;
10 else if $\mathtt{suffix}(x) = [01] \wedge i(x) \le b \wedge q(x) < t + k$ then
11     Apply Sampling;
12 else if $\mathtt{suffix}(x) = [01] \wedge i(x) \le b \wedge q(x) = t + k$ then
13     Apply OptimizeBlock;
14 else if $\mathtt{suffix}(x) = [01] \wedge i(x) = b + 1$ then
15     $x \leftarrow [x_{\ell+bs+s+1} \dots x_{2\ell+bs+s+1} | x_{\ell+1} \dots x_{\ell+bs} | c \dots c]$ with $c \in \mathcal{C} \backslash \{x_{\ell+bs}\}$ u.a.r.;
16     Query $\mathrm{eq}(z, x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z, x))$; //prepares $x$ for
       LinAlg
17 else if $\exists c \in \mathcal{C}$ : $\mathtt{suffix}(x) = [cc] \wedge \ell + bs < \mathrm{tn}(x) \le n - 2$ then
18     LinAlg;
19 else if $\exists c \in \mathcal{C}$ : $\mathtt{suffix}(x) = [cc] \wedge \mathrm{tn}(x) = n - 1$ then
20     Sample $y \in \{[x_1 \dots x_{n-2} | p] \mid p \in \mathcal{C}^2\} \backslash \{x\}$ uniformly at random;
21     Query $\mathrm{eq}(z, y)$;
22     if $\mathrm{eq}(z, y) = n$ then $\mathcal{M} \leftarrow \{(y, \mathrm{eq}(z, y))\}$ ; //secret code found
23 Go to line 2;
```

replies $\mathrm{eq}(z, x)$. By adopting a worst-case view below, we implicitly still allow Carole to change z as long as the new choice is consistent with all previous replies.

If in any iteration we find an x with $\mathrm{eq}(z, x) = n$, we have $x = z$ and we are done. Thus, in what follows we always assume $\mathrm{eq}(z, x) < n$.

Initialization of Algorithm 22, lines 1–4. For initialization, Paul picks a $c \in \mathcal{C}$ uniformly at random and guesses the "all-c's string" of length n, $x = [c \dots c]$. He updates the memory $\mathcal{M} \leftarrow \{(x, \mathrm{eq}(z, x))\}$ accordingly. This memory satisfies all conditions of line 1 of routine `LinAlg` (Algorithms 20 and 21) with $\mathrm{tn}(x) = 1$.

Phase 0 of Algorithm 22, lines 5–6. To this string, Paul applies successive calls to the routine `LinAlg`. By Lemma 7.4 he finds a string $y \in \mathcal{C}^n$ with $y_i = z_i$ for all $i \le \ell$, and $\mathrm{tn}(y) = \ell + 1$ in an expected number of $O(\ell)$ guesses. These bits shall be used in phase 1 of Algorithm 22 to indicate the status of the `Sampling` routine (first position) and for storing Carole's latest reply $\mathrm{eq}(z, x) \in [0..n]$ (positions $\{2, \dots, \ell\}$).

We shall describe this in more detail below.

Intermediate step, lines 7–9. After Paul has determined the first ℓ entries, he needs to prepare the string for the random guessing phase, which is the main part of Algorithm 22. Since we want to use the first ℓ entries to store reference values, we need to make a copy of the prefix (which, by construction, coincides with Carole's hidden code). To this end, we query in line 7 the string

$$y = [\ \underbrace{0\dots0}_{\ell+bs \text{ entries}}\ |x_1 \dots x_\ell|\ \underbrace{0 \quad \dots \quad 0}_{n-(2\ell+bs+\ell_s+2) \text{ entries}}\ |\underbrace{\texttt{binary}_{\ell_s}(1)}_{\ell_s \text{ entries}}|01]\,,$$

where x is the string that is currently in the memory, i.e., the string we obtained through phase 0[1] and $bs = n - \Theta(n/\log n)$ is the number of positions Paul determines via random guessing. As mentioned in the overview, the last two entries $\texttt{suffix}(x) = [x_{n-1}x_n] = [01]$ indicate that we are entering the second phase. Throughout the game we have

$$\texttt{suffix}(x) = \begin{cases} [01], & \text{if we are in the second phase of the algorithm}\,, \\ [cc], & \text{for some } c \in \mathcal{C}, \text{ otherwise.} \end{cases}$$

In positions $\{n - \ell_s - 1, \dots, n-2\}$ we initialize $i(x) = 1$ to indicate that, in what follows, we aim at determining the first block, B_1, of the secret code z.

After guessing y and updating the memory by replacing the current one with $\{(y, \mathrm{eq}(z,y))\}$, Paul enters the first phase. The overall expected number of queries needed until this point is $O(\ell) = O(\log n)$.

First phase of Algorithm 22, lines 10–13. The first phase is the main phase of Algorithm 22. In this phase, Paul determines all but $n - \Theta(n/\log n)$ entries by iteratively determining blocks of length s via random guessing. In total, he determines b such blocks in this phase. The description of the routines `Sampling` (in which k reference strings and t random samples are queried for the $i(x)$-th block $B_{i(x)}$) and `OptimizeBlock` (in which we use the reference strings and the random guesses to determine $\texttt{BLOCK}_{i(x)}(z)$, the $i(x)$-th block of the secret code z) is quite technical. We present the details after the description of the remaining phases.

Second phase of Algorithm 22, lines 14–18. In the second phase of Algorithm 22 we again apply successive calls to routine `LinAlg` to determine all but the last two remaining entries. To this end, we first need to prepare the string. This is done in lines 14–16 of Algorithm 22 (intermediate step). The string x queried in line 16 satisfies $x_i = z_i$ for all $1 \le i \le \ell + bs$. And, by definition, it also satisfies $x_{\mathrm{tn}(x)-1} \neq x_{\mathrm{tn}(x)}$ with $\mathrm{tn}(x) = \ell + bs + 1$.

From Lemma 7.4 we infer that via routine `LinAlg` we find a string x with $\mathrm{tn}(x) = n-1$ and $x_i = z_i$ for all $1 \le i \le n-2$ in an expected number of $O(n-2-(\ell+bs)) = O(n/\log n)$ queries. These are lines 17 and 18 of Algorithm 22.

Third phase of Algorithm 22, lines 19–22. Comparable to the last step of Algorithm 19, all we need to do in the last phase of Algorithm 22 is to determine the last two entries. This is done by sampling y uniformly at random from the set of possible target strings $\{[x_1 \dots x_{n-2}|p] \mid p \in \{0,1\}^2\} \backslash \{x\}$ and we find $y = z$ after a

[1]That is, we have $\forall i \le \ell : x_i = z_i$ and $\exists c \in \mathcal{C}\backslash\{x_\ell\} \forall i \ge \ell+1 : x_i = c$.

constant expected number of queries. This phase is recognized by the algorithm by the fact that $\mathrm{tn}(x) = n-1$. Note that we have $\mathrm{tn}(x) \leq n-2$ in the `LinAlg` phases—phases 0 and 2—and that we have $\mathrm{tn}(x) = n$ in phase 1.

In the remainder of this proof we present the details of the first phase of Algorithm 22, the random sampling routine `Sampling` and the `OptimizeBlock` routine. As mentioned above, this description requires some technicalities. Therefore, we split it into the following parts:

Part I In the first part, we present the general structure of the guesses made in the sampling phase. Here, we shall also show that the n positions are indeed sufficient to store, for any of the b blocks of length s, all necessary information about the samples.

Part II The second part, which is brief, provides further notation used in the pseudo-code of Algorithm 23.

Part III The main part is the third one. Here we show how the contributions $\Delta_{i(x)}(r) \in [0..s]$ of the random samples $r \in \mathcal{C}^s$ can be computed. This also shows that indeed after sampling the t random guesses for block $B_{i(x)}$, it is possible to regain the full query history using only the information that has been stored in the memory. This is clearly the most technical part of this proof.

Part IV We conclude the description of phase 1 in the fourth part, where we explain how the memory is being updated, once the entries $z_{|B_{i(x)}} = \texttt{BLOCK}_{i(x)}(z)$ of the secret code z in the $i(x)$-th block have been determined.

Part I. The general structure of a query x in the first phase is the following.

$$\begin{aligned} x = [\, & \underbrace{x_1}_{(1)} \,|\, \underbrace{\texttt{binary}_{\ell_n}(\mathrm{eq}(z,y))}_{(2)} \,|\, \underbrace{\texttt{opt}(B_1)|\ldots|\texttt{opt}(B_{i(x)})}_{(3)} \,|\, \underbrace{r}_{(4)} \,|\, \underbrace{0\ldots0}_{(5)} \,|\, \underbrace{z_1\ldots z_\ell}_{(6)} \,| \\ & \underbrace{\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x^0))|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x^1))|1}_{(7)} \,| \\ & \underbrace{\texttt{binary}_{\ell_n}(\mathrm{eq}(z,\texttt{ref}^{(1)}))|r^{(1)}|\Delta_{i(x)}(r^{(1)})|1|\ldots}_{(8)} | \\ & \underbrace{\texttt{binary}_{\ell_n}(\mathrm{eq}(z,\texttt{ref}^{(t')}))|r^{(t')}|\Delta_{i(x)}(r^{t'})|1}_{(8)\text{ continued}} \,|0\ldots0|\, \underbrace{\texttt{binary}_{\ell_s}(i(x))}_{(9)} \,|\, \underbrace{01}_{(10)} \,]\,, \end{aligned} \tag{7.1}$$

where we use

(1) the first entry $x_1 \in \{0,1\}$ to indicate whether this string contains new and not yet stored information ($x_1 = 1$) or whether we are just doing a storage operation ($x_1 = 0$), through which we add to x all necessary information from the previous guess. An explanation of these operation will be given below;

(2) ℓ_n entries for encoding the value $\mathrm{eq}(z,y) \in [0..n]$ of the string y that is currently stored in the memory cell ($\mathrm{eq}(z,y)$ serves as reference value),

(3) $(i(x)-1)s$ entries for the already determined blocks $B_1, \ldots, B_{i(x)-1}$,

(4) s entries for the current block $B_{i(x)}$ of interest. If we are sampling new information (i.e., if $x_1 = 1$), then the substring r is a string taken from $\mathcal{C}^s$ uniformly at random and r is the all-zeros string of length s otherwise;

(5) $(b - i(x))s$ zeros for the yet untouched blocks $B_{b'}$ with $i(x) < b' \leq b$,

(6) ℓ entries for storing the length-ℓ prefix that coincides with Carole's hidden code (this we obtained trough phase 0),

(7) $2\ell_n + 1$ entries for storing the values $\text{eq}(z, x^0)$ and $\text{eq}(z, x^1)$ of the two reference strings x^0 and x^1 (explanation follows),

(8) $t'(\ell_n + s + \ell_s + 1)$, $t' \leq t$, entries for storing, for each random sample, (i) the value $\text{eq}(z, \texttt{ref})$ for a reference string $\texttt{ref}$ (in binary, requires ℓ_n positions), (ii) the random sample $r \in \mathcal{C}^s$ itself, (iii) its contribution $\Delta_{i(x)}(r) \in [0..s]$ to Carole's reply (in binary, requires ℓ_s positions), and (iv) one additional "1" (to ease the computation of the number of guesses $q(x)$ via $p_1(x)$; details follow),

(9) ℓ_s entries for encoding in binary $i(x)$, the index of the block, which we are currently trying to determine, and

(10) the last two entries, $\texttt{suffix}(x)$, for indicating the current phase of the algorithm.

Clearly, one critical part is the limited storage capacity. For this reason, let us show next that we have enough positions to store all the information needed to compute $\mathcal{S}_{i(x)}^{\text{consistent}}$, the set of all strings consistent with Carole's replies for the random guesses in the $i(x)$-th block $B_{i(x)}$.

For determining the $i(x)$-th block $\texttt{BLOCK}_{i(x)}(z)$ of z, we sample $t = \Theta(s/\log s)$ random guesses. In addition, equivalently to the proof of Theorem 7.2, we need again two reference strings x^0 and x^1 (reference (7) in equation (7.1)). These two reference strings will be needed to infer the contributions $\Delta_{i(x)}(r)$ of the random samples $r \in \mathcal{C}^s$ in the $i(x)$-th block.

From the structure of the guesses presented in equation (7.1) above, we infer that the total storage requirement can be bounded from above by

$$\begin{aligned} 1 + \ell_n + &bs + \ell + 2\ell_n + 1 + t(\ell_n + s + \ell_s + 1) + \ell_s + 2 \\ &= bs + ts + o(n/\log n) \leq n\left(1 - \frac{K}{\log_2 n}\right) + \Theta(n/\log n) + o(n/\log n) < n \end{aligned}$$

for sufficiently large, but constant K and sufficiently large n. This shows that, for sufficiently large n, Paul indeed can store all information needed to compute $\mathcal{S}_{i(x)}^{\text{consistent}}$ in one single string of length n.

Part II. The pseudo-code of routine $\texttt{Sampling}$ is Algorithm 23. For all $b' < i(x)$ we set

$$\texttt{opt}(B_{b'}) := [x_{\ell+(b'-1)s+1} \ldots x_{\ell+b's}]\,,$$

the entries of x in the b'-th block. The notation "opt" is justified by the fact that we shall have $\texttt{opt}(B_{b'}) = \texttt{BLOCK}_{b'}(z)$ for all $b' < i(x)$. Furthermore, let

$$\texttt{AddReferenceStringInfo}(x) := [\underbrace{0\dots0}_{(1),(2)}|\underbrace{\texttt{opt}(B_1)|\dots|\texttt{opt}(B_{i(x)-1})}_{(3)}|\underbrace{0\dots0}_{(4)}|$$
$$\underbrace{x_{\ell+i(x)s+1}\dots x_{2\ell+bs}}_{(5),(6)}|\underbrace{x_2\dots x_\ell|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x))|1}_{(7)}|\underbrace{x_{2\ell+bs+2\ell_n+2}\dots x_n}_{(*)}],$$

where the references in the expression above are the same as the ones used in equation (7.1) and where $(*)$ is simply a copy of the last entries of x. That is, the $\texttt{AddReferenceStringInfo}(x)$ operation adds to x the values $\mathrm{eq}(z,x^0)$ and $\mathrm{eq}(z,x^1)$ to the memory and each of these values is stored in binary notation of length ℓ_n. Lastly, we denote by $\texttt{Add}(\mathrm{eq}(z,x))$ the operation

$$\texttt{Add}(\mathrm{eq}(z,x)) := [\underbrace{0\dots0}_{(1),(2)}|\underbrace{\texttt{opt}(B_1)|\dots|\texttt{opt}(B_{i(x)-1})}_{(3)}|\underbrace{0\dots0}_{(4)}|\underbrace{x_{\ell+i(x)s+1}\dots x_{p_1(x)}}_{(5),(6),(7),(8)}|$$
$$\underbrace{x_2\dots x_\ell|\texttt{BLOCK}_{i(x)}(x)|\texttt{binary}_{\ell_s}(\Delta_{i(x)}(\texttt{BLOCK}_{i(x)}(x)))|1}_{(\dagger)}|$$
$$\underbrace{x_{p_1(x)+\ell_n+s+\ell_s+2}\dots x_n}_{(*)}], \tag{7.2}$$

which adds (in substring $(\dagger)$) to the memory

- a copy of the value $\mathrm{eq}(z,\texttt{ref})$ of a reference string $\texttt{ref}$ (which was previously stored in positions $\{2,\dots,\ell\}$),
- the random sample $\texttt{BLOCK}_{i(x)}(x)$ of the last guess,
- the contribution $\Delta_{i(x)}(\texttt{BLOCK}_{i(x)}(x))$ of the random sample $\texttt{BLOCK}_{i(x)}(x)$ to $\mathrm{eq}(z,x)$, and
- the one additional "1" that shall ease the computation of $q(x)$, the number of already queried samples.

All other but the first ℓ entries (which are set to zero) are copied from x. The references in equation (7.2) are the same as in equation (7.1).

Part III. Let us now show in detail how to infer the contributions $\Delta_{i(x)}(r)$ of the random guesses. For clarity, we show how to do this for the first block, i.e., for the positions $\{\ell+1\dots\ell+s\}$. The procedure is similar for all other blocks and we shall comment on this case at the end of this part.

First note that after the intermediate step in lines 7 to 9 of Algorithm 22, the algorithm enters the routine $\texttt{Sampling}$ with $\mathcal{M} = \{(x^0, \mathrm{eq}(z,x^0))\}$, where

$$x^0 = [\underbrace{0\ \dots\ 0}_{\ell+bs \text{ entries}}|z_1\dots z_\ell|\underbrace{0\ \dots\ 0}_{n-(2\ell+bs+\ell_s+2)\text{ entries}}|\underbrace{\texttt{binary}_{\ell_s}(1)}_{\ell_s\text{ entries}}|01],$$

Algorithm 23: The `Sampling` routine for $k = 2$ colors.

```
1  Assumption: Memory $\mathcal{M} = \{(x, \mathrm{eq}(z,x))\}$ satisfies $\mathrm{eq}(z,x) < n$,
   $\texttt{suffix}(x) = [01]$, $i(x) \le b$, and $q(x) < t + 2$;
2  if $q(x) = 0 \wedge x_1 = 0$ then
3      $x \leftarrow [1|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x))|\texttt{opt}(B_1)|\dots|\texttt{opt}(B_{i(x)-1})|1\dots1|x_{\ell+i(x)s+1}\dots x_n]$;
4      Query $\mathrm{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z,x))$;
5  else if $q(x) = 0 \wedge x_1 = 1$ then
6      $x \leftarrow \texttt{AddReferenceStringInfo}(x)$ ;
7      Query $\mathrm{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z,x))$;
8  else if $2 \le q(x) < t + 2 \wedge x_1 = 0$ then
9      $x \leftarrow [1|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x))|\texttt{opt}(B_1)|\dots|\texttt{opt}(B_{i(x)-1})|r|x_{\ell+i(x)s+1}\dots x_n]$ for
       $r \in \mathcal{C}^s$ chosen u.a.r.;
10     Query $\mathrm{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z,x))$;
11 else if $2 \le q(x) < t + 2 \wedge x_1 = 1$ then
12     $x \leftarrow \texttt{Add}(\mathrm{eq}(z,x))$;
13     Query $\mathrm{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z,x))$;
```

and it queries in the first sampling iteration (lines 2–4 of Algorithm 23)

$$x^1 = [\underbrace{1|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x^0))}_{1+\ell_n=\ell \text{ entries}}|\underbrace{1\dots1}_{s \text{ entries}}|\underbrace{0 \quad \dots \quad 0}_{(b-1)s \text{ entries}}|z_1\dots z_\ell|$$
$$\underbrace{0 \quad \dots \quad 0}_{n-(2\ell+bs+\ell_s+2) \text{ entries}} \qquad |\texttt{binary}_{\ell_s}(1)|01]$$

with the all-ones substring in the first block. We can compute the contribution $\tilde{f}(x^1)$ of the first ℓ entries $[1|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x^0))]$ to the value $\mathrm{eq}(z,x^1)$ via

$$\tilde{f}(x^1) := \mathrm{eq}([z_1\dots z_\ell],[1|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x^0))]) = \mathrm{eq}([x^1_{\ell+bs+1}\dots x^1_{2\ell+bs}],[x^1_1\dots x^1_\ell])\,,$$

and, by the same reasoning, the contribution of the first ℓ entries in x^0 to $\mathrm{eq}(z,x^0)$ via $\tilde{f}(x^0) = \mathrm{eq}([x^0_{\ell+bs+1}\dots x^0_{2\ell+bs}],[0\dots0])$. Let us, for a moment, assume that we now had $\mathcal{M} = \{(x^1, \mathrm{eq}(z,x^1))\}$ and that we had sampled another string

$$y = [z_1\dots z_\ell|\underbrace{\;r\;}_{s \text{ entries}}|\underbrace{0\dots0}_{(b-1)s \text{ entries}}|z_1\dots z_\ell|\underbrace{0\dots0}_{n-(2\ell+bs+\ell_s+2) \text{ entries}}|\texttt{binary}_{\ell_s}(1)|01]$$

for some random substring $r \in \mathcal{C}^s$. Then we could compute the contribution of the random entries r in the first block B_1 of y via

$$\tilde{\Delta}_1(r) = \mathrm{eq}(z,y) - \frac{\mathrm{eq}(z,x^0) + \mathrm{eq}(z,x^1) + (\ell - \tilde{f}(x^0)) + (\ell - \tilde{f}(x^1)) - s}{2}\,.$$

Key to this equality is the fact that the first ℓ entries of y contribute ℓ to Carole's response $\mathrm{eq}(z,y)$ to guess y, whereas the first ℓ entries of x^0 and x^1 contribute $\tilde{f}(x^0) + \tilde{f}(x^1)$ to the sum $\mathrm{eq}(z,x^0) + \mathrm{eq}(z,x^1)$ and the fact that the entries in the first block,

$[0\dots0]$ and $[1\dots1]$, respectively, contribute in total s toward $\text{eq}(z,x^0)+\text{eq}(z,x^1)$. All other entries $x_i, y_i, i > \ell + s$ contribute either 2 or 0 to the sum $\text{eq}(z,x^0) + \text{eq}(z,x^1)$ and every entry contributes 2 if and only if it contributes 1 to the value $\text{eq}(z,y)$.

Note however, that we would now have to choose now which of the strings to keep in the memory and we would eventually loose the information $\text{eq}(z,x^1)$. Therefore, in lines 6 and 7 in Algorithm 23, we first query the reference string

$$\begin{aligned} x^2 &= \texttt{AddReferenceStringInfo}(x^1)\\ &= [\underbrace{0\dots0}_{\ell\text{ entries}} \mid \underbrace{0\dots0}_{s\text{ entries}} \mid \underbrace{0\dots0}_{(b-1)s} | z_1\dots z_\ell | \underbrace{\texttt{binary}_{\ell_n}(\text{eq}(z,x^0))|\texttt{binary}_{\ell_n}(\text{eq}(z,x^1))|1}_{2\ell_n+1\text{ entries}} | \\ &\qquad \underbrace{0\dots0}_{\substack{n-(2\ell+bs+2\ell_n+\\ 1+\ell_s+2)\text{ entries}}} \qquad |\texttt{binary}_{\ell_s}(1)|01]\,. \end{aligned}$$

This query is needed only to store the values $\text{eq}(z,x^0)$ and $\text{eq}(z,x^1)$ of both reference strings. Since adding the substring $[\texttt{binary}_{\ell_n}(\text{eq}(z,x^0))|\texttt{binary}_{\ell_n}(\text{eq}(z,x^1))|1]$ to the memory string again changes the number of positions in which the guess and Carole's hidden string coincide, we need to store this information in the next query as well. More precisely, we have that x^0 and x^2 differ in exactly the substring $[\texttt{binary}_{\ell_n}(\text{eq}(z,x^0))|\texttt{binary}_{\ell_n}(\text{eq}(z,x^1))|1]$ (i.e., they differ in positions $\{bs+2\ell+1,\dots,bs+2\ell+2\ell_n+1\}$), and the contribution of this substring (compared to the all-zeros substring which it replaces) is $\text{eq}(z,x^2)-\text{eq}(z,x^0)$.

Furthermore, we need to indicate that we are sampling a new random substring. This is the first position in the string. Instead of querying y as above, Algorithm 23 queries (lines 9 and 10 of Algorithm 23) as first random sample for the first block a string

$$\begin{aligned} x^3 = [1|&\texttt{binary}_{\ell_n}(\text{eq}(z,x^2))|r^{(1)}|0\dots0|z_1\dots z_\ell|\texttt{binary}_{\ell_n}(\text{eq}(z,x^0))|\\ &\texttt{binary}_{\ell_n}(\text{eq}(z,x^1))|1|0\dots0|\texttt{binary}_{\ell_s}(1)|01]\,, \end{aligned}$$

where the substring $r^{(1)} \in \mathcal{C}^s$ in B_1 is taken uniformly at random. The number of zeros in the first all-zeros substring is again $(b-1)s$ and in the second all-zeros substring it is $n-(2\ell+bs+2\ell_n+1+\ell_s+2)$. Now, in the same fashion as above, we can compute the contribution $\Delta_1(r^{(1)}) = \text{eq}([z_{\ell+1}\dots z_{\ell+s}])$ of the substring $r^{(1)} \in \mathcal{C}^s$ via

$$\begin{aligned} \Delta_1(r^{(1)}) = \text{eq}(z,x^3) - \Big(&\frac{\text{eq}(z,x^0)+\text{eq}(z,x^1)+(\ell-\tilde f(x^1))+(\ell-\tilde f(x^0))-s}{2}\\ &+(\text{eq}(z,x^2)-\text{eq}(z,x^0))-(\ell-\tilde f(x^3))\Big)\,. \end{aligned} \tag{7.3}$$

All information needed for this computation is contained in the string x^3 itself.

In the next guess we store both the reference value $\text{eq}(z,x^2)$ (positions $\{2,\dots,\ell$ of $x^3)$ as well as the contribution $\Delta_1(r^{(1)})$. Of course, we also need to store the random guess $r^{(1)} = \texttt{BLOCK}_1(x^3)$ itself. Thus, we query (lines 12 and 13 in Algorithm 23) in the next iteration of Algorithm 22

$$x^4 = [\underbrace{0\dots0}_{\ell}|\underbrace{0\dots0}_{s}|\underbrace{0\dots0}_{(b-1)s}|z_1\dots z_\ell|\texttt{binary}_{\ell_n}(\text{eq}(z,x^0))|\texttt{binary}_{\ell_n}(\text{eq}(z,x^1))|1|$$
$$\underbrace{\texttt{binary}_{\ell_n}(\text{eq}(z,x^2))}_{=[x^3_2\dots x^3_\ell]}|\underbrace{\texttt{BLOCK}_1(x^3)}_{=r^{(1)}}|\underbrace{\texttt{binary}_{\ell_s}(\Delta_1(r^{(1)}))}_{\text{see equation (7.3)}}|1|0\dots0|\texttt{binary}_{\ell_s}(1)|01].$$

Note that since $\Delta_1(r^{(1)}) \in [0..s]$, we can encode this value using ℓ_s positions only.

Continuing like this we are able to compute, in any iteration of the first phase, the contributions $\Delta_{i(x)}(r)$ of the random substrings $r \in \mathcal{C}^s$.

As in the proof of Theorem 7.2 we need to be able to compute how many random guesses have queried already for the current block of interest. As indicated above, this can be derived from $p_1(x)$ as follows. For any random guess $r \in \mathcal{C}^s$ we need $\ell_n + s + \ell_s + 1$ entries for storing all information that will be needed later to regain the full guessing history. Furthermore, we need $2\ell_n + 1$ entries for storing the values $\text{eq}(z,x^0)$ and $\text{eq}(z,x^1)$ of the two reference strings x^0 and x^1, and we store information only in positions $i > 2\ell + bs$. Hence, the number of guesses that have been queried already for block $B_{i(x)}$ can be computed as

$$q(x) = \begin{cases} 0, & \text{if } p_1(x) \le 2\ell + bs \text{ and } x_1 = 0\,, \\ 1, & \text{if } p_1(x) \le 2\ell + bs \text{ and } x_1 = 1\,, \\ 2 + \frac{p_1(x) - (2\ell + bs + 2\ell_n + 1)}{\ell_n + s + \ell_s + 1}, & \text{otherwise.} \end{cases}$$

Algorithm 24: The `OptimizeBlock` routine

```
1 Assumption: Memory $\mathcal{M} = \{(x, \text{eq}(z,x))\}$ satisfies $\text{eq}(z,x) < n$,
  $\texttt{suffix}(x) = [01]$, $i(x) \le b$, and $q(x) = t + 2$;
2 if $x_1 = 0$ then
3     $y \leftarrow [1\dots1|\texttt{opt}(B_1)|\dots|\texttt{opt}(B_{i(x)-1})|w|x_{\ell+i(x)s+1}\dots x_n]$ for $w \in \mathcal{S}^{\text{consistent}}_{i(x)}$
      chosen u.a.r.;
4     Query $\text{eq}(z,y)$;
5     if $\Delta_{i(x)}(\mathit{BLOCK}_{i(x)}(y)) = s$ then $\mathcal{M} \leftarrow \{(y, \text{eq}(z,y))\}$ ; //$w = \texttt{BLOCK}_{i(x)}(z)$
6 else
7     $x \leftarrow \texttt{Update}(x)$;
8     Query $\text{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \text{eq}(z,x))$; //string prepared
      for determining the next block
```

After querying t random guesses (i.e., after querying a total number of $t+k$ guesses) for the first block, we regain the full guessing history from the string x then stored in the memory as follows. The i-th random sample $r^{(i)} \in \mathcal{C}^s$ which we guessed for the first block is

$$r^{(i)} := [x_{2\ell+bs+2\ell_n+1+(i-1)(\ell_n+s+\ell_s+1)+\ell_n+1} \cdots x_{2\ell+bs+2\ell_n+1+(i-1)(\ell_n+s+\ell_s+1)+\ell_n+s}]\,,$$

and the corresponding query was

$$y^{(i)} := [1|x_{2\ell+bs+2\ell_n+1+(i-1)(\ell_n+s+\ell_s+1)+1} \cdots x_{2\ell+bs+2\ell_n+1+(i-1)(\ell_n+s+\ell_s+1)+\ell_n}| \\ r^{(i)}|x_{\ell+s+1} \dots x_{2\ell+bs+2\ell_n+1+(i-1)(\ell_n+s+\ell_s+1)}|0\dots0|x_{n-\ell_s-1}\dots x_n] \,.$$

We have stored in binary the contribution $\Delta_1(r^{(i)})$ of $r^{(1)}$ to the overall function value $\mathrm{eq}(z, y^{(i)})$ in positions

$$\{2\ell + bs + 2\ell_n + 1 + (i-1)(\ell_n + s + \ell_s + 1) + \ell_n + s + 1 \dots \\ 2\ell + bs + 2\ell_n + 1 + (i-1)(\ell_n + s + \ell_s + 1) + \ell_n + s + \ell_s\}$$

and thus we have

$$\Delta_1(r^{(i)}) = \sum_{i=0}^{\ell_s-1} 2^i x_{2\ell+bs+2\ell_n+1+(i-1)(\ell_n+s+\ell_s+1)+\ell_n+s+\ell_s-i} \,.$$

By Theorem 7.3, the expected size of

$$\mathcal{S}_1^{\text{consistent}} = \{w \in \{0,1\}^s \mid \forall i \le t : \mathrm{eq}(w, r^{(i)}) = \Delta_1(r^{(i)})\}$$

is bounded from above by $1+1/s$. That is, we can now identify $\texttt{BLOCK}_1(z)$ in a constant number of guesses. These are lines 3 – 5 of routine `OptimizeBlock` (Algorithm 24).

As mentioned above, determining the other blocks $2, \dots, b$ is similar. In these iterations, the $(i(x)-1)s$ entries in positions $\{\ell+1, \dots, \ell+(i(x)-1)s\}$ are already optimized, that is, they coincide with Carole's hidden string z. Thus, they are not changed in any further iteration of Algorithm 24.

Part IV. Once $\texttt{BLOCK}_{i(x)}(z) = [z_{\ell+(i(x)-1)s+1} \cdots z_{\ell+i(x)s}]$ has been determined, we need to update the memory such that we can start determining the entries of the next block. These are lines 7 and 8 in Algorithm 24. Here we abbreviate

$$\texttt{Update}(x) := [\underbrace{0\dots0}_{(a)}|\underbrace{\texttt{opt}(B_1)|\dots|\texttt{opt}(B_{i(x)})}_{(b)}|\underbrace{0\dots0}_{(c)}|\underbrace{x_{\ell+bs+1}\dots x_{2\ell+bs}}_{(d)}| \\ \underbrace{0\dots0}_{(e)}|\underbrace{\texttt{binary}_{\ell_s}(i(x)+1)}_{(f)}|\underbrace{01}_{(g)}] \,,$$

where

(a) the first ℓ entries are set to zero,

(b) we now have $i(x)$ already determined blocks,

(c) the new block of interest, block $i(x)+1$, as well as all blocks $b' > i(x)+1$ are (still) set to zero,

(d) we keep the copy of the prefix $[z_1 \dots z_\ell]$ in positions $\{\ell+bs+1, \dots, 2\ell+bs\}$,

(e) all information that we have used in the previous query to determine block $B_{i(x)}$ is removed (and set to zero),

(f) the index for the current block of interest is increased by one, and

(g) the last two entries still indicate the second phase.

The case of $k \geq 3$ colors. For the general case, the main strategy as given by Algorithm 22 remains the same. What needs to be changed is the `Sampling` routine, where instead of sampling only two reference strings x^0 and x^1, we need to sample k reference strings $x^0, x^1, \dots, x^{k-1}$ with $\texttt{BLOCK}_{i(x)}(x^c) = [c \dots c]$ for all $c \in [0..k-1]$.

Algorithm 25: The `Sampling` routine for $k \geq 3$ colors.

