Theoretical Computer Science 471 (2013) 84–106

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Black-box complexities of combinatorial problems☆

Benjamin Doerr [a], Timo Kötzing [a], Johannes Lengler [b], Carola Winzen [a,*]

[a] Max Planck Institute for Informatics, Saarbrücken, Germany
[b] ETH Zürich, Institut für Theoretische Informatik, Zürich, Switzerland

ARTICLE INFO

Article history:
Received 4 January 2012
Received in revised form 31 August 2012
Accepted 26 October 2012
Communicated by X. Yao

Keywords:
Black-box complexity
Runtime analysis
Combinatorial optimization
Theory

ABSTRACT

Black-box complexity, counting the number of queries needed to find the optimum of a problem without having access to an explicit problem description, was introduced by Droste, Jansen, and Wegener [S. Droste, T. Jansen, I. Wegener, Upper and lower bounds for randomized search heuristics in black-box optimization, Theory of Computing Systems 39 (2006) 525–544] to measure the difficulty of solving an optimization problem via generic search heuristics such as evolutionary algorithms, simulated annealing or ant colony optimization.

Since then, a number of similar complexity notions were introduced. However, so far these new notions were only analyzed for artificial test problems. In this paper, we move a step forward and analyze the different black-box complexity notions for two classic combinatorial problems, namely the minimum spanning tree and the single-source shortest path problem. Besides proving bounds for their black-box complexities, our work reveals that the choice of how to model the optimization problem has a significant influence on its black-box complexity. In addition, when regarding the unbiased (symmetry-invariant) black-box complexity of combinatorial problems, it is important to choose a meaningful definition of unbiasedness.

1. Introduction

Black-box complexity is a notion trying to capture the difficulty of solving a problem with general-purpose, possibly randomized, search heuristics (sometimes called black-box optimization algorithms, since they do not exploit an explicit representation of the problem instance, but only have access to the objective values of the solution candidates). Roughly speaking, the black-box complexity of a problem (a class of functions to be optimized) is the expected number of function evaluations needed to find the optimum of an unknown member of the class. This is the *unrestricted* black-box model, which was first analyzed in the context of randomized search heuristics by Droste, Jansen, and Wegener [2].

This unrestricted black-box model sometimes gives unrealistically small complexity values—unrealistically as compared with runtimes exhibited by standard randomized search heuristics. A way to overcome this is to restrict the class of randomized algorithms regarded (of course, in a way that as many classic randomized search heuristics as possible are still included). To this aim, Lehre and Witt [3] suggested an *unbiased black-box* model. In their model, algorithms are only allowed to generate new solutions from existing ones, and only via so-called unbiased variation operators, which roughly speaking have to be invariant under problem symmetries. Doerr and Winzen [4] regard the restriction that the algorithm has no access to the absolute objective values of solutions, but only to the ranking implied by their fitnesses. This leads to an (unrestricted or unbiased) *ranking-based black-box* model.

☆ A preliminary version of this work has been presented at the Genetic and Evolutionary Computation Conference (GECCO 2011) Doerr et al. (2011) [1].

* Correspondence to: Max Planck Institute for Informatics, AG 1, Campus E 1 4, 66123 Saarbrücken, Germany. Tel.: +49 681 9325 1013; fax: +49 681 9325 1099.
E-mail address: winzen@mpi-inf.mpg.de (C. Winzen).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.10.039

A number of deep and sometimes unexpected results exist for these new notions. Unfortunately, these only regard artificial test problems like OneMax, LeadingOnes, or Jump functions. The focus of this paper is to undertake an in-depth analysis of the different black-box complexity notions for combinatorial problems. Besides the natural interest in the problem difficulty of combinatorial optimization problems, a second motivation is to learn how the previously defined complexity notions behave in these more complicated settings. For this reason, we regard both the minimum spanning tree (MST) problem, where it is generally agreed on that a bit-string representation is most natural, and the single-source shortest path (SSSP) problem, where current-best evolutionary approaches use representations different from bit-strings. Here, for example, it is not clear a priori how to extend the notion of unbiasedness of Lehre and Witt [3] to combinatorial representations.

1.1. Minimum spanning trees

In one of the earliest theoretical works on evolutionary algorithms for combinatorial optimization problems, Neumann and Wegener [5,6] analyze the optimization time of the $(1+1)$ evolutionary algorithm (EA) for the MST problem. They prove that the expected number of generations until an MST is found, is $O(m^2 \log(nw_{\max}))$, where n is the number of vertices, m the number of edges and $w_{\max}$ is the maximum of the positive and integral edge weights. Since then, it remains a major open problem to decide whether the dependence on the maximum edge weight is necessary.

The same bound was 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 [7,8] 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).

Since the MST problem has a natural representation via bit-strings, for this combinatorial problem we can without much further discussion use the four black-box complexity notions discussed above. The only minor detail to take care of is that in the MST usually the fitness is a two-criteria one, that is, the fitness function returns both the number of connected components and the total weight of the solution. This is the natural way to obtain feasible, namely connected, solutions. For all black-box notions apart from the ranking-based one, this provides no difficulties. For the ranking-based black-box complexity, allowing only relative comparisons of fitnesses, but not the access to the absolute values, we have to decide how to deal with the two-criteria fitness. Since one motivation of ranking-based black-box complexity is to obtain results that are invariant under rescaling the fitness (that is, under monotonic transformations), it makes sense to use the ranking information for each component of the fitness separately (as it is hard to imagine a reasonable rescaling notion that does not treat both components separately).

The original work of Neumann and Wegener [5,6] considers a single-criterion fitness function, where the number of connected components exceeding one is penalized by some large value $C > n^2 w_{\max}$. As is easy to see (and is argued for in Section 5.2), in the non ranking-based settings, the value C can be computed easily. Once C is known, the single-criterion and the two-criterion fitness functions provide the same information. Therefore, all bounds presented for the non ranking-based black-box models hold also for the fitness function used in [5,6].

Our results for the different black-box complexities of the MST problem are summarized in Table 1. A particular observation to derive from the table is that by allowing higher-arity operators, we gain lower unbiased black-box complexities. A similar observation was made for the run-time of concrete evolutionary algorithms for the all-pairs shortest path problem recently (see [9–11]).

1.2. Single-source shortest paths

In another one of the earliest theoretical works on evolutionary algorithms for combinatorial optimization problems, Scharnow, Tinnefeld, and Wegener [12,13] analyze how a $(1+1)$ EA solves the SSSP problem.

Since in the SSSP problem shortest paths between the source and any other vertex are sought for, a bit-string representation for solution candidates is 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 [13], 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 [14].

In addition, also the choice of the fitness function is subtle. In [13], a *multi-criteria fitness* function was suggested. It returns, for a given solution, the vector of distances of each vertex (except the source) from the source (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, Scharnow, Tinnefeld, and Wegener prove an expected optimization time of $O(n^3)$. Doerr, Happ, and Klein [15] improved this to a bound of $O(n^2 \max\{\ell, \log n\})$, where ℓ is the smallest height of a shortest path tree.

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.

Table 1
Upper and lower bounds for the black-box complexity of MST in different models. Abbreviations: unrestr. = unrestricted, rb = ranking-based, unb. = unbiased. † $O(mn\log(m/n))$ if all edge weights are distinct.

	(rb) unrestr.	$*$-ary unb.	Unary unb.
Upper bound	$2m+1$	$O(m)$	$O(mn\log n)^\dagger$
Lower bound	$n-2$	$\Omega(n)$	$\Omega(m\log n)$
	rb unary unb.	(rb) binary unb.	(rb) 3-ary unb.
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 2
Upper bounds for the black-box complexity of the SSSP problem with the single-criteria fitness function in different models. A lower bound of $\Omega(n^2)$ for the redirecting unbiased black-box complexity is shown in Theorem 30. Abbreviations: unrestr. = unrestricted, rb = ranking-based, redir = redirecting unbiased.

	unrestr.	rb unrestr.	rb unary redir	binary redir
Upper bound	$n(n-1)/2$	$(n-1)^2$	$O(n^3)$	$O(n^2\log n)$

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.[1] Droste, Jansen, and Wegener [2] showed that the unrestricted black-box complexity of the SSSP problem 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.[2] 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 3 in Section 5.1. 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 (e.g., by adding expensive dummy edges).

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 [13] 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 [16], it was observed that this (artificial) problem dissolves if each unconnected vertex only contributes a large value (e.g., a value larger than the sum of all edge weights) to the objective value. This is the common way to implement the ∞-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. Again, it is a well-known open problem whether the dependence on $w_{\max}$ is necessary or not.

