Chapter 3
Complexity Theory for Discrete Black-Box Optimization Heuristics

Carola Doerr

Abstract A predominant topic in the theory of evolutionary algorithms and, more generally, theory of randomized black-box optimization techniques is *running-time analysis.* Running-time analysis is aimed at understanding the performance of a given heuristic on a given problem by bounding the number of function evaluations that are needed by the heuristic to identify a solution of a desired quality. As in general algorithms theory, this running-time perspective is most useful when it is complemented by a meaningful *complexity theory* that studies the limits of algorithmic solutions.

In the context of discrete black-box optimization, several *black-box complexity models* have been developed to analyze the best possible performance that a black-box optimization algorithm can achieve on a given problem. The models differ in the classes of algorithms to which these lower bounds apply. This way, black-box complexity contributes to a better understanding of how certain algorithmic choices (such as the amount of memory used by a heuristic, its selective pressure, or properties of the strategies that it uses to create new solution candidates) influence performance.

In this chapter we review the different black-box complexity models that have been proposed in the literature, survey the bounds that have been obtained for these models, and discuss how the interplay of running-time analysis and black-box complexity can inspire new algorithmic solutions to well-researched problems in evolutionary computation. We also discuss in this chapter several interesting open questions for future work.

Carola Doerr
Sorbonne Université, CNRS, LIP6, Paris, France

B. Doerr, F. Neumann (eds.), *Theory of Evolutionary Computation*,
Natural Computing Series, https://doi.org/10.1007/978-3-030-29414-4_3

3.1 Introduction and Historical Remarks

One of the driving forces in theoretical computer science is the fruitful interplay between *complexity theory* and the *theory of algorithms.* While the former measures the minimum computational effort that is needed to solve a given problem, the latter is aimed at designing and analyzing efficient algorithmic solutions which prove that a problem can be solved with a certain computational effort. When, for a problem, the lower bounds on the resources needed to solve it are identical to (or not much smaller than) the upper bounds attained by some specific algorithm, we can be certain that we have an (almost) optimal algorithmic solution to this problem. Big gaps between lower and upper bounds, in contrast, indicate that more research effort is needed to understand the problem: it may be that more efficient algorithms for the problem exist, or that the problem is indeed "harder" than what the lower bound suggests.

Many different complexity models coexist in the theoretical computer science literature. The arguably most classical one measures the number of arithmetic operations that an algorithm needs to perform on the problem data until it obtains a solution for the problem. A solution can be a "yes/no" answer (a decision problem), a classification of a problem instance according to some criteria (a classification problem), a vector of decision variables that maximize or minimize some objective function (an optimization problem), etc. In the optimization context, we are typically interested only in algorithms that satisfy some minimal quality requirements such as a guarantee that the suggested solutions ("the output" of the algorithm) are always optimal or are optimal with some large enough probability, or that they are not worse than an optimal solution by more than some additive or multiplicative factor C, etc.

In the *white-box* setting, in which the algorithms have full access to the data describing the problem instance, complexity theory is a well-established and very intensively studied research objective. In *black-box optimization,* where the algorithms do not have access to the problem data and can learn about the problem at hand only through the evaluation of potential solution candidates, complexity theory is a much less present topic, with rather large fluctuations in the number of publications. In the context of heuristic solutions to black-box optimization problems, which is the topic of this book, complexity theory has been systematically studied only since 2010, using the notion of *black-box complexity.* Luckily, black-box complexity theory can build on results in related research domains such as information theory, discrete mathematics, cryptography, and others.

In this chapter, we review the state of the art in this currently very active area of research, which is concerned with bounding the best possible performance that an optimization algorithm can achieve in a black-box setting.

3.1.1 Black-Box vs. White-Box Complexity

Most of the traditional complexity measures assume that the algorithms have access to the problem data, and count the number of steps that are needed until the algorithm outputs a solution. In the black-box setting, these complexity measures are not very meaningful, as the algorithms are asked to optimize a problem without having direct access to it. As a consequence, the performance of a black-box optimization algorithm is therefore traditionally measured by the number of function evaluations that the algorithm does until it queries for the first time a solution that satisfies some specific performance criteria. In this book, we are mostly interested in the expected number of evaluations needed until an *optimal* solution is evaluated for the first time. It is therefore natural to define black-box complexity as the *minimum number of function evaluations that any black-box algorithm needs to perform, on average, until it queries an optimal solution for the first time.*

We typically consider classes of problems, for example, the set of traveling salesperson instances of planar graphs with integer edge weights. For such a class $\mathcal{F} \subseteq \{f : S \to \mathbb{R}\}$ of problem instances, we take a worst-case view and measure the expected number of function evaluations that an algorithm needs to optimize any instance $f \in \mathcal{F}$. That is, the black-box complexity of a problem $\mathcal{F}$ is $\inf_A \sup_{f \in \mathcal{F}} E[T(A, f)]$, the best (among all algorithms A) worst-case (among all problem instances f) expected number $E[T(A, f)]$ of function evaluations that are needed to optimize any $f \in \mathcal{F}$. A formal definition will be given in Section 3.2.

The black-box complexity of a problem can be very different from its white-box counterpart. We will discuss, for example, in Sections 3.2.4 and 3.6.3.5 the fact that there are a number of NP-hard problems whose black-box complexity is of small polynomial order.

3.1.2 Motivation and Objectives

The ultimate objective of black-box complexity is to support the investigation and design of efficient black-box optimization techniques. This is achieved in several complementary ways.

A first benefit of black-box complexity is that it enables the above-mentioned evaluation of how well we have understood a black-box optimization problem, and how suitable the state-of-the-art heuristics are. Where large gaps between lower and upper bounds exist, we may want to explore alternative algorithmic solutions, in the hope of identifying more efficient solvers. Where the lower and upper bounds match or are close, we can stop striving for more efficient algorithms.

Another advantage of black-box complexity studies is that they allow us to investigate how certain algorithmic choices influence the performance: By re-

stricting the class of algorithms under consideration, we can judge how these restrictions increase the complexity of a black-box optimization problem. In the context of evolutionary computation, interesting restrictions include the amount of memory that is available to the algorithms, the number of solutions that are sampled in every iteration, the way new solution candidates are generated, the selection principles according to which it is decided which search points to keep for future reference, etc. Comparing the unrestricted with the restricted black-box complexity of a problem (i.e. its black-box complexity with respect to *all* versus that with respect to a *subclass* of all algorithms) quantifies the performance loss caused by these restrictions. This way, we can understand, for example, the effects of not storing the set of all previously evaluated solution candidates, but only a small subset.

The black-box complexity of a problem can be significantly smaller than the performance of a best known 'standard' heuristic. In such cases, the small complexity is often attained by a problem-tailored black-box algorithm, which is not representative of common black-box heuristics. Interestingly, it turns out that we can nevertheless learn from such highly specific algorithms, as they often incorporate some ideas that could be beneficial far beyond the particular problem at hand. As we shall demonstrate in Section 3.9, even for very well-researched optimization problems, such ideas can give rise to the design of novel heuristics which are provably more efficient than standard solutions. This way, black-box complexity serves as a source of inspiration for the development of novel algorithmic ideas that lead to the design of better search heuristics.

3.1.3 Relationship to Query Complexity

As indicated above, black-box complexity is studied in several different contexts, which reach far beyond evolutionary computation. In the 1960s and 1970s, for example, this complexity measure was very popular in the context of combinatorial games, such as coin-weighing problems of the type "given n coins of two different types, what is the minimum number of weighings that is needed to classify the coins according to their weight?" Interpreting a weighing as a function evaluation, we see that such questions can be formulated as black-box optimization problems.

Black-box complexity also plays an important role in cryptography, where a common research question concerns the minimum amount of information that suffices to break a secret code. Quantum computing, communication complexity, and information theory are other research areas where (variants of) black-box complexity are intensively studied performance measures. While in these settings the precise model is often not exactly identical to a model of the kind we are faced with in black-box optimization, some of the tools developed in these related areas can be useful in our context.

A significant part of the literature studies the performance of deterministic algorithms. Randomized black-box complexities are much less understood. They can be much smaller than their deterministic counterparts. Since deterministic algorithms form a subclass of randomized ones, any lower bound proven for the randomized black-box complexity of a problem also applies to any deterministic algorithm. In some cases, a strict separation between deterministic and randomized black-box complexities can be proven. This is the case for the LeadingOnes function, as we shall briefly discuss in Section 3.3.6. For other problems, the deterministic and randomized black-box complexities coincide. Characterizing those problems for which access to random bits can provably decrease the complexity is a wide-open research question.

In several contexts, in particular the research domains mentioned above, black-box complexity is typically referred to as *query* or *oracle complexity*, with the idea that the algorithms do not evaluate the function values of the solution candidates themselves but rather query them from an oracle. This interpretation is mostly identical to the black-box scenario classically considered in evolutionary computation.

3.1.4 Scope of This Chapter

In this chapter, as in the remainder of this book, we restrict our attention to discrete optimization problems, i.e., the maximization or minimization of functions $f : S \to \mathbb{R}$ that are defined over finite search spaces S. As in the previous chapters, we will mostly deal with the optimization of pseudo-Boolean functions $f : \{0,1\}^n \to \mathbb{R}$, permutation problems $f : S_n \to \mathbb{R}$, and functions $f : [0..r-1]^n \to \mathbb{R}$ defined for strings over an alphabet of bounded size, where, here and in the following, we use the following abbreviations: $[0..r-1] := \{0,1,\ldots,r-1\}$ represents the set of non-negative integers smaller than r, $[n] := \{1,2,\ldots,n\}$, and S_n represents the set of all permutations (one-to-one maps) $\sigma : [n] \to [n]$.

We point out that black-box complexity notions are also studied for infinite search spaces S. In the context of continuous optimization problems, studies of black-box complexity are aimed at bounding the best possible *convergence rates* that a derivative-free black-box optimization algorithm can achieve, see [57, 86] for examples.

3.1.5 Target Audience and Complementary Material

This chapter is written with a reader in mind who is familiar with black-box optimization, and who brings with them some background in theoretical

running-time analysis. We will give an exhaustive survey of existing results. Where appropriate, we provide proof ideas and discuss some historical developments. Readers interested in a more gentle introduction to the basic concepts of black-box complexity are referred to [61]. A slide presentation on selected aspects of black-box complexity, along with a summary of complexity bounds known back in spring 2014, can be found in the tutorial [20].

3.1.6 Overview of the Content

Black-box complexity is formally defined in Section 3.2. We also provide there a summary of useful tools. In Section 3.2.4 we discuss why classical complexity statements such as NP-hardness results do not necessarily imply hardness in the black-box complexity model.

In Sections 3.3-3.7 we review the different black-box complexity models that have been proposed in the literature. For each model, we discuss the main results that have been achieved for it. For several benchmark problems, including most notably OneMax, LeadingOnes, and Jump, but also combinatorial problems such as the minimum spanning tree problem and shortest-paths problems, bounds have been derived for various complexity models. For OneMax and LeadingOnes, we compare these different bounds in Section 3.8, to summarize where gaps between upper and lower bounds exist, and to highlight the increasing complexities imposed by the restrictive models.

We will demonstrate in Section 3.9 that the complexity-theoretic view of black-box optimization can inspire the design of more efficient optimization heuristics. This is made possible by questioning some of the state-of-the-art choices that are made in evolutionary computation and neighboring disciplines.

Finally, we show in Section 3.10 that research efforts originally motivated by the study of black-box complexity have yielded improved bounds for long-standing open problems in classical computer science.

In Section 3.11, we conclude this chapter with a summary of open questions and problems in discrete black-box complexity and directions for future work.

3.2 The Unrestricted Black-Box Model

In this section we introduce the most basic black-box model, which is the *unrestricted* one. This model contains all black-box optimization algorithms. Any lower bound in this model therefore immediately applies to any of the restricted models which we discuss in Sections 3.4-3.7. We also discuss in this section some useful tools for the analysis of black-box complexity and

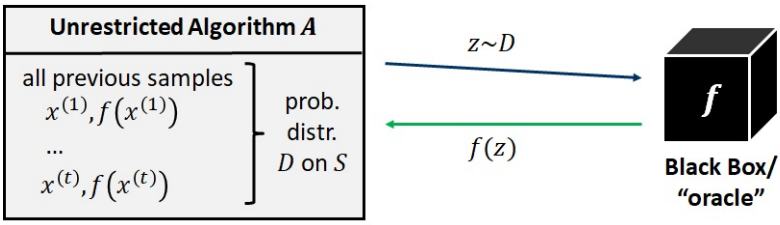

Fig. 3.1 In the unrestricted black-box model, the algorithm can store the full history of previously queried search points. For each of these already evaluated candidate solutions x, the algorithm has access to its absolute function value $f(x) \in \mathbb{R}$. There are no restrictions on the structure of the distributions D from which new solution candidates are sampled.

demonstrate that the black-box complexity of a problem can be very different from its classical white-box complexity.

The unrestricted black-box model was introduced by Droste, Jansen, Wegener in [54]. The only assumption that it makes is that the algorithms do not have any information about the problem at hand other than the fact that it stems from some function class $\mathcal{F} \subseteq \{f : S \to \mathbb{R}\}$. The only way an unrestricted black-box algorithm can learn about the instance f is by evaluating the function values $f(x)$ of potential solution candidates $x \in S$. We can assume that the evaluation is done by some oracle, from which $f(x)$ is queried. In the unrestricted model, the algorithms can update after any such query the strategy by which the next search point(s) are generated. In this book, we are mostly interested in the performance of *randomized black-box heuristics*, so that these strategies are often *probability distributions* over the search space from which the next solution candidates are sampled. This process continues until an optimal search point $x \in \arg\max f$ is queried for the first time.

The algorithms that we are interested in are thus those that maintain a probability distribution D over the search space S. In every iteration, a new solution candidate x is sampled from this distribution and the function value $f(x)$ of this search point is evaluated. After this evaluation, the probability distribution D is updated according to the information gathered through the sample $(x, f(x))$. The next iteration starts again by sampling a search point from this updated distribution D, and so on. This structure is summarized in Algorithm 3.1, which models *unrestricted randomized black-box algorithms.* A visualization is provided in Fig. 3.1.

Note that in Algorithm 3.1, in every iteration only one new solution candidate is sampled. In contrast, many evolutionary algorithms and other black-box optimization techniques generate and evaluate several search points *in parallel.* It is not difficult to see that lower bounds obtained for the unrestricted black-box complexity described here apply immediately to such *population-based* heuristics, since an unrestricted algorithm is free to ignore

Algorithm 3.1: Blueprint of an unrestricted randomized black-box algorithm

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

information obtained from previous iterations. As will be commented on in Section 3.7.2, the *parallel* black-box complexity of a function can be (much) larger than its sequential variant. Taking this idea to the extreme, i.e., requiring the algorithm to neglect information obtained through previous queries yields so-called *nonadaptive* black-box algorithms. A prime example for a nonadaptive black-box algorithm is random sampling (with and without repetitions). Nonadaptive algorithms play only a marginal role in evolutionary computation. From a complexity point of view, however, it can be interesting to study how much adaptation is needed for an efficient optimization; see also the discussions in Sections 3.3.2 and 3.10. For most problems, the adaptive and nonadaptive complexity differ by large factors. For some other problems, however, the two complexity notions coincide; see Section 3.3.1 for an example.

Note also that unrestricted black-box algorithms have access to the full history of previously evaluated solutions. The effects of restricting the available memory to a *population* of a certain size will be the focus of the memory-restricted black-box models discussed in Section 3.4.

In line 3 of Algorithm 3.1 we do not specify how the probability distribution $D^{(t)}$ is chosen. Thus, in principle, the algorithm can spend significant time on choosing this distribution. This can result in small polynomial black-box complexities for NP-hard problems; see Section 3.2.4. Droste, Jansen, and Wegener [54] therefore suggested restricting the set of algorithms to those that execute the choice of the distributions $D^{(t)}$ in a polynomial number of algebraic steps (i.e., polynomial time in the input length, where "time" refers to the classically considered complexity measure). They called this model the *time-restricted model*. In this chapter, we will not study this time-restricted model. That is, we allow the algorithms to spend arbitrary time on the choice of the distributions $D^{(t)}$. This way, we obtain very general lower bounds. Almost all upper bounds stated in this chapter nevertheless apply also to the time-restricted model. The polynomial bounds for NP-hard problems form, of course, an exception to this rule.

We comment, finally, on the fact that Algorithm 3.1 runs forever. As we have seen in previous chapters in this book, the pseudocode in Algorithm 3.1 is a common representation of black-box algorithms in the theory of heuris-

tic optimization. Not specifying the termination criterion is justified by our performance measure, which is the expected number of function evaluations that an algorithm performs until (and including) the first iteration in which an optimal solution is evaluated; see Definition 3.2.1 below. Other performance measures for black-box heuristics have been discussed in the literature [11, 34, 65], but in the realm of black-box complexity, the *average optimization time* is still the predominant performance indicator. See Section 3.11 for a discussion of the possibility of extending existing results to other, possibly more complex performance measures.

3.2.1 Formal Definition of Black-Box Complexity

In this section, we give a very general definition of black-box complexity. More precisely, we formally define the black-box complexity of a class $\mathcal{F}$ of functions with respect to some class $\mathcal{A}$ of algorithms. The unrestricted black-box complexity will be the complexity of $\mathcal{F}$ with respect to all black-box algorithms that follow the blueprint provided in Algorithm 3.1.

For a black-box optimization algorithm A and a function $f : S \to \mathbb{R}$, let $T(A, f) \in \mathbb{R} \cup \{\infty\}$ be the number of function evaluations that algorithm A does until and including the evaluation in which it evaluates for the first time an optimal search point $x \in \arg\max f$. As in previous chapters, we call $T(A, f)$ the *running time of A for f* or, synonymously, the *optimization time of A for f*. When A is a randomized algorithm, $T(A, f)$ is a random variable that depends on the random decisions made by A. We are mostly interested in its expected value $E[T(A, f)]$.

With this performance measure in place, the definition of the black-box complexity of a class $\mathcal{F}$ of functions $S \to \mathbb{R}$ with respect to some class $\mathcal{A}$ of algorithms now follows the usual approach in complexity theory.

Definition 3.2.1. For a given black-box algorithm A, the *A-black-box complexity of $\mathcal{F}$* is

$$E[T(A, \mathcal{F})] := \sup_{f \in \mathcal{F}} E[T(A, f)],$$

the worst-case expected running time of A on $\mathcal{F}$.