```
Assumption: Memory $\mathcal{M} = \{(x, \mathrm{eq}(z,x))\}$ satisfies $\mathrm{eq}(z,x) < n$, $\texttt{suffix}(x) = [01]$, $i(x) \leq b$, and $q(x) < t + k$;
if $q(x) = 0 \wedge x_1 = 0$ then
    $x \leftarrow [1|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x))|\texttt{opt}(B_1)|\dots|\texttt{opt}(B_{i(x)-1})|1\dots1|x_{\ell+i(x)s+1}\dots x_n]$;
    Query $\mathrm{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z,x))$;
else if $q(x) = 0 \wedge x_1 = 1$ then
    $x \leftarrow \texttt{AddReferenceStringInfo}(x)$ ;
    Query $\mathrm{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z,x))$;
else if $2 \leq q(x) < k \wedge x_1 = 0$ then
    $x \leftarrow$
    $[1|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x))|\texttt{opt}(B_1)|\dots|\texttt{opt}(B_{i(x)-1})|q(x)\dots q(x)|x_{\ell+i(x)s+1}\dots x_n]$;
    Query $\mathrm{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z,x))$;
else if $2 \leq q(x) < k \wedge x_1 = 1$ then
    $x \leftarrow \texttt{AddReferenceStringInfo}_2(x)$ ;
    Query $\mathrm{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z,x))$;
else if $k \leq q(x) < t + k \wedge x_1 = 0$ then
    $x \leftarrow [1|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x))|\texttt{opt}(B_1)|\dots|\texttt{opt}(B_{i(x)-1})|r|x_{\ell+i(x)s+1}\dots x_n]$ for
    $r \in \mathcal{C}^s$ chosen u.a.r.;
    Query $\mathrm{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z,x))$;
else if $k \leq q(x) < t + k \wedge x_1 = 1$ then
    $x \leftarrow \texttt{Add}(\mathrm{eq}(z,x))$;
    Query $\mathrm{eq}(z,x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z,x))$;
```

Algorithm 25 shows the generalized sampling routine. Here we define

$$\begin{aligned}\texttt{AddReferenceStringInfo}_2(x) :=&\\ &[\underbrace{0\dots0}_{(1),(2)}|\underbrace{\texttt{opt}(B_1)|\dots|\texttt{opt}(B_{i(x)-1})}_{(3)}|\underbrace{0\dots0}_{(4)}|\underbrace{x_{\ell+i(x)s+1}\dots x_{p_1(x)}}_{(5),(6),(7),(7')}|\\ &\underbrace{x_2\dots x_\ell|\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x))|1}_{(\dagger)}|\underbrace{x_{p_1(x)+\ell_n+s+\ell_s+2}\dots x_n}_{(*)}],\end{aligned}$$

where

(1)–(7) are the same references as in equation (7.1), (7') are the additional positions needed for storing the values $\mathrm{eq}(z, x^j)$ of the already queried reference strings $x^2, \ldots, x^{q(x)-1}$ (each requiring $2\ell_n + 1$ positions),

(†) we add the information of the $q(x)$-th reference string $x^{q(x)}$ to the memory (again requiring $2\ell_n + 1$ positions), and

($*$) is simply a copy of the last entries of the previous guess.

The substring $[x_2 \ldots x_\ell]$ is needed again to infer the contribution of the positions in which we added the information of the previous reference string $x^{q(x)-1}$. The reasoning is the same as in the case of $k = 2$ colors.

Since we added more reference string information, we have to adjust the definition of $q(x)$ accordingly. We need $2\ell_n + 1$ additional bits for each reference string x^j, $2 \leq j \leq k$, and we use $\ell_n + s + \ell_s + 1$ entries for storing the information of each random guess. Hence,

$$q(x) := \begin{cases} 0, \text{ if } p_1(x) \leq 2\ell + bs \text{ and } x_1 = 0\,, \\ 1, \text{ if } p_1(x) \leq 2\ell + bs \text{ and } x_1 = 1\,, \\ j, \text{ if } p_1(x) = 2\ell + bs + 2\ell_n + 1 + (j-2)(2\ell_n + 1) \text{ and } 2 \leq j < k\,, \\ k + \frac{p_1(x) - (2\ell + bs + (k-1)(2\ell_n + 1))}{\ell_n + s + \ell_s + 1}, \text{ otherwise.} \end{cases}$$

It is easily verified that all statements made in the above proof for $k = 2$ colors remain correct if we consider the general case of $k \geq 3$ colors. Only the computation of the contributions $\Delta_{i(x)}(\texttt{BLOCK}_{i(x)}(x))$, equation (7.3), becomes a bit more tedious. However, all calculations are straightforward. □

7.5. Conclusions

We have shown that determining the black-box complexity of $\textsc{OneMax}_n$ is closely related to identifying optimal winning strategies for the black answer-peg only version of the Mastermind game with $k = 2$ colors. We have analyzed the n holes, k color Mastermind game with the additional restriction that only one guess and its corresponding answer can be memorized by the codebreaker. We showed that this does not change the asymptotic runtime of an optimal winning strategy. That is, we have shown that—even with a memory restricted to one—the codebreaker has a winning strategy that requires only $\Theta(n/\log n)$ guesses. This implies a size-one memory-restricted black-box complexity of $\textsc{OneMax}_n$ of order $\Theta(n/\log n)$.

In terms of developing a useful complexity theory for randomized search heuristics, our result indicates that adding a memory-restriction does not dissolve the problem of too low black-box complexities. We note without proof that the methods developed in this section (i.e., imitating iteration and program counters, storing useful information in nonrelevant parts of the search points etc.) can as well be applied to an unbiased memory-restricted black-box setting, provided large enough arity. Similarly, they can be applied to other problem classes as well. Hence we conclude that other restrictions to the black-box model are needed to obtain more useful complexity bounds. We present one such approach in the next section.

8 Ranking-Based Black-Box Models

In this section we propose an alternative black-box model. It builds on the paradigm that many randomized search heuristics—such as many evolutionary algorithms, hill climbers like Randomized Local Search, and ant colony optimization—do not exploit knowledge of absolute fitness values. Instead they use the objective values only to *compare* search points, cf. Section 2.2. To understand the influence of this behavior we introduce a new black-box model in which allow the black-box algorithms to exploit only the relative ranking of the search points queried so far. In other words, throughout the optimization process the black-box algorithm knows for any two already queried search points x and y only whether $f(x) < f(y)$, $f(x) = f(y)$, or $f(x) > f(y)$ holds.

We show that our new *ranking-based black-box model* gives more realistic complexity estimates for some problems. For example, the class of all binary-value functions, BinaryValue^*_n, has a black-box complexity of $O(\log n)$ in the unrestricted and in the unbiased black-box model, but has a ranking-based black-box complexity of $\Theta(n)$. Here, the lower bound is clearly the more interesting one. It holds already for the much smaller subclass BinaryValue_n of BinaryValue^*_n. That is, even if we know the order of the bit weights, we cannot expect to optimize a worst-case BinaryValue instance using less than $\Omega(n)$ queries. The upper bound is easily verified by a simple hill-climber, that, in arbitrary order, changes a single bit value of the current solution and accepts the new solution if it is better than the previous one. In summary, we see that for this function class, the ranking-based black-box complexity seems to give us a more natural complexity measure than the previous approaches.

On the other hand, we also present a ranking-based black-box algorithm that solves any OneMax_n function with $\Theta(n/\log n)$ queries. This indicates that for some problems the ranking-based model still allows too powerful algorithms.

The ranking-based black-box model has been studied implicitly in the conference version [DJTW03] of [DJW06]. In a proof studying a generalized class of the LeadingOnes_n functions, Droste et al. prove that the ranking-based black-box complexity of LeadingOnes_n is between $n/2 - o(n)$ and $n + 1$. Likewise, in the journal version [DJW06] they prove that the ranking-based black-box complexity of BinaryValue$_n$ is $\Omega(n/\log n)$. As mentioned above, here in this section, we improve the second bound to $\Omega(n)$ in Section 8.3 and, by similar arguments, we show that the ranking-based black-box complexity of LeadingOnes$_n$ is at least $n-1$.

The results presented in this section are based on the conference publication [DW11c]. They are joint work with Benjamin Doerr.

8.1. The Ranking-Based Black-Box Model

It has been commented by Nikolaus Hansen (INRIA Saclay, France; personal communication) that many standard randomized search heuristics do not take advantage of knowing the *exact* objective values. Rather, they create new search points based on how the objective values of the previously queried search points compare. That is, after having queried t fitness values $f(x^{(0)}), \ldots, f(x^{(t-1)})$, they rank the corresponding search points $x^{(0)}, \ldots, x^{(t-1)}$ according to their relative fitness. The selection of input individuals $x^{(i_1)}, \ldots, x^{(i_k)}$ for the next variation operator is based solely on this ranking.

We define the ranking induced by f as follows.

Definition 8.1 (Ranking induced by f). *Let S be a set, let $f : S \to \mathbb{R}$ be a function, and let C be a finite subset of S. The* ***ranking ρ_C of C induced by f*** *assigns to each element $c \in C$ the number of elements in C with a smaller f-value plus 1, formally, $\rho_C(c) := 1 + |\{c' \in C \mid f(c') < f(c)\}|$.*

Note that two elements with the same f-value are assigned the same rank.

As discussed above, when selecting a parent population for generating new search points, many randomized search heuristics do only use the ranking of the search points queried so far. In this work we are interested in how this fact influences the complexity of standard test function classes. Therefore, we regard here the restricted class of black-box algorithms that use no other information than this ranking. This yields the following black-box models.

The *unrestricted ranking-based black-box complexity* of some class of functions is the complexity with respect to all algorithms following the scheme of Algorithm 26.

Algorithm 26: Scheme of an unrestricted ranking-based black-box algorithm

```
1 Initialization: Sample $x^{(0)}$ according to some probability distribution $p^{(0)}$ over
  $\{0,1\}^n$ and query $f(x^{(0)})$;
2 Optimization: for $t = 1, 2, 3, \ldots$ do
3     Depending on the ranking of $\{x^{(0)}, \ldots, x^{(t-1)}\}$ induced by $f$, choose a
      probability distribution $p^{(t)}$ over $\{0,1\}^n$ and sample $x^{(t)}$ according to $p^{(t)}$;
4     Query the ranking of $\{x^{(0)}, \ldots, x^{(t)}\}$ induced by $f$;
```

Similarly, the *k-ary unbiased ranking-based black-box complexity* of some class of functions is the complexity with respect to all algorithms following the scheme of Algorithm 27.

Algorithm 27: Scheme of a k-ary unbiased ranking-based black-box algorithm

```
1 Initialization: Sample $x^{(0)} \in \{0,1\}^n$ uniformly at random and query $f(x^{(0)})$;
2 Optimization: for $t = 1, 2, 3, \ldots$ do
3     Depending on the ranking of $\{x^{(0)} \ldots, x^{(t-1)}\}$ induced by $f$, choose $k$
      indices $i_1, \ldots, i_k \in [0..t-1]$ and a $k$-ary unbiased distribution
      $(D(. \mid y^{(1)}, \ldots, y^{(k)}))_{y^{(1)},\ldots,y^{(k)} \in \{0,1\}^n}$;
4     Sample $x^{(t)}$ according to $D(. \mid x^{(i_1)}, \ldots, x^{(i_k)})$ and query the ranking of
      $\{x^{(0)}, \ldots, x^{(t)}\}$ induced by $f$;
```

Both ranking-based black-box models capture many common search heuristics such as evolutionary algorithms using elitist selection, ant colony optimization, and Randomized Local Search. They do not include algorithms like Simulated Annealing, Threshold Accepting, or evolutionary algorithms using fitness proportional selection.

To distinguish the unrestricted and the unbiased black-box model from their ranking-based counterparts, we shall sometimes refer to them as the *basic* unrestricted black-box model and the *basic* unbiased black-box model, respectively.

When working with the ranking-based models, the fact that the rank of a search point x varies with the number of already queried search points may be distracting. However, the ranking-based models can be equivalently modeled via a *monotone perturbation* of the fitness values. By this we mean that instead of ranking all previously queried search points, the oracle may as well reply to any query $x^{(t)}$ with some value $g(f(x^{(t)}))$, where f is the secret function to be optimized and $g : \mathbb{R} \to \mathbb{R}$ is a strictly monotone function that depends on all search points $x^{(1)}, \ldots, x^{(t)}$ queried so far.

To make this model more precise, let us first recall that a function is said to be *strictly monotone* if for all $\alpha < \beta$ we have $g(\alpha) < g(\beta)$. We show how the oracle can construct such a strictly monotone function "on the fly", preserving the ranking of the search points. Let A be a black-box algorithm. When algorithm A queries a search point $x^{(0)}$ for initialization, the oracle responds to A with "$(g \circ f)(x^{(0)}) = 0$". That is, it sets $g(f(x^{(0)})) := 0$. For any iteration t, if algorithm A queries $x^{(t)}$, the oracle returns to A the value

$$g(f(x^{(t)})) =$$

$$\begin{cases} g(f(x^{(i)})), & \text{if } \rho(x^{(t)}) = \rho(x^{(i)}) \text{ for some } i \in [0..t-1]\,, \\ \max\{g(f(x^{(i)})) \mid i \in [0..t-1]\} + 2^n, & \text{if } \rho(x^{(t)}) = 1\,, \\ \min\{g(f(x^{(i)})) \mid i \in [0..t-1]\} - 2^n, & \text{if } \rho(x^{(t)}) = t+1\,, \\ \left(g(f(x^{(h)})) - |g(f(x^{(i)}))|\right)/2, & \text{if } \rho(x^{(h)}) = \max\{\rho(x^{(\ell)}) \mid \rho(x^{(\ell)}) < \rho(x^{(t)})\} \\ & \text{and } \rho(x^{(i)}) = \min\{\rho(x^{(\ell)}) \mid \rho(x^{(\ell)}) > \rho(x^{(t)})\}\,, \end{cases}$$

where we abbreviate $\rho := \rho_{\{x^{(i)} | i \in [0..t]\}}$, the ranking of $\{x^{(i)} \mid i \in [0..t]\}$ induced by f. It is easily verified that indeed we have $g(f(x^{(i)})) > g(f(x^{(j)}))$ if and only if $f(x^{(i)}) > f(x^{(j)})$, i.e., g can be extended to a strictly monotone function $\mathbb{R} \to \mathbb{R}$.

The information revealed by the $(g \circ f)$-values and the information revealed by the ranking of the search points is the same. Therefore, the two models are equivalent.

We sometimes refer to the model with a $(g \circ f)$-oracle as the (unrestricted or unbiased, respectively) *monotone black-box model*. In particular for proving upper bounds this model is more convenient to work with.

Convention. In what follows, we shall always denote by g the monotone perturbation. That is, g is the function, which is used for representing the ranking of the already queried search points.

8.2. The Ranking-Based Black-Box Complexity of OneMax

In Section 4 we have shown that the n-ary unbiased black-box complexity of $\textsc{OneMax}_n$ is $O(n/\log n)$. Furthermore, we claimed without proof that for $2 \leq k < n$, the k-ary unbiased black-box complexity of $\textsc{OneMax}_n$ is $O(n/\log k)$. The following theorem shows that we can achieve the same bound in the (much weaker) unbiased ranking-based model. In addition, the unary unbiased black-box complexity of $\Theta(n \log n)$ carries over to the unary unbiased ranking-based black-box model as well.

Theorem 8.2. *The unary unbiased ranking-based black-box complexity of* $\textsc{OneMax}_n$ *is* $\Theta(n \log n)$*. For* $2 \leq k \leq n$*, the* k*-ary unbiased ranking-based black-box complexity of* $\textsc{OneMax}_n$ *is* $O(n/\log k)$*.*

Recall that this statement is asymptotically tight for $k = n^{\Omega(1)}$ by Theorem 4.1.

Corollary 8.3. *The* $*$*-ary unbiased ranking-based black-box complexity of* $\textsc{OneMax}_n$ *is* $\Theta(n/\log n)$*.*

In the spirit of Section 7 let us relate our statement to the Mastermind game. Assume that we played the black answer-peg only version of the Mastermind game with the additional rule that Carole, the codemaker, does not reply with the exact number of black answer-pegs but instead she ranks Paul's, i.e., the codebreaker's, guesses. Carole's ranking is the one induced by the black-answer pegs (which, like the $\textsc{Om}_z$ function, indicate in how many positions Carole's secret code and Paul's guesses coincide). Theorem 8.2 shows that—as in the non-ranking-based version of the Mastermind game—Paul still has an optimal winning strategy using $\Theta(n/\log n)$ guesses only. This generalizes previous results on the Mastermind game to a ranking-only version.

To ease reading, we split the proof of Theorem 8.2 into three parts. The first part, Section 8.2.1, is the easiest. It deals with constant values of k. We show that any class of *generalized strictly monotone functions* can be optimized by a binary unbiased ranking-based algorithm in $O(n)$ queries. $\textsc{OneMax}_n$ is such a class of generalized strictly monotone functions. For the unary setting, Theorem 8.2 follows from the two facts that (i) already the basic unbiased unary black-box complexity of $\textsc{OneMax}_n$ is $\Omega(n \log n)$ and (ii) that Randomized Local Search is a unary unbiased ranking-based algorithm which optimizes $\textsc{OneMax}_n$ in $O(n \log n)$ queries.

The second case is the most interesting one. We show that for $k = n$ there exists an unbiased ranking-based algorithm which optimizes every function $\textsc{Om}_z \in \textsc{OneMax}_n$ using only $O(n/\log n)$ queries. As in all previous results, the main strategy is again random sampling. As we have seen in Sections 4 and 7, after $O(n/\log n)$ samples chosen from $\{0,1\}^n$ independently and uniformly at random, with high probability, it is possible to uniquely identify the target string z. However, we need to be more careful here as, other than in all previous works, we do not have exact knowledge of the fitness values.

Lastly, we show how to deal with the case of arbitrary $k = \omega(1), k < n$. We prove that, for any such k, we can independently optimize substrings ("blocks") of size k in $O(k/\log k)$ iterations, using only k-ary unbiased variation operators. We optimize these blocks sequentially. Since there are $\Theta(n/k)$ such blocks of length k, the desired $O(n/\log k)$ bound follows. These are Sections 8.2.3 and 8.2.4.

8.2.1. Proof of Theorem 8.2 for Constant Values k

As mentioned above, for $k = 1$ the lower bound in Theorem 8.2 follows from [LW10a, Theorem 6], which states that the unary unbiased black-box complexity of any class of functions $\{0,1\}^n \to \mathbb{R}$ having a single global optimum is $\Omega(n \log n)$. Clearly, $\textsc{OneMax}_n$ is such a class. Since the k-ary unbiased black-box complexity of any class $\mathcal{F}$ of functions is a lower bound for the k-ary unbiased ranking-based black-box complexity of $\mathcal{F}$, this also shows that the unary unbiased ranking-based black-box complexity of $\textsc{OneMax}_n$ is $\Omega(n \log n)$.

The upper bound is certified by a simple hill-climber, Randomized Local Search (cf. Algorithm 1). It is easily verified that the variation operator implicit in the mutation step is a unary unbiased one, cf. Lemma 6.1. The selection depends only on the ranking of the current search point x and its neighbor y. Hence, Randomized Local Search is a unary unbiased ranking-based black-box algorithm. By the coupon collector's problem (cf. Section 2.4.5) the expected runtime of RLS on any $\textsc{OneMax}_n$ function is $O(n \log n)$. This concludes our comments on the unary unbiased ranking-based black-box complexity of $\textsc{OneMax}_n$.

For $k \geq 2$ we prove a more general statement, Lemma 8.4, which shows that all classes of generalized strictly monotone functions have a binary unbiased ranking-based black-box complexity that is at most linear in n. Since any $\textsc{Om}_z$ function is strictly monotone with respect to z, this shows that for any $k \geq 2$, the k-ary unbiased black-box complexity of $\textsc{OneMax}_n$ is $O(n)$.

Lemma 8.4. *Let $k \geq 2$ and let $\mathcal{F}$ be a class of generalized strictly monotone functions. The k-ary unbiased ranking-based black-box complexity of $\mathcal{F}$ is at most $4n - 5$.*

For proving Lemma 8.4 we show that two bit strings $x, y \in \{0,1\}^n$ suffice for encoding which bits have been optimized already. Exploiting this, only binary operators are needed to test for each bit individually whether it should be set to zero or to one.

Proof of Lemma 8.4. Note that it suffices to prove the statement for $k = 2$. We claim that Algorithm 28 certifies Lemma 8.4.

First we note that the selection/update step (lines 7,8,11) depends only on the rankings of the search points. Therefore, Algorithm 28 is a ranking-based one.

Apart from the uniform sampling variation operator, the algorithm certifying Lemma 8.4 (Algorithm 28) makes use of the following variation operators.

Algorithm 28: A binary unbiased ranking-based black-box algorithm for optimizing generalized strictly monotone functions $f \in \mathcal{F}$.

```
1  Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and query $g(f(x))$;
2  Set $y \leftarrow$ complement($x$) and query $g(f(y))$;
3  Optimization: for $t = 1, 2, 3, \ldots$ do
4      Sample $w \leftarrow$ flipOneWhereDifferent($x, y$) and query $g(f(w))$;
5      if $g(f(w)) > g(f(x))$ then
6          Set $w' \leftarrow$ dist$_1$($x, w$) and query $g(f(w'))$;
7          if $g(f(w')) = g(f(x))$ then Update $x \leftarrow w$ ; //$x$ and $w$ differ in 1 bit
8          else if $g(f(w)) > g(f(y))$ then Update $y \leftarrow w$ ; //$y$ and $w$ differ in 1 bit
9      else if $g(f(w)) > g(f(y))$ then
10         Set $w' \leftarrow$ dist$_1$($y, w$) and query $g(f(w'))$;
11         if $g(f(w')) = g(f(y))$ then Update $y \leftarrow w$ ; //$y$ and $w$ differ in 1 bit
```

- `complement`$(\cdot)$ is a unary variation operator which, given some $x \in \{0,1\}^n$, returns `complement`$(x) := \bar{x}$, the bitwise complement of x.

- `flipOneWhereDifferent`$(\cdot,\cdot)$ is a binary variation operator which, given some $x, y \in \{0,1\}^n$, first chooses uniformly at random a bit position $j \in \{i \in [n] \mid x_i \neq y_i\}$. With probability 1/2, it returns $x \oplus e_j^n$ and with probability 1/2 it returns $y \oplus e_j^n$. That is, with equal probability `flipOneWhereDifferent`(x, y) either flips exactly one bit in x, in which x and y differ, or it flips one such bit in y.

- `dist`$_1(\cdot,\cdot)$ is a binary operator which, given some $x, w \in \{0,1\}^n$ returns `dist`$_1(x, w) = x$ if the Hamming distance $|x \oplus w|_1$ of x and w equals 1 and it returns w otherwise.

It is easily verified that `complement`$(\cdot)$ is an unbiased unary variation operator, cf. Section 4.3. By a similar reasoning and obeying the fact that the position to be flipped is chosen uniformly, one can easily show that `flipOneWhereDifferent`$(\cdot,\cdot)$ is unbiased as well. Lastly, we also have $\sigma($`dist`$_1(x, w)) =$ `dist`$_1(\sigma(x), \sigma(w))$ and `dist`$_1(x \oplus y, w \oplus y) =$ `dist`$_1(x, w) \oplus y$ for all $x, y, w \in \{0,1\}^n$ and all $\sigma \in S_n$. This shows that `dist`$_1(\cdot,\cdot)$ is also unbiased.

For proving that Algorithm 28 indeed certifies Lemma 8.4, let us fix a function $f \in \mathcal{F}$ and let us assume that f is strictly monotone with respect to some fixed $z \in \{0,1\}^n$.

We show that throughout the run of Algorithm 28 the following invariant holds: for all $i \in [n]$ we have $x_i = y_i$ only if $x_i = z_i$. After initialization we have $x_i \neq y_i$ for all $i \in [n]$. Hence, by construction, the invariant is satisfied. Once we accept a bit flip of position $i \in [n]$, we necessarily have $x_i = y_i$. Hence, the bit will not be flipped in any further iteration of the algorithm. Furthermore, for any two bit strings $x, w \in \{0,1\}^n$ with Hamming distance $|x \oplus w|_1 = 1$ we have $f(w) > f(x)$ (and, by construction, $g(f(w)) > g(f(x))$) if and only if $w_i = z_i$ where $i \in [n]$ is the one position in which x and w differ. This shows that the invariant is always satisfied.

Next we show that Algorithm 28 terminates and that the expected runtime is at most $4n-5$. First we bound from above the number of queries needed to reduce the Hamming distance of x and y from n to two. To this end, let $x, y \in \{0,1\}^n$ with $|x-y|_1 > 2$. Then, for $w = \texttt{flipOneWhereDifferent}(x,y)$ either we have $|x \oplus w|_1 = 1$ or $|y \oplus w|_1 = 1$. Both events are equally likely and exactly one of them yields an update of x or y, respectively. Therefore, the probability of an update equals $1/2$. If we update any one of the two strings, the Hamming distance of x and y decreases by 1. This implies an expected number of $2(n-2)$ iterations for decreasing the Hamming distance of x and y from n to 2. Any such iteration requires at most two queries, the one for $g(f(w))$ and the one for $g(f(w'))$. Therefore, on average, we need at most $4(n-2)$ queries to reduce the Hamming distance from x and y from n to 2.

Once the Hamming distance of x and y is reduced to two, either we have $z \in \{x,y\}$ or we have that both $|x \oplus z|_1 = 1$ and $|y \oplus z|_1 = 1$. By the random initialization of x and the fact that we replace x and y only by bit strings of Hamming distance one, all bits x_i for which $x_i \neq y_i$ satisfy $\Pr[x_i = 1] = \Pr[x_i = 0] = 1/2$ and, likewise, $\Pr[y_i = 1] = \Pr[y_i = 0] = 1/2$. Therefore, the event $z \in \{x,y\}$ occurs with probability $1/2$. In this case we are done, because both $g(f(x))$ as well as $g(f(y))$ have been queried already. Therefore, let us assume that $z \notin \{x,y\}$. Let again $w = \texttt{flipOneWhereDifferent}(x,y)$. Then both $|x \oplus w|_1 = 1$ and $|y \oplus w|_1 = 1$ must hold. Since only two such strings with Hamming distance one from both x and y exist, we have $\Pr[w = z] = 1/2$. Furthermore, $g(f(w)) > g(f(x))$ or $g(f(w)) > g(f(y))$ holds only if $w = z$. That is, if $w \neq z$, then neither x nor y will be updated. Therefore, once $|x \oplus y|_1 = 2$, it takes, on average, $1/2 \cdot 0 + 1/2 \cdot 2 = 1$ query until Algorithm 28 queries $z = \arg\max f$ for the first time.

Together with the two queries needed for initialization (lines 1 and 2 of Algorithm 28), the runtime of our algorithm can be bounded from above by $2+4(n-2)+1 = 4n-5$. □

8.2.2. Proof of Theorem 8.2 for $k = n$

We need to show that there exists a ranking-based black-box algorithm which employs only unbiased variation operators of arity at most n and which optimizes any function $\text{OM}_z \in \text{ONEMAX}_n$ in $O(n/\log n)$ iterations.

To ease reading of the following description, let us already fix here some (unknown) function $\text{OM}_z \in \text{ONEMAX}_n$. In order to optimize OM_z, the algorithm has to query the target string $z \in \{0,1\}^n$.

We work with the monotone model, i.e., whenever the algorithm queries from the oracle a search point x, it receives from it the value $g\big(\text{OM}_z(x)\big)$.

The rough description of the algorithm certifying Theorem 8.2 for $k = n$ is fairly easy. It first samples $s \in O(n/\log n)$ search points $x_1, \ldots, x_s$ from $\{0,1\}^n$ mutually independent and uniformly at random. We show that, with high probability, knowing only the $(g \circ \text{OM}_z)$-values $\{g\big(\text{OM}_z(x_i)\big) \mid i \in [s]\}$ suffices to create the target string z using only two additional (unbiased) iterations. These last two queries, however, require some technical effort. This is the main part of this section.

In what follows, let $\kappa \geq 2$ be a constant, let $\beta := e^{-4\kappa^2}(2\sqrt{\pi})^{-1}$, and let α be a constant that is at least $8\left(1-2e^{-2\kappa^2}\right)^{-1}$. Furthermore, let $s := \alpha n/\log n$ and let

$x_1, \ldots, x_s$ be sampled from $\{0,1\}^n$ independently and uniformly at random.

We divide the proof of Theorem 8.2 for the case $k = n$ into three steps. We feel that the intermediate steps are interesting on their own. Each of the following statements holds with exponentially small probability of failure. In particular, they hold with probability at least $1 - o(n^{-\lambda})$ for all constant values of λ.

- First we show that for each $\ell \in [\frac{n}{2} \pm \kappa\sqrt{n}]$ there is at least one bit string x_i such that $\textsc{Om}_z(x_i) = \ell$. Furthermore, the set of all samples with $\textsc{Om}_z$-value in $[\frac{n}{2} \pm \kappa\sqrt{n}]$ has size at least $\frac{s}{2}(1 - 2e^{-2\kappa^2})$.
- In the second part we show how to identify $g(\frac{n}{2})$ and how this knowledge suffices to identify the interval $g([\frac{n}{2} \pm \kappa\sqrt{n}])$. This allows us to calculate $\textsc{Om}_z(x)$ for all x with $\textsc{Om}_z(x) \in [\frac{n}{2} \pm \kappa\sqrt{n}]$. That is, for such x we are able to undo the monotone perturbation caused by g.
- Lastly, we show that there does not exist any $y \in \{0,1\}^n \backslash \{z\}$ with $\textsc{Om}_y(x) = \textsc{Om}_z(x)$ for all such samples $x \in \{x_1, \ldots, x_s\}$ with $\textsc{Om}_z(x) \in [\frac{n}{2} \pm \kappa\sqrt{n}]$. Thus, we can unambiguously determine z.

Part 1: Flooding the interval $[\frac{n}{2} \pm \kappa\sqrt{n}]$

In the next two lemmata we show that, with probability at least $1 - 3\kappa\sqrt{n}\exp(-\frac{\alpha\beta\sqrt{n}}{8\log n})$, for all $\ell \in [\frac{n}{2} \pm \kappa\sqrt{n}]$ there exists at least one $i \in [s]$ such that $\textsc{Om}_z(x_i) = \ell$.

Lemma 8.5. *Let $\ell \in [\frac{n}{2} \pm \kappa\sqrt{n}]$ and let x be sampled from $\{0,1\}^n$ uniformly at random. For large enough n we have that $\Pr[\textsc{Om}_z(x) = \ell] \geq \beta n^{-1/2}$.*

Proof. Clearly, $\Pr[\textsc{Om}_z(x) = \ell] = \binom{n}{\ell}2^{-n}$. Thus, we have to prove that, for large enough n, $\binom{n}{\ell} \geq \beta 2^n n^{-1/2}$. Let $\gamma \in [-\kappa, +\kappa]$ such that $\ell = \frac{n}{2} + \gamma\sqrt{n}$.

By definition we have

$$\tbinom{n}{\ell} = \tfrac{n!}{\ell!(n-\ell)!} = \tfrac{n!}{(n/2+\gamma\sqrt{n})!(n/2-\gamma\sqrt{n})!}\,.$$

By Lemma 2.10 we can bound

$$n! \geq \sqrt{2\pi} n^{n+1/2} e^{-n} e^{1/(12n+1)} \geq \sqrt{2\pi} n^{n+1/2} e^{-n}\,,$$

$$\left(\frac{n}{2} + \gamma\sqrt{n}\right)! \leq \sqrt{2\pi}\left(\frac{n}{2} + \gamma\sqrt{n}\right)^{\frac{n+1}{2}+\gamma\sqrt{n}} e^{-(\frac{n}{2}+\gamma\sqrt{n})} e^{1/(12(\frac{n}{2}+\gamma\sqrt{n}))}\,, \text{ and}$$

$$\left(\frac{n}{2} - \gamma\sqrt{n}\right)! \leq \sqrt{2\pi}\left(\frac{n}{2} - \gamma\sqrt{n}\right)^{\frac{n+1}{2}-\gamma\sqrt{n}} e^{-(\frac{n}{2}-\gamma\sqrt{n})} e^{1/(12(\frac{n}{2}-\gamma\sqrt{n}))}\,.$$

We rewrite

$$\left(\frac{n}{2} + \gamma\sqrt{n}\right)^{\frac{n+1}{2}+\gamma\sqrt{n}} = \left(\frac{n}{2}\right)^{\frac{n+1}{2}+\gamma\sqrt{n}} \underbrace{\left(1 + \frac{2\gamma}{\sqrt{n}}\right)^{\gamma\sqrt{n}}}_{(*)} \left(1 + \frac{2\gamma}{\sqrt{n}}\right)^{\frac{n+1}{2}}\,,$$

where term $(*)$ equals $\left((1+\frac{2\gamma}{\sqrt{n}})^{\frac{\sqrt{n}}{2\gamma}}\right)^{2\gamma^2}$. This term converges to $e^{2\gamma^2}$. For all n, term $(*)$ can be bounded from above by $e^{2\gamma^2}$.