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. In this work, 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, the operators do not regard the *labels* of different nodes, but only their structure (cf. Definitions 20–22 in Section 5.2.2 for the formal definition of the different unbiasedness notions), as well as the *generalized* unbiased model as proposed by Rowe and Vose [17] yield almost the same black-box complexities as the unrestricted black-box model. As we shall prove in Section 5, the 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 models seem to be a very natural extension of the unbiasedness notion of Lehre and Witt [3].

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 path 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 2 gives a brief summary of our main findings. More results are given in Section 5.

2. The four black-box models

In Section 2.1 we present the two basic black-box models, the *unrestricted* black-box model by Droste, Jansen, and Wegener [2] and the *unbiased* model by Lehre and Witt [3]. Furthermore, we shall introduce the unrestricted and the

[1] The only other result published on the black-box complexity of a combinatorial problem is the proof that the *NP*-complete MaxClique problem has a polynomial black-box complexity, see again [2].

[2] Note that the upper bound in [2] still holds in a more restricted setting, cf. Section 5.1.

unbiased *ranking-based* black-box models in Section 2.2. We give only a short presentation of the models and refer the reader to [2–4] for a more detailed exposure.

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 $[k] := \{1, \dots, k\}$. Analogously, we define $[0..k] := [k] \cup \{0\}$. 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 $\bar{x}_i = 1 - x_i$). The bitwise exclusive OR is denoted by $\oplus$. By $|x|_1 = \sum_{i \in [n]} x_i$ we denote the Hamming weight of x. For any set S we denote by 2^S the power set of S, i.e., the set of all subsets of S. By S_n we denote the set of all permutations of $[n]$. Lastly, with $\log_a$ we denote the logarithm to base a, and with log we denote the natural logarithm to base $e := \exp(1)$.

2.1. The unrestricted and the unbiased black-box model

We are interested in measuring the complexity of a problem's *optimizability by randomized search heuristics*. Black-box complexity follows the usual approach to take as a measure the *performance of the best algorithm* out of some class of algorithms. As our main interest is in the performance of randomized search heuristics, we restrict our attention to the class of all algorithms which obtain information about the problem to be solved only by learning the objective value of possible solutions (black-box optimization algorithms). The objective function is given by an oracle, or as a *black-box*. Using this oracle, the algorithm may query the objective value of all possible solutions, but any such query does only return this solution's objective value and no other information about the objective function.

Naturally, we allow that the algorithms are adaptive and that they use random decisions. However, the only type of action the algorithm may perform is, based on the objective values learned so far, deciding on a probability distribution on the search space $\mathcal{S}$, sampling from it a solution ("search point") $x \in \mathcal{S}$, and querying its objective value ("fitness") from the oracle. This leads to the **unrestricted black-box model** by Droste, Jansen, and Wegener [2] which contains all algorithms following the scheme of an *unrestricted black-box algorithm*, cf. Algorithm 1.

In typical applications of randomized search heuristics, the evaluation of the fitness of a search point is more costly than the generation of a new one. Thus, we take as performance measure of a black-box algorithm the number of queries to the oracle until the algorithm queries an optimal solution for the first time. Since we mainly talk about randomized algorithms, we regard the *expected* number of such queries and call this value the *runtime* of the black-box algorithm.

For a class $\mathcal{F}$ of functions defined on $\mathcal{S}$, the *complexity* of an algorithm A for $\mathcal{F}$ is the worst-case runtime, i.e., the maximum runtime of A on a function $f \in \mathcal{F}$. The complexity of $\mathcal{F}$ with respect to a class $\mathcal{A}$ of algorithms is the minimum ("best") complexity among all $A \in \mathcal{A}$ for $\mathcal{F}$. Hence, the *unrestricted black-box complexity* of $\mathcal{F}$ is the complexity of $\mathcal{F}$ with respect to the class of all unrestricted black-box algorithms.

Algorithm 1: Scheme of an unrestricted black-box algorithm

1 **Initialization:** Sample x^0 according to some probability distribution p^0 on $\mathcal{S}$. Query $f(x^0)$.
2 **Optimization: for** $t = 1, 2, 3, \dots$ **until** *termination condition met* **do**
3 Depending on $\big((x^0, f(x^0)), \dots, (x^{t-1}, f(x^{t-1}))\big)$ choose a probability distribution p^t on $\mathcal{S}$.
4 Sample x^t according to p^t, and query $f(x^t)$.

Already the authors of [2] noted that the unrestricted black-box is very powerful. As an example, consider a single objective function f. Clearly, the unrestricted black-box complexity of $\{f\}$ is 1—the algorithm which queries an optimal solution of f as first action shows this bound.

This motivated Lehre and Witt [3] to introduce a more restrictive black-box model, the *unbiased black-box model*, where algorithms may generate new solution candidates only from random or previously generated search points and only by using *unbiased* variation operators. Note that Lehre and Witt formulated their model only for the hypercube $\{0,1\}^d$ as search space. In Section 5, we propose two ways to carry over the notion to a different setting.

We note that all search spaces considered in this paper are finite. However, the following definition applies more generally.

Definition 1. Let $k \in \mathbb{N}$. Let $\mathcal{S}$ be a 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 $\big(D(\cdot \mid y^1, \dots, y^k)\big)_{y^1,\dots,y^k \in \mathcal{S}}$ over $\mathcal{S}$ such that for all *inputs* $y^1, \dots, y^k \in \mathcal{S}$ the condition

$$\forall x \in \mathcal{S}\ \forall g \in \mathcal{G} : D(x \mid y^1, \dots, y^k) = D(g(x) \mid g(y^1), \dots, g(y^k))$$

holds.

An operator sampling from a k-ary, $\mathcal{G}$-unbiased distribution is called a *k-ary, $\mathcal{G}$-unbiased variation operator*.

1-ary (i.e., mutation only) operators are also called *unary* and we refer to 2-ary (i.e., crossover type) operators as *binary* ones. If arbitrary arity is considered, we call the corresponding model the $*$-ary, $\mathcal{G}$-unbiased black-box model. If no confusion can arise, we use the term *unbiased* instead of *$\mathcal{G}$-unbiased*.

For a family of probability distributions $\big(D(\cdot \mid y^1, \ldots, y^k)\big)_{y^1,\ldots,y^k \in \mathcal{S}}$ we also write $\big(D(\cdot \mid \vec{y})\big)_{\vec{y} \in \mathcal{S}^k}$ or just $\big(D(\cdot \mid \vec{y})\big)_{\vec{y}}$ when the search space is understood.

A k-ary, $\mathcal{G}$-unbiased black-box algorithm [3] can now be described via the scheme of Algorithm 2. 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 2: 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** $t = 1, 2, 3, \ldots$ **until** *termination condition met* **do**
3 Depending on $\big(f(x^0), \ldots, f(x^{t-1})\big)$ choose $\ell \leq k$ indices $i_1, \ldots, i_\ell \in [0..t-1]$, and a ℓ-ary, $\mathcal{G}$-unbiased distribution $D(\cdot \mid x^{i_1}, \ldots, x^{i_\ell})$.
4 Sample x^t according to $D(\cdot \mid x^{i_1}, \ldots, x^{i_\ell})$ and query $f(x^t)$.

Lehre and Witt formulated their model for the hypercube $\{0,1\}^d$ 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 $z \in \{0,1\}^d$ is an XOR shift in direction z and $\sigma \in S_d$ is a permutation of the bit positions (cf. Lemma 3). 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.

Contrary to the unrestricted model, Lehre and Witt [3] could show that all pseudo-Boolean functions with a single global optimum have a unary Hamming-unbiased black-box complexity of $\Omega(n \log n)$, a bound which, for several standard test problems, is met by different unary randomized search heuristics, such as the $(1+1)$ EA or the randomized local search algorithm RLS. For results on higher arity models we refer to the work of Doerr et al. [18].

2.2. Ranking-based black-box models

Another possible restriction of the black-box model was introduced by Doerr and Winzen [4]. The authors observed that many standard randomized search heuristics do not take advantage of knowing *exact* objective values. Rather, for creating the next search points, many randomized search heuristics always select those individuals with largest fitness values, examples are given below.

Definition 2. Let $\mathcal{S}$ be a finite set, let $f : \mathcal{S} \to \mathbb{R}$ be a function, and let $\mathcal{C}$ be a subset of $\mathcal{S}$. The *ranking* ρ of $\mathcal{C}$ with respect to f assigns to each element $c \in \mathcal{C}$ the number of elements in $\mathcal{C}$ with a smaller f-value plus 1, formally, $\rho(c) := 1 + \big|\{c' \in \mathcal{C} \mid f(c') < f(c)\}\big|$.