The *$\mathcal{A}$-black-box complexity of $\mathcal{F}$* is

$$E[T(\mathcal{A}, \mathcal{F})] := \inf_{A \in \mathcal{A}} E[T(A, \mathcal{F})],$$

the minimum ("best") complexity among all $A \in \mathcal{A}$ for $\mathcal{F}$.

Thus, formally, the *unrestricted black-box complexity* of a problem class $\mathcal{F}$ is $E[T(\mathcal{A}, \mathcal{F})]$, where $\mathcal{A}$ is the collection of all unrestricted black-box algorithms, i.e., all algorithms that can be expressed in the framework of Algorithm 3.1.

The following lemma formalizes the intuition that every lower bound for the unrestricted black-box model also applies to any restricted black-box model.

Lemma 3.2.2. *Let $\mathcal{F} \subseteq \{f : S \to \mathbb{R}\}$. For every collection $\mathcal{A}'$ of black-box optimization algorithms for $\mathcal{F}$, the $\mathcal{A}'$-black-box complexity of $\mathcal{F}$ is at least as large as its unrestricted black-box complexity.*

Formally, this lemma holds because $\mathcal{A}'$ is a subclass of the set $\mathcal{A}$ of all unrestricted black-box algorithms. The infimum in the definition of $E[T(\mathcal{A}', \mathcal{F})]$ is therefore taken over a smaller class, thus giving values that are at least as large as $E[T(\mathcal{A}, \mathcal{F})]$.

3.2.2 Tools for Proving Lower Bounds

Lemma 3.2.2 shows that the unrestricted black-box complexity of a class $\mathcal{F}$ of functions is a lower bound for the performance of any black-box algorithm on $\mathcal{F}$. In other words, no black-box algorithm can optimize $\mathcal{F}$ more efficiently than what the unrestricted black-box complexity of $\mathcal{F}$ indicates. We are therefore particularly interested in proving *lower bounds* for the black-box complexity of a problem. This is the topic of this section.

To date, the most powerful tool to prove lower bounds for randomized query complexity models such as our unrestricted black-box model is the so-called *minimax principle* of Yao [88]. In order to discuss this principle, we first need to recall that we can interpret every randomized unrestricted black-box algorithm as a probability distribution over deterministic algorithms. In fact, randomized black-box algorithms are often defined this way.

Deterministic black-box algorithms are those for which the probability distributions in line 3 of Algorithm 3.1 are one-point distributions. That is, for every t and for every sequence $\big((x^{(0)}, f(x^{(0)})), \dots, (x^{(t-1)}, f(x^{(t-1)}))\big)$ of previous queries, there exists a search point $s \in S$ such that $D^{(t)}\Big(\big((x^{(0)}, f(x^{(0)})), \dots, (x^{(t-1)}, f(x^{(t-1)}))\big)\Big)(s) = 1$ and $D^{(t)}\Big(\big((x^{(0)}, f(x^{(0)})), \dots, (x^{(t-1)}, f(x^{(t-1)}))\big)\Big)(y) = 0$ for all $y \neq s$. In other words, we can interpret deterministic black-box algorithms as *decision trees.* A decision tree for a class $\mathcal{F}$ of functions is a rooted tree in which the nodes are labeled by the search points that the algorithm queries. The first query is the label of the root node, say $x^{(0)}$. The edges from the root node to its neighbors are labeled with the possible objective values $\{g(x^{(0)}) \mid g \in \mathcal{F}\}$. After evaluating $f(x^{(0)})$, the algorithm follows the (unique) edge $\{x^{(0)}, x^{(1)}\}$ which is labeled with the value $f(x^{(0)})$. The next query is the label of the endpoint $x^{(1)}$ of this edge. We call $x^{(1)}$ a level-1 node. The level-2 neighbors of $x^{(0)}$ (i.e., all neighbors of $x^{(1)}$ except the root node $x^{(0)}$) are the potential search points to be queried in the next iteration. As before, the algorithm

chooses as the next query the neighbor $x^{(2)}$ of $x^{(1)}$ to which the unique edge labeled with the value $f(x^{(1)})$ leads. This process continues until an optimal search point has been queried. The optimization time $T(A, f)$ of the algorithm A on the function f equals the depth of this node plus one (the "plus one" accounts for the evaluation of the root node).

We can easily see that, in this model, it does not make sense to query the same search point twice. Such a query would not reveal any new information about the objective function f. For this reason, on every rooted path in the decision tree, every search point appears at most once. This shows that the *depth of the decision tree* is bounded by $|S|-1$. The *width* of the tree, however, can be as large as the size of the set $\mathcal{F}(S) := \{g(s) \mid g \in \mathcal{F}, s \in S\}$, which can be infinite or even uncountable, for example, if $\mathcal{F}$ equals the set of all linear or monotone functions $f : \{0,1\}^n \to \mathbb{R}$. As we shall see below, Yao's minimax principle can only be applied to problems for which $\mathcal{F}(S)$ is finite. Luckily, it is often possible to identify subclasses $\mathcal{F}'$ of $\mathcal{F}$ for which $\mathcal{F}'(S)$ is finite and whose complexity is identical to or not much smaller than that of the whole class $\mathcal{F}$.

When S and $\mathcal{F}(S)$ are finite, the number of (nonrepetitive) deterministic decision trees, and hence the number of deterministic black-box algorithms for $\mathcal{F}$, is finite. In this case, we can apply Yao's minimax principle. This theorem, intuitively speaking, allows us to restrict our attention to bounding the expected running time $E[T(A, f)]$ of a best possible *deterministic* algorithm A on a *random* instance f taken from $\mathcal{F}$ according to some probability distribution p. By Yao's minimax principle, this best possible expected running time is a lower bound for the expected performance of a best possible *randomized* algorithm on an *arbitrary* input. In our words, it is thus a lower bound on the unrestricted black-box complexity of the class $\mathcal{F}$.

Analyzing deterministic black-box algorithms is often considerably easier than directly bounding the performance of any possible randomized algorithm. An a priori challenge in applying this theorem is the identification of a probability distribution p on $\mathcal{F}$ for which the expected optimization time of a best possible deterministic algorithm is large. Luckily, for many applications some rather simple distributions on the inputs suffice, for example the uniform distribution, which assigns equal probability to each problem instance $f \in \mathcal{F}$. Another difficulty in the application of the theorem is the above-mentioned identification of subclasses $\mathcal{F}'$ of $\mathcal{F}$ for which $\mathcal{F}'(S)$ is finite.

Formally, Yao's minimax principle reads as follows.

Theorem 3.2.3 (Yao's minimax principle). *Let Π be a problem with a finite set $\mathcal{I}$ of input instances (of a fixed size) permitting a finite set $\mathcal{A}$ of deterministic algorithms. Let p be a probability distribution over $\mathcal{I}$ and let 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 denotes a random algorithm chosen from $\mathcal{A}$ according to q, and $T(I,A)$ denotes the running time of algorithm A on input I.

The formulation of Theorem 3.2.3 is taken from the book by Motwani and Raghavan [77], where an extended discussion of this principle can be found.

A straightforward but still quite handy application of Yao's minimax principle gives the following lower bound.

Theorem 3.2.4 (simple information-theoretic lower bound, Theorem 2 in [54]). *Let S be finite. Let $\mathcal{F}$ be a set of functions $\{f : S \to \mathbb{R}\}$ such that for every $s \in S$ there exists a function $f_s \in \mathcal{F}$ for which the size of $f_s(S) := \{f_s(x) \mid x \in S\}$ is bounded by k and for which s is a unique optimum, i.e., $\arg\max f_s = \{s\}$ and $|f_s(S)| \leq k$. The unrestricted black-box complexity of $\mathcal{F}$ is at least $\lceil \log_k(|S|) \rceil - 1$.*

To prove Theorem 3.2.4 it suffices to select for every $s \in S$ one function f_s as in the statement and to consider the uniform distribution over the set $\{f_s \mid s \in S\}$. Every deterministic black-box algorithm that eventually solves any instance f_s has to have at least one node labeled s. We therefore need to distribute all $|S|$ potential optima on the decision tree that corresponds to this deterministic black-box algorithm. Since the outdegree of every node is bounded from above by k, the average distance from a node to the root is at least $\lceil \log_k(|S|) \rceil - 2$.

An informal interpretation of Theorem 3.2.4, which in addition ignores the rounding of the logarithms, is as follows. In the setting of Theorem 3.2.4, optimizing a function f_s corresponds to *learning* s. A binary encoding of the optimum s requires $\log_2(|S|)$ bits. With every query, we obtain at most $\log_2(k)$ bits of information, namely, the number of bits needed to encode which of the at most k possible objective values is assigned to the queried search point. We therefore need to query at least $\log_2(|S|)/\log_2(k) = \log_k(|S|)$ search points to obtain the information that is required to decode s. This "hand-wavy" interpretation often gives a good first idea of the lower bounds that can be proven by Theorem 3.2.4.

This intuitive proof for Theorem 3.2.4 shows that it works best if at every search point *exactly* k answers are possible, and each of them is equally likely. This situation, however, is not typical for black-box optimization processes, where usually only a (possibly small) subset of function values are likely to appear next. As a rule of thumb, the larger the difference of the potential function value from the function value of the current best solution, the less likely an algorithm is to obtain it in the next iteration. Such *transition probabilities* are not taken into account in Theorem 3.2.4. The theorem also does not cover very well the situation in which, at a certain step, fewer than k answers are possible. Even for fully symmetric problem classes, this situation is likely to appear in the later parts of the optimization process, where those problem instances that are still aligned with all previously evaluated function values all map the next query to one out of fewer than k possible function

values. Covering these two shortcomings of Theorem 3.2.4 is one of the main challenges in black-box complexity. One step in this direction is the *matrix lower bound theorem* presented in [9] and the subsequent publication [7]. As also acknowledged there, however, the verification of the conditions under which these two generalizations apply is often quite tedious, so that the two methods are unfortunately not yet easily and very generally applicable. So far, they have been used to derive lower bounds for the black-box complexity of the OneMax and the Jump benchmark functions; see Sections 3.3.2 and 3.3.7.

Another tool that will be very useful in the subsequent sections is the following theorem, which allows us to transfer lower bounds proven for a simpler problem to a problem that is derived from it by a composition with another function. Most notably, it allows us to bound the black-box complexity of functions of unitation (i.e; functions for which the function value depends only on the number of ones in the string) by that of the OneMax problems. We will apply this theorem to show that the black-box complexity of the jump functions is at least as large as that of OneMax; see Section 3.3.7.

Theorem 3.2.5 (generalization of Theorem 2 in [28]). *For all problem classes $\mathcal{F}$, all classes of algorithms $\mathcal{A}$, and all maps $g : \mathbb{R} \to \mathbb{R}$ that are such that for all $f \in \mathcal{F}$ it holds that $\{x \mid g(f(x)) \text{ optimal}\} = \{x \mid f(x) \text{ optimal}\}$ the $\mathcal{A}$-black-box complexity of $g(\mathcal{F}) := \{g \circ f \mid f \in \mathcal{F}\}$ is at least as large as that of $\mathcal{F}$.*

The intuition behind Theorem 3.2.5 is that with a knowledge of $f(x)$, we can compute $g(f(x))$, so that every algorithm that optimizes $g(\mathcal{F})$ can also be used to optimize $\mathcal{F}$, by evaluating the $f(x)$ values, feeding $g(f(x))$ to the algorithm, and querying the solution candidates that this algorithm suggests.

3.2.3 Tools to Prove Upper Bounds

We now present general upper bounds for the black-box complexity of a problem. We recall that, by definition, a small upper bound for the black-box complexity of a problem $\mathcal{F}$ shows that there exists an algorithm which solves every problem instance $f \in \mathcal{F}$ efficiently. When the upper bound for a problem is smaller than the expected performance of well-understood search heuristics, the question of whether these state-of-the-art heuristics can be improved or whether the unrestricted black-box model is too generous arises.

The simplest upper bound for the black-box complexity of a class $\mathcal{F}$ of functions is the expected performance of random sampling without repetitions.

Lemma 3.2.6. *For every finite set S and every class $\mathcal{F} \subset \{f : S \to \mathbb{R}\}$ of real-valued functions over S, the unrestricted black-box complexity of $\mathcal{F}$ is at most $(|S|+1)/2$.*

This simple bound can be tight, as we shall discuss in Section 3.3.1. A similarly simple upper bound is presented in the next subsection.

3.2.3.1 Function Classes vs. Individual Instances

In all of the above we have discussed the black-box complexity of a *class* of functions, and not of individual problem instances. This is justified by the following observation, which also explains why in the following we will usually consider generalizations of the benchmark problems typically studied in the theory of randomized black-box optimization.

Lemma 3.2.7. *For every function $f : S \to \mathbb{R}$, the unrestricted black-box complexity of the class $\{f\}$ that consists only of f is one. The same holds for any class $\mathcal{F}$ of functions that all have their optimum at the same point, i.e., for which there exists a search point $x \in S$ such that, for all $f \in \mathcal{F}$, $x \in \arg\max f$ holds.*

More generally, if $\mathcal{F}$ is a collection of functions $f : S \to \mathbb{R}$ and $X \subseteq S$ is such that for all $f \in \mathcal{F}$ there exists at least one point $x \in X$ such that $x \in \arg\max f$, the unrestricted black-box complexity of $\mathcal{F}$ is at most $(|X|+1)/2$.

For every finite set $\mathcal{F}$ of functions, the unrestricted black-box complexity is bounded from above by $(|\mathcal{F}|+1)/2$.

The proof of this lemma is quite straightforward. For the first statement, the algorithm which queries any point in $\arg\max f$ in the first query certifies this bound. Similarly, the second statement is certified by the algorithm that queries x in the first iteration. The algorithm which queries the points in X in random order proves the third statement. Finally, note that the third statement implies the fourth by letting X be the set that contains, for each function $f \in \mathcal{F}$, one optimal solution $x_f \in \arg\max f$.

Lemma 3.2.7 indicates that function classes $\mathcal{F}$ for which $\cup_{f\in\mathcal{F}} \arg\max f$ or, more precisely, for which a small set X as in the third statement of Lemma 3.2.7 exists are not very interesting research objects in the unrestricted black-box model. We therefore typically choose generalizations of the benchmark problems in such a way that any set X which contains for each objective function $f \in \mathcal{F}$ at least one optimal search point has to be large. We shall often even have $|X| = |\mathcal{F}|$, i.e., the optima of any two functions in $\mathcal{F}$ are pairwise different.

We will see in Section 3.6 that Lemma 3.2.7 does not apply to all of the restricted black-box models. In fact, in the unary unbiased black-box model considered there, the black-box complexity of a single function can be of order $n \log n$. That is, even if the algorithm "knows" where the optimum is, it may still need $\Omega(n \log n)$ steps to generate it.

3.2.3.2 Upper Bounds via Restarts

In several situations, rather than bounding the expected optimization time of a black-box heuristic, it can be easier to show that the probability that it solves a given problem within s iterations is at least p. If p is large enough (for an asymptotic bound, it suffices that this success probability is constant), then a restarting strategy can be used to obtain upper bounds on the black-box complexity of the problem. Either the algorithm is finished after at most s steps, or it is initialized from scratch, independently of all previous runs. This way, we obtain the following lemma.

Lemma 3.2.8 (Remark 7 in [37]). *Suppose for a problem $\mathcal{F}$ that there exists an unrestricted black-box algorithm A that, with constant success probability, solves any instance $f \in \mathcal{F}$ in s iterations (that is, it queries an optimal solution within s queries). Then the unrestricted black-box complexity of $\mathcal{F}$ is at most $O(s)$.*

Lemma 3.2.8 also applies to almost all of the restricted black-box models that we will discuss in Sections 3.4-3.7. In general, it applies to all black-box models in which restarts are allowed. It does not apply to the (strict version of the) elitist black-box model, which we discuss in Section 3.7.4.

3.2.4 Polynomial Bounds for NP-Hard Problems

Our discussion in Section 3.1.1 indicates that the classical complexity notions developed for white-box optimization and decision problems are not very meaningful in the black-box setting. This is impressively demonstrated by a number of NP-hard problems that have a small polynomial black-box complexity. We present such an example here, taken from [54, Section 3].

One of the best-known NP-complete problems is MaxClique. For a given graph $G = (V, E)$ of $|V| = n$ nodes and for a given parameter k, it asks whether there exists a complete subgraph $G' = (V' \subseteq V, E' := E \cap \{\{u, v\} \in E \mid u, v \in V'\})$ of size $|V'| \geq k$. A complete graph is a graph in which every two vertices are connected by a direct edge between them. The optimization version of MaxClique asks us to find a complete subgraph of the largest possible size. A polynomial-time optimization algorithm for this problem implies P=NP.

The unrestricted black-box complexity of MaxClique is, however, only of order n^2. This bound can be achieved as follows. In the first $\binom{n}{2}$ queries, the algorithm queries the presence of individual edges. This way, it learns the structure of the problem instance. From this information, all future solution candidates can be evaluated without any oracle queries. That is, a black-box algorithm can now compute an optimal solution *offline*, i.e., without the need for further function evaluations. This offline computation may take exponential time, but in the black-box complexity model, we do not charge

the algorithm for the time needed between two queries. The optimal solution of the MaxClique instance can be queried in the $\binom{\binom{n}{2}+1}{}$-st query.

Theorem 3.2.9 (Section 3 in [54]). *The unrestricted black-box complexity of* MaxClique *is at most* $\binom{n}{2}+1$ *and thus* $O(n^2)$.

Several similar results can be obtained. For most of the restricted black-box complexity models this has been explicitly done; see also Section 3.6.3.5.

One way to avoid such small complexities would be to restrict the time that an algorithm can spend between any two queries. This suggestion was made in [54]. In our opinion, this requirement would, however, carry a few disadvantages such as a mixture of different complexity measures. We will therefore, in this chapter, not explicitly verify that the algorithms run in polynomial time. Most upper bounds are nevertheless easily seen to be obtained by polynomial-time algorithms. Where polynomial bounds are proven for NP-hard problems, there must be at least one iteration for which the respective algorithm, according to today's knowledge, needs excessive time.

3.3 Known Black-Box Complexities in the Unrestricted Model

We survey existing results for the unrestricted black-box model, and proceed by problem type. For each benchmark problem considered, we first introduce its generalization to classes of similar problem instances. We discuss which of characteristics of the original problem are maintained in these generalizations. We will see that for some classical benchmark problems, different generalizations have been proposed in the literature.