Similarly we rewrite

$$\left(\frac{n}{2}-\gamma\sqrt{n}\right)^{\frac{n+1}{2}-\gamma\sqrt{n}} = \left(\frac{n}{2}\right)^{\frac{n+1}{2}-\gamma\sqrt{n}} \underbrace{\left(1-\frac{2\gamma}{\sqrt{n}}\right)^{-\gamma\sqrt{n}}}_{(**)} \left(1-\frac{2\gamma}{\sqrt{n}}\right)^{\frac{n+1}{2}},$$

where term $(**)$ equals $\left(\left(1-\frac{2\gamma}{\sqrt{n}}\right)^{\frac{\sqrt{n}}{2\gamma}}\right)^{-2\gamma^2}$. This term also converges to $e^{2\gamma^2}$. By Lemma 2.8, for any n, expression $(**)$ can be bounded from below by $e^{2\gamma^2}$. However, by convergence, there exists a $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ we have $\left(1-\frac{2\gamma}{\sqrt{n}}\right)^{-\gamma\sqrt{n}} \leq 2e^{2\gamma^2}$.

Finally, let us note that,

$$\left(1+\frac{2\gamma}{\sqrt{n}}\right)^{\frac{n+1}{2}} \left(1-\frac{2\gamma}{\sqrt{n}}\right)^{\frac{n+1}{2}} = \left(1-\frac{4\gamma^2}{n}\right)^{\frac{n+1}{2}} < 1$$

and, for large enough n,

$$e^{1/(12(\frac{n}{2}+\gamma\sqrt{n}))} e^{1/(12(\frac{n}{2}-\gamma\sqrt{n}))} = e^{1/(3n-12\gamma^2)} \leq \sqrt{2}\,.$$

Altogether we obtain for large enough n that

$$\binom{n}{\ell} \geq \frac{2^{n+1}}{\sqrt{2\pi}\sqrt{n}2e^{4\gamma^2}\sqrt{2}} \geq \frac{2^n}{2\sqrt{\pi n}e^{4\kappa^2}} = \beta\frac{2^n}{\sqrt{n}}\,.$$

□

By applying a Chernoff bound (Lemma 2.12) to the result of Lemma 8.5, we immediately obtain the following.

Corollary 8.6. $\Pr\left[\forall \ell \in [\frac{n}{2} \pm \kappa\sqrt{n}]\, \exists i \in [s] : \textsc{Om}_z(x_i) = \ell\right] \geq 1 - 3\kappa\sqrt{n}\exp(-\frac{\alpha\beta\sqrt{n}}{8\log n})\,.$

Proof. Throughout this proof assume that n is sufficiently large.

Let $\ell \in [\frac{n}{2} \pm \kappa\sqrt{n}]$. For all $i \in [s]$ let X_i^ℓ be the indicator variable of the event $\textsc{Om}_z(x_i) = \ell$. Let $X^\ell := \sum_{i=1}^s X_i^\ell$. By Lemma 8.5 we have $\mathrm{E}[X^\ell] \geq s\beta n^{-1/2} = \alpha\beta n^{1/2}\log^{-1} n$. By applying a Chernoff bound (cf. Lemma 2.12, (2.2)) we derive

$$\begin{aligned}\Pr[X^\ell < \tfrac{\alpha\beta}{2} n^{1/2}\log^{-1} n] &\leq \Pr[X^\ell < \tfrac{1}{2}\mathrm{E}[X^\ell]] \leq \exp(-\tfrac{1}{8}\mathrm{E}[X^\ell]) \\ &\leq \exp(-\tfrac{\alpha\beta}{8} n^{1/2}\log^{-1} n)\,.\end{aligned}$$

The statement follows from a simple union bound argument. The probability of not sampling at least one of the values in $[\frac{n}{2} \pm \kappa\sqrt{n}]$ can be bounded from above by $(2\kappa\sqrt{n}+1)\exp(-\frac{\alpha\beta}{8}n^{1/2}\log^{-1} n)$. □

We conclude the first part by the following elementary lemma, which again is not best possible, but good enough for our purposes. It shows that almost one half of the s samples $x_1, \ldots, x_s$ lie in the interval $[\frac{n}{2} \pm \kappa\sqrt{n}]$.

Lemma 8.7. ***(i)*** *If x is drawn from $\{0,1\}^n$ uniformly at random, then $\Pr[\mathrm{OM}_z(x) \in [\frac{n}{2} \pm \kappa\sqrt{n}]] \geq 1 - 2e^{-2\kappa^2}$.*

(ii) *Let S be the number $|\{i \in [s] \mid \mathrm{OM}_z(x_i) \in [\frac{n}{2} \pm \kappa\sqrt{n}]\}|$ of samples with OM_z-value in $[\frac{n}{2} \pm \kappa\sqrt{n}]$. With probability at least $1 - \exp(-\frac{(1-2e^{-2\kappa^2})\alpha n}{8 \log n})$ we have $S \geq \frac{s}{2}(1 - 2e^{-2\kappa^2})$.*

Proof. Let x be drawn from $\{0,1\}^n$ uniformly at random. Then, by Chernoff's bound (cf. (2.1) in Lemma 2.12),

$$\Pr[\mathrm{OM}_z(x) \in [\tfrac{n}{2} \pm \kappa\sqrt{n}]] = 1 - \Pr[|\mathrm{OM}_z(x) - \mathrm{E}[\mathrm{OM}_z(x)]| > \kappa\sqrt{n}] \geq 1 - 2e^{-2\kappa^2} .$$

This shows **(i)**. Furthermore, we expect at least $(1-2e^{-2\kappa^2})s$ samples to have an OM_z-value in $[\frac{n}{2} \pm \kappa\sqrt{n}]$. By again applying a Chernoff bound (cf. (2.1) in Lemma 2.12), we bound

$$\Pr[S \leq \tfrac{1}{2}(1 - 2e^{-2\kappa^2})s] \leq \Pr[S \leq \tfrac{1}{2}\,\mathrm{E}[S]] \leq \exp(-\tfrac{1}{8}\,\mathrm{E}[S]) = \exp(-\tfrac{1}{8}(1 - 2e^{-2\kappa^2})s) .$$

□

Part 2: Identification of $g(\frac{n}{2})$ and of $g\left([\frac{n}{2} \pm \kappa\sqrt{n}]\right)$

From the previous part we know that after drawing $s = \alpha n / \log n$ samples independently and uniformly at random, we can assume that for each value $\ell \in [\frac{n}{2} \pm \kappa\sqrt{n}]$ there exists at least one $i \in [s]$ such that $\mathrm{OM}_z(x_i) = \ell$. Furthermore, we have bounded the number of samples that fall into the interval $[\frac{n}{2} \pm \kappa\sqrt{n}]$. As we shall see in the third part of this section, if we could identify these samples with OM_z-value in $[\frac{n}{2} \pm \kappa\sqrt{n}]$, then, with high probability, we could determine the target string z. In this part we show that on top of the s samples $x_1, \ldots, x_s$ we need only one additional query to determine $g(\frac{n}{2})$. Once we have identified the value $g(\frac{n}{2})$, from Part 1 we infer that we also learned $g(\ell)$ for all $\ell \in [\frac{n}{2} \pm \kappa\sqrt{n}]$.

We first explain how to identify $g(\frac{n}{2})$. We do this by exploiting the strong monotonicity of g. To be more precise, we make use of the fact that g preserves the median of a set of objective values. Here in our context we define the median of a finite multiset S to be the smallest value $m \in S$ such that the number of elements in S which are smaller or equal to m is at least half the size of S. Formally, $m = \min\{m' \in S \mid |\{s \in S \mid s \leq m'\} \geq |S|/2\}$. For our context we set

$$m' := \min\{\ell \in [0..n] \mid |\{i \in [s] \mid \mathrm{OM}_z(x_i) \leq \ell\}| \geq n/2\} .$$

For all statements that follow, we assume that n is large enough.

Lemma 8.8. *The probability that $m' \in [\frac{n}{2} \pm \sqrt{n}]$ is at least $1 - 2\exp(-2\alpha\beta^2 n / \log n)$.*

Proof. We bound the probability that more than $s/2$ samples have an $\textsc{Om}_z$-value that is less than $\frac{n}{2} - \sqrt{n}$ and we bound the probability that more $s/2$ samples have an $\textsc{Om}_z$-value that is larger than $\frac{n}{2} + \sqrt{n}$.

Let $x \in \{0,1\}^n$ be sampled uniformly at random. By symmetry, for all γ it holds that

$$\Pr[\textsc{Om}_z(x) = \tfrac{n}{2} + \gamma] = \Pr[\textsc{Om}_z(x) = \tfrac{n}{2} - \gamma]\,. \tag{8.1}$$

Furthermore, by Lemma 8.5 we have

$$\begin{aligned}\Pr\left[\textsc{Om}_z(x) \notin [\tfrac{n}{2} \pm \sqrt{n}]\right] &= 1 - \sum_{i=\frac{n}{2}-\sqrt{n}}^{\frac{n}{2}+\sqrt{n}} \Pr[\textsc{Om}_z(x) = i] \leq 1 - (2\sqrt{n}+1)\beta n^{-1/2} \\ &\leq 1 - 2\beta\,.\end{aligned} \tag{8.2}$$

Equations (8.1) and (8.2) imply $\Pr[\textsc{Om}_z(x) < \frac{n}{2} - \sqrt{n}] \leq \frac{1}{2} - \beta$. By the linearity of expectation we can thus bound the expected number X of samples with $\textsc{Om}_z$-value less than $\frac{n}{2} - \sqrt{n}$ by $s(\frac{1}{2} - \beta)$.

From the Chernoff bound (2.1) in Lemma 2.12 we derive

$$\Pr\left[X \geq \tfrac{s}{2}\right] \leq \Pr\left[X \geq \mathrm{E}[X] + s\beta\right] \leq \exp\left(-2s^2\beta^2/s\right) = \exp(-2\alpha\beta^2 n/\log n)$$

By symmetry, the same reasoning proves $\Pr\left[Y \geq \frac{s}{2}\right] \leq \exp(-2\alpha\beta^2 n/\log n)$ for the number Y of samples with $\textsc{Om}_z$-value larger than $\frac{n}{2} + \sqrt{n}$. The statement follows from a union bound over the two events $X \geq \frac{s}{2}$ and $Y \geq \frac{s}{2}$. □

With Lemma 8.8 at hand, the identification of $g(\frac{n}{2})$ and $g\left([\frac{n}{2} \pm \kappa\sqrt{n}]\right)$ is easy.

Lemma 8.9. *If we know the $(g \circ \textsc{Om}_z)$-values of $x_1, \ldots, x_s$, we can apply a unary unbiased variation operator to one of these samples in order to create one additional search point x' such that after querying $g(\textsc{Om}_z(x'))$ we can identify $g(\lceil \frac{n}{2} \rceil)$ and $g\left([\lceil \frac{n}{2} \rceil \pm \kappa\sqrt{n}]\right)$, with probability at least $1 - c\sqrt{n}\exp(-\frac{\alpha\beta\sqrt{n}}{8\log n})$ for some constant value c.*

Proof. Let m be the median of the multiset $\{g(\textsc{Om}_z(x_1)), \ldots, g(\textsc{Om}_z(x_s))\}$. Since g is a strictly monotone function, we have $m = g(m')$. Lemma 8.8 yields $m \in g\left([\lceil \frac{n}{2} \rceil \pm \sqrt{n}]\right)$, with probability at least $1 - 2\exp(-2\alpha\beta^2 n/\log n)$.

According to Lemma 8.5, there exists a sample $x_i \in \{x_1, \ldots, x_s\}$ such that $g(\textsc{Om}_z(x_i)) = m$, again with probability at least $1 - 3\kappa\sqrt{n}\exp(-\frac{\alpha\beta\sqrt{n}}{8\log n})$. We show how sampling the bitwise complement $\overline{x_i}$ of x_i reveals $g(\frac{n}{2})$.

Before we prove this claim let us first note that $\overline{x_i}$ can be obtained from x_i by the unary unbiased variation operator `complement`$(\cdot)$ that we introduced in the proof of Lemma 8.4.

For even values of n, our algorithm to identify $g(\frac{n}{2})$ is Algorithm 29. Here we denote by $\text{median}(g(\textsc{Om}_z(y)), g(\textsc{Om}_z(x_i)) | g(\textsc{Om}_z(x_1)), \ldots, g(\textsc{Om}_z(x_s)))$ the median of the sampled $(g \circ \textsc{Om}_z)$-values in $[g(\textsc{Om}_z(x_i)), g(\textsc{Om}_z(y))]$ (if $g(\textsc{Om}_z(x_i)) < g(\textsc{Om}_z(y))$) or in $[g(\textsc{Om}_z(y)), g(\textsc{Om}_z(x_i))]$ (otherwise), respectively, where each sampled value is counted with multiplicity one only.

Algorithm 29: Identifying $g(\frac{n}{2})$ for even values of n.

```
1 Sample $i \in \{j \in [s] \mid g(\textsc{Om}_z(x_j)) = m\}$ uniformly at random;
2 Set $y \leftarrow \texttt{complement}(x)$ and query $g(\textsc{Om}_z(y))$;
3 if $g(\textsc{Om}_z(y)) = g(\textsc{Om}_z(x_i))$ then $m_g \leftarrow m$;
4 else $m_g \leftarrow \text{median}(g(\textsc{Om}_z(y)), g(\textsc{Om}_z(x_i)) | g(\textsc{Om}_z(x_1)), \ldots, g(\textsc{Om}_z(x_s)))$;
5 Output $m_g$
```

To show the correctness of Algorithm 29, let us first assume that $g(\textsc{Om}_z(y)) = g(\textsc{Om}_z(x_i))$ holds. Since g is a strictly monotone function, this implies $\textsc{Om}_z(y) = \textsc{Om}_z(x_i)$. But then $m' = \textsc{Om}_z(x_i) = \frac{n}{2}$ must hold by the symmetry property that we mentioned already in equation (8.1) in the proof of Lemma 8.8.

Therefore, we may assume without loss of generality that $g(\textsc{Om}_z(x_i)) \neq g(\textsc{Om}_z(y))$. As mentioned above, by Lemma 8.8, with probability at least $1 - 2\exp(-2\alpha\beta^2 n/\log n)$ we have $\textsc{Om}_z(x_i) = g^{-1}(m) \in [\frac{n}{2} \pm \sqrt{n}]$. So is $g(\textsc{Om}_z(y))$ by the symmetry of the $\textsc{Om}_z$ function (equation (8.1)). The symmetry also implies $\frac{n}{2} = (\textsc{Om}_z(x_i) + \textsc{Om}_z(y))/2$.

Assume $\textsc{Om}_z(x_i) < \textsc{Om}_z(y)$. Then $\frac{n}{2}$ is exactly the median of the integer values in $[\textsc{Om}_z(x_i), \textsc{Om}_z(y)]$. But since we have—by Corollary 8.6—with probability at least $1 - 3\kappa\sqrt{n}\exp(-\frac{\alpha\beta\sqrt{n}}{8\log n})$

$$[\textsc{Om}_z(x_i), \textsc{Om}_z(y)] \subseteq \{\textsc{Om}_z(x_1), \ldots, \textsc{Om}_z(x_s)\},$$

the median of the integer values in $[\textsc{Om}_z(x_i), \textsc{Om}_z(y)]$ equals the median of the integer values $\{\textsc{Om}_z(x_1), \ldots, \textsc{Om}_z(x_s)\} \cap [\textsc{Om}_z(x_i), \textsc{Om}_z(y)]$.

Since g is a strictly monotone function we also have, with the same probability,

$$g([\textsc{Om}_z(x_i), \textsc{Om}_z(y)]) \subseteq \{g(\textsc{Om}_z(x_1)), \ldots (\textsc{Om}_z(x_s))\}.$$

Therefore, the median of the sampled values

$$\{g(\textsc{Om}_z(x_1)), \ldots, g(\textsc{Om}_z(x_s))\} \cap [g(\textsc{Om}_z(x_i)), g(\textsc{Om}_z(y))]$$

equals $g(\frac{n}{2})$, if we count each sampled value with multiplicity one.

By Corollary 8.6, once we have identified $g(\frac{n}{2})$ we also know $g\left([\frac{n}{2} \pm \kappa\sqrt{n}]\right)$, with high probability.

For odd values of n a similar reasoning shows that Algorithm 30 computes $g(\lceil\frac{n}{2}\rceil)$: Either we have $\textsc{Om}_z(x_i) \in \{\frac{n}{2} - 1, \frac{n}{2} + 1\}$ (lines 3–8) in which case the two values $g(\textsc{Om}_z(y))$ and $g(\textsc{Om}_z(x_i))$ must be two consecutive values in $\{g(\textsc{Om}_z(x_j)) \mid j \in [s]\}$[1] or we identify as above $g(\lceil\frac{n}{2}\rceil)$ as the median integer value of $[\textsc{Om}_z(x_i), \textsc{Om}_z(y)]$ (if $\textsc{Om}_z(x_i) < \textsc{Om}_z(y)$) or $[\textsc{Om}_z(y), \textsc{Om}_z(x_i)]$ (if $\textsc{Om}_z(y) < \textsc{Om}_z(x_i)$), respectively. □

[1] That is, $\{g(\textsc{Om}_z(x_1)), \ldots, g(\textsc{Om}_z(x_s))\} \cap [\textsc{Om}_z(x_i), \textsc{Om}_z(y)] = \{\textsc{Om}_z(x_i), \textsc{Om}_z(y)\}$ (if $\textsc{Om}_z(x_i) < \textsc{Om}_z(y)$) or $\{g(\textsc{Om}_z(x_1)), \ldots, g(\textsc{Om}_z(x_s))\} \cap [\textsc{Om}_z(y), \textsc{Om}_z(x_i)] = \{\textsc{Om}_z(y), \textsc{Om}_z(x_i)\}$ (if $\textsc{Om}_z(y) < \textsc{Om}_z(x_i)$), respectively.

Algorithm 30: Identifying $g(\lceil \frac{n}{2} \rceil)$ for odd values of n.

```
1  Sample $i \in \{j \in [s] \mid g(\textsc{Om}_z(x_j)) = m\}$ uniformly at random;
2  Set $y \leftarrow$ complement($x$) and query $g(\textsc{Om}_z(y))$;
3  if $g(\textsc{Om}_z(y)) < g(\textsc{Om}_z(x_i))$ then
4      if $\forall j \in [s] : (g(\textsc{Om}_z(x_j)) \leq g(\textsc{Om}_z(y))) \vee (g(\textsc{Om}_z(x_j)) \geq g(\textsc{Om}_z(x_i)))$ then
5          $m_g \leftarrow m$;
6  else if $g(\textsc{Om}_z(y)) > g(\textsc{Om}_z(x_i))$ then
7      if $\forall j \in [s] : (g(\textsc{Om}_z(x_j)) \geq g(\textsc{Om}_z(y))) \vee (g(\textsc{Om}_z(x_j)) \leq g(\textsc{Om}_z(x_i)))$ then
8          $m_g \leftarrow g(\textsc{Om}_z(y))$;
9  else $m_g \leftarrow \text{median}(g(\textsc{Om}_z(y)), g(\textsc{Om}_z(x_i)) | g(\textsc{Om}_z(x_1)), \ldots, g(\textsc{Om}_z(x_s)))$;
10 Output $m_g$
```

Part 3: Calculation of z

In this section, we prove that the s random samples and the one additional sample needed to identify $g(\lceil \frac{n}{2} \rceil)$ suffice to determine the target string z. We do so by showing that the probability that there exists a bit string $y \neq z$ with $\textsc{Om}_y(x_i) = \textsc{Om}_z(x_i)$ for all $i \in [s]$ with $\textsc{Om}_z(x_i) \in [\frac{n}{2} \pm \kappa\sqrt{n}]$ is very small.

Lemma 8.10. *Let $S := \{i \in [s] \mid \textsc{Om}_z(x_i) \in [\frac{n}{2} \pm \kappa\sqrt{n}]\}$, the set of all samples with $\textsc{Om}_z$-value close to $\frac{n}{2}$. Let $F := \{y \mid \forall i \in S : \textsc{Om}_z(x_i) = \textsc{Om}_y(x_i)\}$, the set of all y that are consistent with the $\textsc{Om}_z$-values for all x_i with $i \in S$.*

Then, with probability at least $1 - \exp(-\frac{(1-2e^{-2\kappa^2})\alpha n}{8 \log n})$, we have $\mathrm{E}[|F|] \leq 1 + 2^{-t/4}$ for $t := \frac{s}{2}(1 - 2e^{-2\kappa^2})$.

In particular we have that, with probability at least $1 - c\exp(-\frac{(1-2e^{-2\kappa^2})\alpha n}{8 \log n})$ for some constant c, there does not exist a string $y \in \{0,1\}^n \backslash \{z\}$ such that $\textsc{Om}_z(x_i) = \textsc{Om}_y(x_i)$ for all $i \in S$.

Proof. First note that, by Lemma 8.7, we can assume that $|S| \geq \frac{s}{2}(1 - 2e^{-2\kappa^2}) = t$, with probability at least $1 - \exp(-\frac{(1-2e^{-2\kappa^2})\alpha n}{8 \log n})$.

Let $y \in \{0,1\}^n \backslash \{z\}$ and let $h := |y \oplus z|_1$, the Hamming distance of y and z. We bound the probability that for all $i \in S$ we have $\textsc{Om}_y(x_i) = \textsc{Om}_z(x_i)$.

If we consider one particular sample x chosen from $\{0,1\}^n$ uniformly at random, we have

$$\Pr\left[\textsc{Om}_y(x) = \textsc{Om}_z(x) \mid \textsc{Om}_z(x) \in [\tfrac{n}{2} \pm \kappa\sqrt{n}]\right] \leq \frac{\Pr\left[\textsc{Om}_y(x) = \textsc{Om}_z(x)\right]}{\Pr\left[\textsc{Om}_z(x) \in [\frac{n}{2} \pm \kappa\sqrt{n}]\right]}.$$

By Lemma 8.7 the probability $\Pr\left[\textsc{Om}_z(x) \in [\frac{n}{2} \pm \kappa\sqrt{n}]\right]$ that the $\textsc{Om}_z$-value of x lies in the interval $[\frac{n}{2} \pm \kappa\sqrt{n}]$ can be bounded from below by $1 - 2e^{-2\kappa^2}$.

Furthermore, $\textsc{Om}_y(x) = \textsc{Om}_z(x)$ holds if and only if x coincides with z in exactly half of the h bits in which z and y differ. Thus, $\Pr[\textsc{Om}_y(x) = \textsc{Om}_z(x)] = \binom{h}{h/2}2^{-h}$ if

h is even and $\Pr[\textsc{Om}_y(x) = \textsc{Om}_z(x)] = 0$ for odd values of h. In particular,

$$\Pr\left[\textsc{Om}_y(x) = \textsc{Om}_z(x) \mid x \in S\right] \leq \frac{\binom{h}{h/2}2^{-h}}{1-2e^{-2\kappa^2}}\,,$$

for all even values h and $\Pr\left[\textsc{Om}_y(x) = \textsc{Om}_z(x) \mid x \in S\right] = 0$ for odd values h.

Assume h to be even. As the samples $x_1, \ldots, x_s$ are drawn independently, the probability that $\textsc{Om}_y(x_i) = \textsc{Om}_z(x_i)$ for all $i \in S$ can be bounded as follows.

$$\Pr\Big[\bigwedge_{i \in S}\left(\textsc{Om}_y(x_i) = \textsc{Om}_z(x_i)\right)\Big] = \prod_{i \in S} \Pr\left[\textsc{Om}_y(x_i) = \textsc{Om}_z(x_i)\right] \leq \left(\frac{\binom{h}{h/2}2^{-h}}{1-2e^{-2\kappa^2}}\right)^t.$$

As there are $\binom{n}{h}$ different bit strings y with Hamming distance $|y \oplus z|_1 = h$ from z, we bound the expected number of bit strings $y \neq z$ with $\textsc{Om}_y(x_i) = \textsc{Om}_z(x_i)$ for all $i \in S$ from above by

$$\sum_{h \in [n]; h\, even} \binom{n}{h}\left(\frac{\binom{h}{h/2}2^{-h}}{1-2e^{-2\kappa^2}}\right)^t = \left(1-2e^{-2\kappa^2}\right)^{-t} \sum_{h \in [n]; h\, even} \binom{n}{h}\left(\binom{h}{h/2}2^{-h}\right)^t.$$

We have proven in Proposition 4.6 that for sufficiently large n and $\tilde{t} \geq 2\big(1+\frac{4\log_2\log_2 n}{\log_2 n}\big)\frac{n}{\log_2 n}$, it holds that

$$\sum_{h \in [n]; h\, even} \binom{n}{h}\left(\binom{h}{h/2}2^{-h}\right)^{\tilde{t}} \leq (n/2)\cdot 2^{-3\tilde{t}/4}\,.$$

In our case the condition $\alpha \geq 8\big(1-2e^{-2\kappa^2}\big)^{-1} > 4\big(1-2e^{-2\kappa^2}\big)^{-1}\big(1+\frac{4\log_2\log_2 n}{\log_2 n}\big)$ ensures that t satisfies this condition. Furthermore, we have for large enough n that $\frac{n}{2} \leq 2^{t/4}$ and lastly, the requirement $\kappa \geq 2$ implies that $2e^{-2\kappa^2} << 0.15$. Hence, $\big(1-2e^{-2\kappa^2}\big)^{-1} \leq 0.85^{-1} \leq 1.18 \leq 2^{1/4}$ and finally, $\big(1-2e^{-2\kappa^2}\big)^{-t} \leq 2^{t/4}$. We thus conclude that, with probability at least $1-\exp(-\frac{(1-2e^{-2\kappa^2})\alpha n}{8\log n})$,

$$\mathrm{E}[|\{y \neq z \mid \forall x \in S : \textsc{Om}_y(x) = \textsc{Om}_z(x)\}|] \leq 2^{-t/4}\,.$$

□

We are now ready to prove Theorem 8.2 for the special case of arity $k = n$. To this end, we fix the values of $\kappa := 2$ and $\alpha := 9 = \left\lceil 8\big(1-2e^{-2\kappa^2}\big)^{-1}\right\rceil$.

Proof of Theorem 8.2 for $k = n$. We need to show that there exists a ranking-based unbiased algorithm which optimizes any function $\textsc{Om}_z \in \textsc{OneMax}_n$ in an expected number of $O(n/\log n)$ queries.

We claim that Algorithm 31 certifies Theorem 8.2 for $k = n$ and even values n. For odd values, the part in which we identify $g(\lceil \frac{n}{2} \rceil)$ (lines 5–8 of Algorithm 31) needs to be replaced by lines 1–9 of Algorithm 30. We show that the probability that Algorithm 31 queries z after $O(n/\log n)$ iterations is $1-o(n^{-\lambda})$ for all constant values

Algorithm 31: An n-ary unbiased ranking-based black-box algorithm optimizing $\textsc{OneMax}_n$ in $O(n/\log n)$ queries.

```
1  Initialization:
2    for $i = 1, \ldots, s$ do
3      Sample $x_i \in \{0,1\}^n$ uniformly at random and query $g(\textsc{Om}_z(x_i))$;
4  Identification of $g(\frac{n}{2})$:
5    Sample $i \in \{j \in [s] \mid g(\textsc{Om}_z(x_j)) = m\}$ uniformly at random;
6    Set $y \leftarrow$ complement$(x_i)$ and query $g(\textsc{Om}_z(y))$;
7    if $g(\textsc{Om}_z(y)) = g(\textsc{Om}_z(x_i))$ then $m_g \leftarrow m$;
8    else $m_g \leftarrow \text{median}(g(\textsc{Om}_z(y)), g(\textsc{Om}_z(x_i)) | g(\textsc{Om}_z(x_1)), \ldots, g(\textsc{Om}_z(x_s)))$;
9  Compute $S \leftarrow \{i \in [s] \mid \textsc{Om}_z(x_i) \in [\frac{n}{2} \pm 2\sqrt{n}]\}$;
10 Compute $F \leftarrow \{y \in \{0,1\}^n \mid \forall i \in S : \textsc{Om}_z(x_i) = \textsc{Om}_y(x_i)\}$;
11 Sample $x \in F$ uniformly at random and query $g(\textsc{Om}_z(x))$;
```

λ. By Corollary 2.15 this implies the desired bound for the n-ary black-box complexity of $\textsc{OneMax}_n$.

First we show that the algorithm employs only unbiased variation operators of arity at most n. We have already argued that sampling uniformly at random from $\{0,1\}^n$ is a 0-ary unbiased variation operator and that `complement`$(\cdot)$ is a unary unbiased one. Therefore, we need to show that the operator "sample $x \in F$ uniformly at random" is unbiased and of arity at most n. The latter follows from the fact that the size of S is at most $s = O(n/\log n)$. The unbiasedness of the variation operator follows essentially from the fact that we sample from F uniformly. More precisely, let us define the following family of $|S|$-ary distributions over $\{0,1\}^n$. Abbreviate $F(w^1, \ldots, w^{|S|}) := \{y \in \{0,1\}^n \mid \forall i \in [|S|] : \textsc{Om}_z(w_i) = \textsc{Om}_y(w_i)\}$ and set

$$D(x|w^1, \ldots, w^{|S|}) :=$$

$$\begin{cases} |F(w^1, \ldots, w^{|S|})|^{-1}, & \text{if } F(w^1, \ldots, w^{|S|}) \neq \emptyset \text{ and } x \in F(w^1, \ldots, w^{|S|}), \\ 0, & \text{if } F(w^1, \ldots, w^{|S|}) \neq \emptyset \text{ and } x \notin F(w^1, \ldots, w^{|S|}), \\ 2^{-n}, & \text{otherwise.} \end{cases}$$

It is now easy to verify that $D(\cdot \mid w^1, \ldots, w^{|S|})_{w^1, \ldots, w^{|S|} \in \{0,1\}^n}$ is a family of unbiased distributions: Let $y \in F(w^1, \ldots, w^{|S|})$ and $v \in \{0,1\}^n$. For all $j \in [|S|]$ clearly we have $\textsc{Om}_{y \oplus v}(w^j \oplus v) = \textsc{Om}_y(w^j)$ and, consequently, we have $y \oplus v \in F(w^1 \oplus v, \ldots, w^{|S|} \oplus v)$. Similarly we conclude that for all permutations σ of $[n]$ we have $\sigma(y) \in F(\sigma(w^1), \ldots, \sigma(w^{|S|}))$. The same reasoning also proves that $F(w^1, \ldots, w^{|S|}) = \emptyset$ if and only if $F(\sigma(w^1 \oplus v), \ldots, \sigma(w^{|S|} \oplus v)) = \emptyset$.

Each run of Algorithm 31 requires $\alpha n/\log n + 2 = O(n/\log n)$ queries. The total probability of failure is at most the sum of the probabilities that

- the median of the $\textsc{Om}_z(x_i)$-values is not in $[\frac{n}{2} \pm \sqrt{n}]$,
- the probability that there exists a value $\ell \in [\frac{n}{2} \pm 2\sqrt{n}]$ with $\textsc{Om}_z(x_j) \neq \ell$ for all $j \in [s]$,

- the probability that the size of $S = \{i \in [s] \mid \textsc{Om}_z(x_i) \in [\frac{n}{2} \pm 2\sqrt{n}]\}$ is less than $\frac{s}{2}(1-2e^{-8})$, and
- the probability that in line 11 we do not sample z.

Each of these probabilities is at most $o(n^{-\lambda})$ for any constant value $\lambda \in \mathbb{R}$. By a simple union bound we infer that the probability that the target string z is sampled in one run of Algorithm 31 is at least $1 - o(n^{-\lambda})$. This concludes the proof. □

8.2.3. Proof of Theorem 8.2 for $k = \Omega(\log^3 n) \cap [n]$

The proof for the case $k = \Omega(\log^3 n)$ uses a simple idea (which we first exploited in [DJK+11], the paper underlying Section 4 and which we also used for proving Theorem 7.1 in Section 7): Given some arity k, $\log^3(n) \leq k < n$, we subdivide the whole bit string into blocks of length k. We show that these blocks can be optimized one after the other, each in $O(k/\log k)$ steps. As there are $\lceil n/k \rceil$ such blocks, the desired $O(n/\log k)$ runtime bound follows.

Proof of Theorem 8.2 for $k = \Omega(\log^3 n)$. To ease reading we assume that k is even. For odd values of k, in the following proof all occurrences of $k/2$ must be replaced by $\lceil k/2 \rceil$. Further we skip the "with probability at least..."-statements. Instead, we bound the total failure probability at the end of this proof. Since k is large, a simple union bound will show that we can assume all statements to hold with high enough probability.