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

Following [4], we restrict the two black-box models which we introduced in the previous section to black-box algorithms that use no other information than this ranking.

Unrestricted Ranking-Based Black-Box Model. The unrestricted ranking-based black-box model can be described via the scheme of Algorithm 1 where we replace the third line by "Depending on the ranking of $\{x^0, \ldots, x^{t-1}\}$ with respect to f, choose a probability distribution p^t on $\mathcal{S}$."

Unbiased Ranking-Based Black-Box Model. For the definition of the unbiased ranking-based model we consider the scheme of Algorithm 2 and replace the third line by "Depending on the ranking of $\{x^0 \ldots, x^{t-1}\}$ with respect to f, choose $\ell \leq k$ indices $i_1, \ldots, i_l \in [0..t-1]$ and a ℓ-ary, $\mathcal{G}$-unbiased distribution $D(\cdot \mid x^{i_1}, \ldots, x^{i_l})$."

Both ranking-based black-box models capture many common search heuristics, such as $(\mu+\lambda)$ evolutionary algorithms, some ant colony optimization algorithms (for example simple versions of Max–Min Ant Systems as analyzed in [19,20], which can be modeled as $*$-ary (unbiased, ranking-based) black-box algorithms), and the randomized local search algorithm RLS. They do not include algorithms like simulated annealing algorithms, threshold accepting algorithms, or evolutionary algorithms using fitness proportional selection.

Doerr and Winzen [4] showed that, while the basic and the ranking-based models yield the same asymptotic bounds for some problems (e.g., the OneMax function class), whether fitness values are considered or not does matter for other problems, e.g., the BinaryValue function class. Here in this work we shall see that the ranking-based black-box complexities may be larger than their non-ranking-based counterparts. However, for both of the problems analyzed in this work, the differences are rather small. This stems from the nature of the two problems MST and SSSP where in both problems the best known algorithms builds a solution in a sequential manner, each time testing to add the least costly edge/node.

Nevertheless we expect the ranking-based notion to make a decisive difference for other combinatorial problems, e.g., the partition problem [21] which we leave for future research.

3. Mathematical background

We have stated in Section 2.1 the following lemma. The statement can be found without proof in [22]. We recall that the Hamming distance of two vectors $x, y \in \{0,1\}^n$ is defined as $H(x,y) := |x \oplus y|_1$.

Lemma 3. *Every bijection of the hypercube $\{0,1\}^d$ that preserves Hamming distance can be written in the form $g(x) = \sigma(x \oplus z)$, where $z \in \{0,1\}^d$ is an XOR shift in direction z and $\sigma \in S_d$ is a permutation of the bit positions. Conversely, every function of the form $g(x) = \sigma(x \oplus z)$ is a bijection that preserves Hamming distance.*

Proof. The converse direction is trivial, since both XOR shifts and permutations of the bit positions are bijective and preserve Hamming distance.

So let g be a bijection of the hypercube that preserves Hamming distance. Since g is a bijection, there is a $z \in \{0,1\}^d$ such that $g(z) = 0$. Consider the function g' on the hypercube defined by $g'(x) := g(x \oplus z)$. Clearly g' is also a bijection and preserves Hamming distance. Furthermore, $g'(0) = 0$. Therefore the map g' also preserves the Hamming *weights* $|.|_1$, since for all $x \in \{0,1\}^d$,

$$|g'(x)|_1 = H(g'(x), 0) = H(g'(x), g'(0)) = H(x, 0) = |x|_1.$$

We show that g' is a permutation of the bit positions. For $i \in [d]$, let x^i be the ith unit vector; i.e., let with $x^i_i = 1$ and $x^i_j = 0$ for $j \neq i$. Then $X := \{x^1, \ldots, x^n\}$ is the set of all vectors in the hypercube of Hamming weight one. Therefore g' must be bijective on X. Let σ be the permutation of bit positions defined by $g'(x^i) = \sigma(x^i)$.

Now consider any vector $x \in \{0,1\}^d$ with Hamming weight $k \geq 1$. Then in the set X there are exactly k vectors $x^{i_1}, \ldots, x^{i_k}$ with Hamming distance $k-1$ from x, and they correspond one-to-one to the one-bits of x. In particular, x is uniquely determined by these positions. Since g' preserves Hamming weights, $g'(x)$ has Hamming weight $|g'(x)|_1 = |x|_1 = k$. In other words, $g'(x)$ contains exactly k one-bits. On the other hand, as g' preserves Hamming distances, we also have

$$H(g'(x), \sigma(x^{i_j})) = H(g'(x), g'(x^{i_j})) = H(x, x^{i_j}) = k - 1$$

for all $j \in [k]$. Since $\sigma(x^{i_j})$ has Hamming weight 1, each $\sigma(x^{i_j})$ must correspond to a one-bit of $g'(x)$. But $g'(x)$ has only k one-bits, and therefore $g'(x) = \sigma(x)$. So we have shown the identity $g'(x) = \sigma(x)$ for all search points of Hamming weight ≥ 1, and it also holds trivially for the zero vector. Therefore, for all $x \in \{0,1\}^d$,

$$g(x) = g'(x \oplus z) = \sigma(x \oplus z),$$

which was to be shown. □

For lower bounds in black-box complexities, the minimax principle by Yao [23] has proven to be very helpful. The following presentation is taken from the book by Motwani and Raghavan [24].

Theorem 4 (*Yao's Minimax Principle [23]*)**.** *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,*

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

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 runtime of algorithm A on input I.

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

Theorem 5 (*Additive Drift [25]*)**.** *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 exist $c > 0$ such that*

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

then

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

If there exist $d > 0$ such that

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

then

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

For the lower bounds we will use repeatedly the fact that a coupon collector that samples uniformly at random from a set of n coupons will need $O(n \log n)$ many samples to collect each coupon at least once with high probability. The following theorem makes this concept precise.

Theorem 6 (*Coupon Collector*). *Let $n \in \mathbb{N}$, $\beta > 0$, and $N \geq \beta \cdot n \log n$. Let $Y_1, \ldots, Y_N$ be independent random variables, drawn uniformly at random from $[n]$. Then the probability that the Y_i do not cover $[n]$ is*

$$\Pr\left[\exists j \in [n] : \forall i \in [N] : Y_i \neq j\right] \leq n^{-\beta+1}.$$

Proof. See [24, Section 3.6.1.]. □

Rather than bounding the expected runtime of an algorithm, it is sometimes easier to show that it solves the given problem with good probability in some number s of iterations. If we are only interested in asymptotic black-box complexities, the following remark allows us to use such statements for upper bounds. Its proof is a standard argument often used in randomized computation. The statement is sometimes referred to as "repetition of independent phases" or "typical runs" in the evolutionary computation literature [26]. We present it here only for the sake of completeness.

Remark 7. Suppose for a problem P there exists a (unrestricted/$\mathcal{G}$-unbiased), (ranking-based) black-box algorithm $\mathcal{A}$ that, with constant success probability, solves P in s iterations. Then the corresponding (unrestricted/$\mathcal{G}$-unbiased), (ranking-based) black-box complexity of P is at most $O(s)$.

Proof. Let c be an upper bound for the failure probability of algorithm $\mathcal{A}$ after s iterations. We call the s iterations of $\mathcal{A}$ a *run* of $\mathcal{A}$. If X_i denotes the indicator variable for the event that the ith independent run of $\mathcal{A}$ is successful (i.e., computes an optimum), then $\Pr[X_i = 1] \geq 1 - c$. Since the X_i's are i.i.d. random variables, the function $Y := \min\{k \in \mathbb{N} \mid X_k = 1\}$ is a geometric random variable with success probability at least $1 - c$. Hence, $E[Y] \leq (1 - c)^{-1}$, i.e., the expected number of independent runs of $\mathcal{A}$ 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. □

For proving lower bounds, we will often encounter the following (or a similar) situation. Suppose we want to investigate all n Hamming neighbors of a search point x. In an unrestricted setting, we can do this in time $O(n)$. With unary unbiased operations, it is in general not possible to access the neighbors one by one, but we can repeatedly sample from the set of all neighbors of x. In this way, we will see all neighbors in expected time $O(n \log n)$ by the Coupon Collector Theorem. However, since we cannot identify the search points (unless the weight function allows us to do so), we will in general not know when we have seen all neighbors. The following lemma tells us that it still suffices to sample $\beta n \log n$ times, for some constant β. For suitable β, the overall algorithm will have a constant probability of success if we encounter such a situation only polynomially often.