3.3.1 Needle

Our first benchmark problem is an example that shows that the simple upper bound given in Lemma 3.2.6 can be tight. The function that we generalize is the Needle function, which assigns 0 to all search points $s \in S$ except for one distinguished optimum, which has a function value of one. In order to obtain the above-mentioned property that every function in the generalized class has a different optimum than any other function (see the discussion after Lemma 3.2.7), while at the same time maintaining the characteristics of the problem, the following generalization is made. For every $s \in S$, we let $f_s : S \to \mathbb{R}$ be the function which assigns the function value 1 to the unique optimum $s \in S$ and 0 to all other search points $x \neq s$. We let $\textsc{Needle}(S) := \{f_s \mid s \in S\}$ be the set of all such functions.

Confronted with such a function f_s, we do not learn anything about the target string s until we have found it. It seems quite intuitive that the best we can do in such a case is to query search points at random, without repetitions. That this is indeed optimal is the statement of the following theorem, which can be easily proven by Yao's minimax principle applied to NEEDLE(S) with the uniform distribution.

Theorem 3.3.1 (Theorem 1 in [54]). *For every finite set S, the unrestricted black-box complexity of* NEEDLE(S) *is* $(|S|+1)/2$.

3.3.2 OneMax

The best-studied benchmark function in the theory of randomized black-box optimization is certainly ONEMAX. ONEMAX assigns to each bit string x of length n the number $\sum_{i=1}^{n} x_i$ of ones in it. The natural generalization of this particular function to a nontrivial class of functions is as follows.

Definition 3.3.2 (OneMax). For all $n \in \mathbb{N}$ and all $z \in \{0,1\}^n$ let

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

the function that assigns to each length-n bit string x the number of bits in which x and z agree. Being the unique optimum of OM_z, the string z is called its *target string*.

We refer to $\textsc{OneMax}_n := \{\mathrm{OM}_z \mid z \in \{0,1\}^n\}$ as the set of all (generalized) ONEMAX functions. We will often omit the subscript n.

We easily observe that, for every n, the original ONEMAX function OM counting the number of ones corresponds to $\mathrm{OM}_{(1,\dots,1)}$. It is, furthermore, not difficult to prove that, for every $z \in \{0,1\}^n$, the *fitness landscape* of OM_z is *isomorphic* to that of OM. This can be seen by observing that $\mathrm{OM}_z(x) = \mathrm{OM}(x \oplus z \oplus (1,\dots,1))$ for all $x, z \in \{0,1\}^n$, which shows that $\mathrm{OM}_z = \mathrm{OM} \circ \alpha_z$ for the Hamming automorphism $\alpha_z : \{0,1\}^n \to \{0,1\}^n, x \mapsto x \oplus z \oplus (1,\dots,1)$. As we shall discuss in Section 3.6, a Hamming automorphism is a one-to-one map $\alpha : \{0,1\}^n \to \{0,1\}^n$ such that for all x and all z the Hamming distance between x and z is identical to that between $\alpha(x)$ and $\alpha(z)$. This shows that the generalization of OM to functions OM_z preserves its problem characteristics. In essence, the generalization is just a "relabeling" of the search points.

3.3.2.1 The Unrestricted Black-Box Complexity of OneMax

With Definition 3.3.2 at hand, we can study the unrestricted black-box complexity of this important class of benchmark functions.

Interestingly, it turns out that the black-box complexity of $\textsc{OneMax}_n$ has been studied in several different contexts, long before Droste, Jansen, and Wegener introduced black-box complexity. In fact, Erdős and Rényi [55] as well as several other authors studied it in the early 1960s, inspired by a question about so-called *coin-weighing problems.*

In our terminology, Erdős and Rényi [55] showed that the unrestricted black-box complexity of OneMax is at least $(1-o(1))n/\log_2(n)$ and at most $(1+o(1))\log_2(9)n/\log_2(n)$. The upper bound was improved to $(1+o(1))2n/\log_2(n)$ in [10, 73, 74]. Identical or weaker bounds have been proven several times in the literature. Some publications appeared at the same time as the work of Erdős and Rényi (see the discussion in [6]), and some much later [2, 6, 54].

Theorem 3.3.3 ([10, 55, 73, 74]). *The unrestricted black-box complexity of* OneMax *is at least* $(1-o(1))n/\log_2(n)$ *and at most* $(1+o(1))2n/\log_2(n)$. *It is thus* $\Theta(n/\log n)$.

The lower bound in Theorem 3.3.3 follows from Yao's minimax principle, applied to $\textsc{OneMax}_n$ with the uniform distribution. Informally, we can use the arguments given after Theorem 3.2.4: since the optimum can be anywhere in $\{0,1\}^n$, we need to learn the n bits of the target string z. With each function evaluation, we receive at most $\log_2(n+1)$ bits of information, namely the objective value, which is an integer between 0 and n. We therefore need at least (roughly) $n/\log_2(n+1)$ iterations. Using Theorem 3.2.4, this reasoning can be turned into a formal proof.

The upper bound given in Theorem 3.2.4 is quite interesting because it is obtained by a very simple strategy. Erdős and Rényi showed that $O(n/\log n)$ bit strings sampled independently and uniformly at random from the hypercube $\{0,1\}^n$ have a high probability of revealing the target string. That is, an asymptotically optimal unrestricted black-box algorithm for OneMax can just sample $O(n/\log n)$ random samples. From these samples and the corresponding objective values, the target string can be identified without further queries. Its computation, however, may not be possible in polynomial time. The fact that $\textsc{OneMax}_n$ can be optimized in $O(n/\log n)$ queries also in polynomial time was proven in [6].[1] The reader interested in a formal analysis of the strategy used by Erdős and Rényi may refer to Section 3 of [35], where a detailed proof of the $O(n/\log n)$ random sampling strategy is presented.

In the context of *learning*, it is interesting to note that the random sampling strategy of Erdős and Rényi is *nonadaptive*, i.e., the t-th search point does not depend on the previous $t-1$ evaluations. In the black-box context, a last query, in which the optimal solution is evaluated, is needed. This query certainly depends on the previous $O(n/\log n)$ evaluations, but note

[1] Bshouty [6] mentions that also the constructions of Lindström [73, 74] and Cantor and Mills [10] can be done in polynomial time. But this was not explicitly mentioned in the latter publications. The method of Bshouty also has the advantage that it generalizes to OneMax functions over alphabets larger than $\{0,1\}$; see also Section 3.10.

that here we know the answer to this evaluation already (with high probability). For nonadaptive strategies, learning z with $(1+o(1))2n/\log n$ queries is optimal [55]. The intuitive reason for this lower bound is that a random guess typically has an objective value close to $n/2$. More precisely, instead of using the whole range of $n+1$ possible answers, almost all function values are in an $O(\sqrt{n})$ range around $n/2$, giving, very informally, the lower bound $\log_2(n)/\log_2(O(\sqrt{n})) = \Omega(2n/\log n)$.

Using the probabilistic method (or the constructive result of Bshouty [6]), the random sampling strategy can be derandomized. This derandomization says that for every n, there is a sequence of $t = \Theta(n/\log n)$ strings $x^{(1)}, \dots, x^{(t)}$ such that the objective values $\mathrm{OM}_z(x^{(1)}), \dots, \mathrm{OM}_z(x^{(t)})$ uniquely determine the target string z. Such a derandomized version will be used in later parts of this chapter, for example, in the context of the k-ary unbiased black-box complexity of OneMax studied in Section 3.6.3.2.

Theorem 3.3.4 (from [55] and others). *For every n there is a sequence $x^{(1)}, \dots, x^{(t)}$ of $t = \Theta(n/\log n)$ bit strings such that for every two length-n bit strings $y \neq z$ there exists an index i with $\mathrm{OM}_z(x^{(i)}) \neq \mathrm{OM}_y(x^{(i)})$.*

For some very concrete OneMax instances, i.e., for instances of bounded dimension n, very precise bounds for the black-box complexity are known; see [7] and the pointers in [29, Section 1.4] for details. Here, in this chapter, we are only concerned with the asymptotic complexity of OneMax_n with respect to the problem dimension n. Unsurprisingly, this benchmark problem will also be studied in almost all of the restricted black-box models that we describe in the subsequent sections. A summary of known results can be found in Section 3.8.

3.3.3 BinaryValue

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

Two straightforward generalizations of BV to function classes exist. The first one is the collection of all functions

$$\mathrm{BV}_z : \{0,1\}^n \to [0..2^n], x \mapsto \sum_{i=1}^{n} 2^{i-1} \mathbb{1}(x_i, z_i),$$

where $\mathbb{1}(a,b) := 1$ if and only if $a = b$, and $\mathbb{1}(a,b) := 0$ otherwise. In light of Definition 3.3.2, this may seem like a natural extension of BV to a class of functions. It also satisfies our sought condition that for any two functions $\mathrm{BV}_z \neq \mathrm{BV}_{z'}$ the respective optima z and z' differ, so that the smallest set

containing its optimum for each function is the full n-dimensional hypercube $\{0,1\}^n$. However, we can easily see that the unrestricted black-box complexity of the set $\textsc{BinaryValue}^*_n := \{\mathrm{BV}_z \mid z \in \{0,1\}^n\}$ so defined is very small.

Theorem 3.3.5 (Theorem 4 in [54]). *The unrestricted black-box complexity of* $\textsc{BinaryValue}^*_n$ *is* $2-2^{-n}$.

Proof. The lower bound follows from observing that, for an instance BV_z for which z is chosen uniformly at random, the probability of querying the optimum z in the first query is 2^{-n}. In all other cases, at least two queries are needed.

For the upper bound, we only need to observe that for any two target strings $z \neq z'$ and for every search point $x \in \{0,1\}^n$ we have $\mathrm{BV}_z(x) \neq \mathrm{BV}_{z'}(x)$. More precisely, it is easy to see that from $\mathrm{BV}_z(x)$ we can easily determine for which bits $i \in [n]$ the bit value of x_i is identical to z_i. This shows that by querying the objective value of a random string in the first query we can compute the optimum z, which we query in the second iteration if the first value is not already optimal. □

Theorem 3.3.5 is possible because the objective values disclose a lot of information about the target string. A second generalization of BV has therefore been suggested in the literature. In light of the typical behavior of black-box heuristics, which do not discriminate between bit positions, and in particular with respect to the unbiased black-box model defined in Section 3.6, this variant seems to be the more "natural" choice in the context of evolutionary algorithms. This second generalization of BV collects together all functions $\mathrm{BV}_{z,\sigma}$, defined as

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

Denting by $\sigma(x)$ the string $(x_{\sigma(1)} \dots x_{\sigma(n)})$, we easily see that $\mathrm{BV}_{z,\sigma}(x) = \mathrm{BV}\big(\sigma(x \oplus z \oplus (1,\dots,1))\big)$, thus showing that the class $\{\mathrm{BV}_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\}$ can be obtained from BV by composing it with an $\oplus$-shift of the bit values and a permutation of the indices $i \in [n]$. Since $z = \arg\max \mathrm{BV}_{z,\sigma}$, we call z the *target string* of $\mathrm{BV}_{z,\sigma}$. Similarly, we call σ the *target permutation* of $\mathrm{BV}_{z,\sigma}$.

Going through the bit string one by one, i.e., flipping one bit at a time, shows that at most $n+1$ function evaluations are needed to optimize any $\mathrm{BV}_{z,\sigma}$ instance. This simple upper bound can be improved by observing that for each query x and for each $i \in [n]$ we can derive from $\mathrm{BV}_{z,\sigma}(x)$ whether or not $x_{\sigma(i)} = z_{\sigma(i)}$, even if we cannot yet locate $\sigma(i)$. Hence, all we need to do is to identify the target permutation σ. This can be done by a binary search, which gives the following result.

Theorem 3.3.6 (Theorem 16 in [44]). *The unrestricted black-box complexity of* $\textsc{BinaryValue}_n := \{\mathrm{BV}_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\}$ *is at most* $\lceil \log_2 n \rceil + 2$.

In a learning-related sense, in which we want to *learn* both z and σ, the bound in Theorem 3.3.6 is tight, as, informally, the identification of σ requires us to learn $\Theta(\log(n!)) = \Theta(n \log n)$ bits, while with every query we obtain $\log_2(2^n) = n$ bits of information. In our optimization context, however, we do not necessarily need to learn σ in order to optimize $\mathrm{BV}_{z,\sigma}$. A similar situation will be discussed in Section 3.3.6, where we study the unrestricted black-box complexity of LeadingOnes. For LeadingOnes, it can be formally proven that the complexities of optimization and learning are identical (up to at most n queries). We are not aware of any formal statement showing whether or not a similar argument holds for the class $\textsc{BinaryValue}_n$.

3.3.4 Linear Functions

OM and BV are representatives of the class of linear functions $f : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^n f_i x_i$. We can generalize this class in the same way as above to obtain the collection

$$\textsc{Linear}_n := \left\{ f_z : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^n f_i \mathbb{1}(x_i, z_i) \mid z \in \{0,1\}^n \right\}$$

of generalized linear functions. $\textsc{OneMax}_n$ and $\textsc{BinaryValue}_n$ are both contained in this class.

Not much is known about the black-box complexity of this class. The only known bounds are summarized by the following theorem.

Theorem 3.3.7 (Theorem 3.3.3 above and Theorem 4 in [54]). *The unrestricted black-box complexity of the class* $\textsc{Linear}_n$ *is at most* $n+1$ *and at least* $(1-o(1))n/\log_2 n$.

The upper bound is attained by an algorithm that starts with a random or a fixed bit string x and flips one bit at a time, using the better of the parent and the offspring as the starting point for the next iteration. A linear lower bound seems likely, but has not been formally proven.

3.3.5 Monotone and Unimodal Functions

For the sake of completeness, we mention that the class $\textsc{Linear}_n$ is a subclass of the class of generalized monotone functions.

Definition 3.3.8 (monotone functions). Let $n \in \mathbb{N}$ and let $z \in \{0,1\}^n$. A function $f : \{0,1\}^n \to \mathbb{R}$ is said to be *monotone with respect to* z if for all $y, y' \in \{0,1\}^n$ with $\{i \in [n] \mid y_i = z_i\} \subsetneq \{i \in [n] \mid y'_i = z_i\}$ it holds that

$f(y) < f(y')$. The class $\textsc{Monotone}_n$ contains all such functions that are monotone with respect to some $z \in \{0,1\}^n$.

The above-mentioned algorithm which flips one bit at a time (see the discussion after Theorem 3.3.7) solves any of these instances in at most $n+1$ queries, giving the following theorem.

Theorem 3.3.9. *The unrestricted black-box complexity of the class* $\textsc{Monotone}_n$ *is at most* $n+1$ *and at least* $(1-o(1))n/\log_2 n$.

Monotone functions are instances of so-called *unimodal functions.* A function f is unimodal if and only if for every nonoptimal search point x there exists a direct neighbor y of x with $f(y) > f(x)$. The unrestricted black-box complexity of this class of unimodal functions was studied in [54, Section 8], where a lower bound that depends on the number of different function values that the objective functions can map to was presented.

3.3.6 LeadingOnes

After OneMax, probably the second most investigated function in the theory of discrete black-box optimization is the leading-ones function $\mathrm{LO} : \{0,1\}^n \to [0..n]$, which assigns to each bit string x the length of the longest prefix of ones, i.e., $\mathrm{LO}(x) := \max\{i \in [0..n] \mid \forall j \in [i] : x_j = 1\}$. Like for BinaryValue, two generalizations have been studied, an $\oplus$-invariant version and an $\oplus$- and permutation-invariant version. As a consequence of the unbiased black-box complexity model which we will discuss in Section 3.6, the latter is the more frequently studied.

Definition 3.3.10 (LeadingOnes function classes). Let $n \in \mathbb{N}$. For any $z \in \{0,1\}^n$, let

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

the length of the maximal joint prefix of x and z. Let $\textsc{LeadingOnes}_n^* := \{\mathrm{LO}_z \mid z \in \{0,1\}^n\}$.

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

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

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

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

The unrestricted black-box complexity of the set $\textsc{LeadingOnes}_n^*$ is easily seen to be around $n/2$. This is the complexity of the algorithm which starts

with a random string x and, given an objective value of $\mathrm{LO}_z(x)$, replaces x by the string that is obtained from x by flipping the $\mathrm{LO}_z(x)+1$-st bit in x. The lower bound is a simple application of Yao's minimax principle to the uniform distribution over all possible problem instances. It is crucial here to note that the algorithms do not have any information about the "tail" $(z_j \dots z_n)$ until they have seen for the first time a search point of function value at least $j-1$.

Theorem 3.3.11 (Theorem 6 in [54]). *The unrestricted black-box complexity of the set* $\textsc{LeadingOnes}^*_n$ *is* $n/2 \pm o(n)$. *The same holds for the set* $\{\mathrm{LO}_{z,\sigma} \mid z \in \{0,1\}^n\}$, *for any fixed permutation* $\sigma \in S_n$.

The unrestricted black-box complexity of $\textsc{LeadingOnes}_n$ is also quite well understood.

Theorem 3.3.12 (Theorem 4 in [1]). *The unrestricted black-box complexity of* $\textsc{LeadingOnes}_n$ *is* $\Theta(n \log\log n)$.

Both the upper and the lower bounds in Theorem 3.3.12 are quite involved. For the lower bound, Yao's minimax principle is applied to the uniform distribution over the instances $\mathrm{LO}_{z,\sigma}$ with $z_{\sigma(i)} := (i \mod 2), i = 1, \dots, n$. Informally, this choice indicates that the complexity of the $\textsc{LeadingOnes}$ problem originates in the difficulty of identifying the target permutation. Indeed, as soon as we know the permutation, we need at most $n+1$ queries to identify the target string z (and only around $n/2$ on average, by Theorem 3.3.11). To measure the amount of information that an algorithm can have about the target permutation σ, a potential function is designed that maps each search point x to a real number. To prove the lower bound in Theorem 3.3.12, it is necessary to show that, for every query x, the expected increase in this potential is not very large. Using drift analysis, this can be used to bound the expected time needed to accumulate the amount of information needed to uniquely determine the target permutation.

The proof of the upper bound will be sketched in Section 3.6.3.3, in the context of the unbiased black-box complexity of $\textsc{LeadingOnes}_n$.

It may be interesting to note that the $O(n \log\log n)$ bound in Theorem 3.3.12 cannot be achieved by deterministic algorithms. In fact, Theorem 3 in [1] states that the *deterministic unrestricted black-box complexity* of $\textsc{LeadingOnes}_n$ is $\Theta(n \log n)$.