Fix $\alpha := 9 = \lceil 8(1-2e^{-8})^{-1} \rceil$ and $s := \alpha k/\log k$. For a better presentation of the ideas let us fix the unknown target function $\textsc{Om}_z \in \textsc{OneMax}_n$.

By Corollary 2.15 it suffices to show that, with high probability, Algorithm 32 queries the target string z. Each run of Algorithm 32 requires $O(n/\log k)$ queries.

The notation used in Algorithm 32 is as follows.

For $x, y \in \{0,1\}^n$ by $O(x,y)$ we denote the set $\{i \in [n] \mid x_i = y_i\}$ of all indices in which x and y coincide. As we shall see below, throughout the run of Algorithm 32, the set $O(x,y)$ equals the set of positions for which we know that $x_i = z_i$ must hold. We call these positions *optimized*.

The variation operator `flipKWhereDifferent`$(\cdot,\cdot)$ is a binary operator that given two strings $x, y \in \{0,1\}^n$ picks uniformly at random $k' := \min\{k, n - |O(x,y)|\}$ different elements $i_1, \ldots, i_{k'}$ from the set of positions $[n]\backslash O(x,y)$ in which x and y disagree. It outputs the string $x \oplus e^n_{i_1} \oplus \ldots \oplus e^n_{i_{k'}}$. That is, it flips k' positions in x in which x and y differ. This is easily verified to be an unbiased operator. As in the proof of the unbiasedness of the operator "sample $x \in F$ uniformly at random" in Section 8.2.2 this follows essentially from the fact that (i) the k' positions are sampled uniformly at random from $[n]\backslash O(x,y)$, that (ii) for all $w \in \{0,1\}^n$ the equation $O(x \oplus w, y \oplus w) = O(x,y)$ holds, and that (iii) for all $\sigma \in S_n$ we have $O(\sigma(x), \sigma(y)) = \sigma(O(x,y))$.

Let $j \in [\lceil n/k \rceil]$. The strings x and $x^{(j,1)}$ are used to encode which substring ("block") is to be optimized in the j-th *phase*. Namely, throughout the j-th phase all entries in positions

$$I^{(j)} := O(x, x^{(j,1)}) = \{i \in [n] \mid x_i = x^{(j,1)}_i\}$$

Algorithm 32: A k-ary unbiased ranking-based black-box algorithm for $k = \Omega(\log^3 n)$ optimizing $\textsc{OneMax}_n$ in $O(n/\log k)$ queries.

```
1  Initialization:
2     Sample $x \in \{0,1\}^n$ uniformly at random and query $g(\textsc{Om}_z(x))$;
3     Set $y \leftarrow$ complement($x$) and query $g(\textsc{Om}_z(y))$;
4  for $j = 1, \dots, \lceil n/k \rceil$ do
5     Sample $x^{(j,1)} \leftarrow$ flipKWhereDifferent($x, y$) and query $g(\textsc{Om}_z(x^{(j,1)}))$;
6     for $i = 2, \dots, s$ do
7        Sample $x^{(j,i)}$ from $\{v \in \{0,1\}^n \mid \forall \ell \in I^{(j)} : v_\ell = x_\ell\}$ uniformly at
         random and query $g\big(\textsc{Om}_z(x^{(j,i)})\big)$;
8     Identify $g(\frac{k}{2} + c^{(j)})$ ; //$c^{(j)}$ is the contribution of bits in $I^{(j)}$ to the
      $\textsc{Om}_z$-values
9     Compute $S^{(j)}$ ; //set of samples with $(g \circ \textsc{Om}_z)$-value in $[\frac{k}{2} \pm 2\sqrt{k} + c^{(j)}]$
10    Compute $F^{(j)}$ ; //set of all feasible bit strings
11    Sample $z^{(j)} \in F^{(j)}$ uniformly at random and query $g(\textsc{Om}_z(z^{(j)}))$;
12    Update $y \leftarrow$ Update($y, x, x^{(j,1)}, z^{(j)}$) and query $g(\textsc{Om}_z(y))$;
13    Update $x \leftarrow z^{(j)}$;
```

remain untouched. We only allow the entries in positions

$$R^{(j)} := [n] \backslash I^{(j)}$$

to be flipped. We call $I^{(j)}$ the set of *irrelevant indices* and we call $R^{(j)} = [n] \backslash I^{(j)}$ the set of *relevant indices*. Unless we are in the very last phase $j = \lceil n/k \rceil$ we have $|I^{(j)}| = n - k$ and $|R^{(j)}| = k$.

Similarly one easily verifies that the operator "Sample $x^{(j,i)}$ from $\{v \in \{0,1\}^n \mid \forall \ell \in I^{(j)} : v_\ell = x_\ell\}$ uniformly at random" employed in line 7 of Algorithm 32 is an unbiased one. The underlying distribution can be specified as

$$D(c \mid a, b) := \begin{cases} 2^{-|a \oplus b|_1}, & \text{if } x_i = a_i \text{ for all } i \in [n] \text{ with } a_i = b_i\,, \\ 0, & \text{otherwise} \end{cases}$$

for all $a, b, c \in \{0,1\}^n$.

Since we do not touch the entries in positions $I^{(j)}$, all search points sampled in the j-th phase (lines 5–13) have an $\textsc{Om}_z$-value of at least $c^{(j)} := |\{i \in I^{(j)} \mid x_i = z_i\}|$. Our algorithm, of course, does not know the value of $c^{(j)}$. Nevertheless we are able to infer $g(\frac{k}{2} + c^{(j)})$. This can be done as follows. Let $x^{(j,i)}$ be one of the search points $x^{(j,2)}, \dots, x^{(j,s)}$ sampled in line 7 such that $g(\textsc{Om}_z(x^{(j,i)}))$ equals the median of the multiset $\{g(\textsc{Om}_z(x^{(j,1)})), \dots, g(\textsc{Om}_z(x^{(j,s)}))\}$ of the sampled $(g \circ \textsc{Om}_z)$-values.

Let $\tilde{x}^{(j,i)}$ be the bit string which, on the relevant k bits, is the bitwise complement of $x^{(j,i)}$. Formally, $\tilde{x}^{(j,i)}_\ell := 1 - x^{(j,i)}_\ell$ for all $\ell \in R^{(j)}$ and $\tilde{x}^{(j,i)}_\ell := x^{(j,i)}_\ell = x_\ell$ for all $\ell \in I^{(j)}$. Clearly, $\tilde{x}^{(j,i)}$ can be obtained from $x^{(j,i)}$, x, and $x^{(j,1)}$ from the 3-ary unbiased variation operator that, given $a, b, c, d \in \{0,1\}^n$ satisfies

$$D(d \mid a,b,c) = \begin{cases} 1, & \text{if } \forall \ell \in [n] : (b_\ell \neq c_\ell \Rightarrow d_\ell = 1 - a_\ell) \wedge (b_\ell = c_\ell \Rightarrow d_\ell = a_\ell)\,, \\ 0, & \text{otherwise.} \end{cases}$$

To identify $g(\frac{k}{2}+c^{(j)})$, in line 8 we query $g\left(\mathrm{OM}_z(\tilde{x}^{(j,i)})\right)$. Clearly, $g\big(\mathrm{OM}_z(\tilde{x}^{(j,i)})\big) = g\big(\frac{k}{2}+c^{(j)}\big)$ if and only if $g\big(\mathrm{OM}_z(\tilde{x}^{(j,i)})\big) = g\big(\mathrm{OM}_z(x^{(j,i)})\big)$. In case $g\big(\mathrm{OM}_z(\tilde{x}^{(j,i)})\big) \neq g\big(\mathrm{OM}_z(x^{(j,i)})\big)$ we know by Lemma 8.9 that $g\big(\frac{k}{2}+c^{(j)}\big)$ is the median of the sampled values between $\big[g\big(\mathrm{OM}_z(\tilde{x}^{(j,i)})\big), g\big(\mathrm{OM}_z(x^{(j,i)})\big)\big]$ or $\big[g\big(\mathrm{OM}_z(\tilde{x}^{(j,i)})\big), g\big(\mathrm{OM}_z(x^{(j,i)})\big)\big]$, respectively, where each sampled $(g \circ \mathrm{OM}_z)$-value is counted with multiplicity one.

Note that once we have determined $g(\frac{k}{2}+c^{(j)})$, we can compute the sets

$$S^{(j)} := \{i \in [s] \mid \mathrm{OM}_z(x^{(j,i)}) \in [\tfrac{k}{2} \pm 2\sqrt{k} + c^{(j)}]\} \text{ and}$$
$$F^{(j)} := \{v \in \{0,1\}^n \mid \big(\forall \ell \in I^{(j)} : v_\ell = x_\ell\big) \wedge \big(\forall i \in S^{(j)} : \mathrm{OM}_z(x^{(j,i)}) = \mathrm{OM}_v(x^{(j,i)})\big)\}\,.$$

This is due to Lemma 8.6 where we have shown that for all $\ell \in [\frac{k}{2} \pm 2\sqrt{k}]$ there exists at least one i such that $\mathrm{OM}_z(x^{(j,i)}) = \ell + c^{(j)}$.

We call $F^{(j)}$ the set of all *feasible* bit strings for the j-th block. In line 11 we sample from this set uniformly at random. Note that for identification of $F^{(j)}$ we need at most all samples in $S^{(j)}$ plus the two strings x and $x^{(j,1)}$ which encode the current block we are optimizing. This is, the arity of the corresponding variation operator "sample $x \in F^{(j)}$ uniformly at random" is at most $|S| + 2 \leq 9k/\log k + 2$. Since we assume that $k = \Omega(\log^3 n)$, this expression can be bounded by k for large enough n.

For the same reason we can assume that $|F^{(j)}| = 1$. This is due to Lemma 8.10 where we have shown that $|F^{(j)}| = 1$ with probability at least $1 - 2^{-\frac{s}{2}(1-2e^{-8})}$. Note that under this assumption in line 11 we actually do not sample randomly but we deterministically sample the search point $z^{(j)}$ which coincides with z on all relevant bits $R^{(j)}$ and it coincides with x on all other bits $I^{(j)}$.

In the last step of the j-th phase we need to update x and y. Recall that by $O(x,y)$ we indicate which bits have been optimized already. So we need to update $O(x,y)$ by adding to it $R^{(j)}$. This can be done in the following way. First we update y by replacing it with

$$\mathtt{Update}(y, x, x^{(j,1)}, z^{(j)}) := \begin{cases} z_\ell^{(j)}, & \text{if } \ell \in R^{(j)}\,, \\ y_\ell, & \text{otherwise.} \end{cases}$$

Formally, $\mathtt{Update}(\cdot,\cdot,\cdot,\cdot)$ is a 4-ary variation operator that, given some $a, b, c, d \in \{0,1\}^n$ returns a vector with $(\mathtt{Update}(a,b,c,d))_i = a_i$ for all i with $b_i = c_i$ and $(\mathtt{Update}(a,b,c,d))_i = d_i$ for all i with $b_i \neq c_i$. Clearly, $\mathtt{Update}(\sigma(a \oplus w), \sigma(b \oplus w), \sigma(c \oplus w), \sigma(d \oplus w)) = \sigma(\mathtt{Update}(a,b,c,d) \oplus w)$ for all bit strings $w \in \{0,1\}^n$ and all permutations $\sigma \in S_n$. Therefore, $\mathtt{Update}(\cdot,\cdot,\cdot,\cdot)$ is an unbiased variation operator.

In line 13 we finally update x by replacing it with $z^{(j)}$, i.e., we set $x \leftarrow z^{(j)}$. This concludes the j-th phase. Summarizing all the above, we have reduced the Hamming distance from x to y by k'. We also have $y_i = x_i = z_i^{(j)}$ for all $i \in R^{(j)}$ and, as we shall see below, with high probability, this translates to $x_i = z_i$ for all $i \in R^{(j)}$.

The total number of search points queried in the j-th phase is $s+3 = O(k/\log k)$. Hence, the total number of queries made by the algorithm is $\lceil n/k \rceil (s+3) + 2 = O(n/\log k)$.

To conclude the proof let us bound the total probability of failure. For each block j the total probability of failure is at most the sum of the probabilities that

- the median of the $\textsc{Om}_z(x_i)$-values is not in $[\frac{k}{2} \pm \sqrt{k} + c^{(j)}]$,
- the probability that there exists a value $\ell \in [\frac{k}{2} \pm 2\sqrt{k} + c^{(j)}]$ with $\textsc{Om}_z(x^{(j,i)}) \neq \ell$ for all $i \in [s]$,
- the probability that the size of $S = \{i \in [s] \mid \textsc{Om}_z(x_i) \in [\frac{k}{2} \pm 2\sqrt{k} + c^{(j)}]\}$ is less than $\frac{s}{2}(1-2e^{-8})$, and
- the probability that in line 11 we do not sample z.

Each of these probabilities is at most $O(\sqrt{k}\exp(-\sqrt{k}/\log k))$. By the union bound the total failure probability is at most $O(n/k)O(\sqrt{k}\exp(-\sqrt{k}/\log k))$, which due to the fact that $k = \Omega(\log^3 n)$, is $o(1)$, as desired. □

8.2.4. Proof of Theorem 8.2 for $k = O(\log^3 n)$

In the last part of the proof for Theorem 8.2, we consider here in this section the case $k = O(\log^3 n)$. Again we do a block-wise optimization of the target function. However, for such small values of k, the union bound does not suffice for a high probability statement. Instead, we need to identify ways to ensure that each block-wise optimization yields the desired equality $z_i^{(j)} = z_i$ for all $i \in R^{(j)}$, in the notation of the previous section. This can be done by optimizing the first length-k block using the linear query time strategy implicit in Lemma 8.4. We use this block as a reference block. By flipping all bits in the reference block and flipping all bits in the block currently under investigation, we can probe whether or not all bits in this block coincide with the corresponding entries of the target string.

Proof of Theorem 8.2 for $k = O(\log^3 n)$. As we have seen in Section 8.2.1, for constant values of k, Theorem 8.2 is a special case of Lemma 8.4. Therefore, we can assume in the following that $k = k(n)$ grows with n. Further we assume that k is even. For odd values of k, in the following proof all occurrences of $k/2$ must be replaced by $\lceil k/2 \rceil$.

Fix $\alpha := 9 = \lceil 8(1-2e^{-8})^{-1} \rceil$ and $s := \alpha k/\log k$. For a better presentation of the ideas let us fix the unknown target function $\textsc{Om}_z \in \textsc{OneMax}_n$.

First we show that Algorithm 33 creates in $k+2$ queries two search points x' and y' of Hamming distance $n-k$ with the additional property that for all $i \in [n]$ with $x'_i = y'_i$ we know $x'_i = z_i$ with certainty.

To this end, let us first fix the notation. `flip1WhereDifferent`$(\cdot,\cdot)$ is the operator introduced in Algorithm 32. It flips exactly one bit value of the first argument. The position is chosen uniformly at random from the set of positions in which the first and the second argument disagree.

`Update`$_2(\cdot,\cdot,\cdot)$ is a variation operator, that creates from $y', x', w \in \{0,1\}^n$ the string `Update`$_2(y',x',w)$ with $(\texttt{Update}_2(y',x',w))_i = y'_i$ if $x'_i = w_i$ and $(\texttt{Update}_2(y',x',w))_i = x'_i$ if $x'_i \neq w_i$. This is easily verified to be an unbiased variation operator.

Algorithm 33: A binary unbiased ranking-based black-box algorithm for optimizing the first block of length k.

```
1 Initialization: Sample $x' \in \{0,1\}^n$ uniformly at random and query $g(\mathrm{OM}_z(x'))$;
2 Set $y' \leftarrow$ complement$(x')$ and query $g(\mathrm{OM}_z(y'))$;
3 Optimization: for $t = 1, \ldots, k$ do
4     Sample $w \leftarrow$ flip1WhereDifferent$(x', y')$ and query $g(\mathrm{OM}_z(w))$;
5     if $g(\mathrm{OM}_z(w)) > g(\mathrm{OM}_z(x'))$ then Update $x' \leftarrow w$;
6     else Update $y' \leftarrow$ Update$_2(y', x', w)$;
```

In line 5 of Algorithm 33 either we have $g(\mathrm{OM}_z(w)) > g(\mathrm{OM}_z(x'))$—in which case the position i in which x' and w differ satisfies $x'_i \neq z_i = w_i = y'_i$. In this case, updating x' clearly reduces the Hamming distance of x' and y' by one. It preserves the invariant that $x'_i = z_i$ for all positions $i \in [n]$ with $x'_i = y'_i$. If, alternatively, $g(\mathrm{OM}_z(w)) < g(\mathrm{OM}_z(x'))$ holds, then $x'_i = z_i \neq w_i = y'_i$ and we update y' by replacing its i-th bit with x'_i. This also reduces the Hamming distance of x' and y' by one, again preserving the invariant $x'_i = y'_i \Rightarrow x'_i = z_i$.

Therefore, after termination of Algorithm 33 we have two bit strings x', y' of Hamming distance $|x' \oplus y'|_1 = k$ that satisfy $(x'_i = y'_i) \Rightarrow (x'_i = z_i)$. In what follows, we call the bit positions $O(x', y') = \{i \in [n] \mid x'_i = y'_i\}$ the "reference block". Next we show how this block allows us to verify that another block of length k is optimized (i.e., that the entries of this block coincide with the entries of the target string z).

The basic idea is simple: first we create from x' and y' two strings x and y such that $O(x, y) = O(x', y')$ but $x_i \neq x'_i$ for all $i \in O(x', y')$. Certainly we have $x_i \neq z_i$ for all such $i \in O(x', y')$. Starting from x and y, we run the same block-wise optimization routine as in Subsection 8.2.3 where the blocks are chosen from $[n]\backslash O(x', y')$. If we want to test that k specific bits of some candidate string $z^{(j)}$ coincide with the entries of z, all we need to do is to flip in $z^{(j)}$ all k bits of interest as well as all bits in block $O(x', y')$. Flipping the bits in block $O(x', y')$ increases the OM_z-value by k since for all $i \in O(x', y')$, by construction, $z^{(j)}_i = x_i \neq z_i$. Therefore, the OM_z-values of the candidate solution $z^{(j)}$ and its offspring $\tilde{z}^{(j)}$ coincide if and only if $z^{(j)}$ and z coincide in the k bits of interest.

The notation in Algorithm 34 is the same as the one used in Algorithm 32. In addition, we make use of the following operators.

The string $\texttt{initialize}_1(x', y')$ is defined via

$$(\texttt{initialize}_1(x', y'))_i := \begin{cases} 1 - x'_i, & \text{if } i \in O(x', y')\,, \\ x'_i, & \text{if } i \notin O(x', y')\,. \end{cases}$$

Similarly, we set

$$(\texttt{initialize}_2(x', y'))_i := \begin{cases} 1 - x'_i, & \text{if } i \in O(x', y')\,, \\ y'_i, & \text{if } i \notin O(x', y')\,. \end{cases}$$

Algorithm 34: A k-ary unbiased ranking-based black-box algorithm for $k = O(\log^3 n) \cap \omega(1)$ optimizing $\textsc{OneMax}_n$ in $O(n/\log k)$ queries.

```
1  Input: Two bit strings $x'$ and $y'$ with $O(x',y') = k$ and $x'_i = z_i$ for all $i \in O(x',y')$;
2  Initialization:
3    Set $x \leftarrow$ initialize₁$(x',y')$ and query $g(\textsc{Om}_z(x))$;
4    Set $x \leftarrow$ initialize₂$(x',y')$ and query $g(\textsc{Om}_z(y))$;
5  for $j = 1, \ldots, \lceil (n-k)/k \rceil$ do
6      repeat
7          Sample $x^{(j,1)} \leftarrow$ flipKWhereDifferent$(x,y)$ and query $g(\textsc{Om}_z(x^{(j,1)}))$;
8          for $i = 2, \ldots, s$ do
9              Sample $x^{(j,i)}$ from $\{v \in \{0,1\}^n \mid \forall \ell \in I^{(j)} : v_\ell = x_\ell\}$ uniformly at random and query $g\big(\textsc{Om}_z(x^{(j,i)})\big)$;
10         Identify $g(\frac{k}{2} + c^{(j)})$ ; //$c^{(j)}$ is the contribution of bits in $I^{(j)}$ to the $\textsc{Om}_z$-values
11         Compute $S^{(j)}$ ; //set of samples with $(g \circ \textsc{Om}_z)$-value in $[\frac{k}{2} \pm 2\sqrt{k} + c^{(j)}]$
12         Compute $F^{(j)}$ ; //set of all feasible bit strings
13         Sample $z^{(j)} \in F^{(j)}$ uniformly at random and query $g(\textsc{Om}_z(z^{(j)}))$;
14         Set $\tilde{z}^{(j)} \leftarrow$ test$(z^{(j)}, x', y', x, x^{(j,1)})$ and query $g(\textsc{Om}_z(\tilde{z}^{(j)}))$;
15     until $g(\textsc{Om}_z(z^{(j)})) = g(\textsc{Om}_z(\tilde{z}^{(j)}))$ ;
16     Update $y \leftarrow$ Update$(y, x, x^{(j,1)}, z^{(j)})$ and query $g(\textsc{Om}_z(y))$;
17     Update $x \leftarrow z^{(j)}$;
18 Set $w \leftarrow$ finish$(x, x', y')$ and query $g(\textsc{Om}_z(w))$;
```

The string `initialize`$_1(x',y')$ is obtained through sampling from the distribution $D(w \mid x',y') = 1$ if $w_i = 1 - x'_i$ if and only if $i \in O(x',y')$. This is an unbiased distribution. Hence, both `initialize`$_1(\cdot,\cdot)$ and, by similar reasoning, `initialize`$_2(\cdot,\cdot)$ are unbiased variation operators.

In line 14 we query `test`$(z^{(j)}, x', y', x, x^{(j,1)})$ which is defined via

$$(\texttt{test}(z^{(j)}, x', y', x, x^{(j,1)}))_i := \begin{cases} 1 - z_i^{(j)}, & \text{if } i \in O(x',y') \text{ or } x_i \neq x_i^{(j,1)}\,, \\ z_i^{(j)}, & \text{otherwise.} \end{cases}$$

Again this is easily verified to be sampled from an unbiased (5-ary) distribution.

Lastly, we define `finish`(x, x') via

$$(\texttt{finish}(x, x', y'))_i := \begin{cases} 1 - x_i, & \text{if } i \in O(x',y')\,, \\ x_i, & \text{if } i \notin O(x',y')\,. \end{cases}$$

After having optimized all bits in $[n] \backslash O(x',y')$, this operator finally replaces in x the entries in $O(x',y')$ by their complement. Therefore, `finish`(x, x', y') equals the target string z.

Note that Algorithm 34 queries z in line 18 with certainty. Hence, we only need to argue that the expected number of queries of Algorithm 34 is $O(n/\log k)$. We optimize

$\lceil (n-k)/k \rceil$ blocks of length k. Call each execution of lines 7–14 for optimizing a block B a *run for B*. As argued in Section 8.2.3, for any such block B, the probability that in line 14 of Algorithm 34 we have $g(\text{OM}_z(z^{(j)})) = g(\text{OM}_z(\tilde{z}^{(j)}))$ already after the first run for B is at least $1 - o(k^{-\lambda})$ for all constant values λ. In particular, it is at least constant. This shows that at most a constant number of runs are expected for optimizing any block, compare Lemma 2.14 for a formal proof of this claim. Each run requires $s + 2 = O(k/\log k)$ queries. This shows that we need an expected number of $O(k/\log k)$ queries to optimize any length-k block. Since there are $\lceil (n-k)/k \rceil$ of them, the statement follows by the linearity of expectations, Lemma 2.6. □

8.3. The Different Black-Box Complexities of BinaryValue

In the previous section, we have seen that the additional ranking restriction did not increase the black-box complexity of the OneMax_n functions class. In this section, we show an example where the two kinds of complexities greatly differ. Surprisingly, another simple class of classical test functions does the job, namely the class of generalized binary-value functions.

We show that the unbiased black-box complexity of the larger class BinaryValue^*_n is $O(\log n)$, cf. Theorem 8.11, whereas the unrestricted ranking-based black-box complexity of the smaller class BinaryValue_n is $\Omega(n)$, cf. Theorem 8.12.

Let us begin with the upper bound for the unbiased black-box complexity.

Theorem 8.11. *The $*$-ary unbiased black-box complexity of BinaryValue^*_n (and thus, the one of BinaryValue_n) is at most $\lceil \log_2 n \rceil + 2$.*

For every $z \in \{0,1\}^n$ and $\sigma \in S_n$ the function $\text{Bv}_{z,\sigma}$ has 2^n different function values. That is, the function $\text{Bv}_{z,\sigma} : \{0,1\}^n \to [0..2^n - 1]$ is one-to-one. This is in strong contrast to the functions $\text{OM}_z \in \text{OneMax}_n$ which obtain values in $[0..n]$ only and are thus far from being one-to-one functions. Therefore, from each query to a BinaryValue function we obtain much more information about the underlying target string than we would gain from any OneMax function. In particular, for each query x and for each $i \in [n]$ we can derive from $\text{Bv}_{z,\sigma}(x)$ whether or not $x_{\sigma(i)} = z_{\sigma(i)}$. Hence, all we need to do is to identify σ. This can be done by binary search.

Proof of Theorem 8.11. We show that Algorithm 35 is an unbiased black-box algorithm which optimizes every $\text{Bv}_{z,\sigma} \in \text{BinaryValue}^*_n$ using at most $\lceil \log_2 n \rceil + 2$ queries.

Algorithm 35: An unbiased black-box algorithm optimizing BinaryValue_n in $\lceil \log_2 n \rceil + 2$ queries.

```
1 Sample $x^{(1)} \in \{0,1\}^n$ uniformly at random and query $\text{Bv}_{z,\sigma}(x^{(1)})$;
2 for $k = 2, \ldots, \lceil \log_2 n \rceil + 1$ do
3     Sample $x^{(k)} \leftarrow$ flipHalf$(x^{(1)}, \ldots, x^{(k-1)})$ and query $\text{Bv}_{z,\sigma}(x^{(k)})$;
4 Set $x^{(\lceil \log_2 n \rceil + 2)} \leftarrow$ consistent$(x^{(1)}, \ldots, x^{(\lceil \log_2 n \rceil + 1)})$ and query
  $\text{Bv}_{z,\sigma}(x^{(\lceil \log_2 n \rceil + 2)})$;
```

To describe the variation operators used in Algorithm 35, let $k \in \mathbb{N}$ and let $y^{(1)}, \ldots, y^{(k+1)} \in \{0,1\}^n$. Set

$$F^{(1)}(y^{(1)}, y^{(2)}) := \{j \in [n] \mid y_j^{(1)} \neq y_j^{(2)}\} \text{ and}$$
$$F^{(0)}(y^{(1)}, y^{(2)}) := [n] \backslash F^{(1)}(y^{(1)}, y^{(2)}) .$$

That is, $F^{(1)}(y^{(1)}, y^{(2)})$ contains exactly those bit positions in which $y^{(1)}$ and $y^{(2)}$ disagree and $F^{(0)}(y^{(1)}, y^{(2)})$ is the set of positions in which $y^{(1)}$ and $y^{(2)}$ coincide.

Let $1 \leq \ell < k$. For each $(i_1, \ldots, i_\ell) \in \{0,1\}^\ell$ we set

$$F^{(i_1,\ldots,i_\ell,1)}(y^{(1)}, \ldots, y^{(\ell+2)}) := \{j \in F^{(i_1,\ldots,i_\ell)}(y^{(1)}, \ldots, y^{(\ell+1)}) \mid y_j^{(\ell+1)} \neq y_j^{(\ell+2)}\} \text{ and}$$
$$F^{(i_1,\ldots,i_\ell,0)}(y^{(1)}, \ldots, y^{(\ell+2)}) := F^{(i_1,\ldots,i_\ell)}(y^{(1)}, \ldots, y^{(\ell+1)}) \backslash F^{(i_1,\ldots,i_\ell,1)}(y^{(1)}, \ldots, y^{(\ell+2)}) .$$

This way we iteratively define $F^{(i_1,\ldots,i_k)}(y^{(1)}, \ldots, y^{(k+1)})$ for all $(i_1, \ldots, i_k) \in \{0,1\}^k$. For any such vector $(i_1, \ldots, i_k) \in \{0,1\}^k$ the set $F^{(i_1,\ldots,i_k)}(y^{(1)}, \ldots, y^{(k+1)})$ contains exactly the subset of positions from $F^{(i_1,\ldots,i_{k-1})}(y^{(1)}, \ldots, y^{(k)})$ in which $y^{(k)}$ and $y^{(k+1)}$ agree ($i_k = 0$) and for $i_k = 1$ it contains the subset of positions in $F^{(i_1,\ldots,i_{k-1})}(y^{(1)}, \ldots, y^{(k)})$ in which $y^{(k)}$ and $y^{(k+1)}$ disagree.

Let

$$Z(y^{(1)}, \ldots, y^{(\ell)}) := \big\{y^{\ell+1} \in \{0,1\}^n \mid \forall (i_1, \ldots, i_{\ell-1}) \in \{0,1\}^{\ell-1} :$$
$$|F^{(i_1,\ldots,i_{\ell-1},1)}(y^{(1)}, \ldots, y^{(\ell+1)})| = \lfloor |F^{(i_1,\ldots,i_{\ell-1})}(y^{(1)}, \ldots, y^{(\ell)})|/2 \rfloor \big\} .$$

That is, $Z(y^{(1)}, \ldots, y^{(\ell)})$ is the set of bit strings $y^{(\ell+1)}$, which, for every $(i_1, \ldots, i_{\ell-1})$ partitions the set $F^{(i_1,\ldots,i_{\ell-1})}(y^{(1)}, \ldots, y^{(\ell-1)})$ into two subsets $F^{(i_1,\ldots,i_{\ell-1},1)}(y^{(1)}, \ldots, y^{(\ell+1)})$ and $F^{(i_1,\ldots,i_{\ell-1},0)}(y^{(1)}, \ldots, y^{(\ell+1)})$ of (almost) equal size.

For all $\ell \in \mathbb{N}$, the variation operator `flipHalf`$(\cdot, \ldots, \cdot)$ samples from the ℓ-ary distribution $(D(\cdot|\cdot, \ldots, \cdot))_{y^{(1)},\ldots,y^{(\ell)} \in \{0,1\}^n}$, which for given $y^{(1)}, \ldots, y^{(\ell)} \in \{0,1\}^n$ assigns to each $y \in \{0,1\}^n$ the probability

$$D(y \mid y^{(1)}, \ldots, y^{(\ell)}) := \begin{cases} |Z(y^{(1)}, \ldots, y^{(\ell)})|^{-1}, & \text{if } y \in Z(y^{(1)}, \ldots, y^{(\ell)}) , \\ 0, & \text{otherwise.} \end{cases}$$

This is an unbiased distribution: For all $y, w, y^{(1)}, \ldots, y^{(\ell)} \in \{0,1\}^n$ and all $(i_1, \ldots, i_{\ell-1}) \in \{0,1\}^{\ell-1}$ we have

$$F^{(i_1,\ldots,i_{\ell-1})}(y^{(1)}, \ldots, y^{(\ell)}) = F^{(i_1,\ldots,i_{\ell-1})}(y^{(1)} \oplus w, \ldots, y^{(\ell)} \oplus w) .$$

From this we easily obtain that $y \in Z(y^{(1)}, \ldots, y^{(\ell)})$ if and only if $y \oplus w \in Z(y^{(1)} \oplus w, \ldots, y^{(\ell)} \oplus w)$. In addition, for all $\theta \in S_n$ we have

$$F^{(i_1,\ldots,i_{\ell-1})}(\theta(y^{(1)}), \ldots, \theta(y^{(\ell)})) = \theta(F^{(i_1,\ldots,i_{\ell-1})}(y^{(1)}, \ldots, y^{(\ell)})) ,$$

and thus, $y \in Z(y^{(1)}, \ldots, y^{(\ell)})$ holds if and only if $\theta(y) \in Z(\theta(y^{(1)}), \ldots, \theta(y^{(\ell)}))$. From this and the fact that each bit string in $Z(y^{(1)}, \ldots, y^{(\ell)})$ is assigned the same probability we infer that `flipHalf` is an unbiased variation operator. This is true for all ℓ.

For the description of the second variation operator we abbreviate $t := \lceil \log_2 n \rceil + 1$ and we assume that $\textsc{Bv}_{z,\sigma} \in \textsc{BinaryValue}^*_n$ is the (unknown) function to be optimized.

For all $y^{(1)}, \ldots, y^{(t)} \in \{0,1\}^n$ let

$$\mathcal{F}^{\text{consistent}}(y^{(1)}, \ldots, y^{(t)}) := \\ \{z' \in \{0,1\}^n \mid \exists \sigma' \in S_n \forall i \in [t] : \textsc{Bv}_{z',\sigma'}(y^{(i)}) = \textsc{Bv}_{z,\sigma}(y^{(i)})\}\,,$$

the set of all bit strings that are *consistent* with the queries $y^{(1)}, \ldots, y^{(t)}$. This is the set of all possible target strings. As we shall see below, in line 4 of Algorithm 35 we have $|\mathcal{F}^{\text{consistent}}(x^{(1)}, \ldots, x^{(t)})| = 1$.