Lemma 8. *Suppose P is a problem on some search space, and that algorithms can access elements of the search space only by queries to an oracle. Let C_1 be a routine that samples uniformly at random from a set of size n until it has sampled all elements, and outputs the oracle's answer together with an assertion that all elements have been accessed. Let $C_2(\beta)$ be a routine that samples $\beta n \log n$ many search points from the same set and returns the oracle's answer.*

Suppose further there exists a probabilistic algorithm $\mathcal{A}_1$ that uses C_1 polynomially often (in n) as a subroutine and solves P in an expected number of s queries. Let $\mathcal{A}_2(\beta)$ be the algorithm that is obtained from $\mathcal{A}_1$ by replacing all calls to C_1 by calls to $C_2(\beta)$. Then there is a constant $\beta > 0$ such that $\mathcal{A}_2(\beta)$ solves P with constant probability. In particular, by repeating $\mathcal{A}_2(\beta)$ sufficiently often, P can be solved in an expected number of $O(s)$ queries.

Proof. We say that a call to the routine $C_2(\beta)$ *fails* if there is a search point that is not sampled by the routine. Clearly the algorithm $\mathcal{A}_2(\beta)$ solves P if the no call of $C_2(\beta)$ fails. Assume $\mathcal{C}_2(\beta)$ is called $f(n)$ many times. Then since f is polynomial, there exists $\beta > 0$ such that $f(n) \cdot n^{-\beta+1} < 1/2$ for all $n \in \mathbb{N}$. But by the Coupon Collector Theorem the failure probability of each call of $C_2(\beta)$ is at most $n^{-\beta+1}$. Therefore, by the union bound, the probability that $\mathcal{A}_2(\beta)$ fails is at most

$$\Pr\left[\mathcal{A}_2(\beta) \text{ fails}\right] \leq f(n) \cdot n^{-\beta+1} < \frac{1}{2}.$$

Finally, repeating the algorithm $\mathcal{A}_2(\beta)$ sufficiently often will solve P in expected time $O(s)$ by Remark 7. □

For this paper, 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 a function assigning a positive real number $w(e)$, called the *weight*, to any given edge e.

4. Black-box complexities of the minimum spanning tree problem

Definition 9 (*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. Finally we assume that all edge weights are positive.

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 [27] 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 (where optimizing means minimizing in lexicographic order). 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, one based on the first component and a second one based on the second component.

4.1. 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 10 (*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* $O(m)$.

We will used the remainder of this section to prove Theorem 10. In particular, in Section 4.1.1 we consider the unrestricted black-box model and in Section 4.1.2 the unbiased one for the different arities.

Finally, in Section 4.2 we give lower bounds for the black-box complexity of MST in both the unrestricted and unbiased black-box models.

4.1.1. 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, $i \in [m]$, where e_i denotes the ith unit vector $(0, \dots, 0, 1, 0, \dots, 0)$ of length m. These first m queries are used to extract the weights of the graph. 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 through 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. □

4.1.2. The unbiased settings

The unbiased model is much 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 (in the unbiased model) or order (in the ranking-based model) the weights of the edges (including multiplicities) and we test the inclusion of the edges in increasing order of their weights in the *third step*. From basic facts about Kruskal's algorithm we know that for edges with the same weight, it does not matter in which order we test them.

In the following, we prove upper bounds for the expected number of queries needed to complete each step. Note that, using Remark 7 and a simple union bound over the success probabilities of the three steps, it suffices to achieve a constant success probability in 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 $\texttt{uniform}()$ which samples a bit string $x \in \{0,1\}^m$ uniformly at random, $\texttt{RLS}(\cdot)$ (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 $\texttt{complement}(\cdot)$, which assigns to every $x \in \{0,1\}^m$ its bit-wise complement $\bar{x}$.

Finally, for all $k \in \mathbb{N}$ we use the binary operators $\texttt{RLS}_k(\cdot,\cdot)$ and $\texttt{dist}_k(\cdot,\cdot)$. Given two bit strings $x, y \in \{0,1\}^m$, $\texttt{RLS}_k(x,y)$ 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. The other operator $\texttt{dist}_k(x,y)$ returns x if the Hamming distance $H(x,y) = \sum_{i=1}^m |x_i - y_i|$ equals k and returns a 1-Hamming neighbor of x otherwise. Since all edge weights are positive, the output of $\texttt{dist}_k(x,y)$ has the same weight as x if and only if x and y have Hamming distance k. Therefore we can decide whether any two given search points have Hamming distance k.

The following is straightforward to verify. For a discussion of the properties of unbiased variation operators confer [3].

Remark 11. $\texttt{uniform}()$ is a (0-ary) unbiased variation operator and both $\texttt{RLS}(\cdot)$ and $\texttt{complement}(\cdot)$ are unary unbiased ones. Finally, for all k, $\texttt{RLS}_k(\cdot,\cdot)$ and $\texttt{dist}_k(\cdot,\cdot)$ are binary unbiased variation operators.

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.*

Summary. The first two steps are straightforward. In the third step we start with the empty graph and build the MST as in Kruskal's algorithm by testing the edges in order of their weight and including them if they do not close a cycle. In

order to test the kth edge, we flip one bit in our current search point until the correct edge is included. Since there are no multiple edge weights, the fitness function tells us which bit has been flipped. This enables us to speed up the third phase in the following way. Assume we reject the kth edge (decide not to include it in the solution). Then our current search point remains unchanged, but now we try to include the $(k+1)$st edge. If we are lucky then we have already tested this edge while we tried to flip the kth bit. In this case, we do not need to test it again. This idea decreases the runtime to $O(mn \log(m/n))$.

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 $y \leftarrow \mathtt{RLS}(x)$ 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 $\mathtt{RLS}(\cdot)$ 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 kth iteration of the second step we create $z \leftarrow \mathtt{RLS}(x^{k-1})$ 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 kth 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 $w_1 \leq \cdots \leq w_m$ 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 kth edge to the MST the *kth phase*. We start by applying $\mathtt{RLS}(\cdot)$ 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 ith one. To test the inclusion of the (t_i+1)st edge, we query $z^{(i,t)} \leftarrow \mathtt{RLS}(y^i)$. 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 $z^{(i,t)} \leftarrow \mathtt{RLS}(y^i)$ 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 ith phase (including the lastly included one). By a coupon collector argument the expected number of queries needed in the ith 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 $\sum_{i=1}^{n-1} k_i \leq m$ we conclude that

$$\sum_{i=1}^{n-1} \log k_i = \log\Big(\prod_{i=1}^{n-1} k_i\Big) \leq \log\big((m/n)^n\big) = n \log(m/n),$$

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

$$O(m \log m) + O(m \log m) + O(mn \log(m/n)) = O(mn \log(m/n)). \quad \square$$

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 (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.

Summary. We use the same methods as before. However, when we flip a bit in order to test the kth edge, we cannot decide which bit has been flipped. Therefore we repeat the one-bit flips sufficiently often such that with high probability the correct bit flip has occurred.

First step. We find the empty graph in 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. The only difference is that we cannot check $w(x) = 0$, so we do not know when we have reached the empty graph. However, if we simply move on to the second step after $m \log m$ many queries then we have a constant success probability for the first step, which suffices by Remark 7.

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 $z^{(i,t)} \leftarrow \mathtt{RLS}(y^i)$ to find, with high probability, the edge with the smallest weight that can be included into the current solution y^i. By Lemma 8, $O(m \log m)$ queries suffice to guarantee a constant success probability of the whole algorithm. 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. Moreover, we can often store useful information in search points; e.g., the set of edges we have investigated so far. A binary operator may use the current search point x and a second point y as inputs, thereby manipulating x with the information stored in y.

Proof of the $O(m \log n)$ bound.

Summary. In the first step, we start with a search point x and its complement y. We randomly flip a bit in which x and y differ, and accept whenever the weight goes down. In this way we store the bits that we have already altered as those bits in which x and y coincide. In the second step, for every edge e we create a graph y^e with edge set $\{e\}$. In order to generate them efficiently, we maintain a vector y whose zero-bits correspond to the edges that are yet unknown. In the third step, we may add an edge e to the current search point x as follows. Flip all bits in x that differ from y^e. In this way we may flip too many edges (in fact, we only wanted to flip e), so we need to flip back all edges but e. We flip back randomly half of the edges that we have flipped before. By comparing the Hamming distances to the empty graph and to y^e we can decide whether edge e is still in the graph. If this is the case then we have halved the distance to x. We iterate this procedure to halve the new distance to x, and so on until the Hamming distance to x has been reduced to one, in which case we know that only e makes the difference. This strategy allows us to include a specific edge with $O(\log m) = O(\log n)$ many queries, and to run Kruskal's algorithm with only $O(m \log n)$ many queries.