3.3.7 Classes of Jump Functions

Another class of popular pseudo-Boolean benchmark functions is that of so-called "jump" functions. In black-box complexity, this class is currently one of the most intensively studied problems, with a number of surprising results, which in addition carry some interesting ideas for potential refinements of

state-of-the-art heuristics. For this reason, we discuss this class in more detail, and compare the known complexity bounds with running-time bounds for some standard and recently developed heuristics.

For a nonnegative integer ℓ, the function $\textsc{Jump}_{\ell,z}$ is derived from the $\textsc{OneMax}$ function OM_z with target string $z \in \{0,1\}^n$ by "blanking out" any useful information within the strict ℓ-neighborhood of the optimum z and its bitwise complement $\bar{z}$, by giving all these search points a fitness value of 0. In other words,

$$\textsc{Jump}_{\ell,z}(x) := \begin{cases} n & \text{if } \mathrm{OM}_z(x) = n, \\ \mathrm{OM}_z(x) & \text{if } \ell < \mathrm{OM}_z(x) < n-\ell, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3.1}$$

This definition is mostly similar to the two definitions used in [53, 71] that we shall discuss below, which do not fully agree.

3.3.7.1 Known Running-Time Bounds for Jump Functions

We summarize here the known running-time results for the optimization of jump functions via randomized optimization heuristics. The reader interested only in black-box complexity results can skip this section.

Droste, Jansen, and Wegener [53] analyzed the optimization time of the (1+1) EA on jump functions. From this work, it is not difficult to see that the expected running time of the (1+1) EA on $\textsc{Jump}_{\ell,z}$ is $\Theta(n^{\ell+1})$, for all $\ell \in \{1,\dots,\lfloor n/2 \rfloor - 1\}$ and all $z \in \{0,1\}^n$. This running time is dominated by the time needed to "jump" from a local optimum x with function value $\mathrm{OM}_z(x) = n-\ell-1$ to the unique global optimum z.

The *fast genetic algorithm* proposed in [39] significantly reduces this running time by using a generalized variant of standard bit mutation, which goes through its input and flips each bit independently with probability c/n. By choosing in every iteration the expected step size c in this mutation rate c/n from a power-law distribution with exponent β (more precisely, in every iteration, c is chosen independently of all previous iterations, and independently of the current state of the optimization process), an expected running time of $O(\ell^{\beta-0.5}((1+o(1))e/\ell)^\ell n^\ell)$ on $\textsc{Jump}_{\ell,z}$ is achieved, uniformly for all jump sizes $\ell > \beta - 1$. This is only a polynomial factor worse than the $((1+o(1))e/\ell)^\ell n^\ell$ expected running time of a (1+1) EA which for every jump size ℓ uses a bit flip probability of ℓ/n, which is the optimal static choice.

Dang and Lehre [17] showed an expected running time of $O((n/c)^{\ell+1})$ for a large class of nonelitist population-based evolutionary algorithms with mutation rate c/n, where c is supposed to be a constant.

Jump functions were originally studied to investigate the usefulness of crossover, i.e., the recombination of two or more search points into a new solution candidate. In [64], a $(\mu+1)$ genetic algorithm using crossover was

shown to optimize any $\textsc{Jump}_{\ell,z}$ function in an expected number $O(\mu n^2(\ell-1)^3+4^{\ell-1}/p_c)$ of function evaluations, where $p_c<1/(c(\ell-1)n)$ denotes the (artificially small) probability of doing a crossover. In [18] it was shown that for more "natural" parameter settings, most notably those with a nonvanishing probability of crossover, a standard $(\mu+1)$ genetic algorithm which uses crossover followed by mutation has an expected $O(n^\ell \log n)$ optimization time on any $\textsc{Jump}_{\ell,z}$ function, which is a gain by a factor $O(n/\log n)$ over the above-mentioned bound for the standard (1+1) EA. In [16] it was shown that significant performance gains can be achieved by the usage of diversity mechanisms. We refer the interested reader to Chapter 7.6 of this book for a more detailed description of these mechanisms and running-time statements; see in particular Sections 8.4.4 and 8.4.5.

3.3.7.2 The Unrestricted Black-Box Complexity of Jump Functions

From the definition in (3.3.1), we can easily see that for every $n\in\mathbb{N}$ and for all $\ell\in[0..n/2]$ there exists a function $f:[0..n]\to[0..n]$ such that $\textsc{Jump}_{\ell,z}(x)=f(\textsc{OM}_z(x))$ for all $z,x\in\{0,1\}^n$. By Theorem 3.2.5, we therefore obtain the result that for every class $\mathcal{A}$ of algorithms and for all ℓ, the $\mathcal{A}$-black-box complexity of $\textsc{Jump}_{\ell,n}:=\{\textsc{Jump}_{\ell,z}\mid z\in\{0,1\}^n\}$ is at least as large as that of $\textsc{OneMax}_n$. Quite surprisingly, it turns out that this bound can be met for a broad range of jump sizes ℓ. Building on work [28] on the unbiased black-box complexity of jump functions (see Section 3.6.3.4 for a detailed description of the results proven in [27]), the following bounds were obtained in [9].

Theorem 3.3.13 ([9]). *For $\ell<n/2-\sqrt{n}\log_2 n$, the unrestricted black-box complexity of $\textsc{Jump}_{\ell,n}$ is at most $(1+o(1))2n/\log_2 n$, while for $n/2-\sqrt{n}\log_2 n\le\ell<\lfloor n/2\rfloor-\omega(1)$ it is at most $(1+o(1))n/\log_2(n-2\ell)$ (where, in this latter bound, $\omega(1)$ and $o(1)$ refer to $n-2\ell\to\infty$).*

For the extreme case of $\ell=\lfloor n/2\rfloor-1$, the unrestricted black-box complexity of $\textsc{Jump}_{\ell,n}$ is $n+\Theta(\sqrt{n})$.

For all ℓ and every odd n, the unrestricted black-box complexity of $\textsc{Jump}_{\ell,n}$ is at least $\lfloor\log_{\frac{n-2\ell+1}{2}}\left(2^{n-2}(n-2\ell-1)+1\right)\rfloor-\frac{2}{n-2\ell-1}$. For even n, it is at least $\lfloor\log_{\frac{n-2\ell+2}{2}}\left(1+2^{n-1}\frac{(n-2\ell)^2}{n-2\ell-1}\right)\rfloor-\frac{2}{n-2\ell}$.

The proofs of the results in Theorem 3.3.13 are built to a large extent on the techniques used in [28], which we shall discuss in Section 3.6.3.4. In addition to these techniques, [9] introduced a matrix lower-bound method, which allows one to prove stronger lower bounds than the simple information-theoretic result presented in Theorem 3.2.4 by taking into account the fact that the "typical" information obtained through a function evaluation can be

much smaller than what the whole range $\{f(s) \mid s \in S\}$ of possible f-values suggests.

Note that even for the case of "small" $\ell < n/2 - \sqrt{n}\log_2$, the region around the optimum in which the function values are zero is actually quite large. This plateau contains $2^{(1-o(1))n}$ points and has a diameter that is linear in n.

For the case of the extreme jump functions, note also that, apart from the optimum, only the points x with $\mathrm{OM}_z(x) = \lfloor n/2 \rfloor$ and $\mathrm{OM}_z(x) = \lceil n/2 \rceil$ have a nonzero function value. It is thus quite surprising that these functions can nevertheless be optimized so efficiently. We shall see in Section 3.6.3.4 how this is possible.

One may wonder why, in the definition of $\textsc{Jump}_{\ell,n}$, we have fixed the jump size ℓ, as in this way it is "known" to the algorithm. It has been argued in [38] that the algorithms can learn ℓ efficiently, if this is needed; in some cases, including those of small ℓ-values, knowing ℓ may not be needed to achieve the above-mentioned optimization times. Whether or not knowledge of ℓ is needed can be decided adaptively.

3.3.7.3 Alternative Definitions of Jump Functions

Following up on results for the so-called unbiased black-box complexity of jump functions [28] (see Section 3.6.3.4), Jansen [62] proposed an alternative generalization of the classical jump function considered in [53]. To discuss this extension, we recall that the jump function analyzed in [53, Definition 24] is the $(1,\dots,1)$ version of the maps $f_{\ell,z}$ that assign to every length-n bit string x the function value

$$f_{\ell,z}(x) := \begin{cases} \ell + n & \text{if } \mathrm{OM}_z(x) = n, \\ \ell + \mathrm{OM}_z(x) & \text{if } \mathrm{OM}_z(x) \leq n - \ell, \\ n - \mathrm{OM}_z(x) & \text{otherwise.} \end{cases}$$

We first describe the motivation behind the extension considered in the definition given by Equation (3.3.1). To this end, we first note that in the unrestricted black-box complexity model, $f_{\ell,z}$ can be very efficiently optimized by searching for the bitwise complement $\bar{z}$ of z and then inverting this string to the optimal search point z. Note also that, in this definition, the region around the optimum z provides more information than the functions $\textsc{Jump}_{\ell,z}$ defined via (3.3.1). When we are interested in bounding the expected optimization time of classical black-box heuristics, this additional information most often does not pose any problems. But, for our sought black-box complexity studies, this information can make a crucial difference. Lehre and Witt [71] therefore designed a different set of jump functions consisting of maps $g_{\ell,z}$ that assign to each x the function value

$$g_{\ell,z}(x) := \begin{cases} n & \text{if } \mathrm{OM}_z(x) = n, \\ \mathrm{OM}_z(x) & \text{if } \ell < \mathrm{OM}_z(x) \leq n-\ell, \\ 0 & \text{otherwise.} \end{cases}$$

The definition in Equation (3.3.1) is mostly similar to this definition of Lehre and witt, with the only difference being the function values for bit strings x with $\mathrm{OM}_z(x) = n-\ell$. Note that in (3.3.1) the sizes of the "blanked-out areas" around the optimum and its complement are equal, while for $g_{\ell,z}$ the area around the complement is larger than that around the optimum.

As mentioned, Jansen [62] introduced yet another version of the jump function. His motivation was that the spirit of the jump function is to "[locate] an unknown target string that is hidden in some distance to points a search heuristic can find easily". Jansen's definition also has black-box complexity analysis in mind. For a given $z \in \{0,1\}^n$ and a search point x^* with $\mathrm{OM}_z(x^*) > n-\ell$, his jump function h_{ℓ,z,x^*} assigns to every bit string x the function value

$$h_{\ell,z,x^*}(x) := \begin{cases} n+1 & \text{if } x = x^*, \\ n - \mathrm{OM}_z(x) & \text{if } n-\ell < \mathrm{OM}_z(x) \leq n \text{ and } x \neq x^*, \\ \ell + \mathrm{OM}_z(x) & \text{otherwise.} \end{cases}$$

Since these functions do not reveal information about the optimum other than its ℓ-neighborhood, the unrestricted black-box complexity of the class $\{h_{\ell,z,x^*} \mid z \in \{0,1\}^n, \mathrm{OM}_z(x^*) > n-\ell\}$ is $\left(\sum_{i=0}^{\ell-1} \binom{n}{i} + 1\right)/2$ [62, Theorem 4]. This bound also holds if z is fixed to be the all-ones string, i.e., if we consider the unrestricted black-box complexity of the class $\{h_{\ell,(1,\dots,1),x^*} \mid \mathrm{OM}_z(x^*) > n-\ell\}$. For constant ℓ, the black-box complexity of this class of jump functions is thus $\Theta(n^{\ell-1})$, very different from the results for the unrestricted black-box complexity of the $\textsc{Jump}_{\ell,z}$ functions considered above. In contrast to the latter, the expected optimization times stated for crossover-based algorithms in Section 3.3.7.1 above do not necessarily apply to the functions h_{ℓ,z,x^*}, as for these functions the optimum x^* is not located in the middle of the ℓ-neighborhood of z.

3.3.8 Combinatorial Problems

The results described above are mostly concerned with benchmark functions that were introduced to study some particular features of typical black-box optimization techniques, for example, their hill-climbing capabilities (via the OneMax function) or their ability to jump a plateau of a certain size (this is the focus of the jump functions). Running-time analysis, of course, also studies more "natural" combinatorial problems, such as satisfiability problems,

partition problems, scheduling, and graph-based problems such as routing, vertex cover, and MaxCut, see [79] for a survey of running-time results for combinatorial optimization problems.

Apart from a few results for combinatorial problems derived in [54],[2] the first publication to present a systematic investigation of the black-box complexities of combinatorial optimization problems was [37]. In that publication, the two well-known problems of finding a *minimum spanning tree* (MST) in a graph and the *single-source shortest-paths problem* (SSSP) were considered. The study revealed that, for combinatorial optimization problems, the precise formulation of the problem can make a decisive difference. Modeling such problems therefore needs be done with care.

We will not be able to summarize all results proven in [37], but the following summarizes the most relevant ones for the unrestricted black-box model. [37] also studies the MST and the SSSP problem in various restricted black-box models: more precisely, in the unbiased black-box model (see Section 3.6), the ranking-based model (Section 3.5) and combinations thereof. We will briefly discuss results for the unbiased case in Sections 3.6.3.6 and 3.6.4.1.

3.3.8.1 Minimum Spanning Trees

For a given graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}$, the minimum spanning tree problem asks us to find a connected subgraph $G' = (V, E')$ of G such that the sum $\sum_{e \in E'} w(e)$ of the edge weights in G' is minimized. This problem has a natural bit string representation. Letting $m := |E|$, we can fix an enumeration $\nu : [m] \to \mathrm{e}$. In this way, we can identify a length-m bit string $x = (x_1, \ldots, x_m)$ with the subgraph $G_x := (V, E_x)$, where $E_x := \{\nu(i) \mid i \in [m] \text{ with } x_i = 1\}$ is the set of edges i for which $x_i = 1$. Using this interpretation, the MST problem can be modeled as a pseudo-Boolean optimization problem $f : \{0,1\}^m \to \mathbb{R}$; see [79] for details. This formulation is one of the two most common formulations of the MST problem in the evolutionary computation literature. The other common formulation uses a biobjective fitness function $f : \{0,1\}^m \to \mathbb{R}^2$; the first component maps each subgraph to its number of connected components, while the second component measures the total weight of the edges in this subgraph. In the unrestricted black-box model, the two formulations are almost equivalent.[3]

Theorem 3.3.14 (Theorems 10 and 12 in [37]). *For the biobjective and the single-objective formulation of the MST problem on an arbitrary connected*

[2] More precisely, the following combinatorial problems were studied in [54]: MaxClique (Section 3), Sorting (Section 4), and the single-source shortest-paths problem (Sections 4 and 9).

[3] Note that this is not the case for the *ranking-based model* discussed in Section 3.5, since here it can make a decisive difference whether two rankings for the two components of the biobjective function are reported or whether this information is condensed further into one single ranking.

graph of n nodes and m edges, the unrestricted black-box complexity is strictly larger than $n-2$ and at most $2m+1$.

These bounds also apply if, instead of absolute function values $f(x)$, only their rankings are revealed; in other words, the ranking-based black-box complexity (which will be introduced in Section 3.5) of the MST problem is also at most $2m+1$.

The upper bound is shown by first learning the order of the edge weights and then testing, in increasing order of the edge weights, whether or not the inclusion of the corresponding edge forms a cycle or not. This way, the algorithm imitates the well-known MST algorithm of Kruskal.

The lower bound of Theorem 3.3.14 is obtained by applying Yao's minimax principle with a probability distribution on the problem instances that samples uniformly at random a spanning tree, gives weight 1 to all of its edges, and gives weight 2 to all other edges. By Cayley's formula, the number of spanning trees on n vertices is n^{n-2}. In intuitive terms, a black-box algorithm solving the MST problem therefore needs to learn $(n-2)\log_2 n$ bits. Since 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 running time of at least $n-2$ iterations. To turn this intuition into a formal proof, a drift theorem is used to show that in each iteration the number of consistent possible trees decreases by factor of at most $1/n$.

3.3.8.2 Single-Source Shortest Paths

For the SSSP problem, which asks us to connect all vertices of an edge-weighted graph to a distinguished source node through a path of smallest total weight, several formulations coexist. The first one, which was also considered in [54], uses the following multicriteria objective function. An algorithm can query arbitrary trees on $[n]$ and the objective value of any such tree is an $(n-1)$-tuple of the distances of the $n-1$ nonsource vertices to the source $s=1$ (if an edge is traversed which does not exist in the input graph, the entry in the tuple is ∞).

Theorem 3.3.15 (Theorems 16 and 17 in [37]). *For arbitrary connected graphs with n vertices and m edges, the unrestricted black-box complexity of the multiobjective formulation of the SSSP problem is $n-1$. For complete graphs, it is at least $n/4$ and at most $\lfloor (n+1)/2 \rfloor + 1$.*

Theorem 3.3.15 improves on the previous $n/2$ lower and the $2n-3$ upper bound given in [54, Theorem 9]. For the general case, the proof of the upper bound imitates Dijkstra's algorithm by first connecting all vertices to the source, then all but one of the 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. The lower bound is an application of Yao's minimax principle to a bound on deterministic algorithms, which is obtained through an additive drift analysis.

For complete graphs, it is essentially shown that the problem instance can be learned with $\lfloor (n+1)/2 \rfloor$ queries, while the lower bound is again a consequence of Yao's minimax principle.

The bound in Theorem 3.3.15 is not very satisfactory as already the size of the input is $\Omega(m)$. The discrepancy is due to the large amount of information that the objective function reveals about the problem instance. To avoid such low black-box complexities, and to shed a better light on the complexity of the SSSP problem, [37] considered also an alternative model for the SSSP problem, in which a representation of a candidate solution is a vector $(\rho(2), \dots, \rho(n)) \in [n]^{n-1}$. Such a vector is interpreted such that the predecessor of node i is $\rho(i)$ (the indices run from 2 to n to match the indices with the labels of the nodes - node 1 is the source node to which a shortest path is sought). With this interpretation, the search space becomes the set $S_{[2..n]}$ of permutations of $[2..n]$, i.e., $S_{[2..n]}$ is the set of all one-to-one maps $\sigma : [2..n] \to [2..n]$. For a given graph G, the single-criterion objective function f_G is defined by assigning to each candidate solution $(\rho(2), \dots, \rho(n))$ the function value $\sum_{i=2}^{n} d_i$, where d_i is the distance of the i-th node to the source node. If an edge - including loops - is traversed which does not exist in the input graph, d_i is set to n times the maximum weight $w_{\max}$ of any edge in the graph.