Abbreviate $\mathcal{F}^{\text{consistent}}(y^{(1)}, \ldots, y^{(t)}) = \mathcal{F}^{\text{consistent}}$. For $y \in \{0,1\}^n$ set

$$D'(y \mid y^{(1)}, \ldots, y^{(t)}) := \begin{cases} |\mathcal{F}^{\text{consistent}}|^{-1}, & \text{if } y \in \mathcal{F}^{\text{consistent}}\,, \\ 0, & \text{if } \mathcal{F}^{\text{consistent}} \neq \emptyset \text{ and } y \notin \mathcal{F}^{\text{consistent}}\,, \\ 2^{-n}, & \text{otherwise.} \end{cases}$$

This is the distribution from which the variation operator `consistent`$(y^{(1)}, \ldots, y^{(t)})$ samples. It is an unbiased t-ary distribution as can be easily verified using the fact that $y \in \mathcal{F}^{\text{consistent}}(y^{(1)}, \ldots, y^{(t)})$ if and only if $y \oplus w \in \mathcal{F}^{\text{consistent}}(y^{(1)} \oplus w, \ldots, y^{(t)} \oplus w)$ and if and only if $\theta(y) \in \mathcal{F}^{\text{consistent}}\left(\theta(y^{(1)}), \ldots, \theta(y^{(t)})\right)$ for every $\theta \in S_n$.

In what follows we argue that in line 4 of Algorithm 35 there exists exactly one string $z' \in \mathcal{F}^{\text{consistent}}(x^{(1)}, \ldots, x^{(t)})$. In this case, clearly, $z' = z$ must hold.

Let us first show that from $x^{(1)}, \ldots, x^{(t)}$ we can infer the underlying target permutation σ. We do so by proving that **(a)** for any $2 \leq k \leq t$ and for all $j \in [n]$ we can determine the index $(i_1, \ldots, i_{k-1}) \in \{0,1\}^{k-1}$ with $\sigma(j) \in F^{(i_1,\ldots,i_{k-1})}(x^{(1)}, \ldots, x^{(k)})$. This suffices to determine σ because, by construction, for all vectors $(i_1, \ldots, i_{t-1}) \in \{0,1\}^{t-1}$, we have $|F^{(i_1,\ldots,i_{t-1})}(x^{(1)}, \ldots, x^{(t)})| \leq 1$.

The key argument proving statement **(a)** is the injectivity of $\textsc{Bv}_{z,\sigma}$, which, for any index $j \in [n]$ and for every search point $x \in \{0,1\}^n$, reveals whether or not $x_{\sigma(j)} = z_{\sigma(j)}$. Let us fix an index $j \in [n]$. Clearly, $\sigma(j) \in F^{(1)}(x^{(1)}, x^{(2)})$ if and only if

- $x^{(1)}_{\sigma(j)} = z_{\sigma(j)}$ and $x^{(2)}_{\sigma(j)} \neq z_{\sigma(j)}$, or
- $x^{(1)}_{\sigma(j)} \neq z_{\sigma(j)}$ and $x^{(2)}_{\sigma(j)} = z_{\sigma(j)}$.

Similarly, if $\sigma(j) \in F^{(i_1,\ldots,i_{k-1})}(x^{(1)}, \ldots, x^{(k)})$ is known, then $\sigma(j) \in F^{(i_1,\ldots,i_{k-1},1)}(x^{(1)}, \ldots, x^{(k+1)})$ if and only if

- $x^{(k)}_{\sigma(j)} = z_{\sigma(j)}$ and $x^{(k+1)}_{\sigma(j)} \neq z_{\sigma(j)}$, or
- $x^{(k)}_{\sigma(j)} \neq z_{\sigma(j)}$ and $x^{(k+1)}_{\sigma(j)} = z_{\sigma(j)}$.

Hence, in line 4 of Algorithm 35 we know σ. Let $z' \in \mathcal{F}^{\text{consistent}}(x^{(1)}, \ldots, x^{(t)})$. By definition, there exists a permutation $\sigma' \in S_n$ such that for all $i \in [t]$ we have

$\text{Bv}_{z',\sigma'}(x^{(i)}) = \text{Bv}_{z,\sigma}(x^{(i)})$. Let $j \in [n]$. As we have shown above, we can identify the vector $(i_1, \ldots, i_{t-1})$ such that $\sigma(j) \in F^{(i_1,\ldots,i_{t-1})}(x^{(1)}, \ldots, x^{(t)})$. By construction, also $\sigma'(j) \in F^{(i_1,\ldots,i_{t-1})}(x^{(1)}, \ldots, x^{(t)})$ must hold. This shows $\sigma' = \sigma$. Hence, $\text{Bv}_{z',\sigma'}(x^{(1)}) = \text{Bv}_{z',\sigma}(x^{(1)}) = \text{Bv}_{z,\sigma}(x^{(1)})$. But as this requires $z' = z$, we conclude that indeed $|\mathcal{F}^{\text{consistent}}(x^{(1)}, \ldots, x^{(t)})| = 1$.

Putting everything together we have shown that from the first $t = \lceil \log_2 n \rceil + 1$ samples we can infer both the target permutation σ as well as the target string z. This can be sampled from an unbiased distribution in the $(\lceil \log_2 n \rceil + 2)$nd query. □

Let us now prove that already the unrestricted ranking-based black-box complexity of BinaryValue_n is asymptotically different from the basic unbiased one of BinaryValue^*_n.

Theorem 8.12. *The unrestricted ranking-based black-box complexity of* BinaryValue_n *and* BinaryValue^*_n *is larger than* $n - 1$.

As discussed in the introduction, Droste, Jansen, and Wegener [DJW06] implicitly showed a lower bound of $\Omega(n/\log n)$ for the unrestricted ranking-based black-box complexity of BinaryValue_n. Our lower bound of $n-1$ is almost tight. An upper bounds of $n+1$ for the unrestricted ranking-based black-box complexity of BinaryValue^*_n can be shown exactly in the same way as in [DJW06, Theorem 5]. Intuitively, the algorithm which starts with a random initial search point and then, from left to right, flips in each iteration exactly one bit shows this bound. This is a deterministic version of Algorithm 28, which itself is similar to Algorithm 33.

For the unbiased black-box complexities of BinaryValue_n and BinaryValue^*_n, the situation is as follows. Both the basic as well as the ranking-based unary unbiased black-box complexity of BinaryValue_n are of order $\Theta(n \log n)$. The lower bound follows from the already mentioned theorem in [LW10a, Theorem 6], which implies that any function with a single global optimum has an unary unbiased black-box complexity of $\Omega(n \log n)$. The upper bound follows from the fact that, for example, Randomized Local Search (Algorithm 1) solves any instance $\text{Bv}_{z,\sigma} \in \text{BinaryValue}^*_n$ in an expected number of $O(n \log n)$ queries. The latter follows from the coupon-collector's problem.

For higher arities $k \geq 2$, Theorem 8.12 and Lemma 8.4 from Section 8.2.1 immediately yield the following.

Corollary 8.13. *For all $k \geq 2$, the k-ary unbiased ranking-based black-box complexity of* BinaryValue_n *and* BinaryValue^*_n *is larger than $n - 1$ and it is at most $4n - 5$.*

To derive the lower bound, Theorem 8.12, we employ Yao's minimax principle, cf. Theorem 2.16. We show that in the ranking-based black-box model any deterministic algorithm needs an expected number of more than $n - 1$ iterations to optimize Bv_z, if Bv_z is taken from BinaryValue_n uniformly at random. Theorem 2.16 then implies that for any randomized algorithm A there exist at least one instance $\text{Bv}_z \in \text{BinaryValue}_n$ such that it takes, in expectation, at least $n - 1$ iterations for algorithm A to optimize Bv_z. This implies Theorem 8.12.

The crucial observation is that when optimizing Bv_z with a ranking-based algorithms, then from t samples we can learn at most $t-1$ bits of the hidden bit string z. This is easy to see for two samples x, y. If $\text{Bv}_z(x) > \text{Bv}_z(y)$, we see that $x_k = z_k \neq y_k$, where $k := \max\{j \in [n] \mid x_j \neq y_j\}$, but we cannot infer any information about x_ℓ for $\ell \neq k$. Similarly, if we have t samples $x^{(1)}, \ldots, x^{(t)}$ and their corresponding Bv_z-values, we cannot infer $\binom{t}{2}$ bits of information as one might guess, but at most $t-1$ bits. As we shall see, this is an immediate consequence of the following combinatorial lemma.

Lemma 8.14. *Let $t \in [n]$ and let $x^{(1)}, \ldots, x^{(t)}$ be t pairwise different bit strings. For every pair $(i,j) \in [t]^2$ we set $\ell_{i,j} := \max\{k \in [n] \mid x^{(i)}_k \neq x^{(j)}_k\}$, the largest bit position in which $x^{(i)}$ and $x^{(j)}$ differ. Then $|\{\ell_{i,j} \mid i,j \in [t]\}| \leq t-1$.*

Proof. Let $z \in \{0,1\}^n$. By renaming the bit strings where required, we may assume $\text{Bv}_z(x^{(1)}) > \ldots > \text{Bv}_z(x^{(t)})$.

We prove the statement by induction on t. For $t = 1$ and $t = 2$ there is nothing to show. Therefore, we may assume that we have proven $|\{\ell_{i,j} \mid i,j \in [k]\}| \leq k-1$ for some $k \geq 2$. Now, if $\ell_{h,k+1} \in \{\ell_{i,j} \mid i,j \in [k]\}$ for all $h \in [k]$, then clearly we have $|\{\ell_{i,j} \mid i,j \in [k+1]\}| \leq k-1$. Thus, we may assume without loss of generality that there exists a $h \in [k]$ with $\ell_{h,k+1} \notin \{\ell_{i,j} \mid i,j \in [k]\}$.

Since $\text{Bv}_z(x^{(h)}) > \text{Bv}_z(x^{(k+1)})$, the definition of Bv_z implies that $z_{\ell_{h,k+1}} = x^{(h)}_{\ell_{h,k+1}} \neq x^{(k+1)}_{\ell_{h,k+1}}$.

Now, let $j \in [k]$. We show that either $\ell_{j,k+1} = \ell_{h,k+1}$ or $\ell_{j,k+1} = \ell_{j,h}$.

Let us first consider the case $j \leq h$. Since $\text{Bv}_z(x^{(j)}) > \text{Bv}_z(x^{(h)})$ it holds by the definition of Bv_z that $z_{\ell_{j,h}} = x^{(j)}_{\ell_{j,h}} \neq x^{(h)}_{\ell_{j,h}}$. Now, either $\ell_{j,h} \geq \ell_{h,k+1}$ or $\ell_{j,h} < \ell_{h,k+1}$. In the first case $x^{(k+1)}_{\ell_{j,h}} = x^{(h)}_{\ell_{j,h}} \neq x^{(j)}_{\ell_{j,h}}$, i.e., $\ell_{j,k+1} \geq \ell_{j,h}$. On the other hand, by definition of the $\ell_{i,i'}$'s, for all $\ell > \ell_{j,h}$ we have $x^{(j)}_\ell = x^{(h)}_\ell$. For the same reason, due to the fact $\ell > \ell_{j,h} \geq \ell_{h,k+1}$, we also have for all such ℓ that $x^{(h)}_\ell = x^{(k+1)}_\ell$. From this we infer $\ell_{j,k+1} \leq \ell_{j,h}$. This shows $\ell_{j,k+1} = \ell_{j,h}$.

Equivalently, if $\ell_{j,h} < \ell_{h,k+1}$, then $x^{(j)}_{\ell_{h,k+1}} = x^{(h)}_{\ell_{h,k+1}} \neq x^{(k+1)}_{\ell_{h,k+1}}$. Thus, $\ell_{j,k+1} \geq \ell_{h,k+1}$. On the other hand we have for all $\ell > \ell_{h,k+1} > \ell_{j,h}$ that $x^{(k+1)}_\ell = x^{(h)}_\ell = x^{(j)}_\ell$. This shows $\ell_{j,k+1} \leq \ell_{h,k+1}$ and we conclude $\ell_{j,k+1} = \ell_{h,k+1}$.

The reasoning for $j > h$ is the same. □

We are now ready to prove Theorem 8.12.

Proof of Theorem 8.12. Since the search space $\{0,1\}^n$ is finite, the set $\mathcal{A}$ of all deterministic algorithms on BinaryValue_n is finite, if we restrict our attention to those algorithms which stop querying search points after the n-th iteration.

As mentioned above, we equip BinaryValue_n with the uniform distribution. Let $\text{Bv}_z \in \text{BinaryValue}_n$ be drawn uniformly at random and let $A \in \mathcal{A}$ be a (deterministic) algorithm. In the following, we show that prior to the t-th iteration, the set of still possible target strings has size at least 2^{n-t+1} and that all of these target strings have the same probability to be the desired target string **(A)**. Consequently, the probability to query the correct bit string in the t-th iteration, given that the algorithm has not found it in a previous iteration, is at most 2^{-n+t-1}. This shows that the expected number of iterations $\mathrm{E}[T(\mathrm{BV}_z, A)]$ until algorithm A queries the target string z can be bounded from below by

$$\begin{aligned}\sum_{i=1}^{n} i \cdot \Pr[A \text{ queries } z \text{ in the } i\text{-th iteration}] \geq \sum_{i=1}^{n} i \cdot 2^{-n+i-1} &= \sum_{i=1}^{n} (n-i+1)\, 2^{-i} \\ &= (n+1) \sum_{i=1}^{n} 2^{-i} - \sum_{i=1}^{n} i\, 2^{-i} \,. \end{aligned} \tag{8.3}$$

A simple, but nonetheless very helpful observation shows

$$\begin{aligned}\sum_{i=1}^{n} i\, 2^{-i} = \sum_{i=1}^{n} 2^{-i} + \sum_{i=1}^{n} (i-1)\, 2^{-i} &= (1-2^{-n}) + 2^{-1} \sum_{i=1}^{n} (i-1)\, 2^{-(i-1)} \\ = (1-2^{-n}) + 2^{-1} \sum_{i=1}^{n-1} i\, 2^{-i} \,,\end{aligned}$$

yielding $\sum_{i=1}^{n} i\, 2^{-i} = 2(1-2^{-n}) - n2^{-n} = 2 - (n+2)2^{-n}$.

Plugging this into (8.3), we obtain

$$\mathrm{E}[T(\mathrm{BV}_z, A)] \geq (n+1)(1-2^{-n}) - (2-(n+2)2^{-n}) > n-1 \,.$$

This proves $\min_{A \in \mathcal{A}} \mathrm{E}[T(\mathrm{BV}_z, A)] > n-1$ for BV_z taken from $\textsc{BinaryValue}_n$ uniformly at random. Yao's minimax principle implies that for any distribution q over the set of deterministic algorithms we have $\max_{z \in \{0,1\}^n} \mathrm{E}[T(\mathrm{BV}_z, \tilde{A}_q)] \geq \min_{A \in \mathcal{A}} \mathrm{E}[T(\mathrm{BV}_z, A)] > n-1$. That is, the ranking-based black-box complexity of $\textsc{BinaryValue}_n$ is larger than $n-1$.

It remains to prove **(A)**. Let $t \leq n$ and let $x^{(1)}, \ldots, x^{(t)}$ be the search points which have been queried by the algorithm in the first t iterations. All the algorithm has learned about $x^{(1)}, \ldots, x^{(t)}$ is the ranking of these bit strings induced by BV_z, i.e., it knows for all $i, j \in [t]$ whether $\mathrm{BV}_z\big(x^{(i)}\big) > \mathrm{BV}_z\big(x^{(j)}\big)$, or $\mathrm{BV}_z\big(x^{(i)}\big) < \mathrm{BV}_z\big(x^{(j)}\big)$, or $\mathrm{BV}_z\big(x^{(i)}\big) = \mathrm{BV}_z\big(x^{(j)}\big)$. Note that $\mathrm{BV}_z\big(x^{(i)}\big) = \mathrm{BV}_z\big(x^{(j)}\big)$ implies $x^{(i)} = x^{(j)}$. Thus, this case can be disregarded as one cannot learn any additional information by querying the same bit string twice.

As in Lemma 8.14 we set $\ell_{i,j} := \max\{k \in [n] \mid x^{(i)}_k \neq x^{(j)}_k\}$ for all $i, j \in [t]$ and we set $\mathcal{L} := \{\ell_{i,j} \mid i, j \in [t]\}$.

Let $\ell \in \mathcal{L}$ and let $i, j \in [t]$ such that $\max\{k \in [n] \mid x^{(i)}_k \neq x^{(j)}_k\} = \ell$. We can fix $z_\ell = x^{(i)}_\ell$ if $\mathrm{BV}_z\big(x^{(i)}\big) > \mathrm{BV}_z\big(x^{(j)}\big)$, and we fix $z_\ell = x^{(j)}_\ell$ if $\mathrm{BV}_z\big(x^{(i)}\big) < \mathrm{BV}_z\big(x^{(j)}\big)$. That is, we can fix $|\mathcal{L}|$ bits of z.

Statement **(A)** follows from observing that for every bit string z' with $z'_\ell = z_\ell$ for all $\ell \in \mathcal{L}$ the function $\mathrm{BV}_{z'}$ yields exactly the same ranking as BV_z. Hence, all such z' are possible target strings. Since there is no way to differentiate between them, all of them are equally likely to be the desired target string.

Furthermore, it holds by Lemma 8.14 that $|\mathcal{L}| \leq t-1$. This shows that, at the end of the t-th iteration, there are at least $2^{n-(t-1)}$ possible target strings. By definition of $\mathcal{L}$, the algorithm has queried at most one of them. Consequently, prior to executing the $(t+1)$-st iteration, there are at most $2^{n-(t-1)} - 1 > 2^{n-t}$ bit strings which are equally likely to be the desired target string. This proves **(A)**. □

Note that already a much simpler proof, also applying Yao's minimax principle, shows the following general lower bound.

Theorem 8.15. *Let $\mathcal{F}$ be a class of functions such that each $f \in \mathcal{F}$ has a unique global optimum and such that for all $z \in \{0,1\}^n$ there exists a function $f_z \in \mathcal{F}$ with $z = \arg\max f_z$. Then the unrestricted ranking-based black-box complexity of $\mathcal{F}$ is $\Omega(n/\log n)$.*

8.4. Ranking-Based Black-Box Complexity of LeadingOnes

The proof ideas of the lower bound in the previous section can also be applied to $\textsc{LeadingOnes}^*_n$. Theorem 8.16 closes a gap left open in [DJTW03].

As discussed in the last paragraph of the introduction, Droste et al. in the conference version [DJTW03] of paper [DJW06] implicitly show that the ranking-based black-box complexity of $\textsc{LeadingOnes}_n$ is at least $\frac{n}{2} - o(n)$ and at most $n+1$. Using the same methods as for $\textsc{BinaryValue}_n$, we improve the lower bound to $n-1$.

Theorem 8.16. *The ranking-based black-box complexity of* $\textsc{LeadingOnes}_n$ *(and thus, the ranking-based black-box complexity of* $\textsc{LeadingOnes}^*_n$*) is strictly larger than $n-1$.*

Crucial for the proof is again the combinatorial statement of Lemma 8.14, from which we conclude that after t queries we cannot have learned more than $t-1$ bits of the hidden bit string z.

Let us conclude with two upper bounds for the ranking-based black-box complexity of $\textsc{LeadingOnes}^*_n$.

Remark 8.17. *The binary unbiased ranking-based black-box complexity of* $\textsc{LeadingOnes}^*_n$ *is $O(n \log n)$. For all $k \geq 3$ the k-ary unbiased ranking-based black-box complexity of* $\textsc{LeadingOnes}^*_n$ *is $O(n \log n / \log\log n)$.*

The first statement of Remark 8.17 follows easily from the proof of Theorem 4.7, the second one is Theorem 5.9.

8.5. Conclusions

Motivated by the fact that (i) previous complexity models for randomized search heuristics give unrealistic low complexities and (ii) that many randomized search heuristics only compare objective values, but not regard their absolute values, we added such a restriction to the two existing black-box models. While this does not change the black-box complexity of the $\textsc{OneMax}_n$ function class (this remains relatively low at $\Theta(n/\log n)$), we do gain an advantage for the $\textsc{BinaryValue}_n$ function class. Here the complexity is $O(\log n)$ without the ranking restriction, but $\Theta(n)$ in the ranking-based model. We obtain some more results improving previous work by different authors, summarized in Tables 8.1 and 8.2. These results indicate that the ranking-based black-box complexity might be a promising measure for the hardness of problems for randomized search heuristics.

Model	Arity	$\textsc{OneMax}_n$		$\textsc{BinaryValue}_n$	
unrestr.	n/a	$\Theta(n/\log n)$	[ER63]	$\Omega(n/\log n)$	[DJW06]
unbiased	1	$\Theta(n \log n)$	[LW10a]	$\Theta(n \log n)$	[LW10a]
	$2 \leq k \leq n$	$O(n/\log k)$	[DJK+11]	$O(n)$	[DJK+11]
	$*$			$O(\log n)$	Thm. 8.11
unbiased	1	$\Theta(n \log n)$	[LW10a]	$\Theta(n \log n)$	[LW10a]
ranking-b.	$2 \leq k \leq n$	$O(n/\log k)$	Thm. 8.2	$\Theta(n)$	Thm. 8.12
	$*$			$\Theta(n)$	Thm. 8.12

Table 8.1. Black-Box Complexities of $\textsc{OneMax}_n$ and $\textsc{BinaryValue}_n$. Remarks: The upper bounds for $\textsc{BinaryValue}_n$ also hold for $\textsc{BinaryValue}^*_n$. Abbreviations: unrestr. = unrestricted, ranking-b. = ranking-based

Model	Arity	$\textsc{LeadingOnes}_n$		$\textsc{LeadingOnes}^*_n$	
unrestr.	n/a	$\Theta(n)$	[DJW06]	$O(n \log n/ \log\log n)$	Thm. 5.1
unbiased	1	$\Theta(n^2)$	[LW10a]	$\Theta(n^2)$	[LW10a]
	2	$O(n \log n)$	Thm. 4.7	$O(n \log n)$	Thm. 4.7
	$3 \leq k \leq n$	$O(\frac{n \log n}{\log\log n})$	Thm. 5.8	$O(\frac{n \log n}{\log\log n})$	Thm. 5.8
unbiased	1	$\Theta(n^2)$	[LW10a]	$\Theta(n^2)$	[LW10a]
ranking-b.	2	$O(n \log n)$	Thm. 4.7	$O(n \log n)$	Thm. 4.7
	$3 \leq k \leq n$	$O(\frac{n \log n}{\log\log n})$	Thm. 5.9	$O(\frac{n \log n}{\log\log n})$	Thm. 5.9
	$*$	$> n-1$	Thm. 8.16	$> n-1$	Thm. 8.16

Table 8.2. Black-Box Complexities of $\textsc{LeadingOnes}_n$ and $\textsc{LeadingOnes}^*_n$. Abbreviations: unrestr. = unrestricted, ranking-b. = ranking-based

Part III

Black-Box Complexities of Combinatorial Problems

9 The Minimum Spanning Tree Problem

In the previous sections we have analyzed the black-box complexities of several artificial test functions. In this and the next section, we move a step forward and give a detailed analysis for the two combinatorial problems of finding a minimum spanning tree (MST) and finding single-source shortest paths. We begin with the MST problem. The MST problem can be modeled in a very natural way via (multi-criteria) pseudo-Boolean functions. This has the advantage that we can work in the unbiased model as defined by Lehre and Witt [LW10a] and we do not need to argue how their notion of unbiasedness translates to search spaces that are different from the hypercube $\{0,1\}^n$. This will be done Section 10, where we need to adjust the unbiasedness notion to more general search spaces $\mathcal{S}$.

It is interesting to see how the tools developed in the previous sections can be applied to combinatorial problems as well. We provide several bounds for the MST problem in the different black-box models.

The results presented in this section are based on the conference publication [DKLW11]. They are joint work with Benjamin Doerr, Timo Kötzing, and Johannes Lengler (ETH Zürich).

9.1. Introduction and Problem Definition

When moving from classical test functions to combinatorial problems, a number of additional modeling issues have to be regarded. As mentioned above, we start our analysis with the minimum spanning tree (MST) problem, because here it is generally agreed on that a bit string representation is most natural. That is, we first enumerate the edges E by $1,\ldots,m$ and then interpret a bit string $x \in \{0,1\}^m$ as the set of edges, whose corresponding bit is set to one. This way we can encode every possible subset of E. Each subset E' of E is assigned the objective value $(c(E'), w(E'))$, where $c(E')$ is the number of connected components induced by E' and $w(E')$ is the sum of the edge weights $\sum_{e\in E'} w(e)$. This model allows to use the definition of unbiasedness as in [LW10a]. When talking about ranking-based black-box complexity, the two-criteria fitness (number of connected components, total weight) needs attention, but we feel that the only reasonable model is to treat the two criteria separately. That is, we assume comparability in both criteria.

Previous Results. In one of the earliest theoretical works on evolutionary algorithms for combinatorial optimization problems, Neumann and Wegener [NW04, NW07] analyze the optimization time of the (1+1) EA for the MST problem. They prove that thisalgorithm needs, on average, $O(m^2 \log(nw_{\max}))$ function evaluation to find one. Here n is the number of vertices, m the number of edges and $w_{\max}$ is the maximum of the positive and integral edge weights. It is a major open problem whether the dependence on the maximum edge weight is necessary.

The same bound is proven for a Randomized Local Search (RLS) variant doing one-bit and two-bit flips each with probability 1/2. This can be easily improved to $O(m^2 \log n)$ by noting that the optimization behavior remains exactly the same if we replace the existing edge weights by the numbers from 1 to m (keeping the relative order of the edge weights unchanged) [RS09].

Our Results. A summary of our results can be found in Table 9.1.

	(rb) unrestricted	$*$-ary unbiased	unary unbiased
upper bound	$2m+1$	$O(m)$	$O(mn \log n)^\dagger$
lower bound	$(1-o(1))n$	$\Omega(n)$	$\Omega(m \log n)$

	rb unary unbiased	(rb) binary unbiased	(rb) 3-ary unbiased
upper bound	$O(mn \log n)$	$O(m \log n)$	$O(m)$
lower bound	$\Omega(m \log n)$	$\Omega(m/ \log n)$	$\Omega(m/ \log n)$

Table 9.1. Upper and lower bounds for the black-box complexity of MST in the different models. Abbreviations: rb = ranking-based.
† $O(mn \log(m/n))$ if all edge weights are distinct.

In a nutshell, the results show that, on the one hand, simple algorithms based on unary operators, such as evolutionary algorithms and RLS can get runtimes very close to the theoretically optimal; on the other hand, they show how operators of higher arity can further improve on the runtime. In particular we show that the basic and the ranking-based unbiased black-box complexities are asymptotically different for the unary case and for arities greater than or equal to three. This is the first result showing that the complexity of a combinatorial problem depends on the arity of the black-box model.

The results also indicate that the ranking-based black-box complexities may be larger than their non-ranking-based counterparts if we rule out multiple edge weights. However, both for the MST problem as well as the single-source shortest paths problem, which we analyze in Section 10, the differences are rather small. This stems from the nature of the two problems: for both problems the best known algorithms builds a solution in a sequential manner, each time testing to add the least costly edge.

Nevertheless we expect the ranking-based notion to make a decisive difference for other combinatorial problems, e.g., the partition problem (cf. Section 6), which we leave for future research.

Conventions. For this thesis, a graph G is a triple (V, E, w), where V is a set of vertices, E a set of edges (an edge is a set of two vertices) and $w : E \to \mathbb{R}$ a function assigning to each edge $e \in E$ a positive real number $w(e)$, called the *weight* of e.

Precise Problem Definition.

Definition 9.1 (MST). *The **Minimum Spanning Tree** (MST) problem consists of a connected graph $G = (V, E, w)$ on $n := |V|$ vertices and $m := |E|$ weighted edges. The objective is to find an edge set $E' \subseteq E$ of minimal weight that connects all vertices.*

We encode this problem in binary representation as follows. First, we enumerate the edges in E in arbitrary order $\nu : E \to [m]$. For every bit string $x \in \{0,1\}^m$ we then interpret x as the subset of edges $E_x := \{\nu^{-1}(i) \in E \mid x_i = 1\}$. In the following, we assume that the enumeration of the edges is *not known* to the algorithm. So the algorithm only knows the numbers n and m, but neither knows the geometry of the graph nor which bit corresponds to which edge. However, it may assume that the graph is connected since otherwise no minimum spanning tree for this graph exists.

For $E' \subseteq E$ let $c(E')$ be the number of connected components induced by E', and let $w(E') = \sum_{e \in E'} w(e)$ be the total weight of E'. The book [NW10] argues that the objective function $f(E') = (c(E'), w(E'))$ is "appropriate in the black-box scenario". Thus, in our model, if an algorithm *queries* some $x \in \{0,1\}^m$, it receives $f(E_x) = (c(E_x), w(E_x))$ as answer. In what follows, we shall also use the notation $f(x) = (c(x), w(x))$ instead of $(c(E_x), w(E_x))$.

In the ranking-based models, the objective value consists of a ranking of both components. That is, the oracle reveals two rankings of the search points where one ranking is induced by the number of connected components, and the second ranking is induced by the total edge weights.

9.2. Upper Bounds for the MST Problem

We obtain the following upper bounds by modifying Kruskal's algorithm to fit the black-box setting at hand.

Theorem 9.2 (Upper bounds for MST). *The **(ranking-based) unrestricted** black-box complexity of the MST problem is at most $2m + 1$. The **unary unbiased** black-box complexity is $O(mn \log(m/n))$ if there are no duplicate weights and $O(mn \log n)$ if there are. The **ranking-based unary unbiased** black-box complexity is $O(mn \log n)$.The **(ranking-based) binary unbiased** black box-complexity is $O(m \log n)$. The **(ranking-based) 3-ary unbiased** black-box complexity is $O(m)$.*

In the following, we give the proofs of Theorem 9.2. In Section 9.2.1 we consider the unrestricted black-box model and in Section 9.2.2 the unbiased one for the different arities.

Finally, in Section 9.3 we give lower bounds for the black-box complexity of MST in both the unrestricted and the unbiased black-box models.

9.2.1. The Unrestricted Setting

We start with the statement in the unrestricted setting, which is very simple.

Proof of the $2m+1$ upper bound. Query the empty graph $(0,\dots,0)$ as a reference point, then query all edges e_i^m, $i \in [m]$, where we recall that by e_i^m we denote the i-th unit vector $(0,\dots,0,1,0,\dots,0)$ of length m. Then test all edges in increasing order of their weights (ties broken arbitrarily). Accept an edge if it does not form a cycle (note that this can be checked trough the first component of the bi-objective objective function). This shows how to run Kruskal's algorithm after an initial number of $m+1$ reference queries. □

9.2.2. The Unbiased Settings

The unbiased model is more involved. For all three arities, the basic principle of the algorithm is the same as for the unrestricted algorithm, basically following Kruskal's algorithm for the MST construction. In the *first step*, we create the empty graph. This serves as a reference point for all further iterations.

In the *second step*, we learn the weights of the edges (in the unbiased model) and we learn the ranking/order of the edge weights in the ranking-based model. For both models we need to learn the multiplicities of the edge weights as well.

Finally, in the *third step* we test the inclusion of the edges in increasing order of their weights. For the success of Kruskal's algorithm it does not matter in which order we test edges of the same weight.

In the following, we prove upper bounds for the expected number of queries needed to complete each step. For readability purposes, we split the proof into 4 parts, one for each model.

Let us remark already here that in the proof we apply only few variation operators, namely `uniform`() which samples a bit string $x \in \{0,1\}^m$ uniformly at random, `RLS`$(\cdot)$ (random local search) which, given some $x \in \{0,1\}^m$, creates from x a new bit string $y \in \{0,1\}^m$ by flipping exactly one bit in x, the bit position being chosen uniformly at random. We also use the operator `complement`$(\cdot)$, which assigns to every $x \in \{0,1\}^m$ its bitwise complement $\bar{x}$. Lastly, we use the operator `RLS`$_k(\cdot,\cdot)$. Given some bit stings $x, y \in \{0,1\}^m$, `RLS`$_k(x,y)$ outputs a bit string z that has been created from x by flipping exactly k bits of x, chosen uniformly at random from the set of positions in which x and y differ. If x and y differ in less than k bits, it outputs x.

For the former three variation operators we have shown in previous sections that they are unbiased. For the latter one, this can be easily verified by the fact that it is unbiased if the Hamming distance of x and y is at least k (this has been shown in Section 8.2) and by the fact that returning one of the two arguments is also an unbiased operation.