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 $y \leftarrow \mathtt{complement}(x)$. We then set $z \leftarrow \mathtt{RLS}_1(x,y)$ with probability $1/2$ and $z \leftarrow \mathtt{RLS}_1(y,x)$ 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. This condition ensures that we never add edges to x or y.

With probability $1/2$, this operation decreases the Hamming distance of x and y by 1. Thus, after an expected number of $2m$ calls to $\mathtt{RLS}_1(\cdot,\cdot)$ we will have $x = y$ (just note that initially x and y had a Hamming distance of m). Since x and y do not share any edges, x and y must be the empty graph then.

Note that choosing $\mathtt{RLS}_1(x,y)$ with probability $1/2$ and $\mathtt{RLS}_1(y,x)$ 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 $z \leftarrow \mathtt{RLS}(x^0)$ that differs from x^0 in exactly one bit. By Lemma 8 we may assume that after these samples we 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 $z \leftarrow \mathtt{RLS}(x^0)$. Since x^0 is the empty graph, 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 ith 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 $z \leftarrow \mathtt{RLS}(x^0)$ 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 the binary unbiased operator $\mathtt{dist}_k(\cdot,\cdot)$. If z contains a new weight, 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 ith 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 $z' \leftarrow \mathtt{RLS}_{(k+1)/2}(y,z)$. 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 $\mathrm{RLS}(x^0)$ 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 edges $e_{\sigma(1)}, \dots, e_{\sigma(m)}$ such that their weights are ordered $w_1 = w(e_{\sigma(1)}) \leq \cdots \leq w_m = w(e_{\sigma(m)})$. 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 ith heaviest edge $e_{\sigma(i)}$, given that we have tested already the inclusion of edge $e_{\sigma(i-1)}$, the *ith 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 $v(e_{\sigma(i)})$.

Now set $z \leftarrow \mathrm{RLS}_{\lfloor (k+1)/2 \rfloor}(x, y^{\sigma(i)})$ and query its objective value $(c(z), w(z))$. As above, by testing 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 $z' \leftarrow \mathrm{dist}_{k-\lfloor (k+1)/2 \rfloor + 2}(x^0, z)$ 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 ith 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 ith 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.

Proof of the $O(m)$ bound.

Summary. We proceed mainly as in the binary case. In particular, we generate for each edge e a graph z^e with edge set $\{e\}$. However, in contrast to the binary case, we can include the edge into the current search point by just one query. More precisely, we flip all edges in the current search point in which z^e and the empty graph differ (i.e., we flip exactly the bit corresponding to e). This operation turns out to be ternary unbiased. In this way, we can build the MST with $O(m)$ many queries.

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 $y \leftarrow \mathrm{complement}(x^0)$, the bit-wise 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 $z \leftarrow \mathrm{RLS}_1(x^0, y)$. We update $y \leftarrow \mathrm{update}(y, z, x^0)$, where $\mathrm{update}(\cdot,\cdot,\cdot)$ is the 3-ary variation operator that can be described as follows. Given $a, b, c \in \{0,1\}^m$, the operator $\mathrm{update}(a, b, c)$ returns c for those positions where a and b coincide and it returns a otherwise. Formally, for all $i \in [m]$ we have $\mathrm{update}(a, b, c)_i := c_i$ if $a_i = b_i$ and we have $\mathrm{update}(a, b, c)_i := a_i$ if $a_i \neq b_i$. We then proceed querying $z \leftarrow \mathrm{RLS}_1(x^0, y)$ and updating $y \leftarrow \mathrm{update}(y, z, x^0)$ 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., $2m$ queries as each step requires two queries). Let us again fix an ordering of the edges $e_{\sigma(1)}, \dots, e_{\sigma(m)}$ such that $w_1 = w(e_{\sigma(1)}) \leq \cdots \leq w_m = w(e_{\sigma(m)})$. 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 ith heaviest 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 $e_{\sigma(1)}, \dots, e_{\sigma(i)}$ 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 $z \leftarrow \mathrm{test}(x, x^0, z^{\sigma(i+1)})$, where $\mathrm{test}(\cdot,\cdot,\cdot)$ 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.

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 $O(m)$. □

4.2. Lower bounds for the MST problem

Theorem 12. *The unrestricted black-box complexity of MST for complete graphs is strictly larger than $n-2$.*

Proof. We apply Yao's minimax principle, Theorem 4. To this end, we show that there exists a probability distribution on the input set of all weighted complete graphs such that every deterministic algorithm needs at least $n-2$ queries in expectation to compute a MST. More precisely, we consider the distribution p on 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 all other edges "expensive".

Intuition. 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). In intuitive terms, this implies that every algorithm needs to learn $(n-2)\log_2 n$ many bits. As each query reveals a number between $2k-n+1$ and $2k$ (k being the number of edges included in the corresponding graph), it provides at most $\log_2 n$ bits of information. Hence, in the worst case, we get a runtime of at least $n-2$ iterations. Rigorously, we will use a drift theorem and show that, in each iteration, the number of consistent possible trees decreases by at most a factor of $1/n$.

Formal proof. Let us 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.

For all t, we let X_t be the (random) number of spanning trees consistent with the information that A has after making t queries. Thanks to Cayley's formula, we have $X_0 = n^{n-2}$. Furthermore, note that $X_t = 1$ if A has sampled the optimal tree, as any instance defines a unique optimal search point. Thus, we let T be the (random) variable denoting the minimal t such that $X_t = 1$; now $E[T]$ is a lower bound on the expected time until A queries the optimum.

In order to apply an *additive* drift theorem, we let, for all t, $Y_t = \log_n(X_t)$, so that $Y_0 = n-1$ and T is characterized as the minimal t with $Y_t = 0$. Using the lower-bound version of the additive drift theorem (Theorem 5), it suffices to show that, for all t, $E[Y_t - Y_{t+1} \mid t < T] \le 1$ to get the desired lower bound of $n-2$ iterations until A samples the optimum.

Suppose now $t < T$ and let S be the set of spanning graphs that is consistent with the first t answers to A; let x be the $t+1$th query. The query x allows for up to n different answers, depending on the number of cheap edges contained in x; let, for all $i \le n$, S_i be the set of spanning trees compatible with the ith possible answer. Then the S_i are pairwise disjoint, and their union is S.

Since the distribution over the input space is uniform over all spanning trees, the probability to receive as answer ith answer is proportional to the size of $|S_i|$. Therefore, the expected number $E[X_{t+1}]$ of trees consistent with the first $t+1$ queries is

$$E[X_{t+1}] = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \cdot |S_i| = \frac{n}{|S|} \cdot \frac{\sum_{i=1}^{n} |S_i|^2}{n} \ge \frac{n}{|S|} \left(\frac{\sum_{i=1}^{n} |S_i|}{n} \right)^2 = \frac{|S|}{n},$$

where we have used the quadratic-arithmetic mean inequality. This shows that $E[Y_t - Y_{t+1} \mid t < T] \le 1$, and, with the additive drift theorem (Theorem 5), the claim. 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 auxiliary problem $\textsc{OneMax}_m$. The search space of $\textsc{OneMax}_m$ is the space $\{0,1\}^m$, and for each vector $x \in \{0,1\}^m$ the objective value is given by $\textsc{OneMax}_m(x) := \sum_{i=1}^{m} x_i$, the number of 1-bits in x.

Theorem 13. *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 $\textsc{OneMax}_m$.*

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 $x \in \{0,1\}^m$,

$$f(x) = (\textsc{OneMax}_m(x), m+1-\textsc{OneMax}_m(x)).$$

In particular, any algorithm optimizing f can be used to optimize $\textsc{OneMax}$ with the exact same number of queries. □

It has been shown in [3] that $\textsc{OneMax}_m$ has a unary unbiased complexity of $\Theta(m \log m) = \Theta(m \log n)$ and already Erdős and Rényi [28] showed that the unrestricted black-box complexity of $\textsc{OneMax}_m$ is $\Theta(m/\log m)$. Since unbiased black-box complexity is bounded from below by the unrestricted black-box complexity, these two results imply the following.

Table 3
Upper and lower bounds for the unrestricted black-box complexity of the SSSP problem with the multi-criteria objective function.