Theorem 3.3.16 (Theorem 18 in [37]). *The unrestricted black-box complexity of the SSSP problem with the single-criterion objective function is at most* $\sum_{i=1}^{n-1} i = n(n-1)/2$.

As in the multiobjective case, the bound in Theorem 3.3.16 is obtained by imitating Dijkstra's algorithm. In the single-objective setting, adding the i-th node to the shortest-path tree comes at a cost of at most $n-i$ function evaluations.

3.4 Memory-Restricted Black-Box Complexity

As mentioned in the previous section, as early as in the first publication on black-box complexity [54] it was noted that the unrestricted model can be too generous in the sense that it includes black-box algorithms that are highly problem-tailored and whose expected running time is much smaller than that of typical black-box algorithms. One potential reason for such a discrepancy is the fact that unrestricted algorithms are allowed to store and to access the full search history, while typical heuristics store only a subset ("population" in evolutionary computation) of previously evaluated samples and their corresponding objective values. Droste, Jansen, and Wegener [54] therefore

Algorithm 3.2: The $(\mu+\lambda)$ memory-restricted black-box algorithm for optimizing an unknown function $f : S \to \mathbb{R}$

Initialization:
  $X \leftarrow \emptyset$;
  Choose a probability distribution $D^{(0)}$ over S^μ, sample from it $x^{(1)},\dots,x^{(\mu)} \in S$, and query $f(x^{(1)}),\dots,f(x^{(\mu)})$;
  $X \leftarrow \left\{\left(x^{(1)},f(x^{(1)})\right),\dots,\left(x^{(\mu)},f(x^{(\mu)})\right)\right\}$;
Optimization: for $t = 1,2,3,\dots$ **do**
  Depending only on the multiset X choose a probability distribution $D^{(t)}$ over S^λ, sample from it $y^{(1)},\dots,y^{(\lambda)} \in S$, and query $f(y^{(1)}),\dots,f(y^{(\mu)})$;
  Set $X \leftarrow X \cup \left\{\left(y^{(1)},f(y^{(1)})\right),\dots,\left(y^{(\mu)},f(y^{(\lambda)})\right)\right\}$;
  for $i = 1,\dots,\lambda$ **do** Select $(x,f(x)) \in X$ and update $X \leftarrow X \setminus \{(x,f(x))\}$;

suggested adjusting the unrestricted black-box model to reflect this behavior. In their *memory-restricted model of size* μ,[4] the algorithms can store up to μ pairs $(x,f(x))$ of previous samples. Based only on this information, they decide on the probability distribution D from which the next solution candidate is sampled. Note that this also implies that the algorithm does not have any iteration counter or any other information about the time elapsed so far. Regardless of how many samples have been evaluated already, the sampling distribution D depends only on the μ pairs $\left(x^{(1)},f(x^{(1)})\right),\dots,\left(x^{(\mu)},f(x^{(\mu)})\right)$ stored in the memory.

We extend this model to a $(\mu+\lambda)$ *memory-restricted black-box model*, in which the algorithms have to query λ solution candidates in every round; see Algorithm 3.2 and Fig. 3.2. Following Definition 3.2.1, the $(\mu+\lambda)$ *memory-restricted black-box complexity* of a function class $\mathcal{F}$ is the black-box complexity with respect to the class $\mathcal{A}$ of all $(\mu+\lambda)$ memory-restricted black-box algorithms.

The memory-restricted model of size μ corresponds to the $(\mu+1)$ memory-restricted one, in which only one search point needs to be evaluated per round. Since this variant with $\lambda = 1$ allows the highest degree of adaptation, it is not difficult to verify that for all $\mu \in \mathbb{N}$ and for all $\lambda > 1$ the $(\mu+\lambda)$ memory-restricted black-box complexity of a problem $\mathcal{F}$ is at least as large as its $(\mu+1)$ black-box complexity. The effects of larger λ have been studied in a *parallel black-box complexity model*, which we will discuss in Section 3.7.2.

While it seems intuitive that larger memory sizes yield smaller optimization times, this is not necessarily true for all functions. Indeed, the following discussion shows that memory is not needed for the efficient optimization of OneMax.

[4] In the original publication [54], a memory-restricted algorithm of size μ was called a black-box algorithm with *size bound* μ.

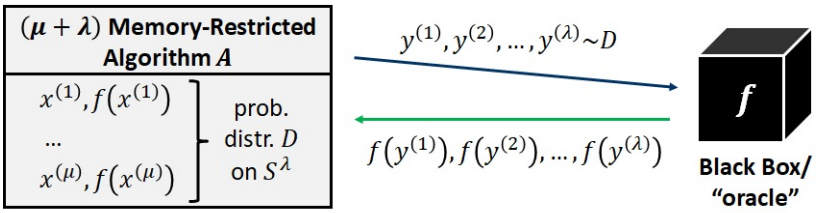

Fig. 3.2 In the $(\mu+\lambda)$ memory-restricted black-box model, the algorithm can store up to μ previously evaluated search points and their absolute function values. In each iteration, it queries the function values of λ new solution candidates. It then has to decide which of the $\mu+\lambda$ search points to keep in the memory for the next iteration.

3.4.1 OneMax

Droste, Jansen, and Wegener conjectured in [54, Section 6] that the $(1+1)$ memory-restricted black-box complexity of OneMax is $O(n \log n)$, in the belief that Randomized Local Search and the (1+1) EA are asymptotically optimal representatives of this class. This conjecture was refuted in [42], where a linear upper bound was presented. This bound was reduced further to $O(n/\log n)$ in [43], showing that even the most restrictive version of the memory-restricted black-box model does not increase the asymptotic complexity of OneMax. By the lower bound presented in Theorem 3.3.3, the $O(n/\log n)$ bound is asymptotically best possible.

Theorem 3.4.1 (Theorem 1 in [43]). *The* $(1+1)$ *memory-restricted black-box complexity of* OneMax *is* $\Theta(n/\log n)$.

The proof of Theorem 3.4.1 makes use of the $O(n/\log n)$ unrestricted strategy of Erdős and Rényi. To respect the memory restriction, the algorithm achieving the $O(n/\log n)$ expected optimization time works in rounds. In every round, a substring of size $s := \sqrt{n}$ of the target string z is identified, using $O(s/\log s)$ queries. The algorithm alternates between querying a string to obtain new information and queries which are used only to store the function value of the last query in the current memory. This works only if sufficiently many bits are available in the guessing string to store this information. It was shown that $O(n/\log n)$ bits suffice. These last remaining $O(n/\log n)$ bits of z are then identified with a constant number of guesses per position, giving an overall expected optimization time of $O(n/s)O(s/\log s) + O(n/\log n) = O(n/\log n)$.

3.4.2 Difference with Respect to the Unrestricted Model

In light of Theorem 3.4.1, it is interesting to find examples for which the $(\mu+1)$ memory-restricted black-box complexity is strictly (and potentially much) larger than its $((\mu+1)+1)$ memory-restricted complexity. This question was addressed in [82].

In the first step, it was shown that having a memory of one can make a decisive difference compared with not being able to store any information at all. In fact, it is easily seen that without any memory, for every function class $\mathcal{F}$ that for every $s \in S$ contains a function f_s such that s is the only optimal solution of f_s, the best one can do without any memory is random sampling, resulting in an expected optimization time of $|S|$. Assume now that there is a (fixed) search point $h \in S$ where a hint is given, in the sense that for all $s \in S$ the objective value $f_s(h)$ uniquely determines where the optimum s is located. Then, clearly, the $(1+1)$ memory-restricted algorithm which first queries h and then, based on $(h, f_s(h))$, queries s solves any problem instance f_s in at most two queries.

This idea can be generalized to a class of functions with two hints hidden in two different distinguished search points h_1 and h_2. Only the combination of $(h_1, f_s(h_1))$ with $(h_2, f_s(h_2))$ defines where the optimum s is located. This way, the $(2+1)$ memory-restricted black-box complexity of this class $\mathcal{F}(h_1, h_2)$ is at most three, while its $(1+1)$ memory-restricted complexity is at least $(S+1)/2$. For, say, $S = \{0,1\}^n$ we thus see that the discrepancies between the $(0+1)$ memory-restricted black-box complexity of a problem $\mathcal{F}$ and its $(1+1)$ memory-restricted complexity can be exponential, and so can be the difference between the $(1+1)$ memory-restricted black-box complexity and the $(2+1)$ memory-restricted complexity. We are not aware of any generalization of this result to arbitrary values of μ.

Theorem 3.4.2 ([82]). *There are classes of functions $\mathcal{F}(h) \subset \{f \mid f : \{0,1\}^n \to \mathbb{R}\}$ and $\mathcal{F}(h_1, h_2) \subset \{f \mid f : \{0,1\}^n \to \mathbb{R}\}$ such that*

- *the $(0+1)$ memory-restricted black-box complexity of $\mathcal{F}(h)$ is exponential in n, while its $(1+1)$ memory-restricted black-box complexity is at most two, and*
- *the $(1+1)$ memory-restricted black-box complexity of $\mathcal{F}(h_1, h_2)$ is exponential in n, while its $(2+1)$ memory-restricted black-box complexity is at most three.*

Storch [82] also presented a class of functions that is efficiently optimized by a standard (2+1) genetic algorithm, which is a $(2+1)$ memory-restricted black-box algorithm, in $O(n^2)$ queries on average, while its $(1+1)$ memory-restricted black-box complexity is exponential in n. This function class is built around so-called royal road functions, the main idea being that the genetic

algorithm is guided towards the two "hints" between which the unique global optimum is located.

3.5 Comparison- and Ranking-Based Black-Box Complexity

Many standard black-box heuristics do not take advantage of knowing *exact* objective values. Instead, they use these function values only to rank the search points. This ranking determines the next steps, so that the absolute function values are not needed. Such algorithms are often referred to as *comparison-based* or *ranking-based*. To understand their efficiency *comparison-based* and *ranking-based* black-box complexity models were suggested in [44, 57, 86].

3.5.1 The Ranking-Based Black-Box Model

In the ranking-based black-box model, the algorithms receive a ranking of the search points currently stored in the memory of the population. This ranking is defined by the objective values of these points.

Definition 3.5.1. Let S be a finite set, let $f : S \to \mathbb{R}$ be a function, and let $\mathcal{C}$ be a subset of 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, $\rho(c) := 1 + |\{c' \in \mathcal{C} \,|\, f(c') < f(c)\}|$.

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

In the ranking-based black-box model without memory restriction, an algorithm thus receives with every query a ranking of *all* previously evaluated solution candidates, while, in the memory-restricted case, naturally, only the ranking of those search points currently stored in the memory is revealed. To be more precise, the ranking-based black-box model without memory restriction subsumes all algorithms that can be described via the scheme of Algorithm 3.3. Fig. 3.3 illustrates these ranking-based black-box algorithms.

Likewise, the $(\mu + \lambda)$ memory-restricted ranking-based model contains those $(\mu+\lambda)$ memory-restricted algorithms that follow the blueprint in Algorithm 3.4; Fig. 3.4 illustrates this pseudocode.

These ranking-based black-box models capture many common search heuristics, such as $(\mu+\lambda)$ evolutionary algorithms, some ant colony optimization algorithms (e.g., simple versions of the Max-Min Ant Systems analyzed in [68, 78]), and Randomized Local Search. They do not include algorithms

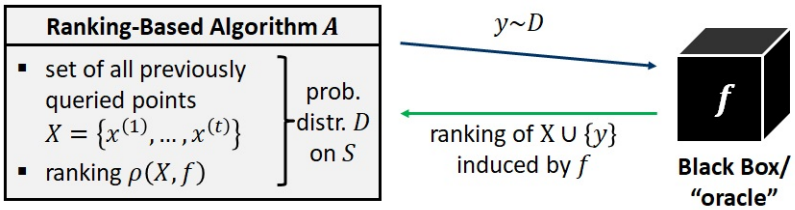

Fig. 3.3 A ranking-based black-box algorithm without memory restriction can store all previously evaluated search points. Instead of knowing their function values, it only has access to the ranking of the search points induced by the objective function f. Based on this, it decides upon a distribution D from which the next search point is sampled.

Algorithm 3.3: Blueprint of a ranking-based black-box algorithm without memory restriction

Initialization:
  Sample $x^{(0)}$ according to some probability distribution $D^{(0)}$ over S;
  $X \leftarrow \{x^{(0)}\}$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  Depending on $\{x^{(0)}, \ldots, x^{(t-1)}\}$ and its ranking $\rho(X, f)$ with respect to f, choose a probability distribution $D^{(t)}$ on S and sample from it $x^{(t)}$;
  $X \leftarrow X \cup \{x^{(t)}\}$;
  Query the ranking $\rho(X, f)$ of X induced by f;

Algorithm 3.4: The $(\mu + \lambda)$ memory-restricted ranking-based black-box algorithm for maximizing an unknown function $f : \{0,1\}^n \to \mathbb{R}$

Initialization:
  $X \leftarrow \emptyset$;
  Choose a probability distribution $D^{(0)}$ over S^μ and sample from it $X = \{x^{(1)}, \ldots, x^{(\mu)}\} \subseteq S$;
  Query the ranking $\rho(X, f)$ of X induced by f;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  Depending only on the multiset X and the ranking $\rho(X, f)$ of X induced by f choose a probability distribution $D^{(t)}$ on S^λ and sample from it $y^{(1)}, \ldots, y^{(\lambda)}$;
  Set $X \leftarrow X \cup \{y^{(1)}, \ldots, y^{(\lambda)}\}$ and query the ranking $\rho(X, f)$ of X induced by f;
  for $i = 1, \ldots, \lambda$ **do** Based on X and $\rho(X, f)$ select a (multi)subset Y of X of size μ and update $X \leftarrow Y$;

with fitness-dependent parameter choices, such as fitness-proportional mutation rates or fitness-dependent selection schemes.

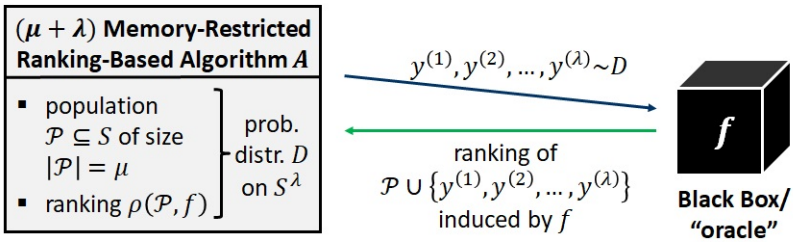

Fig. 3.4 A $(\mu+\lambda)$ memory-restricted ranking-based black-box algorithm can store up to μ previously evaluated search points and the ranking of this population induced by the objective function f. Using only this information, λ new solution candidates are sampled in each iteration and the ranking of the $(\mu+\lambda)$ points is revealed. Based on this ranking, the algorithm needs to select which μ points to keep in the memory.

Surprisingly, the unrestricted and the nonmemory-restricted ranking-based black-box complexities of OneMax coincide in asymptotic terms; the leading constants may be different.

Theorem 3.5.2 (Theorem 6 in [44]). *The ranking-based black-box complexity of* OneMax *without memory restriction is* $\Theta(n/\log n)$.

The upper bound for OneMax is obtained by showing that, for a sufficiently large sample base, a median search point x (i.e., a search point for which half of the search points have a ranking that is at most as large as that of x and the other half of the search points have a ranking that is at least as large as that of x) is very likely to have $n/2$ correct bits. It was shown furthermore that with $O(n/\log n)$ random queries, each of the function values in the interval $[n/2-\kappa\sqrt{n}, n/2+\kappa\sqrt{n}]$ appears at least once. This information is used to translate the ranking of the random queries into absolute function values, for those solution candidates y for which $\mathrm{OM}_z(y)$ lies in the interval $[n/2-\kappa\sqrt{n}, n/2+\kappa\sqrt{n}]$. The proof is then concluded by showing that it suffices to consider only these samples in order to identify the target string z of the problem instance OM_z.

For BinaryValue, in contrast, it makes a substantial difference whether absolute or relative objective values are available.

Theorem 3.5.3 (Theorem 17 in [44]). *The ranking-based black-box complexity of* $\textsc{BinaryValue}_n$ *and* $\textsc{BinaryValue}_n^*$ *is strictly larger than* $n-1$, *even when the memory is not bounded.*

This lower bound of $n-1$ is almost tight. In fact, an $n+1$ ranking-based algorithm is easily obtained by starting with a random initial search point and then, from left to right, flipping exactly one bit in each iteration. The ranking uniquely determines the permutation σ and the string z of the problem instance $\mathrm{BV}_{z,\sigma}$.

Theorem 3.5.3 can be shown with Yao's minimax principle applied to the uniform distribution over the problem instances. The crucial observation is that when optimizing $\mathrm{BV}_{z,\sigma}$ with a ranking-based algorithm, from t samples we can learn at most $t-1$ bits of the hidden bit string z, and not $\Theta(t \log t)$ bits as one might guess from the fact that there are $t!$ permutations of the set $[t]$.

This last intuition, however, gives a very general lower bound. Intuitively, if $\mathcal{F}$ is such that every $z \in \{0,1\}^n$ is the unique optimum for a function $f_z \in \mathcal{F}$, and we learn only the ranking of the search points evaluated so far, then for the t-th query we learn at most $\log_2(t!) = \Theta(t \log t)$ bits of information. Since we need to learn n bits in total, the ranking-based black-box complexity of $\mathcal{F}$ is of order at least $n/\log n$.

Theorem 3.5.4 (Theorem 21 in [44]). *Let $\mathcal{F}$ be a class of functions such that each $f \in \mathcal{F}$ has a unique global optimum and such that for all $z \in \{0,1\}^n$ there exists a function $f_z \in \mathcal{F}$ with $\{z\} = \arg\max f_z$. Then the unrestricted ranking-based black-box complexity of $\mathcal{F}$ is $\Omega(n/\log n)$.*

Results for the ranking-based black-box complexity of the two combinatorial problems MST and SSSP have been derived in [37]. Some of these bounds were mentioned in Section 3.3.8.

3.5.2 The Comparison-Based Black-Box Model

In the ranking-based model, the algorithms receive for every query quite a lot of information, namely the full ranking of the current population and its offspring. One may argue that some evolutionary algorithms use even less information. Instead of considering the full ranking, they base their decisions on a few selected points only. This idea is captured in the *comparison-based black-box model*. In contrast to the ranking-based model, here only the ranking of the queried points is revealed. In this model it can therefore make sense to query a search point more than once, to compare it with a different offspring, for example. Fig. 3.5 illustrates the $(\mu+\lambda)$ memory-restricted comparison-based black-box model. A comparison-based model without memory restriction is obtained by setting $\mu = \infty$.