The Unary Unbiased Model

The bound of $O(mn \log m)$ for multiple edge weights will follow from the bound for the ranking-based unary unbiased model, so we give here only the proof for the $O(mn \log(m/n))$ bound.

Proof of the $O(mn\log(m/n))$ bound.

First step. As required by the unbiased black-box model, we first draw a search point $x \in \{0,1\}^m$ uniformly at random. We construct the empty graph by creating $y \leftarrow \texttt{RLS}(x)$ and accepting $x \leftarrow y$ if and only if $w(y) < w(x)$. We do so until $w(x) = 0$. By the standard coupon collector argument, this takes an expected $O(m \log m)$ queries. In the following, we denote the empty graph by x^0.

Second step. In order to learn the weights, we again employ the operator $\texttt{RLS}(\cdot)$ iteratively to x^0 until we have added all edges. More precisely, we generate a sequence of search points x^k as follows. In the k-th iteration of the second step we create $z \leftarrow \texttt{RLS}(x^{k-1})$ and query the objective value $f(z) = (c(z), w(z))$ of z. If $w(z)$ is larger than $w(x^{k-1})$, we set $x^k \leftarrow z$. Otherwise, we discard z and keep on sampling from x^{k-1}. Then the difference of the objective values of x^k and x^{k-1} is exactly the weight of the edge added in the k-th iteration, so we learn all edge weights. By the same coupon collector argument as before, this takes an expected $O(m \log m)$ queries. Let $w_1 \leq \ldots \leq w_m$ be the ordering of the edge weights.

Third step. We return to the search point x^0, and show how to construct an MST for the underlying graph. For any $k \leq n-1$ we call the queries needed to add the k-th edge to the MST the *k-th phase*. We start by applying $\texttt{RLS}(\cdot)$ to x^0 until we have found an edge of minimal weight w_1. We call the latter sample y^1.

Assume now that we have already added i edges of the MST. Let y^i be this search point and let us assume that we have tested the inclusion of the t_i "cheapest" edges to find the i-th one. To test the inclusion of the (t_i+1)st edge, we query $z^{(i,t)} \leftarrow \texttt{RLS}(y^i)$. If $w(z^{(i,t)}) < w(y^i)$, we discard $z^{(i,t)}$. Otherwise, it holds that the weight of the flipped edge equals $w(z^{(i,t)}) - w(y^i)$. By the first value $c(z^{(i,t)})$ we learn whether we can add this edge without creating a cycle (if and only if $c(z^{(i,t)}) < c(y^i)$).

We do so until we have flipped the (t_i+1)st heaviest edge . This requires an expected number of m queries. If we cannot add this edge to the current solution without creating a cycle, we check whether we have already created in one of the $z^{(i,t)}$s the string which includes the (t_i+2)nd heaviest edge. If so, we check whether or not to add it to the current solution. If we have not created it already, we continue drawing $z^{(i,t)} \leftarrow \texttt{RLS}(y^i)$ until we have found the edge with the lowest weight that can be included to our current solution y^i. We call the new solution y^{i+1} and continue with the $(i+2)$nd phase until we have added a total number of $n-1$ edges to x^0.

To determine an upper bound for the number of queries needed, let $k_i := t_i - t_{i-1}$ be the number of edges for which we have tested the inclusion in the i-th phase (including the lastly included one). By a coupon collector argument the expected number of queries needed in the i-th phase is $m/k_i \cdot k_i \log k_i = m \log k_i$. This follows from the fact that we need to sample all k_i edges ("coupons") but the chance of getting one of them equals only m/k_i, for each query. Note that this argument works only in the case when all edge weights are distinct. Otherwise, we would need to sample more often to be sure that we have seen all edges of the given weight.

This shows that the third phase of the algorithm takes no more than

$O(m\sum_{i=1}^{n-1}\log k_i)$ queries. Since $\sum_{i=1}^{n-1} k_i \le m$ we conclude that

$$\sum_{i=1}^{n-1} \log k_i = \log\Big(\prod_{i=1}^{n-1} k_i\Big) \le \log\big((m/n)^n\big) = n\log(m/n)\,,$$

and thus, $O(m\sum_{i=1}^{n-1}\log k_i) = O(mn\log(m/n))$. Hence, the unary unbiased black-box complexity of MST is

$$O(m\log m) + O(m\log m) + O(mn\log(m/n)) = O(mn\log(m/n))\,.$$

□

The Ranking-Based Unary Unbiased Model

Let us now consider the ranking-based setting. The proof of the $O(mn\log n)$ bound given below carries over to the basic (i.e., the non-ranking-based) unary unbiased setting with duplicate weights.

In the following, we speak of *weights*, even if we are in the ranking-based black-box model. Note however, that we do not need to know the exact value of the weight but only its *rank*.

Proof of the $O(mn\log n)$ bound.

First step. The first step is exactly the same as in the unary unbiased model. Note that, thanks to the information given by the ranking, we are still able to determine if $w(y) < w(x)$ holds.

Second step. For this model, we can skip the second step.

Third step. The difference for the ranking-based unbiased model compared to the unbiased one is quite obvious. Since we cannot query for the objective value $(c(x), w(x))$ but only the relative ranks of $c(x)$ and $w(x)$, we do not know which bits we have flipped in either one of the iterations. Thus, for the inclusion of the $(i+1)$st edge, we perform $O(m\log m)$ queries $z^{(i,t)} \leftarrow \texttt{RLS}(y^i)$ to find, with high probability, the edge with the smallest weight that can be included into the current solution y^i. The high probability statement is again the basic coupon collector's argument. Note that we can check the ranking of the weights via the second component of the objective function and the feasibility of adding it to the current solution via the first component. If and only if the rank of $c(z^{(i,t)})$ is strictly less than the one of $c(y^i)$, we can include the corresponding edge. Since we need to include $n-1$ edges into the MST, we need an expected $O(nm\log m)$ queries until all edges of the MST have been added. □

The Ranking-Based Binary Unbiased Model

Compared to the unary case, in the binary unbiased black-box model we can gain information about the Hamming distance of two search points. This allows more powerful algorithms.

Proof of the $O(m\log n)$ bound.

First step. As required by the unbiased black-box model, we first draw a search point $x \in \{0,1\}^m$ uniformly at random.

We can create the empty graph in an expected number of $2m$ queries as follows. Set $y \leftarrow \texttt{complement}(x)$. We then set $z \leftarrow \texttt{RLS}_1(x, y)$ with probability $1/2$ and $z \leftarrow \texttt{RLS}_1(y, x)$ with probability $1/2$. We update $x \leftarrow z$ (in the first case) and $y \leftarrow z$ (in the second case) if and only if $w(z) < w(x)$ or $w(z) < w(y)$, respectively.

With probability $1/2$, this operation decreases the Hamming distance of x and y by 1 and thus, the expected number of calls to $\texttt{RLS}_1(\cdot, \cdot)$ is $2m$; just note that initially x and y had a Hamming distance of m (compare Lemma 8.4 from Section 8.2.1 for a rigorous proof of this claim). Note that choosing $\texttt{RLS}_1(x, y)$ with probability $1/2$ and $\texttt{RLS}_1(y, x)$ with probability $1/2$ is still an unbiased operation. As before, let x^0 denote the empty graph.

Second step. As we shall see in the following, the binary model allows us to learn all m edge weights with multiplicities in $O(m \log m)$ queries. The key idea is again to sample $O(m \log m)$ times from x^0 a bit string $z \leftarrow \texttt{RLS}(x^0)$ that differs from x^0 in exactly one bit. By the coupon collector's argument, with high probability, these samples suffice to have flipped each bit at least once. The main difference to the unary model is the fact that the binary model allows us to store which edge weights have been "learned" already.

To learn the first weight, we query $z \leftarrow \texttt{RLS}(x^0)$. Since x^0 is the empty graph, we have that the weight of the edge corresponding to the flipped position is $w(z)$. Now we initialize $y \leftarrow z$. Throughout the run of the algorithm we shall have $y_i = x^0_i$ if and only if the weight of the i-th edge $\nu^{-1}(i)$ is not yet known. This is certainly true after initialization of y.

Assume now that we have learned already $k \geq 1$ edge weights, i.e., x^0 and y differ in exactly k bits. We again query a bit string $z \leftarrow \texttt{RLS}(x^0)$ that differs from x^0 in exactly one bit. Note that this is a new weight if and only if the Hamming distance of z to y is $k + 1$. As mentioned above, we can test the Hamming distance using a binary unbiased operator.

More precisely, for all $\ell \in \mathbb{N}$, let $\texttt{dist}_\ell(\cdot, \cdot)$ be the operator which, given a pair $(x', y') \in \{0,1\}^m \times \{0,1\}^m$ of bit strings, returns x' if the Hamming distance $\sum_{i=1}^m |x'_i - y'_i|$ equals ℓ and returns a 1-Hamming neighbor of x' otherwise. It is straightforward to verify that this operator $\texttt{dist}_\ell(\cdot, \cdot)$ is a binary unbiased one and from it we learn in one single query whether the two search points at hand are at Hamming distance ℓ from each other. The latter is true since all edge weights are positive, and hence for all 1-Hamming neighbors z' of x' we have $w(z') \neq w(x')$.

To test whether the Hamming distance of z to y is $k + 1$, we query $\texttt{dist}_{k+1}(y, z)$. If $w(\texttt{dist}_{k+1}(y, z)) \neq w(y)$ we discard the current search point z (we have learned the weight of the edge in E_z already). Otherwise, i.e., if $w(\texttt{dist}_{k+1}(y, z)) = w(y)$, we need to update y. This can be done in $O(\log m)$ queries as follows.

For simplicity, assume for the moment that $k + 1$ is even. Furthermore, let us denote by i the bit position in which z and x^0 differ. For updating y we need to create in $O(\log m)$ queries a search point y' which equals y in all bits but the i-th one. To this end, we create a new search point z' from y and z by flipping exactly half of the bits in which y and z differ. More precisely, let $z' \leftarrow \texttt{RLS}_{(k+1)/2}(y, z)$. By computing the Hamming distance between z' and x^0, we can decide whether $z'_i = z_i$. Indeed the Hamming distance of z' and x^0 equals $k - (k + 1)/2 = (k - 1)/2$ if $z'_i \neq z_i$ and it equals $k - ((k + 1)/2 - 1) + 1 = (k + 3)/2$ otherwise. We update $z \leftarrow z'$ if $z'_i = z_i$, i.e., if $w(\mathtt{dist}_{(k+3)/2}(z', x^0)) = w(z')$ and we do nothing otherwise. Then we proceed by flipping again half of the bits in which z and y differ, updating z whenever $z'_i = z_i$ in the new sample z'.

If $k+1$ is odd, we flip $\lfloor (k+1)/2 \rfloor$ bits in which y and z differ and, by a similar reasoning as above, we replace z with z' if the Hamming distance of x^0 and z' equals $k - (\lfloor (k+1)/2 \rfloor - 1) + 1$ and we discard z' otherwise.

Repeating like this, we have in each step a probability of 1/2 to reduce the Hamming distance between y and z by half and we find $z = y'$ after $O(\log m)$ queries.

Since we need to learn m weights in total, we need to do $O(m \log m)$ 1-bit flips $\mathtt{RLS}(x^0)$ and for m weights in total we need to run the update procedure for y, requiring $O(\log m)$ queries each. Therefore, we can learn all weights with multiplicities in time $O(m \log m)$.

After having learned the different weights, we can fix some enumeration σ of the the edges $e_{\sigma(1)}, \ldots, e_{\sigma(m)}$ such that their weights are ordered $w_1 = w(e_{\sigma(1)}) \leq \ldots \leq w_m = w(e_{\sigma(m)})$. Note that for every i, we have already sampled a search point $y^{\sigma(i)}$ with $E_{y^{\sigma(i)}} = \{e_{\sigma(i)}\}$. In what follows, we say that $y^{\sigma(i)}$ *contains* only the edge $e_{\sigma(i)}$.

Third step. As in the unary model, we successively add edges to our current solution. This time, we call the queries needed to *test* the inclusion of the i-th heaviest edge $e_{\sigma(i)}$, given that we have tested already the inclusion of edge $e_{\sigma(i-1)}$, the *i-th phase*.

We show that such a phase requires at most $O(\log n)$ queries. The key idea is essentially the same as the one in the second phase. Let x be the current search point and let $k := |\{E_x\}|$, the number of edges already included in the current solution. Note that we know k as we are starting with the empty graph and we are adding edges one by one. The Hamming distance from x to the search point $y^{\sigma(i)}$ is $k+1$ as the search points differ in the k edges included in x as well as in the bit position $\nu(e_{\sigma(i)})$.

Now set $z \leftarrow \mathtt{RLS}_{\lfloor (k+1)/2 \rfloor}(x, y^{\sigma(i)})$ and query its objective value $(c(z), w(z))$. As above, by computing the Hamming distance between z and x^0, we may decide whether $e_{\sigma(i)} \in E_z$ or not. Namely, the Hamming distance of x^0 and x is k and, thus, the distance of x^0 and z equals $k - \lfloor (k+1)/2 \rfloor$ if and only if $e_{\sigma(i)} \notin E_z$ and the distance of x^0 and z equals $k - (\lfloor (k+1)/2 \rfloor - 1) + 1 = k - \lfloor (k+1)/2 \rfloor + 2$ otherwise. If $e_{\sigma(i)} \notin E_z$ we discard z and try again. Otherwise, we replace $y^{\sigma(i)}$ with z and again flip half of the bits in which z and x differ, updating z whenever $e_{\sigma(i)}$ is contained in the new sample. Formally, we query $z' \leftarrow \mathtt{dist}_{k - \lfloor (k+1)/2 \rfloor + 2}(x^0, z)$ and we replace $y^{\sigma(i)}$ with z if and only if $w(z') = w(z)$.

Repeating like this, we have in each step a probability of 1/2 to reduce the Hamming distance between z and x by half. Meanwhile, by comparing with x^0 we ensure that $e_{\sigma(i)}$ is always contained in E_z. Therefore, when the Hamming distance of x and z decreases to 1, the search point z differs from x only by the edge $e_{\sigma(i)}$, as desired. Once given these two search points we decide whether or not to include the i-th heaviest edge $e_{\sigma(i)}$. As argued above, we include edge $e_{\sigma(i)}$ if and only if by its inclusion we do not form a cycle, i.e., we include $e_{\sigma(i)}$ if and only if $c(z) < c(x)$. Clearly we have $k < n-1$ and thus, we need $O(\log n)$ queries on average for the i-th phase. In the worst-case we need to check the inclusion of each of the m edges. Hence, the third step requires an expected number of $O(m \log n)$ queries. □

The Ranking-Based 3-Ary Unbiased Model

In the 3-ary model, we have even more flexibility. The main advantage is that we can create any particular 1-bit flip in a linear number of queries (linear in the length of the bit string). Using this, we can optimize the MST problem in a linear number of queries. Again, we use the word "weight" but are aware that we are not given the exact fitness values but only the ranks of the search points.

Proof of the $O(m)$ bound.

First step. The bound for the binary model holds for the 3-ary one as well, and thus, $O(m)$ is an upper bound for the first step in the 3-ary model, too. Let us again denote the empty graph by x^0.

Second step. We show how to learn the weights of the edges in $O(m)$ queries. To encode which edges have been looked at already, we initialize $y \leftarrow \texttt{complement}(x^0)$, the bitwise complement of x^0. Throughout the run of the algorithm it will hold that the edges we have looked at correspond to the bit positions in which x and y coincide.

We learn the first edge weight by querying $z \leftarrow \texttt{RLS}_1(x^0, y)$. We update $y \leftarrow \texttt{Update}(y, z, x^0)$, where $\texttt{Update}(\cdot,\cdot,\cdot)$ is the 3-ary variation operator that can be described as follows. Given $a, b, c \in \{0,1\}^m$, the operator $\texttt{Update}(a, b, c)$ returns c for those positions where a and b coincide and it returns a otherwise. Formally, for all $i \in [m]$ we have $\texttt{Update}(a, b, c)_i := c_i$ if $a_i = b_i$ and we have $\texttt{Update}(a, b, c)_i := a_i$ if $a_i \neq b_i$. Let us briefly remark that the operator $\texttt{Update}(\cdot,\cdot,\cdot)$ is unbiased. This is easily verified using the fact that for all $a, b, d \in \{0,1\}^m$ we have $a_i = b_i$ if and only if $(a \oplus d)_i = (b \oplus d)_i$ and for all $\sigma \in S_m$ we have $a_i = b_i$ if and only if $\sigma(a)_{\sigma^{-1}(i)} = \sigma(b)_{\sigma^{-1}(i)}$.

After having updated $y \leftarrow \texttt{Update}(y, z, x^0)$ we proceed by querying $z \leftarrow \texttt{RLS}_1(x^0, y)$ and updating $y \leftarrow \texttt{Update}(y, z, x^0)$ until we find $x^0 = y$. It is easily verified that this occurs after m such iterations as the Hamming distance of y and x^0 decreases by 1 in each iteration.

Furthermore, we have created all possible 1-bit flips, after only m such iterations (i.e., after $2m$ queries as each step requires two queries). Let us again fix an ordering of the edges $e_{\sigma(1)}, \ldots, e_{\sigma(m)}$ such that $w_1 = w(e_{\sigma(1)}) \leq \ldots \leq w_m = w(e_{\sigma(m)})$. For every $i \in [m]$ let $z_{\sigma(i)} \in \{0,1\}^m$ be the query corresponding to $e_{\sigma(i)}$. That is $z_{\sigma(i)}$ was obtained from x^0 by flipping exactly one bit in which x^0 and $w(z_{\sigma(i)}) = w_i$, the i-th lightest edge weight.

Third step. We now check the inclusion of the edges to the current solution in increasing order of their weights. Since $e_{\sigma(1)}$ can be included without any further consideration, we may assume that we have already tested the inclusion of the i edges $e_{\sigma(1)}, \ldots, e_{\sigma(i)}$ with the lowest weights. Let x be our current solution. To test the inclusion of edge $e_{\sigma(i+1)}$ into the current solution, we query $z \leftarrow \texttt{test}(x, x^0, z_{\sigma(i+1)})$, where $\texttt{test}(\cdot,\cdot,\cdot)$ is again an unbiased 3-ary variation operator that can be described as follows. For any $a, b, c \in \{0,1\}^m$ the operator $\texttt{test}(a, b, c)$ outputs a string that has entries equal to a in all positions for which b and c coincide and entries equal to $1 - a$ otherwise. This is again unbiased by the same reasons as given above.

Note that in our case, we clearly have that the strings x and z differ in exactly one bit position (because x^0 and $z_{\sigma(i+1)}$ do). We update $x \leftarrow z$ if the edge $e_{\sigma(i+1)}$ can be included into the current solution (if and only if $c(z) < c(x)$). We then continue with testing the inclusion of edge $e_{\sigma(i+2)}$.

As this third phase requires at most m queries, the ranking-based unbiased 3-ary black-box complexity of MST can be bounded by $O(m)$. □

9.3. Lower Bounds for the MST problem

Theorem 9.3. *The unrestricted black-box complexity of MST for complete graphs is at least* $(1 - o(1))n$.

Proof. We apply Yao's minimax principle, Theorem 2.16. To this end, we show that there exists a probability distribution over the input set of all weighted complete graphs such that every deterministic algorithm needs at least $(1 - o(1))n$ queries to compute a MST. More precisely, we consider the distribution p over the set of all inputs where we sample uniformly at random a spanning tree, and give weight 1 to all of its edges. All other edges receive weight 2. We call edges of weight 1 "cheap", and we call all other edges "expensive".

Let us now consider a fixed deterministic algorithm A. We assume that the algorithm already knows which bit in the vector corresponds to which edge in the graph. This assumption makes life only easier for the algorithm. Then for each query the algorithm knows in advance how many connected component its query has. So the first component of the objective function does not contain any new information for the algorithm. If the algorithm makes a query consisting of k edges, then the total weight of all these edges is contained in the interval $[2k - n + 1, 2k]$. This is due to the fact that any query contains at most $n - 1$ cheap edges. Therefore, each query gives at most $\log_2(n)$ bits of information.

Obviously, the algorithm A needs to learn the set of all cheap edges. It is well known that the number of spanning trees on n vertices is n^{n-2} (this is the so-called Cayley's formula). Therefore A needs to learn $(n - 2)\log_2(n)$ many bits, so it has in the worst case a runtime of $T := n - 2$. Moreover, for every $0 \le t \le T$, after $T - t$ many queries the probability to find the correct solution is at most n^{-t}. Therefore, the probability that A needs at least T steps is bounded from below by

$$\Pr[T(I_p, A) \ge T] \ge 1 - \sum_{t=1}^{T-1} n^{-t} \ge 1 - \frac{n^{-1}}{1 - n^{-1}} = 1 - o(1).$$

Note that the hidden constant in $o(1)$ does not depend on the algorithm A. By Markov's inequality, the expected runtime is at least

$$\begin{aligned} \mathrm{E}[T(I_p, A)] &\ge T \cdot \Pr[T(I_p, A) \ge T] \\ &\ge T \cdot (1 - o(1)) \\ &= (1 - o(1))n. \end{aligned}$$

Since this holds for all deterministic algorithms A, Yao's minimax principle implies the statement. □

In order to prove a lower bound in the unbiased setting, we compare MST with the OneMax-problem. Since our search space is $\{0,1\}^m$, we slightly deviate from the previous notation and we write $\textsc{Om}_m$ for the function that assigns each vector $x \in \{0,1\}^m$ the objective value $\textsc{Om}_m(x) := \sum_{i=1}^m x_i$, the number of 1-bits in x.

Theorem 9.4. *The k-ary unbiased black-box complexity of MST for m edges is at least as large as the k-ary unbiased black-box complexity of $\textsc{Om}_m$.*

Proof. For a given m, consider a path P on $m+1$ vertices, all m edges having unit weight. For the associated MST fitness function f we have, for all bit strings $x \in \{0,1\}^m$,

$$f(x) = (\textsc{Om}_m(x), m+1-\textsc{Om}_m(x)).$$

In particular, any algorithm optimizing f can be used to optimize $\textsc{Om}_m$ with the exact same number of queries. □

As mentioned several times already, $\textsc{OneMax}_m$ has a unary unbiased complexity of $\Theta(m \log m) = \Theta(m \log n)$ [LW10a, Theorem 6] and already Erdős and Rényi [ER63] showed that the unrestricted black-box complexity of $\textsc{OneMax}_m$ is $\Theta(m/\log m)$. Since the unbiased black-box complexity is bounded from below by the unrestricted black-box complexity for all problems, these two results imply the following.

Corollary 9.5. *The unary unbiased black-box complexity of MST is in $\Omega(m \log n)$; for all other arities, the unbiased black-box complexity of MST is in $\Omega(m/\log n)$.*

10 The Single-Source Shortest Paths Problem

In this section we analyze the black-box complexity of the single-source shortest paths problem (SSSP). We present bounds for this problem in several black-box models.

Our work reveals that the choice of how to model the optimization problem is nontrivial here. This in particular comes true for the SSSP problem as this problem is typically not modeled via bit strings. Therefore, a reasonable definition of unbiasedness has to be agreed on. As we shall see, transforming the definition from Lehre and Witt [LW10a] in a straightforward way leads to not very useful results. Taking the problem semantics into account, we find a reasonable definition for unbiasedness and prove meaningful black-box complexities.

As in the previous section, the results presented here are based on the conference publication [DKLW11]. They are joint work with Benjamin Doerr, Timo Kötzing, and Johannes Lengler.

10.1. Introduction

Even before the results of Neumann and Wegener for the minimum spanning tree problems, in another one of the earliest theoretical works on evolutionary algorithms for combinatorial optimization problems, Scharnow, Tinnefeld, and Wegener [STW02, STW04] analyze how a (1+1) EA-type algorithm solves the SSSP problem. In what follows, we briefly sketch how Scharnow et al. modeled the SSSP problem.

Since in the SSSP problem a shortest path between the source and any other vertex is sought for, a bit string representation for solution candidates seems not very natural. Therefore, most works resort to trees or slightly more general structures as representations. To ease the comparison with most existing works on the SSSP problem, in this work we shall only work with the vertex-based representation employed in [STW04], which, roughly speaking, for each vertex stores its predecessor on the path from the source to it. We note that superior runtimes were recently proven for an edge-based approach [DJ10].

The Multi-Criteria Fitness Function. In addition, also the choice of the fitness function is subtle. In [STW04], a *multi-criteria fitness* was suggested. For each vertex, the objective function returns the distance from the source in the current solution (infinity, if the vertex is not connected to the source). An offspring is only accepted if, in each of these $n-1$ criteria, it is not worse than the parent. For the natural (1+1) EA building on this framework, they prove an expected optimization time of $O(n^3)$. Doerr, Happ, and Klein improved this to a bound of $O(n^2 \max\{\ell, \log(n)\})$, where ℓ is the smallest height of a shortest paths tree [DHK07].

When analyzing the black-box complexity of this formulation of the SSSP problem, we first note that both unbiased and ranking-based complexities make little sense. Since the multi-objective fitness explicitly distinguishes the vertices, treating vertices equally here (as done by unbiased operators) or making individual distances incomparable (as done by component-wise ranking) is ill-natured.

Hence, for the multi-criteria fitness, we shall only regard the unrestricted black-box complexity. Interestingly, this problem is also among the few combinatorial problems for which black-box complexity results exist. Droste, Jansen, (Tinnefeld,) and Wegener [DJTW03, DJW06] showed that the unrestricted black-box complexity of the SSSP in the multi-criteria formulation is at least $n/2$ and at most $2n-3$. We first improve these bounds to exactly $n-1$ for both the upper and the lower bound.[1] Surprisingly, if we may assume that the input graph is a complete graph, we obtain a black-box complexity of at most $n/2 + O(1)$, see Table 10.2 in Section 10.2. That is, the SSSP problem becomes easier (in the black-box complexity sense) if we transform an arbitrary instance to one on a complete graph (but adding expensive dummy edges).

The Single-Criterion Fitness Function. The natural *single-criterion* formulation of the SSSP problem takes as objective the sum of the distances of all vertices to the source. This approach was dismissed in [STW04] for the reason that then all solutions with at least one vertex not connected to the source form a huge plateau of equal fitness.

In [BBD$^+$09], it was observed that this artificial problem dissolves if each unconnected vertex only contributes a large value (e.g., larger than the sum of all edge weights) to the objective value. This is the common way to implement the infinity value in most algorithms. In this setting, also the single-criterion EA is efficient and finds the optimum, on average, in $O(n^3 \log(nw_{\max}))$ iterations [BBD$^+$09].

For the single-criterion version of the SSSP problem, there is no reason not to regard unbiased black-box complexities. However, we shall see that finding a good notion for unbiasedness is a crucial point here. We discuss three different notions of unbiasedness. Whereas all three notions a priori seem to capture different aspects of what unbiasedness in the SSSP problem could mean, we show that two of these models are too powerful. In fact, already the unary version of both the *structure preserving* unbiased model, in which, intuitively speaking, all graphs with the same structure but different node labels are equally likely to be chosen (cf. Definition 10.7) as well as the

[1]Note that the upper bound in [DJW06] still holds in a more restricted setting, cf. Section 10.2.

	unrestricted	rb unrestricted	rb unary redirecting	binary redir. unb.
upper bound	$n(n-1)/2$	$(n-1)^2$	$O(n^3)$	$O(n^2 \log n)$

Table 10.1. Upper bounds for the black-box complexity of SSSP with single-criteria fitness function in the different models. A lower bound of $\Omega(n^2)$ for the redirecting unbiased black-box complexity is shown in Theorem 10.15. Abbreviations: rb = ranking-based, redir. unb.= redirecting unbiased.

generalized unbiased model as proposed by Rowe and Vose [RV11] yield almost the same black-box complexities as the unrestricted black-box model. As we shall prove in Sections 10.3.2 and 10.3.5, these three black-box complexity notions differ by at most one query. Hence, we feel that neither the structure preserving nor the generalized unbiased model sufficiently capture what "unbiasedness" in the SSSP problem should mean. We find this surprising as both model seem to be a very natural extension of the unbiasedness notion of Lehre and Witt [LW10a].

We suggest a third model, the *redirecting* unbiased black-box model in which, intuitively, a node may choose to change its predecessor in the shortest paths tree, but if it decides to do so, then all possible predecessors must be equally likely to be chosen. We show that this model indeed yields more meaningful black-box complexities. Table 10.1 gives a brief summary of our main findings. More results are given in Section 10.2.

Precise Problem Definition.

Definition 10.1 (SSSP). *The **Single-Source Shortest Paths** (SSSP) problem consists of a connected graph $G = (V, E, w)$ on $n := |V|$ vertices and $m := |E|$ edges. There is a distinguished **source vertex** $s \in V$. The objective is to find, for all vertices $v \in V \backslash \{s\}$, a path p_v in G from s to v such that the total weight of p_v, $\sum_{e \in p_v} w(e)$, is minimal among all paths from s to v.*

It is well-known that the set of all edges that are used on a shortest path from s to some vertex v forms a tree. Thus, implicitly, SSSP is the problem of finding a minimum cost tree rooted in s.

Conventions. In what follows, we always assume all input graphs to be connected. Without loss of generality we assume that the nodes are labeled by $1, \dots, n$ and that $s = 1$ is the source for which we need to compute the shortest paths tree.

10.2. SSSP with a Multi-Criteria Fitness Function

The paper [DJW06] argues for a multi-criteria objective function, where any algorithm may query arbitrary trees on $[n]$ and the objective value of any such tree is an $n-1$ tuple of the distances of the $n-1$ non-source vertices to the source $s = 1$ (if an edge is traversed which does not exist in the input graph, the entry of the tuple is ∞).

From [DJW06] we know that the unrestricted black-box complexity of this problem is lower bounded by $n/2$ and upper bounded by $2n - 3$.[2]

[2]Note that the upper bound given in [DJW06] was shown to hold in the 2-memory-restricted

In this section, we first improve both the lower and the upper bound from [DJW06] matching them at $n-1$ exactly. Then we restrict the problem instances to complete graphs, which will avoid objective values of ∞ for the different objectives; for this setting, there are more efficient algorithms available than in the setting allowing incomplete graphs.

The following table summarizes our results for these settings.

	arbitrary connected graph	complete graph
upper	$n-1$	$\lfloor (n+1)/2 \rfloor + 1$
lower	$n-1$	$n/4$

Table 10.2. Upper and lower bounds for the unrestricted black-box complexity of the SSSP problem with multi-criteria objective function.

Theorem 10.2. *The unrestricted black-box complexity of SSSP with arbitrary input graphs is* $n-1$.

Proof. We start with the **upper bound**. We simulate Dijkstra's algorithm by first connecting all vertices to the source, then all but one vertices to the vertex of lowest distance to the source, then all but the two of lowest distance to the vertex of second lowest distance and so on, fixing one vertex with each query. This will cost an overall of $n-1$ queries.

For the **lower bound** consider the set S of all graphs on $\{1,\dots,n\}$ which contain exactly one path as edges (all of weight 1), one of the endpoints being the source $s=1$, and no other edges. We apply Yao's minimax principle, Theorem 2.16. To this end, let a deterministic algorithm A be given; we show that A uses in expectation $n-1$ queries on a graph drawn uniformly at random from S. We do this by showing that, in expectation, each query will give at most one new non-∞ entry in the tuple. To this we shall apply the additive drift theorem for lower bounds, Theorem 2.17.

We can assume without loss of generality that, during the run of algorithm A, the number of finite entries in the objective value does never decrease. Suppose that, after some queries, A has determined $n-1-k$ finite entries in the objective value, so for k vertices A has still not discovered a path to the source. Let T be the next query of A. Let U be the set of all vertices that A connects, in T, with a vertex with finite distance to the source. For all $v \in U$, let $t(v)$ be the number of vertices that A connects to the source *via* v. The expected number of *new* non-∞ entries in the objective value of T is at most

$$\sum_{v \in U} \frac{1+t(v)}{k}, \tag{10.1}$$

as $1/k$ is the probability that a given $v \in U$ is the next vertex in the path after the

setting. That is it holds in the restricted setting where the algorithm may only store up to two previous data points (cf. Section 7). Our algorithm for improving this bound (given in the proof of Theorem 10.2) does not work in this restricted setting. However, our algorithm also does not require the full storage granted by the unrestricted setting, but merely needs to store a linear number of pointers at any given time.

known vertices, and once we get that vertex right, we gain at most $1 + t(v)$ new non-∞ entries. As we have a total of k vertices left to connect with the source, we have $\sum_{v \in U} t(v) = k - |U|$. We have that (10.1) equals $|U|/k + (k - |U|)/k = 1$. Now the additive drift theorem, Theorem 2.17, gives the desired bound. □

Surprisingly, if we may assume that the input graph is complete, we obtain a lower complexity. Note that this includes the case where the complete graph is obtained from an arbitrary one by adding dummy edges with artificially high weight. This shows, again, that even small changes in modeling the combinatorial problem can lead to substantial changes in the black-box complexity.