	Arbitrary connected graph	Complete graph
Upper	$n-1$	$\lfloor (n+1)/2 \rfloor + 1$
Lower	$n-1$	$n/4$

Corollary 14. *The unary unbiased black-box complexity of MST is $\Omega(m \log n)$; for all other arities, the unbiased black-box complexity of MST is $\Omega(m/\log n)$.*

5. Single-source shortest path

In this section we analyze the black-box complexity of the single-source shortest path problem in the different black-box models.

Definition 15 (*SSSP*)**.** The *Single-Source Shortest Path* (SSSP) problem consists of a connected graph $G = (V, E)$ 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 $v \in V \setminus \{s\}$, 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 is a tree. Thus, implicitly, SSSP is the problem of finding an optimal tree, and we use search spaces based on this observation.

However, for the SSSP problem, it not as clear as for the MST problem what a good choice of the search space and the objective function is. We regard two approaches: using multi-criteria fitness is considered in Section 5.1 and in Section 5.2 we consider a single-criterion fitness function.

In what follows, we always assume all input graphs to be connected. Without loss of generality we assume that the nodes are labeled by $1, \ldots, n$ and that $s = 1$ is the source for which we need to compute the shortest path tree.

5.1. SSSP with multi-criteria fitness

The paper [2] 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 ∞).

The paradigm underlying the unbiased black-box complexity is that the algorithm should not be allowed to exploit knowledge about solution candidates stemming from their representation, but only information stemming from their fitness and the population history.

For the multi-objective formulation of the SSSP problem, we thus feel that there is little room for unbiasedness. With the fitness explicitly distinguishing the vertices, imposing certain symmetry conditions among the vertices makes little sense. Similar concerns makes us not regard ranking-based black-box complexities for this problem.

From [2] we know that the unrestricted black-box complexity of this problem is lower bounded by $n/2$ and upper bounded by $2n - 3$.[3]

In this section, we first improve both the lower and the upper bound from [2] 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.

Table 3 summarizes our results for these settings.

Theorem 16. *The unrestricted black-box complexity of the* SSSP *problem with arbitrary input graphs is $n-1$.*

Proof. We start with the **upper bound**. Just as in [2], 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 $\{1, \ldots, n\}$ 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 4. 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 5.

[3] Note that the upper bound given in [2] was shown to hold in a restricted setting where the algorithm may only store up to two previous data points. Our algorithm for improving this bound (given in the proof of Theorem 16) 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.

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

$$\sum_{v \in U} \frac{1+t(v)}{k}, \tag{1}$$

as $1/k$ is the probability that a given $v \in U$ is the next vertex in the path after the 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 (1) equals $|U|/k + (k-|U|)/k = 1$. Now the additive drift theorem, Theorem 5, 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 17. *The unrestricted black-box complexity of the* SSSP *problem 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 by querying for $\lfloor (n+1)/2 \rfloor$ spanning trees; then the next query will be for the optimal spanning tree.

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 [29]. 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 4. 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 path tree consists precisely of the cheap edges (and for each i, the chosen j is the predecessor on that shortest path 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 $i-1$ (a *needle-in-the-haystack problem* is a Boolean function that equals one for exactly one argument, and equals zero for all other, cf. [2, Section 4]). Note that the different haystacks are completely independent; hence, the haystack associated with vertex i requires an expected number of $i/2$ (edge) queries [2, 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. □

5.2. SSSP with single-criterion fitness

We have seen in the previous section that the SSSP problem 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 $\Omega(m)$, where m is the number of edges. The reason for this discrepancy is the large amount of information that the objective function contains. So in order to obtain a more realistic black-box model, we study a more restrictive objective function. We will also find that it is possible to define unbiased black-box complexity in this context.

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 ith 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 as n times the maximum weight of an edge in the graph).

In the remainder of this introduction, we briefly argue that the value C can be learned at a cost that is negligible compared to the overall cost of all bounds presented below. We have not yet formally defined the structure preserving and the redirecting unbiased models. The formal definitions follow in Definitions 20 and 21. We recall from the introduction in Section 1.2 that, intuitively, in the redirecting unbiased model an algorithm may ask to change the predecessor of a node, but it may not ask to change the predecessor to a specific node in the graph—all nodes other than the current predecessor are equally likely to be chosen. In the structure preserving unbiased model, the algorithm may ask for a specific structural change of its current query, but it may not ask for specific nodes to be part of a particular structure.

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.

5.2.1. The unrestricted black-box complexity

In this section we present our results regarding the SSSP problem with black-box models that do not feature any condition of unbiasedness.

Theorem 18. *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. Summary. We show that adding the ith node to the shortest-path 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.

Formal proof. Let $G = (V, E, w)$ be the input graph. We say that a node is *unconnected* if in the current solution there does not yet 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 $(1, 3, 4, \ldots, n)$, $(2, 1, 4, 5, \ldots, n)$, ..., $(2, \ldots, n - 1, 1)$, 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 equally to the fitness function (namely a weight of C). 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 path 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 path 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 kth 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 $v_1, \dots, v_k$. 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. □

5.2.2. Unbiased black-box models for SSSP

We would like to study the SSSP problem in black-box models that require some unbiasedness, as we did for MST. However, the Hamming-unbiased model of Lehre and Witt in [3] only applies to pseudo-Boolean functions, cf. Section 2. As we are dealing with a different representation here, our first step is to find an appropriate notion of unbiasedness for the SSSP problem.

Recall that an unbiased black-box complexity theory is given by its set of invariances, cf. Definition 1. 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 by the authors of this work [1] and the latter one was introduced by Rowe and Vose in a more general framework for unbiasedness notions [17]. 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 $\vec{z} = (z^1, \dots, z^k)$ of search points, and we would like to sample the next search point according to a probability distribution $D_{\vec{z}}$ on 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

$$\begin{aligned}&\text{For all } g \in \mathcal{G} \text{ such that } g(z^i) = z^i \text{ for all } i \in [k], \text{ and for}\\ &\text{all } x \in [n]^{n-1} \text{ it holds that } D_{\vec{z}}(x) = D_{\vec{z}}(g(x)).\end{aligned} \tag{2}$$

The following proposition shows that this condition is also sufficient.

Proposition 19. *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 $\vec{z} = (z^1, \dots, z^k) \in \mathcal{S}^k$ 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 $z^1, \dots, z^k$ fixed.

Then for any probability distribution $D_{\vec{z}}$ on $[n]^{n-1}$, the following two statements are equivalent.

(i) *There exists a k-ary $\mathcal{G}$-unbiased distribution $(D(\cdot \mid \vec{y}))_{\vec{y} \in \mathcal{S}^k}$ on $\mathcal{S}$ such that $D_{\vec{z}} = D(\cdot \mid \vec{z})$.*
(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 1.

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 on 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 $D_{\vec{z}}$. □

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 on 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 20 (*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 may want to 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. Or, put differently, an algorithm may ask to change the predecessor of a particular node but it may not ask for a specific predecessor. Formally, we require the following.

Definition 21 (*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 $x = (x_2, \dots, x_n) \in [n]^{n-1}$, $\vec{s}(x) := \big(\sigma_2(x_2), \dots, \sigma_n(x_n)\big)$.

A possible variant of redirecting unbiased distributions would not only require the possible predecessors to be treated symmetrically, but also to choose the redirected vertex at random. More formally, we would consider the set of invariances $RD' = \{(\tau, \vec{s}) \mid \tau \in S_{n-1}, \vec{s} = (\sigma_2, \dots, \sigma_n) \in RD\}$, where an invariant $(\tau, \vec{s})$ acts on a vector $x = (x_2, \dots, x_n)$ by $(\tau, \vec{s})(x) = \big(\sigma_{\tau(2)}(x_{\tau(2)}), \dots, \sigma_{\tau(n)}(x_{\tau(n)})\big)$.

Any algorithm that is unbiased with respect to RD' is also unbiased with respect to *RD*. Consequently, the unbiased redirecting black-box complexity (with respect to *RD*) is at most as large as the one with respect to RD'. For this reason, the lower bound of $\Omega(n^2)$ for the ($*$-ary) redirecting unbiased black-box complexity of the SSSP problem (Theorem 27) carries over to the RD'-unbiased model. In addition, also the upper bound of $O(n^3)$ for the unary redirecting unbiased black-box complexity of SSSP (Corollary 28) still holds for this variant, with identical proof. However, the upper bound for the binary case (Theorem 29) does not apply to this variant. We do not further pursue the unbiasedness with respect to RD' since we aim to restrict the algorithms as little as possible, and since the problem class SSSP is already permutation-invariant.

At the same time when we developed the structure preserving and the redirecting black-box models, Rowe and Vose [17] independently extended the notion of unbiasedness from the hypercube to arbitrary search spaces. More precisely, Rowe and Vose define the following.