We will not detail this model further, as it has received only marginal attention so far in the black-box complexity literature. We note, however, that Teytaud and co-authors [57, 86] have presented some very general lower bounds for the convergence rate of comparison-based and ranking-based evolutionary strategies in continuous domains. From these studies, results for the comparison-based black-box complexity of problems defined over discrete domains can be obtained. These bounds, however, seem to coincide with the information-theoretic ones that can be obtained through Theorem 3.2.4.

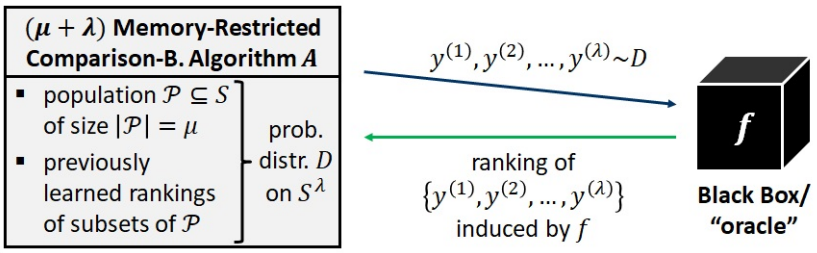

Fig. 3.5 A $(\mu+\lambda)$ memory-restricted comparison-based black-box algorithm can store up to μ previously evaluated search points and the comparison of these points that have been learned through previous queries. In the next iteration, λ solution candidates are queried, possibly containing some of the current population. Only the ranking of the μ queried points is revealed. Based on this ranking and the previous information about the relative fitness values, the algorithm needs to select which μ points to keep in the memory.

3.6 Unbiased Black-Box Complexity

As previously commented, the quest to develop a meaningful complexity theory for evolutionary algorithms and other black-box optimization heuristics seemed to have come to an early end after 2006, the only publication which picked up on this topic being that of Anil and Wiegand on the unrestricted black-box complexity of OneMax [2] (see Section 3.3.2). In 2010, the situation changed drastically. Black-box complexity was revived by Lehre and Witt in [71] (a journal version appeared as [72]). To overcome the drawbacks of the previous unrestricted black-box model, they restricted the class of black-box optimization algorithms in a natural way that still covers a large class of the classically used algorithms.

In their *unbiased black-box complexity model*, Lehre and Witt considered pseudo-Boolean optimization problems $\mathcal{F} \subseteq \{f : \{0,1\}^n \to \mathbb{R}\}$. The unbiased black-box model requires that all solution candidates must be sampled from distributions that are *unbiased.* In the context of pseudo-Boolean optimization, unbiasedness means that the distribution cannot discriminate between bit positions $1, 2, \dots, n$ nor between the bit entries 0 and 1; a formal definition will be given in Sections 3.6.1 and 3.6.2. The unbiased black-box model also admits a notion of arity. A k-ary unbiased black-box algorithm is one that employs only such variation operators that take up to k arguments. This allows one, for example, to talk about mutation-only algorithms (unary unbiased algorithms) and to study the potential benefits of recombining previously sampled search points through distributions of higher arity.

In a crucial difference from the memory-restricted model, in the pure version of the unbiased black-box model the memory is not restricted. That is, the k search points that form the input for the k-ary variation operator can be any random or previously evaluated solution candidate. As in the case of

the comparison- and the ranking-based black-box models, combined unbiased memory-restricted models have also been studied; see Section 3.7.

Before we formally introduce unbiased black-box models for pseudo-Boolean optimization problems in Section 3.6.2, we define and discuss in Section 3.6.1 the concept of unbiased variation operators. Known black-box complexities in the unbiased black-box models are surveyed in Section 3.6.3. In Section 3.6.4 we present extensions of the unbiased black-box models to search spaces different from $\{0,1\}^n$.

3.6.1 Unbiased Variation Operators

In order to formally define the unbiased black-box model, we first introduce the notion of *k-ary unbiased variation operators*. Informally, a k-ary unbiased variation operator takes as input up to k search points. It samples a new point $z \in \{0,1\}^n$ by applying some procedure to these previously evaluated solution candidates that treats all bit positions and the two bit values in an equal way.

Definition 3.6.1 (k-ary unbiased variation operator). Let $k \in \mathbb{N}$. A *k-ary unbiased distribution* $(D(. \mid y^{(1)}, \dots, y^{(k)}))_{y^{(1)},\dots,y^{(k)} \in \{0,1\}^n}$ is a family of probability distributions over $\{0,1\}^n$ such that for all inputs $y^{(1)}, \dots, y^{(k)} \in \{0,1\}^n$ the following two conditions hold:

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

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

To get some intuition about unbiased variation operators, we now summarize a few characterizations and consequences of Definition 3.6.1.

We first note that the combination of $\oplus$- and permutation invariance can be characterized as invariance under Hamming automorphisms. A Hamming automorphism is a one-to-one map $\alpha : \{0,1\}^n \to \{0,1\}^n$ that satisfies the condition that for any two points $x, y \in \{0,1\}^n$ their Hamming distance $H(x,y)$ is equal to the Hamming distance $H(\alpha(x), \alpha(y))$ of their images. A formal proof of the following lemma can be found in [37, Lemma 3].

Lemma 3.6.2. *A distribution $D(\cdot \mid x^1, \dots, x^k)$ is unbiased if and only if, for all Hamming automorphisms $\alpha : \{0,1\}^n \to \{0,1\}^n$ and for all bit strings $y \in \{0,1\}^n$, the probability $D(y \mid x^1, \dots, x^k)$ of sampling y from $(x^1, \dots, x^k)$ equals the probability $D(\alpha(y) \mid \alpha(x^1), \dots, \alpha(x^k))$ of sampling $\alpha(y)$ from $(\alpha(x^1), \dots, \alpha(x^k))$.*

It is not difficult to see that the only 0-ary unbiased distribution over $\{0,1\}^n$ is the uniform one.

1-ary operators, also called *unary* operators, are sometimes referred to as *mutation operators,* in particular in the field of evolutionary computation. Standard bit mutation, as used in several $(\mu+\lambda)$ EAs and $(\mu+\lambda)$ EAs, is a unary unbiased variation operator. The random bit flip operation used by RLS, which chooses at random a bit position $i \in [n]$ and replaces the entry x_i by the value $1-x_i$, is also unbiased. In fact, all unary unbiased variation operators are of a very similar type, as the following definition and lemma, taken from [31], show. These results can be derived from a more general description of unbiased operators offered in [37], which characterizes unbiased operations on arbitrary search spaces. When restricted to pseudo-Boolean optimization, we obtain the following geometric interpretation.

Definition 3.6.3. Let $n \in \mathbb{N}$ and $r \in [0..n]$. For every $x \in \{0,1\}^n$, let flip_r be the variation operator that creates an offspring y from x by selecting r positions $i_1, \dots, i_r$ in $[n]$ uniformly at random (without replacement), setting $y_i := 1-x_i$ for $i \in \{i_1, \dots, i_r\}$, and copying $y_i := x_i$ for all other bit positions $i \in [n] \setminus \{i_1, \dots, i_r\}$.

Using this definition, unary unbiased variation operators can be characterized as follows.

Lemma 3.6.4 (Lemma 1 in [31]). *For every unary unbiased variation operator $(p(\cdot|x))_{x\in\{0,1\}^n}$, there exists a family of probability distributions $(r_{p,x})_{x\in\{0,1\}^n}$ on $[0..n]$ such that for all $x,y \in \{0,1\}^n$ the probability $p(y|x)$ that $(p(\cdot|x))_{x\in\{0,1\}^n}$ samples y from x equals the probability that the routine first samples a random number r from $r_{p,x}$ and then obtains y by applying flip_r to x. On the other hand, each such family of distributions $(r_{p,x})_{x\in\{0,1\}^n}$ on $[0..n]$ induces a unary unbiased variation operator.*

From this characterization, we can easily see that neither the somatic contiguous hypermutation operator used in artificial immune systems (which selects a random position $i \in [n]$ and a random length $\ell \in [n]$ and flips the ℓ consecutive bits in positions $i, i+1 \mod n, \dots, i+\ell \mod n$; see [15, Algorithm 3]), nor the asymmetric nor the position-dependent mutation operators considered in [63] and [12, 27], respectively, are unbiased.

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

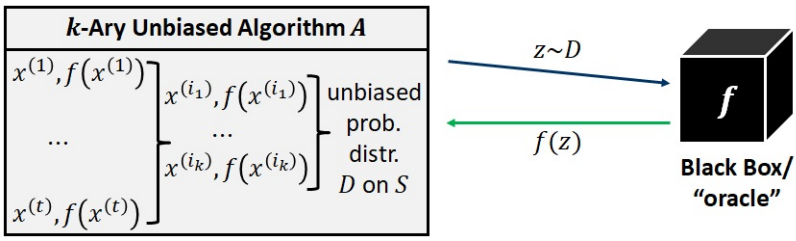

Fig. 3.6 In the k-ary unbiased black-box model, the algorithm can store the full query history. For every already evaluated search point x, the algorithm has access to the absolute function value $f(x) \in \mathbb{R}$. The distributions D from which new solution candidates are sampled have to be unbiased. They can depend on up to k previously evaluated solution candidates.

Some publications refer to the unbiased black-box model allowing variation operators of arbitrary arity as the *$*$-ary unbiased black-box model.* Black-box complexities in the $*$-ary unbiased black-box model are of the same asymptotic order as those in the unrestricted model. This has been formally shown in [81], for a general notion of unbiasedness that is not restricted to pseudo-Boolean optimization problems (see Definition 3.6.16).

Theorem 3.6.5 (Corollary 1 in [81]). *The $*$-ary unbiased black-box complexity of a problem class $\mathcal{F}$ is the same as its unrestricted black-box complexity.*

Apart from [81], most research on the unbiased black-box model assumes a restriction on the arity of the variation operators. We therefore concentrate in the remainder of this chapter on such restricted settings.

3.6.2 The Unbiased Black-Box Model for Pseudo-Boolean Optimization

With Definition 3.6.1 and its characterizations at hand, we can now introduce the unbiased black-box models. The k-ary unbiased black-box model covers all algorithms that follow the blueprint of Algorithm 3.5. Fig. 3.6 illustrates these algorithms. As in previous sections, the *k-ary unbiased black-box complexity* of some class of functions $\mathcal{F}$ is the complexity of $\mathcal{F}$ with respect to all k-ary unbiased black-box algorithms.

As Fig. 3.6 indicates, it is important to note that in line 3 of Algorithm 3.5 the k selected previously evaluated search points $x^{(i_1)}, \ldots, x^{(i_k)}$ do not necessarily have to be the k *immediately previously* queried ones. That is, the algorithm can store and is allowed to choose from *all* previously sampled search points.

Algorithm 3.5: Scheme of a k-ary unbiased black-box algorithm

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

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

The unary unbiased black-box model captures most of the commonly used mutation-based algorithms, such as the $(\mu+\lambda)$ EA and the (μ,λ) EA, Simulated Annealing, the Metropolis algorithm, and Randomized Local Search. The binary unbiased model subsumes many traditional genetic algorithms, such as the $(\mu+\lambda)$ GAs and the (μ,λ) GAs using uniform crossover. As we shall discuss in Section 3.9, the $(1+(\lambda,\lambda))$ GA introduced in [25] is also binary unbiased.

As a word of warning, we note that in [85] and [87] lower bounds are proven for what the authors of those publications call *mutation-based algorithms.* Their definitions are more restrictive than what Algorithm 3.5 proposes. The lower bounds proven in [85, 87] therefore do not (immediately) apply to the unary unbiased black-box model. A comparison of Theorem 12 in [85] and Theorem 3.1(5) in [87] with Theorem 9 in [31] shows that there can be substantial differences (in this case, a multiplicative factor $\approx e$ in the lower bound for the complexity of OneMax with respect to all mutation-based and all unary unbiased black-box algorithms, respectively). One of the main differences between the different models is that in [85, 87] only algorithms using standard bit mutation are considered. This definition excludes algorithms such as RLS for which the radius r fed into the variation operator flip_r is deterministic and is thus not sampled from a binomial distribution $\text{Bin}(n,p)$. When using the term "mutation-based algorithms," we should therefore always make precise which algorithmic framework we are referring to. Here, in this chapter, we will exclusively refer to the unary unbiased black-box algorithms defined via Algorithm 3.5.

3.6.3 Existing Results for Pseudo-Boolean Problems

In this section we survey existing bounds for the unbiased black-box complexity of several classical benchmark functions. As in previous sections, we proceed by function class, and not in historical order.

3.6.3.1 Functions with a Unique Global Optimum

As discussed in Section 3.2.3.1, the unrestricted black-box complexity of every function class $\mathcal{F} = \{f\}$ containing only one function f is one, certified by the algorithm that simply queries a point $x \in \arg\max f$ in the first step. The situation is different in the unbiased black-box model, as the following theorem reveals.

Theorem 3.6.6 (Theorem 6 in [71]). *Let $f : \{0,1\}^n \to \mathbb{R}$ be a function that has a single global optimum (i.e., in the case of maximization, the size of the set $\arg\max f$ is one). The unary unbiased black-box complexity of f is $\Omega(n \log n)$.*

Theorem 3.6.6 gives an $\Omega(n \log n)$ lower bound on the unary unbiased black-box complexity of several standard benchmark functions, such as OneMax and LeadingOnes. We shall see below that for some of these classes, including OneMax, this bound is tight, since it is met by different unary unbiased heuristics, such as the $(1+1)$ EA or RLS. For other classes, including LeadingOnes, the lower bound can be improved through problem-specific arguments.

The proof of Theorem 3.6.6 uses multiplicative drift analysis. To this end, the potential $P(t)$ of an algorithm at time t is defined as the smallest Hamming distance from any of the previously queried search points $x^{(1)}, \ldots, x^{(t)}$ to the unique global optimum z or its bitwise complement $\bar{z}$. The algorithm has identified z (or its complement) if and only if $P(t) = 0$. The distance to $\bar{z}$ needs to be considered, as the algorithm that first identifies $\bar{z}$ and then flips all bits obtains z from $\bar{z}$ in only one additional query. As we have discussed for jump functions in Section 3.3.7, for some functions it can be substantially easier to identify $\bar{z}$ than to identify z itself. This is true in particular if there are paths leading to $\bar{z}$, such as in the original jump functions $f_{\ell,z}$ discussed in Section 3.3.7.3. The key step in the proof of Theorem 3.6.6 is to show that in one iteration $P(t)$ decreases by at most $200P(t)/n$, in expectation, provided that $P(t)$ is between $c \log \log n$ (for some positive constant $c > 0$) and $n/5$. Put differently, in this case $E[P(t) - P(t+1) \mid P(t)] \leq \delta P(t)$ for $\delta := 200/n$. It can be shown, furthermore, that the probability of making very large gains in potential is very small. These two statements allow the application of a multiplicative drift theorem, which bounds the total expected optimization time by $\Omega((\log(n/10) - \log\log(n))/\delta) = \Omega(n \log n)$, provided that the algorithm reaches a state t with $P(t) \in (n/10, n/5]$. A short proof that every unary unbiased black-box algorithm reaches such a state with probability $1 - e^{-\Omega(n)}$ then concludes the proof of Theorem 3.6.6.

3.6.3.2 OneMax

The Unary Unbiased Black-Box Complexity of OneMax

Since OneMax is a unimodal function, the lower bound of Theorem 3.6.6 certainly applies to that function, thus showing that no unary unbiased black-box optimization can optimize OneMax faster than in expected time $\Omega(n \log n)$. This bound is attained by several classical heuristics, such as RLS, the (1+1) EA, and others. While the (1+1) EA has an expected optimization time of $(1 \pm o(1))en \ln(n)$ [32, 85], that of RLS is only $(1 \pm o(1))n \ln(n)$. More precisely, it is $n \ln(n/2) + \gamma n \pm o(1)$ [23], where $\gamma \approx 0.5772156649\ldots$ is the Euler-Mascheroni constant. The unary unbiased black-box complexity of OneMax is just slightly smaller than this term. It was slightly improved by an additive $\sqrt{n \log n}$ term in [69] through iterated initial sampling. In [31], the following very precise bound for the unary unbiased black-box complexity of OneMax was shown. It is smaller than the expected running time of RLS by an additive term that is between $0.138n \pm o(n)$ and $0.151n \pm o(n)$. It was also proven in [31] that a variant of RLS that uses fitness-dependent neighborhood structures attains this optimal bound, up to additive $o(n)$ lower-order terms.

Theorem 3.6.7 (Theorem 9 in [31]). *The unary unbiased black-box complexity of* OneMax *is* $n \ln(n) - cn \pm o(n)$ *for a constant* c *between* 0.2539 *and* 0.2665.

The Binary Unbiased Black-Box Complexity of OneMax

When Lehre and Witt initially defined the unbiased black-box model, they conjectured that also the binary black-box complexity of OneMax was $\Omega(n \log n)$ [P.K. Lehre, personal communication in 2010]. In light of our understanding of the role and usefulness of crossover in black-box optimization, such a bound would have indicated that crossover cannot be beneficial for simple hill-climbing tasks. Given that in 2010 all results seemed to indicate that it was at least very difficult, if not impossible, to rigorously prove any advantages of crossover for problems with smooth fitness landscapes, this conjecture came along very naturally. It was, however, soon refuted. In [35], a binary unbiased algorithm was presented that achieves linear expected running time on OneMax.

Theorem 3.6.8 (Theorem 9 in [35]). *The binary unbiased black-box complexity of* OneMax *and that of any other monotone function is at most linear in the problem dimension* n.

This bound is attained by the algorithm that keeps in the memory two strings x and y that agree in those positions for which the optimal entry

has been identified already, and which differ in all other positions. In every iteration, the algorithm flips a fair random coin and, depending on the outcome of this coin flip, flips exactly one bit in x or one bit in y. The bit to be flipped is chosen uniformly at random from those bits in which x and y disagree. The offspring so created replaces its parent if and only if its function value is larger. In this case, the Hamming distance between x and y reduces by one. Since the probability of choosing the right parent equals $1/2$, it is not difficult to show that, with high probability, for all constants $\varepsilon > 0$, this algorithm optimizes OneMax within at most $(2+\varepsilon)n$ iterations. Together with Lemma 3.2.8, this proves Theorem 3.6.8.