Theorem 10.3. *The unrestricted black-box complexity of SSSP with* ***complete*** *input graphs is bounded from above by* $\lfloor (n+1)/2 \rfloor + 1$ *and bounded from below by* $n/4$.

Proof. We start with showing the **upper bound**. Essentially we prove that it is possible to learn the problem instance quickly.

We show that the complete graph K_n on n vertices may be written as the union of $\lfloor (n+1)/2 \rfloor$ spanning trees. If n is even then it is well known that K_n may be decomposed into $n/2$ edge-disjoint spanning trees [KK09]. If n is odd, then we choose a node v and all edges adjacent to v, thereby getting a spanning tree. The remaining edges form a K_{n-1}. Since $n - 1$ is even we may decompose the remaining edges into $(n-1)/2$ spanning trees of $K_n - \{v\}$, which we complete to spanning trees of K_n in an arbitrary way. Hence we have written K_n as the union of $(n+1)/2$ spanning trees.

Now we describe our strategy. We choose a cover of $\lfloor (n+1)/2 \rfloor$ spanning trees as above. For each spanning tree T, we make a query to the oracle which contains exactly the vertices in T. After the query, we know for each vertex v its distance from the source s. Since T is a spanning tree, all distances are finite and all nodes can be reached via a unique path from s. Therefore, we can compute all the weights of edges in T. Since the spanning trees cover all edges, we know all edge weights after $\lfloor (n+1)/2 \rfloor$ queries. By our unrestricted computational power, we compute the minimal spanning tree and query for it.

As for the **lower bound**, we apply again Yao's minimax principle, Theorem 2.16. We sample instances by having each vertex $i \geq 2$ choose uniformly at random a $j \in \{1, \ldots, i-1\}$ and then giving 1 as the weight to the edge between i and j, and weight n to all other edges. We call edges of weight 1 "cheap", and all other edges "expensive". By construction, the desired shortest paths tree consists precisely of the the cheap edges (and for each i, the chosen j is the predecessor on that shortest paths tree).

Optimizing an instance as described above only becomes easier if we allow querying arbitrary sets of $n - 1$ edges instead of trees, and it becomes even easier if we allow the algorithm to perform these queries in a sequential manner, i.e., instead of querying a full tree, we allow the algorithm to query the $n - 1$ edges of the tree one by one. For any such edge-query we assume the oracle to reveal the weight of that edge to the algorithm. Furthermore, we assume that the algorithm is done once it has queried each cheap edge at least once. Let a deterministic algorithm A be given. For each $i \leq n$, A has to find a needle-in-the-haystack of size $i - 1$. Note that the different haystacks are completely independent; hence, the haystack associated with vertex i

requires an expected number of $i/2$ (edge) queries [DJW06, Theorem 1]. Due to our simplifying assumptions, it is of no importance in which order the algorithm uses its queries for the different haystacks. We can assume that all haystacks will be queried in order of increasing i, each until the cheap edge for vertex i has been found. Thus, we get an overall expected number of

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

edge queries; hence, A will need an expected number $\geq n/4$ *tree* queries.

$\square$

10.3. SSSP with a Single-Criterion Fitness Function

We have seen in the previous section that SSSP with a multi-criteria objective function has complexity $\Theta(n)$. We feel that this is not satisfactory as already the size of the input is in $\Omega(m)$, where m is the number of edges. The reason for this discrepancy is the large amount of information that the objective function contains. In order to obtain a more realistic black-box model, we study a more restrictive objective function. For this alternative model, it is possible to define notions of unbiased black-box complexity.

In this section we consider the following model for the SSSP problem. A representation of a candidate solution will be a vector $(\rho(2), \ldots, \rho(n)) \in [n]^{n-1}$ to be interpreted as follows. The predecessor of node i is $\rho(i)$. Note that we do not require that $\rho(i) \neq i$, nor do we require that the candidate solution forms a tree; randomized search heuristics without repair mechanisms might generate such solutions. In order to reflect the meaning of the components, the indices of such an x will run from 2 to n, i.e., $x = (x_2, \ldots, x_n)$.

Since we do not want the objective function to give vertex-specific information, we use, for a given graph G, the single-criterion objective function $f_G(\rho(2), \ldots, \rho(n)) := \sum_{i=2}^{n} d_i$ where d_i is the distance of the i-th node to the source. If an edge—including loops—is traversed which does not exist in the input graph, we set $d_i := C$ where C is some very large value (e.g., we could choose $C := nw_{\max}$). Let us mention already here that both in the unrestricted and in the structure preserving model the value C can be learned by the algorithm in a constant number of queries, e.g., by querying the objective value of search point $(2, \ldots, 2)$ in the unrestricted model and dividing it by $n-1$ and, similarly, querying a search point with "all nodes to one non-source node" in the structure preserving unbiased model. In the redirecting unbiased model we learn the value of C by iteratively querying a search point $x \in [n]^{n-1}$ uniformly at random. The probability to obtain in one query a search point where all nodes do not point to the source is $(1 - 1/n)^{n-1} \geq 1/e$. Hence, the probability that after a linear number n of queries no such search point has been sampled is at most $(1 - 1/e)^n = o(1)$. Since neither the constant overhead in the unrestricted and the structure preserving unbiased model nor the linear overhead in the redirecting unbiased model changes the asymptotic bounds given below, we assume that the value C is known to the algorithm.

10.3.1. The Unrestricted Black-Box Complexity

Theorem 10.4. *The unrestricted black-box complexity of the SSSP problem with the single-criterion objective function is at most* $\sum_{i=1}^{n-1} i = n(n-1)/2$ *and the ranking-based unrestricted one is at most* $(n-1)^2$.

Proof of Theorem 10.4. Let $G = (V, E, w)$ be the input graph. We show that adding the i-th node to the shortest paths tree costs at most $n - i$ queries in the unrestricted model and at most $n - 1$ queries in the ranking-based unrestricted model. Basically, we are imitating Dijkstra's algorithm. We say that a node is *unconnected* if in the current solution there does not exist a path from that node to the source and we say it is *connected* otherwise. Recall that the indices run from 2 to n, so $(2, 3, \ldots, n)$ encodes the graph where every node except the source points to itself and is thus not connected to the source.

In the first $n - 1$ iterations query the strings $(1, 3, 4, \ldots, n)$, $(2, 1, 4, 5, \ldots, n)$, $\ldots$, $(2, \ldots, n-1, 1)$, each of which connects exactly one node to the source and lets all other nodes point to themselves. Then each of these strings has $n - 2$ unconnected nodes, which contribute equal to the fitness function. In the non-ranking-based model, we learn the costs of the edges adjacent to the source. In the ranking-based model, we still learn their ranking. In particular, in both models we learn which node $v_1 \in \{2, \ldots, n\}$ can be connected to the source at the lowest cost.

Now assume that $k < n - 1$ nodes $v_1, \ldots, v_k \in \{2, \ldots, n\}$ have been added to the shortest paths tree already.

In the non-ranking-based model, we proceed as follows. Test all $n - k - 1$ possibilities to connect exactly one unconnected node to v_k and let every other unconnected node point to itself. We learn the costs of all edges between unconnected nodes and v_k. Furthermore, we compute for each $j < k$ the lowest cost for connecting an unconnected node to the source via v_j. Note that for $j < k$ we have gathered the required information in previous steps. The cheapest such connection is added to the current solution, and we denote this node by v_{k+1}. Thus, we have constructed the shortest paths tree in $\sum_{i=1}^{n-1} i = n(n-1)/2$ queries.

In the ranking-based model we perform the following $n-1$ queries in the k-th step. In each query, we connect the node $v_1, \ldots, v_k$ as learned before. For the unconnected nodes, we query the following combinations.

- For each unconnected node v make the following query. Connect v to v_k, and let all other unconnected nodes point to themselves.
- For each $j \in [k-1]$, take the unconnected node with minimal edge cost to v_j. Connect this node to v_j, and connect all other unconnected node to themselves.

Note for the second type that we know the unconnected node with minimal edge cost to v_j because we have learned the ranking of all edges adjacent to v_j in an earlier step.

Since all queries have exactly $n-k-2$ unconnected nodes, the contribution of these nodes to the cost function is equal for all queries. Thus we learn which unconnected node produces minimal cost when attached to $v_1, \ldots, v_k$. We call this node v_{k+1}. Moreover, we learn the ranking of all edges from unconnected nodes to v_k, which we need in the forthcoming steps.

Together, the algorithm adds an additional vertex to the current solution using $n-1$ queries. Since we have to add $n-1$ vertices in total, the claim follows. □

10.3.2. Unbiased Black-Box Models for SSSP

In this section we would like to study SSSP in black-box models that require some unbiasedness, as we did for the MST problem. However, the *Hamming-unbiased model* of Lehre and Witt in [LW10a] only applies to pseudo-Boolean functions, cf. Section 3.2. As we are dealing with a different representation here, our first step is to find an appropriate notion of unbiasedness for our setting.

To this end, let us first formulate the unbiased model by Lehre and Witt in a more abstract way.

Definition 10.5. *Let $k \in \mathbb{N}$. Let $\mathcal{S}$ be a set (the search space), and let $\mathcal{G}$ be a set of bijections on $\mathcal{S}$ that forms a group, i.e., that is closed under composition and under inversion. We call $\mathcal{G}$ the* ***set of invariances****.*

A ***k-ary, $\mathcal{G}$-unbiased distribution*** *is a family of probability distributions $\big(D(\cdot \mid y^1, \ldots, y^k)\big)_{y^1,\ldots,y^k \in \mathcal{S}}$ over $\mathcal{S}$ such that for all* ***inputs*** *$y^1, \ldots, y^k \in \mathcal{S}$ the condition*

$$\forall x \in \mathcal{S}\, \forall g \in \mathcal{G} : D(x \mid y^1, \ldots, y^k) = D(g(x) \mid g(y^1), \ldots, g(y^k))$$

holds.

An operator sampling from a k-ary, $\mathcal{G}$-unbiased distribution is called a ***k-ary, $\mathcal{G}$-unbiased variation operator****.*

If no confusion can arise, we use the term unbiased instead of $\mathcal{G}$-unbiased

A k-ary, $\mathcal{G}$-unbiased black-box algorithm can now be described via the scheme of Algorithm 36. The *k-ary, $\mathcal{G}$-unbiased black-box complexity* of some class of functions $\mathcal{F}$ defined on $\mathcal{S}$ is the complexity of $\mathcal{F}$ with respect to all k-ary, $\mathcal{G}$-unbiased black-box algorithms.

Algorithm 36: Scheme of a k-ary, $\mathcal{G}$-Unbiased Black-Box Algorithm

```
1 Initialization: Sample $x^{(0)} \in \mathcal{S}$ from a 0-ary $\mathcal{G}$-unbiased distribution and
  query $f(x^{(0)})$;
2 Optimization: for $t = 1, 2, 3, \ldots$ do
3     Depending on $\big(f(x^{(0)}), \ldots, f(x^{(t-1)})\big)$ choose $k$ indices $i_1, \ldots, i_k \in [0..t-1]$,
      and a $k$-ary, $\mathcal{G}$-unbiased distribution $(D(. \mid y^{(1)}, \ldots, y^{(k)}))_{y^{(1)},\ldots,y^{(k)} \in \{0,1\}^n}$;
4     Sample $x^{(t)}$ according to $D(\cdot \mid x^{(i_1)}, \ldots, x^{(i_k)})$ and query $f(x^{(t)})$;
```

To make very precise the connection between this generalized model and the original model by Lehre and Witt, recall that they formulated their model for the hypercube $\{0,1\}^n$ as search space, where the set $\mathcal{G}$ of invariances is the set of all bijections of the search space that preserve Hamming distances. Note that every such bijection may be written in the form $g(x) = \sigma(x \oplus z)$, where $z \in \{0,1\}^n$ is an XOR shift in direction z and $\sigma \in S_n$ is a permutation of the bit positions. We call operators that are unbiased with respect to this set of invariances *Hamming-unbiased operators*, and similarly for unbiased black-box algorithms and unbiased black-box complexity.

Here in this work we discuss three possible sets of invariances, giving rise to *structure preserving unbiased black-box complexity*, *redirecting unbiased black-box complexity*, and *generalized unbiased black-box complexity*, respectively. The former two unbiased notions were introduced in the original paper underlying this section, cf. [DKLW11], and the latter one was introduced by Rowe and Vose in a more general framework for unbiasedness notions [RV11]. We find that the structure preserving and the generalized unbiased complexity are too powerful to give anything new compared to the unrestricted black-box model, whereas the redirecting model seems more promising.

Before we turn to the specific sets of invariances, we develop some general theory on $\mathcal{G}$-unbiased distribution, for any set of invariances $\mathcal{G}$. Assume we have a k-tuple $\vec{z} = (z^1, \ldots, z^k)$ of search points, and we would like to sample the next search point according to a probability distribution $D_{\vec{z}}$ over the search space. We may do so if and only if there is a k-ary $\mathcal{G}$-unbiased distribution $(D(\cdot \mid \vec{y}))_{\vec{y}}$ such that $D(\cdot \mid \vec{z}) = D_{\vec{z}}$. An obvious necessary condition is

$$\begin{array}{l}\text{For all } g \in \mathcal{G} \text{ such that } g(z^i) = z^i \text{ for all } i \in [k]\text{, and} \\ \text{for all } x \in [n]^{n-1} \text{ it holds that } D_{\vec{z}}(x) = D_{\vec{z}}(g(x)).\end{array} \tag{10.2}$$

The following proposition shows that this condition is also sufficient.

Proposition 10.6. *Let $\mathcal{G}$ be a set of invariances, i.e., a set of permutations of the search space $\mathcal{S} = [n]^{n-1}$ that form a group. Let $k \in \mathbb{N}$, and $\vec{z} = (z^1, \ldots, z^k) \in \mathcal{S}^k$ be a k-tuple of search points. Let*

$$\mathcal{G}_0 := \{g \in \mathcal{G} \mid g(z^i) = z^i \text{ for all } i \in [k]\}$$

be the set of all invariances that leave $z^1, \ldots, z^k$ fixed.

Then for any probability distribution $D_{\vec{z}}$ on $[n]^{n-1}$, the following two statements are equivalent.

(i) The probability distribution $D_{\vec{z}}$ extends to a k-ary $\mathcal{G}$-unbiased distribution $(D(\cdot \mid \vec{y}))_{\vec{y} \in \mathcal{S}^k}$ on $\mathcal{S}$ with $D_{\vec{z}} = D(\cdot \mid \vec{z})$.

(ii) For every $g \in \mathcal{G}_0$ and for all $x \in \mathcal{S}$ it holds that $D_{\vec{z}}(x) = D_{\vec{z}}(g(x))$.

Proof. The implication $(i) \implies (ii)$ follows directly from Definition 10.5.

So assume that $D_{\vec{z}}$ satisfies (ii). We explicitly define the unbiased distribution as follows. For all $\vec{y} \in \mathcal{S}^k$ there are two possibilities.

1. Either there exists a $g \in \mathcal{G}$ such that $g(y^i) = z^i$ for all $i \in [k]$. In this case, we define $D(x \mid \vec{y}) := D_{\vec{z}}(g(x))$ for all $x \in \mathcal{S}$.
2. Or there does not exist such a g. In this case, we let $D(\cdot \mid \vec{y})$ be the uniform distribution over the search space.

Condition (ii) implies that the distributions $D(\cdot \mid \vec{y})$ are well-defined. It is straightforward to check that they form a k-ary structure preserving unbiased distribution that extends $D_{\vec{z}}$. $\square$

Although the formulation of the proposition looks rather technical, it is a key ingredient for determining how powerful unbiased operators are. Statement (i) is the statement we are interested in: It says that a given probability distribution over the search space may be used to determine the next search point. On the other hand, statement (ii) gives us a criterion that can be checked using only the distribution $D_{\vec{z}}$, without reference to the whole family of distributions associated to an unbiased operator.

Now we are ready to introduce the three sets of invariances. The three notions will be defined first, followed by a discussion on the main differences between them.

In the first unbiasedness condition, we want the algorithm to be unbiased with respect to relabeling of the nodes. Intuitively, all subgraphs with the same structure but different labels are equally likely to be chosen.

Definition 10.7 (Structure preserving unbiasedness). *Let $k \in \mathbb{N}$. A **k-ary structure preserving unbiased distribution** is a k-ary SP-unbiased distribution over $[n]^{n-1}$ with set of invariances*

$$SP = \{\hat{\sigma} \mid \sigma \in S_n,\, \sigma(1) = 1\},$$

where for all $x = (x_2, \dots, x_n) \in [n]^{n-1}$, $\hat{\sigma}(x) := \big(\sigma(x_{\sigma^{-1}(2)}), \dots, \sigma(x_{\sigma^{-1}(n)})\big)$.

Alternatively, as search points are just mappings from the vertex set into itself, we might require that all possible images of each vertex are to be treated symmetrically. The idea is that an unbiased probability distribution would then be unbiased with respect to redirecting the pointers to predecessors of the vertices. Formally, we require the following.

Definition 10.8 (Redirecting unbiasedness). *Let $k \in \mathbb{N}$. A **k-ary redirecting unbiased distribution** is a k-ary RD-unbiased distribution over $[n]^{n-1}$ with set of invariances*

$$RD = \{\vec{s} \mid \vec{s} = (\sigma_2, \dots, \sigma_n) \in S_n^{n-1}\},$$

where for all $x = (x_2, \dots, x_n) \in [n]^{n-1}$, $\vec{s}(x) := \big(\sigma_2(x_2), \dots, \sigma_n(x_n)\big)$.

At the same time when we developed the structure preserving and the redirecting black-box models, Rowe and Vose [RV11] independently extended the notion of unbiasedness from the hypercube to arbitrary search spaces. More precisely, Rowe and Vose define the following.

Definition 10.9 (Generalized unbiased distributions [RV11]). *Let $\mathcal{F}$ be a class of functions from search space $\mathcal{S}$ to some set Y. We say that a bijection $\alpha : \mathcal{S} \to \mathcal{S}$ **preserves** $\mathcal{F}$ if for all $f \in \mathcal{F}$ it holds that $f \circ \alpha \in \mathcal{F}$. Let $\Pi(\mathcal{F})$ be the class of all such $\mathcal{F}$-preserving bijections α.*

*A **k-ary generalized unbiased distribution (for $\mathcal{F}$)** is a k-ary $\Pi(\mathcal{F})$-unbiased distribution.*

Note that for the previous definition that it is argued in [RV11] that $\Pi(\mathcal{F})$ indeed forms a group so Definition 10.9 satisfies our definition of unbiasedness given in Definition 10.5.

As for the hypercube, we call an operator sampling from a k-ary structure preserving (redirecting, generalized) unbiased distribution a *k-ary structure preserving (redirecting, generalized) unbiased variation operator*. Similarly, we define a *k-ary structure preserving (redirecting, generalized) unbiased algorithm* to be a k-ary SP-unbiased (RD-unbiased, $\Pi(\mathcal{F})$-unbiased) algorithm.

Now we determine the distributions $D_{\vec{z}}$ satisfying condition (ii) in Proposition 10.6 for the structure preserving model and for the redirecting model, respectively. As the analysis of the generalized model is more involved, we postpone it to Section 10.3.5.

Let us first consider the unary case $k = 1$ and $\vec{z} = (z)$. For the structure preserving model, we need to find all source-preserving permutations that leave z unchanged. These are in one-to-one correspondence with the source-preserving automorphisms of the graph induced by z, i.e., with source-preserving bijections of the vertex set that map neighbors to neighbors. If A denotes the group of these automorphisms, then condition (ii) in Proposition 10.6 is that $D_{\vec{z}}$ must be invariant under A, i.e., for all $\alpha \in A$ and all $x \in [n]^{n-1}$ we require $D_{\vec{z}}(x) = D_{\vec{z}}(\alpha(x))$. For higher arities k, for $\vec{z} = (z^1, \dots, z^k)$, we get a group A_i of source-preserving automorphisms for each search point z^i. In this case, condition (ii) in Proposition 10.6 is that $D_{\vec{z}}$ must be invariant under the intersection $A = \bigcap_{i \in [k]} A_i$ of all these groups.

For the redirecting model, we again treat the unary case first. We need to determine all tuples $s \in S_n^{n-1}$ such that $\vec{s}(z) = z$. Consider any component z_i of z. Then we can choose σ_i to be any permutation of $[n]$ with $\sigma_i(z_i) = z_i$. In particular, for all $s, t \in [n] \setminus \{z_i\}$ there is such a permutation mapping s to t. Therefore, a distribution $D_{\vec{z}}$ over the search space can be extended to a unary redirecting unbiased distribution if and only if the following condition is satisfied. "If $x, y \in [n]^{n-1}$ are vectors such that for every $i \in [2, \dots, n]$ the equations $x_i = z_i$ and $y_i = z_i$ are either both true or are both false, then $D_{\vec{z}}(x) = D_{\vec{z}}(y)$". So for each vertex, we may choose whether we redirect it or not, but we cannot control where it is redirected to. This is an important to note difference to the Hamming-unbiasedness where we may not ask for something like "change the j-th entry of the current search point".

For higher arities k, for $\vec{z} = (z^1, \dots, z^k)$, a distribution $D_{\vec{z}}$ over the search space can be extended to a k-ary redirecting unbiased distribution if and only if the following condition is satisfied. "If $x, y \in [n]^{n-1}$ are vectors such that for every $i \in [2, \dots, n]$ with $x_i \neq y_i$ it holds that $x_i, y_i \in [n] \setminus \{z_i^1, \dots, z_i^k\}$, then $D_{\vec{z}}(x) = D_{\vec{z}}(y)$". So for each vertex, we may choose to direct it to any target that occurs in the points $z^1, \dots, z^k$, or we may choose to direct it to some new destination, but in the latter case we cannot control the exact target.

We see that for a probability distribution over the search space, being redirecting unbiased is a rather strong condition.

10.3.3. The Structure Preserving Unbiased Black-Box Model

As we shall prove now, being structure preserving unbiased is a very weak condition, the reason for this being the fact that many graphs have few automorphisms. In fact, we obtain the following theorem.

Theorem 10.10. *Let $\mathcal{F}$ be a class of real-valued functions on $[n]^{n-1}$. The unary* *structure preserving unbiased black-box complexity of $\mathcal{F}$ is at most $1 + \mathrm{UBBC}(\mathcal{F})$, where $\mathrm{UBBC}(\mathcal{F})$ denotes the unrestricted black-box complexity of $\mathcal{F}$.*

Furthermore, the same holds for the associated ranking-based complexities.

Proof. We may use the following unary structure preserving unbiased algorithm. With the first query, sample a path p starting in the source, i.e., sample from the distribution that has equal positive probability on all search points that represent such a path, and that has probability 0 elsewhere. This is a 0-ary (and hence, also unary) structure preserving unbiased operator. Since p has no source preserving automorphism, *every* probability distribution D_p over the search space can be extended to a unary structure preserving unbiased distribution.

Therefore, we have now the full power of the unrestricted model and may imitate any (ranking-based) unrestricted black-box algorithm. □

Corollary 10.11. *The ranking-based unary structure preserving unbiased black-box complexity of the SSSP problem with the single-criterion objective function is $O(n^2)$.*

10.3.4. The Redirecting Unbiased Black-Box Model

It has been argued in [BBD+09] that the Randomized Local Search variant which in each iteration redirects exactly one vertex chosen uniformly at random, solves the single-source shortest paths problem with the single-criterion fitness function in $O(n^3)$ iterations.

Theorem 10.12 (from [BBD+09]). *The Randomized Local Search algorithm for SSSP with the single-criterion fitness function has runtime $O(n^3)$.*

Since this algorithm is contained in the ranking-based redirecting unbiased black-box model, we immediately gain an upper bound of $O(n^3)$.

Corollary 10.13. *The ranking-based unary redirecting unbiased black-box complexity of SSSP is $O(n^3)$.*

If we are allowed to access the fitness values themselves instead of their ranking, it is possible to learn the problem instance in fewer steps. Once we know the problem instance, we may compute the optimal solution without cost, because we are only charged for queries. Afterwards, we only have to construct the solution by a sequence of queries. For complete input graphs and binary distributions, all this is possible in time $O(n^2 \log n)$.

Theorem 10.14. *The binary redirecting unbiased black-box complexity of SSSP for complete graphs is $O(n^2 \log n)$.*

Proof. We divide the algorithm into three phases. In the first two phases we learn the weights of the problem instance and in the third phase we construct the solution.
Phase 1: Similar to the previous algorithms, we first construct a search point where no vertex is connected to the source, i.e., no vertex is in the same connected component as the source. Again we call such vertices "unconnected" vertices, and we call all other vertices "connected".

As briefly discussed at the beginning of this section, we may obtain such a search point by iteratively sampling search points uniformly at random from $[n]^{n-1}$. The probability to sample a search point with no non-source vertex connected to the source in one query is

$$\left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{1}{e}\,.$$

If we sample n times, then the probability that no such search point occurs is exponentially small in n. Hence, we may assume that we find the global maximum. From the global maximum we derive the punishment value C that is charged for every unconnected vertex by dividing the global maximum by $n-1$. Since $C \geq nw_{\max}$ holds, we are able to decide for each search point how many vertices are connected to the source.

We sample again uniformly at random from the search space and look for search points x^i where exactly one vertex i is connected. In order to find such a search point, i must direct to the source, and every other vertex must direct to one of the vertices $[n] \setminus \{1, i\}$. The probability for this to happen is

$$\frac{1}{n}\left(1 - \frac{2}{n}\right)^{n-2} = \frac{1}{n}\left(1 - \frac{2}{n}\right)^{(\frac{n}{2}-1)2} \geq \frac{1}{e^2 n} = \Omega(1/n)\,.$$

Since these events are disjoint for all $i \in \{2, \ldots, n\}$, the probability to sample a search point x^i for some $i \in \{2, \ldots, n\}$ is constant. By a coupon collector argument, we need $\Theta(n \log n)$ many samples until we have seen all x^i.

In order to determine the vertex i from the search point x^i, we apply the unary redirecting-unbiased operator O_j, which redirects vertex j and leaves the rest unchanged, to the search point x^i, for all $j \in \{2, \ldots, n\}$. For $i = j$ the result is a search point with all vertices unconnected, whereas all other operators give at least one connected vertex. Therefore, we can determine the index i in linear time. We do this procedure for all search points x^i, requiring $O(n^2 \log n)$ queries in total.

Thus, in the first phase we have constructed the empty graph, and for each $i \in \{2, \ldots, n\}$ we have constructed a search point x^i in which only node i is connected to the source. In total we have queried

$$\Theta(n) + \Theta(n \log n) + \Theta(n^2 \log n) = \Theta(n^2 \log n)$$

search points.

From the empty graph we have learned C and we can compute the weight of the edge $\{1, i\}$ by subtracting $(n-2)C$ from the weight of x^i, i.e., $w(\{1, i\}) = w(x^i) - (n-2)C$.

Phase 2: To each search point x^i, we apply the unary unbiased operator which keeps vertex i unchanged and redirects all other vertices. We look for search points $x^{i,j}$ where i points to the source, j points to i, and all other vertices are not connected to the source. By a similar calculation as above, we find out that the probability to find a specific $x^{i,j}$ is in $\Theta(1/n)$, and that for fixed i we need to sample $\Theta(n \log n)$ many times from x^i to have sampled, with high probability, all search points $x^{i,j}$ with $j \in \{2, \ldots, n\}$. This is again the standard coupon collector's argument.

We repeat the process for all i, and store all the points $x^{i,j}$ in a set X_i. In fact, since we cannot distinguish whether two samples connect the same two points to the source, X_i is a multiset with an expected number of $\Theta(n \log n)$ many elements. By the union bound, with probability at least $1/2$ every multiset X_i contains every search point $x^{i,j}$, for all $i, j \in \{2, \dots, n\}$ with $i \neq j$. We will assume henceforth that we have indeed found all these search points. We can increase the probability of success by repeating the whole algorithm several times, cf. Lemma 2.14.

Note, finally, that from a sample $x^{i,j}$ we can compute the weight of the edge $\{i, j\}$, without knowing the vertex j it belongs to. In total, phase 2 needs time $O(n^2 \log n)$.

Phase 3a: We construct the SSSP tree iteratively. Beforehand, we construct some auxiliary search points. For any $i \in \{2, \dots, n\}$, consider the binary operator that on input (x, y) returns a search point z with $z_i = x_i$ and $z_j = y_j$ for all $j \neq i$. This operator is easily seen to be unbiased in the redirecting sense. Applying this operator to the pair (x^i, y) redirects the vertex i in y to the source. By recursive application we can generate a search point y^0 with all vertices pointing to the source, in time $O(n)$. Fix an index i. We redirect i in y^0 and check whether we obtain a search point y^i with fewer connected vertices. This happens if and only if i is redirected to itself, since all other vertices are connected. In expectation, $O(n)$ redirections of the vertex i are necessary until it points to itself. In this way, for every $i \in \{2, \dots, n\}$ we generate a search point y^i with i pointing to itself. Using these search points, we may henceforth redirect any vertex of a given search point to itself by a binary unbiased operator. In particular, we can generate a search point where all vertices point to themselves, in linear time. Altogether, phase 3a takes total time $O(n^2)$.

Phase 3b: We construct the SSSP tree by imitating Dijkstra's algorithm. We start with the search point u^0 where all vertices point to themselves.

Throughout this phase we maintain a search point u^t where all vertices in the connected component of the source point to the correct target (i.e., to the same target as in the SSSP tree), and all other vertices point to themselves. In each step, we will know which vertices are connected to the source in u^t. Finally, we maintain a (multi-)set $X \subseteq \{x^i \mid i \in \{2, \dots, n\}\} \cup X_2 \cup \dots \cup X_n$ of search points representing the edges that may be used to extend y^t. We start with $X := \{x^i \mid i \in \{2, \dots, n\}\}$.

In each step, we choose the search point $x \in X$ that represents the cheapest edge. Assume that the edge connects i_t to j_t, with i_t known to the algorithm. If x is of the form x^i, then i_t is simply the source. We apply the binary operator to (u^t, x) that redirects node j_t to node i_t in u^t. Formally, this is a binary operator $(D(\cdot \mid v, w))_{v,w \in [n]^{n-1}}$ that, given a pair (v, w) of search points, redirects in v to i_t all nodes j which in w point to i_t. All nodes not directing to i_t are not redirected in v. Let us briefly show that this operator is indeed an unbiased one. By Proposition 10.6 we need to verify that for all $v, w \in [n]^{n-1}$ and for all $\vec{s} = (\sigma_2, \dots, \sigma_n) \in S_n^{n-1}$ satisfying $\sigma_i(v_i) = v_i$ and $\sigma_i(w_i) = w_i$ for all $i \in \{2, \dots, n\}$ we have

$$\forall x \in [n]^{n-1} : D(x \mid v, w) = D(\vec{s}(x) \mid v, w)\,,$$

where we recall that, by definition, $D(\vec{s}(x) \mid v, w) = D\left((\sigma_2(x_2), \dots, \sigma_n(x_n)) \mid v, w\right)$. To this end, let $x \in [n]^{n-1}$ be given. Clearly, $D(x \mid v, w) = 1$ if and only if $x_k = i_t$ for all $k \in \{2, \ldots, n\}$ with $w_k = i_t$ and $x_k = v_k$ for all other k. Assume $D(x \mid v, w) = 1$. We show that x is a fixpoint of $\vec{s}$. If $w_k = i_t$, by assumption, we have $\sigma_k(i_t) = \sigma_k(w_k) = w_k = i_t$. Therefore, $\sigma_k(x_k) = \sigma_k(i_t) = i_t = x_k$ for all such k. Similarly we have $\sigma_k(v_k) = v_k$ for all k and therefore, $\sigma_k(x_k) = \sigma_k(v_k) = v_k = x_k$ for all k with $w_k \neq i_t$. Hence, x is indeed a fixpoint of $\vec{s}$. Since all σ_i are bijective, similarly we conclude that $D(\vec{s}(x) \mid v, w) = 0$ if and only if $D(x \mid v, w) = 0$.

Note that in our case we shall always have exactly one node only that points to i_t. Therefore, for the pair (u^t, x), this operator redirects only the vertex j_t to i_t (in u^t). We denote the new search point $\tilde{u}^{t+1}$. If the number of connected components satisfies $c(\tilde{u}^{t+1}) \geq c(u^t)$ (i.e., the number of nodes which are connected to the source remains unchanged or decreases), we conclude that j_t was already connected in u^t. In this case, we remove x from X, reject $\tilde{u}^{t+1}$, and continue with u^t. On the other hand, if the number of connected components decreases (i.e., the number of vertices connected to the source increases), then we know that j_t was not connected so far, and that j_t directs to i_t in the shortest paths tree. In this case, we accept $\tilde{u}^{t+1}$ as new search point by updating $u^{t+1} := \tilde{u}^{t+1}$. Since all unconnected vertices in u^t point to themselves, the same is true for u^{t+1}.