Definition 22 (*Generalized Unbiased Distributions [17]*)**.** 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 [17] that $\Pi(\mathcal{F})$ indeed forms a group. Therefore, Definition 22 satisfies our definition of unbiasedness given in Definition 1.

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*. Finally, 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, and the *k-ary structure preserving/redirecting/generalized unbiased black-box complexity* to be the *k*-ary *SP*-unbiased/*RD*-unbiased/$\Pi(\mathcal{F})$-unbiased black-box complexity.

Now we determine the distributions $D_{\vec{z}}$ satisfying condition (ii) in Proposition 19 for the structure preserving model and for the redirecting model, respectively. Since the analysis of the generalized model is more involved, we postpone it to Section 5.2.5.

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.

The following characterization of source-preserving permutations is not trivial but straightforward to verify.

Lemma 23. *The source-preserving permutations that leave z unchanged are in one-to-one correspondence with the source-preserving automorphisms of the directed 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 the automorphisms described by Lemma 23, then condition (ii) in Proposition 19 is that $D_{\vec{z}}$ must be invariant under A. That is, 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 19 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, the following statement holds.

Lemma 24. *Let $z \in [n]^{n-1}$. A distribution $D_{\vec{z}}$ on the search space can be extended to a unary redirecting unbiased distribution if and only if $D_{\vec{z}}(x) = D_{\vec{z}}(y)$ holds for any two vectors $x, y \in [n]^{n-1}$ satisfying the property that for every index $i \in [2, \dots, n]$ the two equations $x_i = z_i$ and $y_i = z_i$ are either both true or are both false.*

Intuitively, the characterization of Lemma 24 can be read as follows. For each vertex in the graph we may choose whether or not we wish to redirect it, but we cannot control where it is redirected to. It is important to note the difference to the Hamming-unbiasedness, where we may not ask for something like "change the jth entry of the current search point".

For higher arities k, for $\vec{z} = (z^1, \dots, z^k)$, a distribution $D_{\vec{z}}$ on 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 on the search space, being redirecting unbiased is a rather strong condition.

5.2.3. 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 25. *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 the same 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 other than the identity, *every* probability distribution D_p on 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 26. *The unary structure preserving unbiased black-box complexity of the* SSSP *problem with the single-criterion objective function is between* $\mathrm{UBBC}(\mathrm{SSSP})$ *and* $1 + \mathrm{UBBC}(\mathrm{SSSP}) = O(n^2)$, *where* $\mathrm{UBBC}(\mathrm{SSSP})$ *denotes the unrestricted black-box complexity of the* SSSP *problem with the single-criterion objective function.*

Furthermore, the same holds for the associated ranking-based complexities.

5.2.4. The redirecting unbiased black-box model

It has been argued in [16] that the randomized local search algorithm RLS, which in each iteration redirects exactly one vertex chosen uniformly at random, solves the single-source shortest path problem with the single-criterion fitness function in $O(n^3)$ iterations.

Theorem 27 (*From [16]*)**.** *The randomized local search algorithm RLS for the* SSSP *problem 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 28. *The ranking-based unary redirecting unbiased black-box complexity of the* SSSP *problem 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 29. *The binary redirecting unbiased black-box complexity of the* SSSP *problem for complete graphs is $O(n^2 \log n)$.*

Proof. Summary. We divide the algorithm into three phases. In the first phase we find a search point with no vertex connected to the root and we learn both the penalty value C as well as the weights of all edges adjacent to the root. In the second phase we learn the remaining weights of the problem instance; and in the third phase we construct the solution by imitating Dijkstra's algorithm.

Phase 1: 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. We call such vertices "unconnected vertices", and all other vertices "connected vertices".

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 penalty 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

$$\frac{1}{n}\left(1-\frac{2}{n}\right)^{n-2} \geq \frac{1}{e^2 n} = \Omega(1/n).$$

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 randomly. 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. Thus, we are interested in sampling, for each i, all of n different search points $x^{i,j}$; using Lemma 8 we can assume, without loss of generality, that we find *all* different search points in time $O(n \log n)$ (for each i) even though we cannot distinguish whether two samples connect the same two points to the source.

Note 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 $i \in \{2, \ldots, n\}$, 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. 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 $i \in \{2, \ldots, n\}$ 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, the phase 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 $X \subseteq \{x^i \mid i \in \{2, \ldots, n\}\} \cup X_2 \cup \cdots \cup X_n$ of search points representing the edges that may be used to extend y^t. We start with $X := \{x^i \mid i \in \{2, \ldots, n\}\}$.

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 unbiased operator to (u^t, x) that redirects node j_t to node i_t in u^t. Formally, this is a binary operator 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 19 we need to verify that for all $v, w \in [n]^{n-1}$ and for all $\vec{s} = (\sigma_2, \ldots, \sigma_n) \in S_n^{n-1}$ satisfying $\sigma_i(v_i) = v_i$ and $\sigma_i(w_i) = w_i$ for all $i \in \{2, \ldots, n\}$ 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,

$$D(\vec{s}(x) \mid v, w) = D\left((\sigma_2(x_2), \ldots, \sigma_n(x_n)) \mid v, w\right).$$

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 $k \in \{2, \dots, n\}$ 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 path 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 $j \in \{2, \dots, n\} \setminus \{i_t\}$ 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 $\Theta(n^2 \log n)$. 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 $O(n^2)$. Together, we need time $O(n^2 \log n)$ for the third phase. □

Theorem 30. *The (∗-ary) redirecting unbiased black-box complexity of the* SSSP *problem is* $\Omega(n^2)$.

Proof. Summary. We show that a redirecting unbiased algorithm needs $\Omega(n^2)$ iterations to reconstruct a *given* path on n vertices. We argue with drift on the number of correctly connected vertices.

Formal 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 $X_t = n$.

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 16 for a similar argument).

Thus, using the additive drift theorem (see Theorem 5), we get a total runtime of $\Omega(n^2)$. □

5.2.5. The generalized unbiased black-box model

In this section, we study the generalized unbiased black-box model by Rowe and Vose [17]. Unfortunately, their description defines the set of invariances in an indirect way, see Definition 22. 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 20. In particular, we will see that the complexity of the single-criterion SSSP problem increases by at most 1 compared to the unrestricted black-box complexity.

Recall from Definition 22 that, for any set of problems instances $\mathcal{F}$, $\Pi(\mathcal{F})$ is the class of all bijections on the search space α such that for each $f \in \mathcal{F}$ the composition $f \circ \alpha$ is also a member of $\mathcal{F}$. Furthermore, recall that $\hat{\sigma}$ was defined in Definition 20. We note again that the notion of generalized unbiasedness is very specific to the particular modeling of the problem. Different representations of the search space may lead to different results, and the result given below applies to the SSSP model described in the beginning of Section 5.2.

Theorem 31. *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) $\alpha \in \Pi(\text{SSSP})$.
(ii) *There is a* $\hat{\sigma} \in \text{SP}$ *such that for all* $x \in [n]^{n-1}$ *we have* $E(\alpha(x)) = E(\hat{\sigma}(x))$.

Proof. Let $\alpha \in \Pi(\text{SSSP})$. 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 coefficients 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 $\alpha \in \Pi(\text{SSSP})$).

Recall that, for any SSSP problem, there is a large constant 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 constant C for P, and similarly the constant 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, $C' = C$. □

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 constant 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 $\alpha(x)$. □

For all i with $2 \le i \le n$, 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. □

Let σ be as given by Claim 3.

Claim 4. *For all i with $2 \le i \le n$ we have $w'(1, i) = w(1, \sigma(i))$.*

Proof of Claim 4. Let i be such that $2 \le i \le n$. Fix $x \in \mathcal{S}^{(i)}$. Using Claim 3, we have $(n-2)C + w'(1, i) = f'(x) = f(\alpha(x)) = (n-2)C + w(1, \sigma(i))$. □

Now we extend Claim 4 to arbitrary edges.