A drawback of this algorithm is that it is very problem-specific, and it has been an open question whether or not a "natural" binary evolutionary algorithm can achieve an $o(n \log n)$ (or better) expected running time on OneMax. This question was answered affirmatively in [25] and [21], as we shall discuss in Section 3.9.

Whether or not the linear bound in Theorem 3.6.8 is tight remains an open problem. In general, proving lower bounds for unbiased black-box models of arities larger than one remains one of the biggest challenges in black-box complexity theory. Owing to the greatly enlarged computational power of black-box algorithms using higher-arity operators, proving lower bounds in these models seems to be significantly harder than in the unary unbiased model. As a matter of fact, the best lower bound that we have for the binary unbiased black-box complexity of OneMax is the $\Omega(n/\log n)$ one stated in Theorem 3.3.3, and not even constant-factor improvements of this bound exist.

The k-Ary Unbiased Black-Box Complexity of OneMax

In [35], a general bound for the k-ary unbiased black-box complexity of OneMax of order $n/\log k$ was presented (see Theorem 9 in [35]). This bound has been improved in [45].

Theorem 3.6.9 (Theorem 3 in [45]). *For every $2 \leq k \leq \log n$, the k-ary unbiased black-box complexity of* OneMax *is of order at most n/k. For $k \geq \log n$, it is $\Theta(n/\log n)$.*

Note that for $k \geq \log n$, the lower bound in Theorem 3.6.9 follows from the $\Omega(n/\log n)$ unrestricted black-box complexity of OneMax discussed in Theorem 3.3.3.

The main idea used to achieve the results of Theorem 3.6.9 can be easily described. For a given k, the bit string is split into blocks of size $k-2$. This has to be done in an unbiased way, so that the "blocks" are not consecutive bit positions, but some random $k-2$ positions not previously optimized. Similarly to the binary case, two reference strings x and y are used to encode which $k-2$ bit positions are currently under investigation, namely the $k-2$

bits in which x and y disagree. Using the same encoding, two other strings x' and y' store which bits have been optimized already, and which ones have not been investigated so far. To optimize the $k-2$ bits in which x and y differ, the derandomized version of the result of Erdős and Rényi (Theorem 3.3.4) is used. Applied in our context, this result states that there exists a sequence of $\Theta(k/\log k)$ queries which uniquely determines the entries in the $k-2$ positions. Since $\Theta(n/k)$ such blocks need to be optimized, the claimed total expected optimization time of $\Theta(n/\log k)$ follows. Some technical difficulties need to be overcome to implement this strategy in an unbiased way. To this end, in [45] a generally applicable *encoding strategy* was presented that with k-ary unbiased variation operators simulates a memory of 2^{k-2} bits that can be accessed in an unrestricted fashion.

3.6.3.3 LeadingOnes

The Unary Unbiased Black-Box Complexity of LeadingOnes

Since LeadingOnes is a classic benchmark problem, unsurprisingly, Lehre and Witt had already presented in [72] a first bound for the unbiased black-box complexity of this function.

Theorem 3.6.10 (Theorem 2 in [72]). *The unary unbiased black-box complexity of* LeadingOnes *is* $\Theta(n^2)$.

Theorem 3.6.10 can be proven by drift analysis. To this end, in [72] a potential function was defined that maps the state of the search process at time t (i.e., the sequence $\{(x^{(1)}, f(x^{(1)})), \dots, (x^{(t)}, f(x^{(t)}))\}$ of the pairs of search points evaluated so far and their respective function values) to the largest number of initial ones and initial zeros in any of the $t+1$ strings $x^{(1)}, \dots, x^{(t)}$. It was then shown that a given potential k cannot increase in one iteration by more than an additive term $4/(k+1)$, in expectation, provided that k is at least $n/2$. Since with probability at least $1-e^{-\Omega(n)}$ any unary unbiased black-box algorithm reaches a state in which the potential is between $n/2$ and $3n/4$, and since from this state a total potential of at least $n/4$ must be gained, the claimed $\Omega(n^2)$ bound follows from a variant of the additive drift theorem. More precisely, using these bounds, the additive drift theorem shows that the total optimization time of any unary unbiased black-box algorithm is at least $(n/4)/(4/(n/2))) = \Omega(n^2)$.

The Binary Unbiased Black-Box Complexity of LeadingOnes

Similarly to the case of OneMax, the binary unbiased black-box complexity of LeadingOnes is much smaller than its unary counterpart.

Theorem 3.6.11 (Theorem 14 in [35]). *The binary unbiased black-box complexity of* LeadingOnes *is* $O(n \log n)$.

The algorithm achieving this bound borrows its main idea from the binary unbiased algorithm used to optimize OneMax in linear time, which we described after Theorem 3.6.8. We recall that the key strategy was to use two strings to encode those bits that have been optimized already. In the $O(n \log n)$ algorithm for LeadingOnes, this approach is combined with a *binary search* for the (unique) bit position that needs to be flipped next. Such a binary search step requires $O(\log n)$ steps in expectation. Iterating it n times gives the claimed $O(n \log n)$ bound.

As in the case of OneMax, it is not known whether or not the bound in Theorem 3.6.11 is tight. The best known lower bound is the $\Omega(n \log \log n)$ one for the unrestricted black-box model discussed in Theorem 3.3.12.

The Complexity of LeadingOnes in Unbiased Black-Box Models of Higher Arity

The $O(n \log n)$ bound presented in Theorem 3.6.11 reduces further to at most $O(n \log(n)/\log \log n)$ in the ternary unbiased black-box model.

Theorem 3.6.12 (Theorems 2 and 3 in [41]). *For every* $k \geq 3$*, the* k*-ary unbiased black-box complexity of* LeadingOnes *is of order at most* $n \log(n)/\log \log n$*. This bound also holds in the combined* k*-ary unbiased ranking-based black-box model, in which instead of absolute function values the algorithm can make use only of the ranking of the search points induced by the optimization problem instance* f*.*

The algorithm that certifies the upper bound in Theorem 3.6.12 uses the additional power gained through the larger arity to *parallelize* the binary search of the binary unbiased algorithm described after Theorem 3.6.11. More precisely, the optimization process is split into phases. In each phase, the algorithm identifies the entries of up to $k := O(\sqrt{\log n})$ positions. It can be shown that each phase takes $O(k^3/\log k^2)$ steps in expectation. Since there are n/k phases, a total expected optimization time of $O(nk^2/\log k^2) = O(n \log(n)/\log \log n)$ follows.

The idea of parallelizing the search for several indices was later taken up and further developed in [1], where an iterative procedure with *overlapping* phases was used to derive the asymptotically optimal $\Theta(n \log \log n)$ unrestricted black-box algorithm that proves Theorem 3.3.12.

It seems plausible that higher arities allow a larger degree of parallelization, but no formal proof of this intuition exists. In the context of LeadingOnes, it would be interesting to derive a lower bound on the smallest value of k such that an asymptotically optimal k-ary unbiased $\Theta(n \log \log n)$ black-box algorithm for LeadingOnes exists. As a first step towards answering

this question, the encoding and sampling strategies sketched above could be applied to the algorithm presented in [1], to understand the smallest arity needed to implement this algorithm in an unbiased way.

3.6.3.4 Jump Functions

Jump functions are benchmark functions, which are observed to be difficult for evolutionary approaches because of their large plateaus of constant and low fitness around the global optimum. One would expect that this would be reflected in the unbiased black-box complexity, at least in the unary model. Surprisingly, this is not the case. In [28], it was shown that even extreme jump functions that reveal only the three different fitness values 0, $n/2$, and n have a small polynomial unary unbiased black-box complexity. That is, they can be optimized quite efficiently by unary unbiased approaches. This result indicates that efficient optimization is not necessarily restricted to problems for which the function values reveal a lot of information about the instance at hand.

As discussed in Section 3.3.7, the literature is not unanimous with respect to how to generalize the jump function defined in [53] to a problem class. The results stated in the following apply to the jump function defined in (3.3.1). In the unbiased black-box model, we can assume without loss of generality that the underlying target string is the all-ones string $(1,\dots,1)$. That is, to simplify our notation, we drop the subscript z and assume that for every $\ell < n/2$ we consider the function that assigns to every $x \in \{0,1\}^n$ the function value

$$\textsc{Jump}_\ell(x) := \begin{cases} n & \text{if } |x|_1 = n, \\ |x|_1 & \text{if } \ell < |x|_1 < n-\ell, \\ 0 & \text{otherwise.} \end{cases}$$

The results in [28] cover a broad range of different combinations of jump sizes ℓ and arities k.

Arity	Short jump $\ell = O(n^{1/2-\varepsilon})$	Long jump $\ell = (1/2-\varepsilon)n$	Extreme jump $\ell = n/2-1$
$k=1$	$\Theta(n \log n)$	$O(n^2)$	$O(n^{9/2})$
$k=2$	$O(n)$	$O(n \log n)$	$O(n \log n)$
$3 \le k \le \log n$	$O(n/k)$	$O(n/k)$	$\Theta(n)$

Table 3.1 Known bounds for the unbiased black-box complexity of $\textsc{Jump}_\ell$

Theorem 3.6.13 ([28]). *Table 3.1 summarizes the known bounds for the unbiased black-box complexity of* $\textsc{Jump}_\ell$ *in the different models.*

To discuss the bounds in Theorem 3.6.13, we proceed by problem type. Almost all proofs are rather involved, and so we sketch here only the main ideas.

Short Jumps, i.e., $\ell = O(n^{1/2-\varepsilon})$

A comparison with the bounds discussed in Section 3.6.3.2 shows that the bounds for the k-ary unbiased black-box complexities of short jump functions stated above are of the same order as those for OneMax (which can be seen as a jump function with parameter $\ell = 0$). In fact, it was shown in [28, Lemma 3] that a black-box algorithm having access to a jump function with $\ell = O(n^{1/2-\varepsilon})$ can retrieve (with high probability) the true OneMax value of a search point using only a constant number of queries. The other direction is of course also true, since from the OneMax value we can compute the Jump_ℓ value without further queries. This implies that the black-box complexities of short jump functions are of the same asymptotic order as those of OneMax. Any improved bound for the k-ary unbiased black-box complexity OneMax therefore immediately carries over to short jump functions.

The fact that the $\Theta(n \log n)$ bound for the unary unbiased black-box complexity carries over to so-called Plateau functions, which assign to all sub-optimal solutions of Hamming distance at most ℓ the same function value $n - \ell - 1$ and are identical to Jump otherwise, has been discussed in [3].

Long Jumps, i.e., $\ell = (1/2 - \varepsilon)n$

Despite the fact that the above-mentioned Lemma 3 in [28] can probably not be directly extended to long jump functions, the bounds for arities $k \geq 3$ nevertheless coincide with those for OneMax. In fact, it was shown in [28, Theorem 6] that for all $\ell < (1/2 - \varepsilon)n$ and for all $k \geq 3$ the k-ary unbiased black-box complexity of Jump_ℓ is at most of the same asymptotic order as the $(k-2)$-ary unbiased black-box complexity of OneMax. For $k > 3$, this proves the bounds stated in Theorem 3.6.13. The linear bound for $k = 3$ follows from the case of extreme jumps.

A key ingredient for the bound on the *unary* unbiased black-box complexity of long jump functions is a procedure that samples a number of neighbors at some fixed distance d and studies the empirical expected function values of these neighbors to decide upon the direction in which the search for the global optimum is continued. More precisely, it uses the samples to estimate the OneMax value of the currently investigated search point. Strong concentration bounds are used to bound the probability that this approach gives an accurate estimation of the correct OneMax values.

The $O(n \log n)$ bound for the *binary* unbiased black-box complexity of long jump functions follows from its extreme analogue.

Extreme Jump, i.e., $\ell = n/2 - 1$

The work [28] first considered the *ternary* unbiased black-box complexity of the extreme jump function. A strategy that allowed individual bits to be tested was derived. Testing each bit individually in an efficient way (using the encoding strategies originally developed in [35] and described in Section 3.6.3.2 above) gives the linear bound.

In the *binary* case, the bits cannot be tested as efficiently anymore. The main idea is nevertheless to flip individual bits and to test whether the flip was in a "good" or a "bad" direction. This test is done by estimating the distance to a reference string with $n/2$ ones. Implementing this strategy in $O(n \log n)$ queries requires one to overcome a few technical difficulties imposed by the restriction of sampling only from binary unbiased distributions, resulting in a rather complex bookkeeping procedure, and a rather technical proof 4.5 pages long.

Finally, the polynomial unary unbiased black-box complexity of the extreme jump function can be proven as follows. Similarly to the cases discussed above, individual bits are flipped in a current "best" solution candidate x. A sampling routine is used to estimate whether the bit flip was in a "good" or a "bad" direction, i.e., whether it created a string that was closer to the global optimum or to its bitwise complement than the previous string. The sampling strategy works as follows. Depending on the estimated parity of $|x|_1$, exactly $n/2$ or $n/2-1$ bits are flipped in x. The fraction of offspring so created with function value $n/2$ (the only value that is "visible" apart from that of the global optimum) is recorded. This fraction depends on the distance from x to the global optimum $(1,\dots,1)$ or its complement $(0,\dots,0)$ and is slightly different for different distances. A key step in the analysis of the unary unbiased black-box complexity of the extreme jump function is therefore a proof that shows that a polynomial number of such samples are sufficient to determine the OneMax value of x with sufficiently large probability.

Comments on the Upper Bounds in Theorem 3.6.13

Note that even for long jump functions, the search points having a function value of 0 form plateaus around the optimum $(1,\dots,1)$ and its complement $(0,\dots,0)$ of exponential size. For the extreme jump function, all but a $\Theta(n^{-1/2})$ fraction of the search points form one single fitness plateau. Problem-unspecific black-box optimization techniques will therefore typically not find the optimum of long and extreme jump functions in subexponential time.

Lower Bound

The $\Omega(n \log n)$ lower bound in Theorem 3.6.13 follows from the more general result discussed in Theorem 3.2.5 and the $\Omega(n \log n)$ bound for OneMax in the unary unbiased black-box model, which we discussed in Section 3.6.3.2. Note also that Theorem 3.2.5, together with the $\Omega(n/\log n)$ unrestricted black-box complexity of OneMax, implies a lower bound for Jump_ℓ of the same asymptotic order (for all values of ℓ). The linear lower bound for the extreme jump function can be easily proven by the information-theoretic arguments presented in Theorem 3.2.4. Intuitively, the algorithm needs to learn a linear number of bits, while it receives only a constant number per function evaluation.

Insights from These Bounds and Open Questions

The proof sketches provided above highlight the fact that one of the key novelties presented in [28] are the sampling strategies that are used to estimate the OneMax values of a current string of interest. The idea of accumulating some statistical information about the fitness landscape could be an interesting concept for the design of novel heuristics, in particular for optimization in the presence of noisy function evaluations or for dynamic problems, which change over time.

3.6.3.5 Number Partition

Number partition is one of the best-known NP-hard problems. Given a set $S \subset \mathbb{N}^n$ of n positive integers, this partition problem asks us to decide whether or not it is possible to split S into two disjoint subsets such that the sums of the integers in these two subsets are identical, i.e., whether or not two disjoint subsets S_1 and S_2 of S with $S_1 \cup S_2 = S$ and $\sum_{s \in S_1} s = \sum_{s \in S_2} s$ exist. The optimization version of partition asks us to split S into two disjoint subsets such that the absolute discrepancy $\left|\sum_{s \in S_1} s - \sum_{s \in S_2} s\right|$ is as small as possible.

In [26], a subclass of partition was studied in which the integers in S are pairwise different. The problem remains NP-hard under this assumption. It is thus unlikely that it can be solved efficiently. For two different formulations of this problem (using a signed and an unsigned function assigning to each partition S_1, S_2 of S the discrepancy $\sum_{s \in S_1} s - \sum_{s \in S_2} s$ or the absolute value of this expression, respectively), it was shown that the unary unbiased black-box complexity of this subclass is nevertheless of small polynomial order. More precisely, it was shown that there are unary unbiased black-box algorithms that need only $O(n \log n)$ function evaluations to optimize any $\text{Partition}_{\neq}$ instance. The proof techniques are very similar to the ones presented in Sec-

tion 3.2.4: the algorithm achieving the $O(n \log n)$ expected optimization time first uses $O(n \log n)$ steps to learn the problem instance at hand. After some (possibly – and probably – nonpolynomial-time) offline computation of an optimal solution for this instance, this optimum is then created via an additional $O(n \log n)$ function evaluations, needed to move the integers of the partition instance to the right subset. Learning and moving the bits can be done in linear time in the unrestricted model. The factor $\log n$ stems from the fact that here, in this unary unbiased model, in every iteration a random bit is moved, so that a coupon collector process results in a logarithmic overhead.

This result and those for the different versions of jump functions described in Section 3.6.3.4 show that the unary unbiased black-box complexity can be much smaller than the typical performance of mutation-only black-box heuristics. This indicates that the unary unbiased black-box model does not always give a good estimation of the difficulty of a problem when optimized by mutation-based algorithms. As we shall discuss in Section 3.7, a possible direction to obtain more meaningful results could be to restrict the class of algorithms even further, for example through bounds on the memory size or the selection operators.

3.6.3.6 Minimum Spanning Trees

Having a formulation over the search space $\{0,1\}^m$, the minimum spanning tree problem considered in Section 3.3.8.1 can be directly studied in the unbiased black-box model proposed by Lehre and Witt. The following theorem summarizes the bounds proven in [37] for this problem. We see here that [37] also studied the black-box complexity of a model that combines the restrictions imposed by the ranking-based and the unbiased black-box models. We will discuss this model in Section 3.7 but, for the sake of brevity, will state the bounds now for this combined model.

Theorem 3.6.14 (Theorem 10 in [37]). *The* unary unbiased *black-box complexity of the MST problem 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 of the MST problem is* $O(mn \log n)$. *Its* ranking-based binary unbiased *black box-complexity is* $O(m \log n)$ *and its* ranking-based 3-ary unbiased *black-box complexity is* $O(m)$.

For every k, 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 $\textsc{OneMax}_m$.

As in the unrestricted case of Theorem 3.3.14, the upper bounds in Theorem 3.6.14 are obtained by modifying Kruskal's algorithm to fit the black-box setting at hand. For the lower bound, the path P on $m+1$ vertices with unit edge weights shows that $\textsc{OneMax}_m$ is a subproblem of the MST problem. More precisely, for all bit strings $x \in \{0,1\}^m$, the function value

$f(x) = (\textsc{OneMax}_m(x), m+1-\textsc{OneMax}_m(x))$ of the associated MST fitness function reveals the OneMax value of x.