If we move to u^{t+1}, we need to determine the index j_t of the vertex we have just added. If x is of the form x^i, then we already know the index. If x is of the form $x^{i,j}$, we need to find out $j_t = j$. We do this by applying the operators O_j to x, for $j \in \{2, \ldots, n\} \setminus \{i_t\}$ until the number of connected vertices drops to 1. This happens only if $j = j_t$, so we can determine j_t in this way. It may be that the operator O_{j_t} also does not decrease the number of connected vertices because it redirects j_t to the source. But this is an unlikely event, and we only need to apply each operator O_j an expected constant number of times until the number of connected vertices drops to 1. So we need expected time $O(n)$ to find the index j_t. Once we know the index j_t, we add X_{j_t} to the multiset X.

We are finished when all vertices are connected. The runtime of this phase composes as follows. Whenever we choose an edge $x \in X$, we use one sample in order to determine whether the edge connects an unconnected vertex. If this is not the case, then we remove x from X. Since each multiset X_i is added at most once to X, and has expected size $\Theta(n \log n)$ the total number of search points ever added to X is in $\Theta(n^2 \log n)$. On the other hand, if x connects an unconnected vertex, then we use up to linear time to determine the index of the vertex that we have added. However, we add only $n-1$ unconnected vertices, so the total time for this operation is in $O(n^2)$. Together, we need time $O(n^2 \log n)$ for the third phase. □

Theorem 10.15. *The $*$-ary redirecting unbiased black-box complexity of SSSP is $\Omega(n^2)$.*

Proof. Consider the path on $[n]$, connecting adjacent integers with edges of unit weight. Let A be a redirecting unbiased algorithm optimizing this path. Let X_t be the random variable of the maximal number of vertices properly connected to the source among the first t search points sampled by A. Let T be the random variable denoting the minimal t such that $X_t = n$. We argue with drift. We sketch the main arguments. The formal proof is similar to the ones in Theorem 10.2 and 10.3.

Suppose that, at some point t, A has sampled at most $n/2$ search points with $k < n$ properly connected vertices, and none with more than k properly connected vertices. Then, in the next sample, the probability of sampling a search point with more than k properly connected vertices is at most $2/n$. This is due to the fact that in the redirecting unbiased model we can only decide whether or not to redirect a vertex, but we may not choose where it is being redirected to. Here we assumed that we have sampled at most $n/2$ search points with $k < n$ properly connected vertices. Thus, we can exclude at most $n/2$ vertices as target vertices for the redirection. That is, if we decide to redirect a node, all other $\geq n/2$ vertices must be equally likely to be chosen and hence the probability to direct a vertex to the correct one is $\leq 2/n$.

This shows that, after sampling the first search point with k vertices properly connected (and no previous search point had more), the expected number of additional samples until a search point with strictly more than k properly connected vertices is sampled is $\Omega(n)$. Furthermore, when finally such a search point is sampled, the expected number of properly connected vertices in this new search point is $k+O(1)$ (as one more vertex is connected successfully, but any further properly connected vertices must be properly connected by accident, with a probability of $1/n$ per additional vertex, compare with the proof of Theorem 10.2 for a similar argument).

Thus, using the additive drift theorem (see Theorem 2.17), we get a total runtime of $\Omega(n^2)$. □

10.3.5. The Generalized Unbiased Black-Box Model

In this section, we will study the generalized unbiased black-box model by Rowe and Vose [RV11]. Unfortunately, their description defines the set of invariances in an indirect way, see Definition 10.9. Most of this section will be devoted to explicitly determining this set of invariances. Once this is done, we will see that their notion is almost the same for the single-criterion SSSP problem as the structure preserving model we have introduced in Definition 10.7. In particular, we will see that the complexity of the single-criterion SSSP increases by at most 1 compared to the unrestricted black-box complexity.

Theorem 10.16. *For a search point x, let $E(x)$ be the set of edges of the connected component of the source according to the graph encoded by x.*

For any permutation α of $[n]^{n-1}$ the following two statements are equivalent.

(i) $\alpha \in \Pi(SSSP)$.

(ii) There is a $\hat{\sigma} \in \mathrm{SP}$ such that for all $x \in [n]^{n-1}$ we have $E(\alpha(x)) = E(\hat{\sigma}(x))$.

Proof. Let $\alpha \in \Pi(SSSP)$. We fix a specific SSSP instance P which is a complete graph such that the $n-1$ edges that are incident with the source have weights at least 2 and each two are at least 1 apart, and all other edges have weight less than 1. Furthermore, we choose all weights to be $\mathbb{Q}$-linearly independent positive real numbers (i.e., no nontrivial linear combination with rational coefficient vanishes).

We let f be the fitness function associated with P. Let $f' = f \circ \alpha$. We know that f' corresponds to some SSSP instance P' (as $\alpha \in \Pi(SSSP)$).

Recall that, for any SSSP problem, there is a large value C such that a search point is charged C for every vertex that is not connected to the source, and this C exceeds the cost of any legal path from a vertex to the source. We fix that value C for P, and similarly the value C' for P'.

Claim 1: We have $C = C'$.

Proof of Claim 1. The maximum of f on the search space is $(n-1)C$, and it is attained when no vertex is connected to the source. Since α is a bijection, the maximum of $f' = f \circ \alpha$ is the same as that of f, and also $(n-1)C'$; thus, $C' = C$. □ (of Claim 1)

Let $\mathcal{S}$ be the set of all search points x such that $E(x)$ contains exactly one edge.

Claim 2: α restricted to $\mathcal{S}$ is a permutation of $\mathcal{S}$.

Proof of Claim 2. Since the value C is at least n times larger than the maximum weight, we can, for every search point x, derive the number of connected vertices in x from its fitness. Since the value $f'(x) = f(\alpha(x))$ is at the same time the fitness of x with respect to P' and the fitness of $\alpha(x)$ with respect to P, the number of connected vertices coincides for x and $\alpha(x)$. □ (of Claim 2)

For all i with $2 \leq i \leq n$, we let $\mathcal{S}^{(i)}$ be the set of search points x such that $E(x)$ contains only the edge between 1 and i.

Claim 3: There is a permutation σ on $[n]$ such that, for all i, $\alpha(\mathcal{S}^{(i)}) = \mathcal{S}^{(\sigma(i))}$.

Proof of Claim 3. For all $i \in \{2, \dots, n\}$, fix a search point $x^{(i)} \in \mathcal{S}^{(i)}$. Using Claim 2, for all i, $E(\alpha(x^{(i)}))$ contains exactly one edge; let $\sigma(i)$ be such that $E(\alpha(x^{(i)}))$ contains only the edge between 1 and $\sigma(i)$. Furthermore, let $\sigma(1) = 1$. We show that σ is a permutation of $[n]$ with the claimed property.

For each i, we obviously have that all elements from $\mathcal{S}^{(i)}$ have the same value under f (namely the weight of the edge connecting i with 1 in P plus $n-2$ times C); we denote this value $f(\mathcal{S}^{(i)})$. Analogously, they have the same value under f', which we denote $f'(\mathcal{S}^{(i)})$. Because of the different weight of these edges in P we have, for $i \neq j$, $f(\mathcal{S}^{(i)}) \neq f(\mathcal{S}^{(j)})$; similarly we get $f'(\mathcal{S}^{(i)}) \neq f'(\mathcal{S}^{(j)})$, as f' attains the same values as f (just for possibly different arguments).

Let $i, j \in [n]$ be such that $\sigma(i) = \sigma(j)$. Thus, $f(\alpha(x^{(i)})) = f(\alpha(x^{(j)}))$. Using the definition of f' we get equivalently $f'(x^{(i)}) = f'(x^{(j)})$. This shows that $x^{(i)}$ and $x^{(j)}$, for some k, are both elements of $\mathcal{S}^{(k)}$; in particular, $i = k = j$. This shows that σ is in fact a permutation on $[n]$.

Let $y \in \mathcal{S}^{(i)}$. We show $\alpha(y) \in \mathcal{S}^{(\sigma(i))}$. We have

$$f(\alpha(y)) = f'(y) = f'(x^{(i)}) = f(\alpha(x^{(i)})).$$

Thus, $\alpha(y)$ and $\alpha(x^{(i)})$ have the same connected vertex, namely $\sigma(i)$. Hence, $\alpha(y) \in \mathcal{S}^{\sigma((i))}$. This suffices to show the claim. □ (of Claim 3)

Let σ be as given by Claim 3.

Claim 4: For all i with $2 \leq i \leq n$ we have $w'(1, i) = w(1, \sigma(i))$.

Proof of Claim 4. Let i be such that $2 \leq i \leq n$. Fix $x \in \mathcal{S}^{(i)}$. Using Claim 3, we have $(n-2)C + w'(1,i) = f'(x) = f(\alpha(x)) = (n-2)C + w(1,\sigma(i))$.
□ (of Claim 4)

Now we extend Claim 4 to arbitrary edges.

Claim 5: For all $i, j \leq n$ we have $w'(i,j) = w(\sigma(i),\sigma(j))$.

Proof of Claim 5. For all $i \in \{2 \ldots n\}$ let $M_i = \{x \mid (n-3)C + 2w(1,i) < f(x) < (n-3)C + 2w(1,i) + 1\}$. By choice of our instance P and by the definition of C, the elements of M_i correspond to the search points that contain the edge $\{1,i\}$ and some other edge incident with i.

Consider, for all $i \leq n$, $M_i' = \{x \mid (n-3)C + 2w(1,i) < f'(x) < (n-3)C + 2w(1,i) + 1\}$. As the edges incident with 1 in P' have the same weights as the edges incident with 1 in P, we have that M_i' contains all search points with two connected vertices containing the edge $\{1,\sigma^{-1}(i)\}$ and another edge incident with $\sigma^{-1}(i)$ (as only the edge $\{1,\sigma^{-1}(i)\}$ can contribute $w(1,i)$, by Claim 4). As f and f' have the same range, the elements of M_i have the same f-fitnesses as the elements of M_i' have f'-fitnesses. Thus, the P-weights on the edges adjacent to i are the same as the P'-weights adjacent to $\sigma^{-1}(i)$.

Let us now consider two vertices $i, j \leq n$. We have just seen that the P'-weight $w'(i,j)$, which is adjacent to both i and j is also a P-weight adjacent to both $\sigma(i)$ and $\sigma(j)$. But as all weights are different, $w'(i,j) = w(\sigma(i),\sigma(j))$ must necessarily hold, as desired. □ (of Claim 5)

We are now ready to prove the original statement of this theorem.

Claim 6: For all x, $E(\alpha(x)) = E(\hat{\sigma}(x))$.

Proof of Claim 6. Recall that for all search points x, $E(x)$ is the set of edges in the component (as defined by x) of the source. Note that, as all edge weights both in P and P' are $\mathbb{Q}$-linearly independent (and we know which edges have which weight, using Claim 5), we can derive the edges used in a solution from the fitness of the search point. In other words, there are functions D and D' such that, for each x, $D(f(x)) = E(x) = D'(f'(x))$. For a set of edges M, we write $\sigma(M) = \{\{\sigma(i),\sigma(j)\} \mid \{i,j\} \in M\}$.

Let a search point x be given. We have

$$\begin{aligned}
E(\alpha(x)) &= D(f(\alpha(x))) \\
&= \{\{i,j\} \mid w(i,j) \text{ is part of the } f\text{-weight of } \alpha(x)\} \\
&= \{\{\sigma(i),\sigma(j)\} \mid w(\sigma(i),\sigma(j)) \text{ is part of the } f\text{-weight of } \alpha(x)\} \\
&= \{\{\sigma(i),\sigma(j)\} \mid w'(i,j) \text{ is part of the } f'\text{-weight of } x\} \\
&= \sigma(D'(f'(x))) \\
&= \sigma(E(x)) \\
&= E(\hat{\sigma}(x)).
\end{aligned}$$

□ (of Claim 6)

This shows one implication. The converse is straightforward. □

Proposition 10.6 enables us to decide whether a given probability distribution over the search space extends to a generalized unbiased distribution, similar to the discussion for structure preserving and the redirecting model after Definitions 10.7 and 10.8. If $\vec{z} = (z^1, \dots, z^k) \in \mathcal{S}^k$ is a k-tuple of search points, and $D_{\vec{z}}$ is a probability distribution over the search space, the following two statements are equivalent.

(i) The probability distribution $D_{\vec{z}}$ extends to a k-ary generalized unbiased distribution $(D(\cdot \mid \vec{y}))_{\vec{y} \in \mathcal{S}^k}$ on $\mathcal{S}$.

(ii) For every permutation σ of the search space with the property that $E(\sigma(z^i)) = E(z^i)$ for all $i \in [k]$, and for all $x \in \mathcal{S}$ it holds that $D_{\vec{z}}(x) = D_{\vec{z}}(\sigma(x))$.

This characterization shows that the generalized unbiased black-box complexity still gives the algorithms much power, similar to the structure preserving one. In particular, if the graphs corresponding to the search points z^i are connected, then a probability distribution $D_{\vec{z}}$ extends to a generalized unbiased distribution if and only if it extends to a structure preserving unbiased one. Consequently, we get the following.

Corollary 10.17. *The ranking-based unary generalized unbiased black-box complexity of the SSSP problem with the single-criterion objective function is between $O(n^2)$.*

Proof. The proof is exactly as the proof of Theorem 10.10. All the operators used there are in fact also generalized unbiased by Theorem 10.16. □

10.4. Conclusions for Sections 9 and 10

We have analyzed the black-box complexities of the two combinatorial problems of finding a minimum spanning tree and finding single-source shortest paths trees.

In the MST problem, we could apply many of the techniques developed in previous parts of this thesis. We have shown that access to variation operators of arity greater than one provably helps to decrease runtimes. This raises the question whether there exist "natural" bio-inspired algorithms, which outperform standard mutation-based search heuristics, by using higher arity variation operators. We believe that our work indicates ways to design such algorithms.

For the SSSP problem, the main challenge is in finding reasonable ways to generalize the unbiased black-box model by Lehre and Witt to more general search spaces than the hypercube $\{0,1\}^n$. We have analyzed three different approaches. Two seemingly natural models proved not very useful. In fact, both the generalized unbiased black-box model by Rowe and Vose as well as the structure preserving model introduced here in this work turned out to be almost the same as the unrestricted model. These results indicate that much care has to be taken in order to get a reasonable unbiasedness definition for the problem at hand. For the SSSP problem, the redirecting unbiased black-box model seems most promising.

For both the MST and the SSSP problem, the differences between the ranking-based model and their basic counterparts are negligible. This is due to the nature of the two problems. It would certainly be interesting to investigate other combinatorial problems in the different black-box settings.

Bibliography

[Aar04] Scott Aaronson. Lower bounds for local search by quantum arguments. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC'04)*, pages 465–474. ACM, 2004.

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

[Ald83] David Aldous. Minimization algorithms and random walk on the d-cube. *Annals of Probability*, 11:403–413, 1983.

[AW09] Gautham Anil and R. Paul Wiegand. Black-box search by elimination of fitness functions. In *Proceedings of the 10th ACM Workshop on Foundations of Genetic Algorithms (FOGA'09)*, pages 67–78. ACM, 2009.

[BBD$^+$09] Surender Baswana, Somenath Biswas, Benjamin Doerr, Tobias Friedrich, Piyush P. Kurur, and Frank Neumann. Computing single source shortest paths using single-objective fitness functions. In *Proceedings of the 10th ACM Workshop on Foundations of Genetic Algorithms (FOGA'09)*, pages 59–66, 2009.

[CCH96] Zhixiang Chen, Carlos Cunha, and Steven Homer. Finding a hidden code by asking questions. In *Proceedings of the 2nd Annual International Conference on Computing and Combinatorics (COCOON'96)*, pages 50–55. Springer, 1996.

[Chv83] Vasek Chvátal. Mastermind. *Combinatorica*, 3:325–329, 1983.

[CLTY09] Tianshi Chen, Per Kristian Lehre, Ke Tang, and Xin Yao. When is an estimation of distribution algorithm better than an evolutionary algorithm? In *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC'09)*, pages 1470–1477. IEEE, 2009.

[CTCY10] Tianshi Chen, Ke Tang, Guoliang Chen, and Xin Yao. Analysis of computational time of simple estimation of distribution algorithms. *IEEE Transactions on Evolutionary Computation*, 14(1):1–22, Feb. 2010.

[DF11] Benjamin Doerr and Mahmoud Fouz. Asymptotically optimal randomized rumor spreading. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP'11)*, pages 502–513, 2011.

[DHK07] Benjamin Doerr, Edda Happ, and Christian Klein. A tight analysis of the (1+1)-EA for the single source shortest path problem. In *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC'07)*, pages 1890–1895. IEEE, 2007.

[DHK08] Benjamin Doerr, Edda Happ, and Christian Klein. Crossover can provably be useful in evolutionary computation. In *Proceedings of the 10th Annual Genetic and Evolutionary Computation Conference (GECCO'08)*, pages 539–546. ACM, 2008.

[DJ10] Benjamin Doerr and Daniel Johannsen. Edge-based representation beats vertex-based representation in shortest path problems. In *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference (GECCO'10)*, pages 758–766. ACM, 2010.

[DJK$^+$11] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Per Kristian Lehre, Markus Wagner, and Carola Winzen. Faster black-box algorithms through higher arity operators. In *Proceedings of the 11th ACM Workshop on Foundations of Genetic Algorithms (FOGA'11)*, pages 163–172. ACM, 2011.

[DJS$^+$10] Benjamin Doerr, Thomas Jansen, Dirk Sudholt, Carola Winzen, and Christine Zarges. Optimizing monotone functions can be difficult. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN'10), Part I*, LNCS 6238, pages 42–51. Springer, 2010.

[DJTW03] Stefan Droste, Thomas Jansen, Karsten Tinnefeld, and Ingo Wegener. A new framework for the valuation of algorithms for black-box optimization. In *Proceedings of the 7th Workshop on Foundations of Genetic Algorithms (FOGA'03)*, pages 253–270. Morgan Kaufmann, 2003.

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

[DJW06] Stefan Droste, Thomas Jansen, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39:525–544, 2006.

[DJW10a] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Drift analysis and linear functions revisited. In *Proceedings of 2010 IEEE Congress on Evolutionary Computation (CEC'10)*, pages 1967–1974. IEEE, 2010.

[DJW10b] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. In *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference (GECCO'10)*, pages 1449–1456, 2010.

[DKLW11] Benjamin Doerr, Timo Kötzing, Johannes Lengler, and Carola Winzen. Black-box complexities of combinatorial problems. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 981–988. ACM, 2011.

[DKW11] Benjamin Doerr, Timo Kötzing, and Carola Winzen. Too fast unbiased black-box algorithms. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 2043–2050. ACM, 2011.

[DNHW03] Martin Dietzfelbinger, Bart Naudts, Clarissa Van Hoyweghen, and Ingo Wegener. The analysis of a recombinative hill-climber on H-IFF. *IEEE Transactions on Evolutionary Computation*, 7(5):417–423, Oct. 2003.

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomised Algorithms.* Cambridge University Press, 2009.

[DS04] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization.* MIT Press, 2004.

[DW11a] Benjamin Doerr and Carola Winzen. Breaking the $O(n \log n)$ barrier of Leading-Ones. In *Artificial Evolution (EA'11)*, 2011. Accepted for presentation.

[DW11b] Benjamin Doerr and Carola Winzen. Memory-restricted black-box complexity. *ECCC TR11-092*, 2011.

[DW11c] Benjamin Doerr and Carola Winzen. Towards a complexity theory of randomized search heuristics: Ranking-based black-box complexity. In *Proceedings the 6th International Computer Science Symposium in Russia (CSR'11)*, pages 15–28. Springer, 2011.

[ER63] Paul Erdős and Alfréd Rényi. On two problems of information theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 8:229–243, 1963.

[FK86] Johannes B. G. Frenk and Alexander H. G. Rinnooy Kan. The rate of convergence to optimality of the LPT rule. *Discrete Applied Mathematics*, 14:187–197, 1986.

[FW05] Simon Fischer and Ingo Wegener. The one-dimensional Ising model: Mutation versus recombination. *Theoretical Computer Science*, 344(2-3):208–225, 2005.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[Goo09] Michael T. Goodrich. On the algorithmic complexity of the mastermind game with black-peg results. *Information Processing Letters*, 109:675–678, 2009.

[GSW09] Michael Gnewuch, Anand Srivastav, and Carola Winzen. Finding optimal volume subintervals with k points and calculating the star discrepancy are NP-hard problems. *Journal of Complexity*, 25:115–127, 2009.

[Gut07] Walter J. Gutjahr. Mathematical runtime analysis of ACO algorithms: Survey on an emerging issue. *Swarm Intelligence*, 1:59–79, 2007.

[GWW11] Michael Gnewuch, Magnus Wahlström, and Carola Winzen. A randomized algorithm based on threshold accepting to approximate the star discrepancy. *CoRR*, abs/1103.2102, 2011.

[Hro01] Juraj Hromkovič. *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Springer, 2001.

[HY04] Jun He and Xin Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3:21–35, 2004.

[JW02] Thomas Jansen and Ingo Wegener. The analysis of evolutionary algorithms - a proof that crossover really can help. *Algorithmica*, 34:47–66, 2002.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KE01] James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[KK09] Petr Kovár and Michael Kubesa. Factorizations of complete graphs into spanning trees with all possible maximum degrees. In *Proceedings of the 20th International Workshop on Combinatorial Algorithms (IWOCA'09)*, pages 334–344. Springer, 2009.

[Knu77] Donald E. Knuth. The computer as a master mind. *Journal of Recreational Mathematics*, 9:1–5, 1977.

[KST11] Timo Kötzing, Dirk Sudholt, and Madeleine Theile. How crossover helps in pseudo-Boolean optimization. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 989–996, 2011.

[LL02] Pedro Larrañaga and José A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[LTT89] Donna Crystal Llewellyn, Craig Tovey, and Michael Trick. Local optimization on graphs. *Discrete Applied Mathematics*, 23:157–178, 1989. Erratum: 46:93–94, 1993.

[LW10a] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. In *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference (GECCO'10)*, pages 1441–1448. ACM, 2010.

[LW10b] Per Kristian Lehre and Carsten Witt. Black-box search by unbiased variation. *Electronic Colloquium on Computational Complexity (ECCC)*, page 16, 2010.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[Müh92] Heinz Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In *Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature (PPSN'92)*, pages 15–26. Elsevier, 1992.

[NW04] Frank Neumann and Ingo Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. In *Proceedings of the 6th Annual Genetic and Evolutionary Computation Conference (GECCO'04)*, pages 713–724. Springer, 2004.

[NW07] Frank Neumann and Ingo Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378:32–40, 2007.

[NW09] Frank Neumann and Carsten Witt. Runtime analysis of a simple ant colony optimization algorithm. *Algorithmica*, 54:243–255, 2009.

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

[OHY08] Pietro S. Oliveto, Jun He, and Xin Yao. Analysis of population-based evolutionary algorithms for the vertex cover problem. In *Proceedings of IEEE World Congress on Computational Intelligence (WCCI 2008), Hong Kong, June 1-6, 2008*, pages 1563–1570, 2008.

[Rob55] Herbert Robbins. A remark on Stirling's formula. *The American Mathematical Monthly*, 62:26–29, 1955.

[RS09] Joachim Reichel and Martin Skutella. On the size of weights in randomized search heuristics. In *Proceedings of the 10th ACM Workshop on Foundations of Genetic Algorithms (FOGA'09)*, pages 21–28. ACM, 2009.

[Rud97] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovac, 1997.

[RV11] Jonathan Rowe and Michael Vose. Unbiased black box search algorithms. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 2035–2042. ACM, 2011.

[STW02] Jens Scharnow, Karsten Tinnefeld, and Ingo Wegener. Fitness landscapes based on sorting and shortest path problems. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN'02)*, pages 54–63. Springer, 2002.

[STW04] Jens Scharnow, Karsten Tinnefeld, and Ingo Wegener. The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms*, pages 349–366, 2004.

[Sud05] Dirk Sudholt. Crossover is provably essential for the Ising model on trees. In *Proceedings of the 7th Annual Genetic and Evolutionary Computation Conference (GECCO'05)*, pages 1161–1167, 2005.

[SW04] Tobias Storch and Ingo Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320(1):123–134, 2004.

[SZ06] Jeff Stuckman and Guo-Qiang Zhang. Mastermind is NP-complete. *INFOCOMP Journal of Computer Science*, 5:25–28, 2006.

[Win11] Carola Winzen. Direction-reversing quasi-random rumor spreading with restarts. *CoRR*, abs/1103.2429, 2011.

[Wit05] Carsten Witt. Worst-case and average-case approximations by simple randomized search heuristics. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS'05)*, pages 44–56. Springer, 2005.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 222–227, 1977.

[Zha06] Shengyu Zhang. New upper and lower bounds for randomized and quantum local search. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06)*, pages 634–643. ACM, 2006.

A Further Contributions

A.1. Theory of Randomized Search Heuristics

Multiplicative Drift Analysis

In this work we introduce multiplicative drift analysis as a suitable way to analyze the runtime of randomized search heuristics such as evolutionary algorithms.

We give a multiplicative version of the classical drift theorem. This allows easier analyses in those settings where the optimization progress is roughly proportional to the current distance to the optimum.

To display the strength of this tool, we regard the classical problem of how the (1+1) evolutionary algorithm optimizes an arbitrary linear pseudo-Boolean function. Here, we first give a relatively simple proof for the fact that any linear function is optimized in expected number of $O(n \log n)$ function evaluations, where n is the length of the bit string. Afterwards, we show that in fact any such function is optimized in expected time at most $(1 + o(1))1.39en \log n$, again using multiplicative drift analysis. We also prove a corresponding lower bound of $(1 - o(1))en \log n$ which actually holds for all functions with a unique global optimum.

We further demonstrate how our drift theorem immediately gives natural proofs (with better constants) for the best known runtime bounds for the (1+1) EA on combinatorial problems like finding minimum spanning trees, shortest paths, or Euler tours in graphs.

In: *Proceedings of Gecco '10* (cf. [DJW10b]). Invited and submitted to a special issue of *Algorithmica*.

Non-Existence of Linear Universal Drift Functions

Drift analysis is a powerful tool to prove upper and lower bounds on the runtime of randomized search heuristics. Its most famous application is a simple proof for the classical problem how the (1+1) evolutionary algorithm optimizes linear pseudo-Boolean functions. A relatively simple potential function allows to track the progress of the EA optimizing any linear function.

In this work, we show that such beautiful proofs cease to exist if the mutation probability is slightly larger than the standard value of $1/n$. In fact, we show that no universal liner drift function exists once we increase the mutation probability to c/n for some small constant $c > 1$.

In: *Proceedings of Gecco '10 and CEC '10* (cf. [DJW10b, DJW10a]). A journal version is currently under submission.

Mutation Rate Matters Even When Optimizing Monotone Functions

Extending previous analyses on function classes like linear functions, we analyze how the (1+1) evolutionary algorithm optimizes pseudo-Boolean functions that are strictly monotone. These functions have the property that whenever only 0-bits are changed to 1, then the objective value strictly increases. Contrary to what one would expect, not all of these functions are easy to optimize. The choice of the constant c in the mutation probability $p(n) = c/n$ can make a decisive difference.

We show that if $c < 1$, then the (1+1) EA finds the optimum of every such function in $\Theta(n \log n)$ iterations. For $c = 1$, we can still prove an upper bound of $O(n^{3/2})$. However, for $c \geq 16$, we present a strictly monotone function such that the (1+1) EA with overwhelming probability needs $2^{\Omega(n)}$ iterations to find the optimum. This is the first time that we observe that a constant factor change of the mutation probability changes the run-time by more than constant factors.

In: *Proceedings of PPSN '10* (cf. [DJS$^+$10]). A journal version is currently under submission.

A.2. Estimating Geometric Discrepancies

A New Randomized Algorithm to Approximate the Star Discrepancy Based on Threshold Accepting

We present a new algorithm for estimating the star discrepancy of arbitrary point sets. Similar to the algorithm for discrepancy approximation of Winker and Fang [SIAM J. Numer. Anal. 34 (1997), 2028–2042] it is based on the optimization algorithm Threshold Accepting. Our improvements include, amongst others, a non-uniform sampling strategy, which is more suited for higher-dimensional inputs and which additionally takes into account the topological characteristics of given point sets, and rounding steps, which transform axis-parallel boxes, on which the discrepancy is to be tested, into *critical test boxes*. These critical test boxes provably yield higher discrepancy values, and contain the box that exhibits the maximum value of the local discrepancy. We provide comprehensive experiments to test the new algorithm. Our randomized algorithm computes the exact discrepancy frequently in all cases where this can be checked (i.e., where the exact discrepancy of the point set can be computed in feasible time). Most importantly, in higher dimension the new method behaves clearly better than all previously known methods.

This result will be presented at the Monte Carlo and Quasi-Monte-Carlo Methods in Scientific Computing conference (MCQMC 2012). A technical report is available on arXiv [GWW11]. A journal version is currently under submission.

A.3. Randomized Rumor Spreading

Direction-Reversing Quasi-Random Rumor Spreading with Restarts

In a recent work, Doerr and Fouz [DF11] present a new quasi-random PUSH algorithm for the rumor spreading problem (also known as gossip spreading or message propagation problem). Their *hybrid protocol* outperforms all known PUSH protocols.

In this work, we add to the hybrid protocol a direction-reversing element. We show that this *direction-reversing quasi-random rumor spreading protocol with random restarts* yields a constant factor improvement over the hybrid model, if we allow the same dose of randomness.

Put differently, our protocol achieves the same broadcasting time as the hybrid model by employing only roughly half the number of random choices.

This result is available on arXiv [Win11]. A journal version is currently under submission.

B Curriculum Vitae

Personal Details

Name	:	Carola Winzen
Date of birth	:	March 5, 1984
Address	:	Ohmstr. 17, 66123 Saarbrücken, Germany

Professional Experience

Since Jan 2010	:	Max-Planck-Institut für Informatik, Saarbrücken Department 1: Algorithms and Complexity
Dec 2007 to Nov 2009	:	Business Consultant at McKinsey&Company, Inc. Focus on network optimization strategies in logistics
July to Oct 2006	:	Internship with Deutsche Lufthansa AG Department for Network Planning Data and Systems
Oct 2004 to Feb 2007	:	Teaching Assistant at the Department of Mathematics, Kiel

Education

Since Jan 2010	:	PhD student, Universität des Saarlandes, Saarbrücken
August 2007	:	Diploma in mathematics. Final grade: sehr gut Christian-Albrechts-Universität zu Kiel
Oct 2005	:	Intermediate diploma/Vordiplom in mathematics
Feb 2005	:	Intermediate diploma/Vordiplom in economics

June 2003	:	Abitur at the Heinrich-Suso-Gymnasium, Konstanz Final grade: 1.0
2000 to 2001	:	1-year school exchange program in Tobatí, Paraguay

Awards and Scholarships

July 2010	:	Best Paper Award at GECCO 2010
Since Jan 2010	:	Google Europe Fellowship in Randomized Algorithms
March 2007	:	Admission to the smART program for excellent interns Deutschen Lufthansa AG
2006 to 2007	:	Scholarship from e-fellows.net
May 2005	:	Award from the Association of Friends and Alumni of the Department of Mathematics Kiel
Nov 2004 to Aug 2007	:	Klaus-Murmann Fellowship Program Stiftung der Deutschen Wirtschaft
July 2003	:	Karl-von-Frisch-Award, Verband Deutscher Biologen

Academic Activities

Program committee	:	GECCO 2011 (Genetic Algorithms track)
Reviews for conferences	:	RANDOM 2011, STACS 2011 & 2012, FOGA 2011, CEC 2011, PPSN 2010
Reviews for journals	:	Theory of Computing Systems, Information Processing Letters
Student advisor	:	Vijay Ingalalli (Master student, “Estimating Geometric Star Discrepancies via Evolutionary Algorithms”)
Teaching (Universität des Saarlandes)	:	Randomized Algorithms (TA), Reading Groups in Algorithms (with Kurt Mehlhorn)
Talks	:	GECCO 2011, CSR 2011, Chinese Academy of Sciences 2011, SPP Treffen “Algorithm Engineering” 2011, ThRaSH 2010 & 2011, Dagstuhl seminar 10361, KolKom 2010, CEC 2010, DTU Copenhagen 2010, Universität zu Kiel 2010, Mittagsseminar MPI 2010 & 2011

Extracurricular Engagement at the University

since 2010	:	PhD representative, Algorithms and Complexity group
2005 to 2007	:	Spokesman of the Group of scholarship holders in Kiel
2005 to 2007	:	Organizer of the “MINToring”-Project in Kiel
2004 to 2007	:	Member of various bodies of Kiel University

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, 23. September 2011

Carola Winzen