Claim 5. *For all $i, j \le n$ we have $w'(i, j) = w(\sigma(i), \sigma(j))$.*

Proof of Claim 5. For all $i \in \{2 \dots n\}$ let $M_i = \{x \mid (n-3)C + 2w(1, i) < f(x) < (n-3)C + 2w(1, i) + 1\}$. By the 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 $i \le n$, $M_i' = \{x \mid (n-3)C + 2w(1, i) < f'(x) < (n-3)C + 2w(1, i) + 1\}$. 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 $i, j \le n$. 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. □

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

$$\begin{aligned}
E(\alpha(x)) &= D(f(\alpha(x))) \\
&= \{\{i, j\} \mid w(i, j) \text{ is part of the } f\text{-weight of } \alpha(x)\} \\
&= \{\{\sigma(i), \sigma(j)\} \mid w(\sigma(i), \sigma(j)) \text{ is part of the } f\text{-weight of } \alpha(x)\} \\
&= \{\{\sigma(i), \sigma(j)\} \mid w'(i, j) \text{ is part of the } f'\text{-weight of } x\} \\
&= \sigma(D'(f'(x))) \\
&= \sigma(E(x)) \\
&= E(\hat{\sigma}(x)). \quad \square
\end{aligned}$$

This shows one implication. The converse is straightforward. $\square$

Theorem 31 enables us to decide whether a given probability distribution on the search space extends to a generalized unbiased distribution, similar to the discussion for structure preserving and the redirecting model after Definitions 20 and 21. If $\vec{z} = (z^1, \ldots, z^k) \in \mathcal{S}^k$ is a k-tuple of search points, and $D_{\vec{z}}$ is a probability distribution on 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 32. *The unary generalized unbiased black-box complexity of the* SSSP *problem with the single-criterion objective function is between* UBBC(SSSP) *and* $1+$UBBC(SSSP) $= O(n^2)$*, where* UBBC(SSSP) *denotes the unrestricted black-box complexity of the* SSSP *problem with the single-criterion objective function.*

Furthermore, the same holds for the associated ranking-based complexities.

Proof. The proof is exactly as the proof of Theorem 25. All the operators used there are in fact also generalized unbiased by Theorem 31. $\square$

6. Conclusions

Our analysis of the recently introduced versions of the black-box complexity notion for two classic combinatorial optimization problems showed the following. In general, all notions make sense for combinatorial problems as well, though some care has to be taken of how to implement unbiasedness conditions. The particular bounds we find are reasonably close to actual runtimes observed by existing randomized search heuristics, that is, the black-box complexities give reasonable bounds for the problem difficulties here. In cases where our bounds are smaller than those observed by current best search heuristics, further studies are needed to determine whether current heuristics can be improved, e.g., by using higher-arity variation operators, or whether additional restrictions to the black-box model are needed to exclude artificial algorithms.

Furthermore, it might be interesting to analyze various (ranking-based) black-box complexities for other combinatorial optimization problems, including NP-hard problems like Partition, where an initial analysis was made in [21].

Acknowledgments

Timo Kötzing was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant NE 1182/5-1.

Carola Winzen is a recipient of the Google Europe Fellowship in Randomized Algorithms, and this research is supported in part by this Google Fellowship.

References

[1] B. Doerr, T. Kötzing, J. Lengler, C. Winzen, Black-box complexities of combinatorial problems, in: Proc. of Genetic and Evolutionary Computation Conference, GECCO'11, ACM, 2011, pp. 981–988.

[2] S. Droste, T. Jansen, I. Wegener, Upper and lower bounds for randomized search heuristics in black-box optimization, Theory of Computing Systems 39 (2006) 525–544.

[3] P. K. Lehre, C. Witt, Black-box search by unbiased variation, in: Proc. of Genetic and Evolutionary Computation Conference, GECCO'10, ACM, 2010, pp. 1441–1448. (a journal version of this paper is to appear in *Algorithmica*. The DOI of the full paper, which is available online, is http://dx.doi.org/10.1007/s00453-012-9616-8).
[4] B. Doerr, C. Winzen, Towards a complexity theory of randomized search heuristics: ranking-based black-box complexity, in: Proc. of Computer Science Symposium in Russia, CSR'11, Springer, 2011, pp. 15–28.
[5] F. Neumann, I. Wegener, Randomized local search, evolutionary algorithms, and the minimum spanning tree problem, in: Proc. of Genetic and Evolutionary Computation Conference, GECCO'04, Springer, 2004, pp. 713–724.
[6] F. Neumann, I. Wegener, Randomized local search, evolutionary algorithms, and the minimum spanning tree problem, Theoretical Computer Science 378 (2007) 32–40.
[7] J. Reichel, M. Skutella, Evolutionary algorithms and matroid optimization problems, in: Proc. of Genetic and Evolutionary Computation Conference, GECCO'07, ACM, 2007, pp. 947–954.
[8] J. Reichel, M. Skutella, On the size of weights in randomized search heuristics, in: Proc. of Foundations of Genetic Algorithms, FOGA'09, ACM, 2009, pp. 21–28.
[9] B. Doerr, E. Happ, C. Klein, Crossover can provably be useful in evolutionary computation, Theoretical Computer Science 425 (2012) 17–33.
[10] B. Doerr, M. Theile, Improved analysis methods for crossover-based algorithms, in: Proc. of Genetic and Evolutionary Computation Conference, GECCO'09, ACM, 2009, pp. 247–254.
[11] B. Doerr, D. Johannsen, T. Kötzing, F. Neumann, M. Theile, More effective crossover operators for the all-pairs shortest path problem, in: Proc. of Parallel Problem Solving From Nature, PPSN'10, Springer, 2010, pp. 184–193.
[12] J. Scharnow, K. Tinnefeld, I. Wegener, Fitness landscapes based on sorting and shortest path problems, in: Proc. of Parallel Problem Solving From Nature, PPSN'02, Springer, 2002, pp. 54–63.
[13] J. Scharnow, K. Tinnefeld, I. Wegener, The analysis of evolutionary algorithms on sorting and shortest paths problems, Journal of Mathematical Modelling and Algorithms 3 (2004) 349–366.
[14] B. Doerr, D. Johannsen, Edge-based representation beats vertex-based representation in shortest path problems, in: Proc. of Genetic and Evolutionary Computation Conference, GECCO'10, ACM, 2010, pp. 758–766.
[15] B. Doerr, E. Happ, C. Klein, Tight analysis of the $(1+1)$-EA for the single source shortest path problem, Evolutionary Computation 19 (2011) 673–691.
[16] S. Baswana, S. Biswas, B. Doerr, T. Friedrich, P. P. Kurur, F. Neumann, Computing single source shortest paths using single-objective fitness functions, in: Proc. of Foundations of Genetic Algorithms, FOGA'09, ACM, 2009, pp. 59–66.
[17] J. Rowe, M. Vose, Unbiased black box search algorithms, in: Proc. of Genetic and Evolutionary Computation Conference, GECCO'11, ACM, 2011, pp. 2035–2042.
[18] B. Doerr, D. Johannsen, T. Kötzing, P. K. Lehre, M. Wagner, C. Winzen, Faster black-box algorithms through higher arity operators, in: Proc. of Foundations of Genetic Algorithms, FOGA'11, ACM, 2011, pp. 163–172.
[19] F. Neumann, C. Witt, Runtime analysis of a simple ant colony optimization algorithm, Algorithmica 54 (2009) 243–255.
[20] T. Kötzing, F. Neumann, D. Sudholt, M. Wagner, Simple max–min ant systems and the optimization of linear pseudo-boolean functions, in: Proc. of Foundations of Genetic Algorithms, FOGA'11, 2011, pp. 209–218.
[21] B. Doerr, T. Kötzing, C. Winzen, Too fast unbiased black-box algorithms, in: Proc. of Genetic and Evolutionary Computation Conference, GECCO'11, ACM, 2011, pp. 2043–2050.
[22] A. Brouwer, A. Cohen, A. Neumaier, Distance-regular graphs, in: Ergebnisse der Mathematik und ihrer Grenzgebiete, Springer, 1989.
[23] A. C.-C. Yao, Probabilistic computations: toward a unified measure of complexity, in: Proc. of Foundations of Computer Science, FOCS'77, IEEE, 1977, pp. 222–227.
[24] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.
[25] J. He, X. Yao, A study of drift analysis for estimating computation time of evolutionary algorithms, Natural Computing 3 (2004) 21–35.
[26] R. Sarker, M. Mohammadian, X. Yao, I. Wegener, Methods for the analysis of evolutionary algorithms on pseudo-boolean functions, in: Evolutionary Optimization, in: International Series in Operations Research & Management Science, vol. 48, Springer, US, 2003, pp. 349–369.
[27] F. Neumann, C. Witt, Bioinspired Computation in Combinatorial Optimization, Springer, 2010.
[28] P. Erdős, A. Rényi, On two problems of information theory, Magyar Tudományos Akadémia Matematikai Kutató Intézet Közleményei 8 (1963) 229–243.
[29] P. Kovár, M. Kubesa, Factorizations of complete graphs into spanning trees with all possible maximum degrees, in: Proc. of International Workshop on Combinatorial Algorithms, IWOCA'09, Springer, 2009, pp. 334–344.