3.6.3.7 Other Results

Motivated to introduce a class of functions for which the unary unbiased black-box complexity is $\Theta(2^m)$ for some parameter m that can be scaled between 1 and n, Lehre and Witt introduced in [72] the following function:

$$\text{OM-Needle} : \{0,1\}^n \to [0..n], x \mapsto \sum_{i=1}^{n-m} x_i + \prod_{i=1}^{n} x_i.$$

It is easily seen that this function has its unique global optimum in the all-ones string $(1,\dots,1)$. All other search points whose first $n-m$ entries are equal to one are located on a plateau of function value $n-m$. In the unbiased model, this part is thus similar to the Needle functions discussed in Section 3.3.1. Lehre and Witt showed that for $0 \le m \le n$ the unary unbiased black-box complexity of this function is at least 2^{m-2} [72, Theorem 3]. Note that this function is similar in flavor to the version of the jump function proposed in [62] (see Section 3.3.7.3).

3.6.4 Beyond Pseudo-Boolean Optimization: Unbiased Black-Box Models for Other Search Spaces

In this section we discuss an extension of the pseudo-Boolean unbiased black-box model of Lehre and Witt [72] to more general search spaces. To this end, we first recall from Definition 3.6.1 that the unbiased model is defined through a set of invariances that must be satisfied by the probability distributions from which unbiased algorithms sample their solution candidates. It is therefore quite natural to first generalize the notion of an unbiased operator in the following way.

Definition 3.6.15 (Definition 1 in [37]). Let $k \in \mathbb{N}$, let S be some arbitrary set, and let $\mathcal{G}$ be a set of bijections on S that forms a group, i.e., a set of one-to-one maps $g : S \to S$ that is closed under composition $(\cdot \circ \cdot)$ and under inversion $(\cdot)^{-1}$. We call $\mathcal{G}$ the *set of invariances*.

A *k-ary $\mathcal{G}$-unbiased distribution* is a family of probability distributions $\big(D(\cdot \mid y^1,\dots,y^k)\big)_{y^1,\dots,y^k \in S}$ over S such that for all *inputs* $y^1,\dots,y^k \in S$ the condition

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

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

For $S := \{0,1\}^n$ and when $\mathcal{G}$ is the set of Hamming automorphisms, it is not difficult to verify that Definition 3.6.15 extends Definition 3.6.1. A k-ary $\mathcal{G}$-unbiased black-box algorithm is one that samples all search points from k-ary $\mathcal{G}$-unbiased variation operators.

In [81], Rowe and Vose gave the following very general, but rather indirect, definition of unbiased distributions.

Definition 3.6.16 (Definition 2 in [81]). Let $\mathcal{F}$ be a class of functions from a search space S to some set Y. We say that a one-to-one map $\alpha : S \to S$ *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.

It was argued in [81] that $\Pi(\mathcal{F})$ in fact forms a group, so that Definition 3.6.16 satisfies the requirements of Definition 3.6.15.

To apply the framework of Definition 3.6.16, one has to make precise the set of invariances covered by the class $\Pi(\mathcal{F})$. This can be quite straightforward in some cases [81] but may require some more effort in others [37]. In particular, it is often inconvenient to define the whole family of unbiased distributions from which a given variation operator originates. Luckily, in many cases this effort can be considerably reduced, to proving only the unbiasedness of the variation operator itself. The following theorem demonstrates this for the case $S = [n]^{n-1}$, which is used, for example, in the single-source shortest-path problem considered in the next subsection. In this case, condition (ii) in the theorem states that it suffices to show the k-ary $\mathcal{G}$-unbiasedness of the distribution $D_{\mathbf{z}}$, without making precise the whole family of distributions associated to it.

Theorem 3.6.17. *Let $\mathcal{G}$ be a set of invariances, i.e., a set of permutations of the search space $S = [n]^{n-1}$ that form a group. Let $k \in \mathbb{N}$, and let $\mathbf{z} = (z^1, \dots, z^k) \in S^k$ 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 $D_{\mathbf{z}}$ on $[n]^{n-1}$, the following two statements are equivalent.

(i) There exists a k-ary $\mathcal{G}$-unbiased distribution $(D(\cdot \mid \mathbf{y}))_{\mathbf{y} \in S^k}$ on S such that $D_{\mathbf{z}} = D(\cdot \mid \mathbf{z})$.
(ii) For every $g \in \mathcal{G}_0$ and for all $x \in S$, it holds that $D_{\mathbf{z}}(x) = D_{\mathbf{z}}(g(x))$.

3.6.4.1 Alternative Extensions of the Unbiased Black-Box Model for the SSSP problem

As discussed in Section 3.3.8.2, several formulations of the single-source shortest-path problem (SSSP) coexist. In the unbiased black-box setting, the multicriteria formulation is not very meaningful, as the function values explicitly distinguish between the vertices, so that treating them in an unbiased fashion seems unreasonable. For this reason, in [37] only the single-objective formulation was investigated in the unbiased black-box model. Note that for this formulation, the unbiased black-box model for pseudo-Boolean functions needs to be extended to the search space $S_{[2..n]}$. The work [37] discussed three different extensions:

(a) a *structure-preserving* unbiased model in which, intuitively speaking, the operators do not consider the *labels* of different nodes, but only their local structure (e.g., the size of their neighborhoods);
(b) the *generalized* unbiased model proposed in [81] (this model follows the approach presented in Section 3.6.4 above); and
(c) a *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.

Whereas all three notions a priori seem to capture different aspects of what unbiasedness in the SSSP problem could mean, two of them were shown to be too powerful. More precisely, it was shown that even the *unary* structure-preserving unbiased black-box complexity of SSSP and the *unary* generalized unbiased black-box complexity are almost identical to the unrestricted black-box complexity. The three models were proven to differ by at most one query in [37, Theorem 25 and Corollary 32].

It was then shown that the redirecting unbiased black-box model yields more meaningful black-box complexities.

Theorem 3.6.18 (Corollary 28, Theorem 29, and Theorem 30 in [37]). *The unary ranking-based redirecting unbiased black-box complexity of SSSP is $O(n^3)$. Its binary ranking-based redirecting unbiased black-box complexity is $O(n^2 \log n)$. For all $k \in \mathbb{N}$, the k-ary redirecting unbiased black-box complexity of SSSP is $\Omega(n^2)$.*

The unary bound is obtained by a variant of RLS which, in every step, redirects one randomly chosen node to a random predecessor. For the binary bound, the problem instance is learned in a two-phase step. An optimal solution is then created by an imitation of Dijkstra's algorithm. For the lower bound, drift analysis can be used to prove that every redirecting unbiased algorithm needs $\Omega(n^2)$ function evaluations to reconstruct a given path on n vertices.

3.7 Combined Black-Box Complexity Models

The black-box models discussed in the previous sections either study the complexity of a problem with respect to *all* black-box algorithms (in the unrestricted model) or restrict the class of algorithms with respect to *one* particular feature of common optimization heuristics, such as the size of their memory, their selection behavior, or their sampling strategies. As we have seen, many classical black-box optimization algorithms are members of several of these classes. At the same time, a nonnegligible number of the upper bounds stated in the previous sections can, to date, be certified only by algorithms that satisfy an individual restriction, but clearly violate other requirements that are not controlled by the respective model. In the unbiased black-box model, for example, several of the upper bounds are obtained by algorithms that make use of a rather large memory size. It is therefore natural to ask if and how the black-box complexity of a problem increases if two or more of the different restrictions proposed in the previous sections are combined in a new black-box model. This is the focus of this section, which surveys results obtained in such combined black-box complexity models.

3.7.1 Unbiased Ranking-Based Black-Box Complexity

Even some of the very early publications on the unbiased black-box model considered a combination with the ranking-based model. In fact, the binary unbiased algorithm in [35], which solves OneMax with $\Theta(n)$ queries on average, uses only comparisons, and does not make use of knowledge of absolute fitness values. It was shown in [44] that, also, the other upper bounds for the k-ary black-box complexity of OneMax proven in [35] hold also in the ranking-based version of the k-ary unbiased black-box model.

Theorem 3.7.1 (Theorem 6 and Lemma 7 in [44]). *The unary unbiased ranking-based black-box complexity of* $\textsc{OneMax}_n$ *is* $\Theta(n \log n)$. *For constant* k, *the* k*-ary unbiased ranking-based black-box complexity of* $\textsc{OneMax}_n$ *and that of every strictly monotone function is at most* $4n-5$. *For* $2 \leq k \leq n$, *the* k*-ary unbiased ranking-based black-box complexity of* $\textsc{OneMax}_n$ *is* $O(n/\log k)$.

In light of Theorem 3.6.9, it seems plausible that the upper bounds for the case $2 \leq k \leq n$ can be reduced to $O(n/k)$, but we are not aware of any result proving such a claim.

Also, the binary unbiased algorithm achieving an expected $O(n \log n)$ optimization time on LeadingOnes uses only comparisons.

Theorem 3.7.2 (follows from the proof of Theorem 14 in [35]; see Theorem 3.6.11). *The binary unbiased ranking-based black-box complexity of* LeadingOnes *is* $O(n \log n)$.

For the ternary black-box complexity we have mentioned already in Theorem 3.6.12 that the $O(n\log(n)/\log\log n)$ bound also holds in the ranking-based version of the ternary unbiased black-box model.

For the two combinatorial problems MST and SSSP, it has already been mentioned in Theorems 3.6.14 and 3.6.18 that the bounds hold also in models in which we require the algorithms to base all decisions on only the ranking of previously evaluated search points, and not on absolute function values.

3.7.2 Parallel Black-Box Complexity

The (unary) unbiased black-box model was also the starting point for the authors of [4], who introduced a black-box model to investigate the effects of a parallel optimization. Their model can be seen as a unary unbiased $(\infty+\lambda)$ memory-restricted black-box model. More precisely, their model covers all algorithms following the scheme of Algorithm 3.6.

The model covers the $(\mu+\lambda)$ EA and the (μ,λ) EA, cellular evolutionary algorithms, and unary unbiased evolutionary algorithms working in the island model. The restriction to unary unbiased variation operators can of course be relaxed to obtain general models for λ-parallel k-ary unbiased black-box algorithms.

We see that Algorithm 3.6 forces the algorithm to query λ new solution candidates in every iteration. Thus, intuitively, for every two positive integers k and ℓ with $k/\ell \in \mathbb{N}$ and for all problem classes $\mathcal{F}$, the ℓ-parallel unary unbiased black-box complexity of $\mathcal{F}$ is at most as large as its k-parallel unary unbiased black-box complexity.

Algorithm 3.6: A blueprint for λ-parallel unary unbiased black-box algorithms for the optimization of an unknown function $f : S \to \mathbb{R}$

Initialization:
  for $i = 1,\dots,\mu$ **do** Sample $x^{(i,0)}$ uniformly at random from S and query $f\big(x^{(i,0)}\big)$;
  $\mathcal{I} \leftarrow \{f\big(x^{(1,0)}\big),\dots,f\big(x^{(\lambda,0)}\big)\}$;
Optimization: for $t = 1,2,3,\dots$ **do**
  for $i = 1,\dots,\lambda$ **do**
    Depending only on the multiset $\mathcal{I}$ choose a pair of indices $(j,\ell) \in [\lambda] \times [0..t-1]$;
    Depending only on the multiset $\mathcal{I}$ choose a unary unbiased probability distribution $D^{(i,t)}(\cdot)$ on S, sample $x^{(i,t)} \leftarrow D^{(i,t)}(x^{(j,\ell)})$ and query $f\big(x^{(i,t)}\big)$;
  $\mathcal{I} \leftarrow \mathcal{I} \cup \{f\big(x^{(1,t)}\big),\dots,f\big(x^{(\lambda,t)}\big)\}$;

The following bounds for the λ-parallel unary unbiased black-box complexity are known.

Theorem 3.7.3 (Theorems 1, 3 and 4 in [4]). *The λ-parallel unary unbiased black-box complexity of* LeadingOnes *is $\Omega\big(\frac{\lambda n}{\max\{1,\log(\lambda/n)\}} + n^2\big)$. It is of order at most $\lambda n + n^2$.*

For any $\lambda \leq e^{\sqrt{n}}$, the λ-parallel unary unbiased black-box complexity of any function having a unique global optimum is $\Omega\big(\frac{\lambda n}{\log(\lambda)} + n\log n\big)$. This bound is asymptotically tight for OneMax.

For LeadingOnes, the upper bound is attained by a $(1+\lambda)$ EA investigated in [70]. The lower bound was shown by means of drift analysis, building upon the arguments used in [72] to prove Theorem 3.6.10.

For OneMax, a $(1+\lambda)$ EA with fitness-dependent mutation rates was shown to achieve an $O\big(\frac{\lambda n}{\log(\lambda)} + n\log n\big)$ expected optimization time in [4, Theorem 4]; see Chapter 5.6 for details.

The lower bound for the λ-parallel unary unbiased black-box complexity of functions having a unique global optimum uses additive drift analysis. The proof is similar to the proof of Theorem 3.6.6 in [72], but requires a very precise tail bound for hypergeometric variables (Lemma 2 in [4]).

3.7.3 Distributed Black-Box Complexity

To study the effects of the migration topology on the efficiency of distributed evolutionary algorithms, the λ-parallel unary unbiased black-box model was extended in [5] to a distributed version, in which islands exchange their accumulated information along a given graph topology. Commonly employed topologies are the complete graph (in which all nodes exchange information with each other), the ring topology, the grid of equal side lengths, and the torus. [5] presents an unrestricted and a unary unbiased version of the distributed black-box model. In this context, it is interesting to study how the black-box complexity of a problem increases with sparser migration topologies or with the infrequency of migration. The model of [5] allows all nodes to share all the information that they have accumulated so far. Another interesting extension of the distributed model would be to study the effects of bounding the amount of information that can be shared in any migration phase. We will not present the model, nor all results obtained in [5], in detail. The main result which is interpretable and comparable to the others presented in this chapter is summarized by the following theorem.

Theorem 3.7.4 (Table 1 in [5]). *The λ-distributed unary unbiased black-box complexity of the class of all unimodal functions with $\Theta(n)$ different function values satisfies the bounds stated in Table 3.2. The lower bound for*

	Ring Topology	Grid/Torus	Complete Topology
Upper Bound	$O(\lambda n^{3/2} + n^2)$	$O(\lambda n^{4/3} + n^2)$	$\Theta(\lambda n + n^2)$
Lower Bound	$\Omega(\lambda n + \lambda^{2/3} n^{5/3} + n^2)$	$\Omega(\lambda n + \lambda^{3/4} n^{3/2} + n^2)$	

Table 3.2 The λ-distributed unary unbiased black-box complexity of the class of all unimodal functions with $\Theta(n)$ different function values

the grid applies to arbitrary side lengths, while the upper bound holds for the grid with $\sqrt{\lambda}$ islands in each of the two dimensions.

The upper bounds in Theorem 3.7.4 are achieved by a parallel (1+1) EA, in which every node migrates its complete information after every round. The lower bounds were shown to hold even for a subproblem called the "random short path," which is a collection of problems which all have a global optimum in some point with exactly $n/2$ ones. A short path of Hamming-1 neighbors leads to this optimum. The paths start at the all-ones string. Search points that do not lie on the path lead the optimization process towards the all-ones string; their objective values equal the number of ones in the string.

3.7.4 Elitist Black-Box Complexity

One of the most relevant questions in black-box optimization is how to avoid getting stuck in local optima. Essentially, two strategies have been developed.

- **Nonelitist selection.** The first idea is to allow the heuristics to direct their search towards search points that are, a priori, less favorable than the current best solutions in the memory. This can be achieved, for example, by accepting into the memory ("population") search points with function values that are smaller than the current best solutions. We refer to such selection procedures as *nonelitist selection*. Nonelitist selection is used, for example, in the Metropolis algorithm [75], Simulated Annealing [66], and, more recently, the biology-inspired "Strong Selection, Weak Mutation" framework [80].
- **Global sampling.** A different strategy to overcome local optima is *global sampling*. This approach is used, most notably, by evolutionary and genetic algorithms, but also by swarm optimizers such as ant colony optimization techniques [52] and estimation-of-distribution algorithms (EDAs, see Chapter 7 in this book). The underlying idea of global sampling is to select new solution candidates not only locally in some predefined neighborhood of the current population, but also to reserve some positive probability to sample far away from these solutions. Very often, a truly global sampling operation is used, in which *every* point $x \in S$ has a positive probability of being sampled. This probability typically decreases with increasing dis-

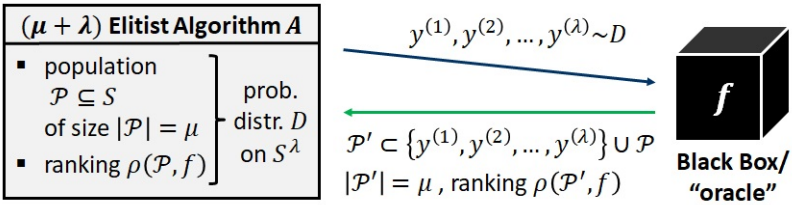

Fig. 3.7 A $(\mu+\lambda)$ elitist black-box algorithm stores the μ previously evaluated search points of largest function value (ties broken arbitrarily or according to some specified rule) and the ranking of these points induced by f. Based on this information, it decides upon a strategy according to which the next λ search points are sampled. From the $\mu+\lambda$ parent and offspring solutions, those μ search points that have the largest function values form the population for the next iteration.

tance to the current best solutions. Standard bit mutation with bit flip probabilities $p < 1/2$ is such a global sampling strategy.

Global sampling and nonelitist selection can certainly be combined, and several attempts in this direction have been made. The predominant selection strategy used in combination with global sampling, however, is *truncation selection*. Truncation selection is a natural implementation of Darwin's "survival of the fittest" paradigm in an optimization context: given a collection $\mathcal{P}$ of search points and a population size μ, truncation selection chooses from $\mathcal{P}$ the μ search points of largest function values and discards the others, breaking ties arbitrarily or according to some rule such as favoring offspring over parents or favoring genotypic or phenotypic diversity.

To understand the influence that this *elitist selection* behavior has on the performance of black-box heuristics, the *elitist black-box model* was introduced in [46] (a journal version has appeared as [49]). The elitist black-box model combines features of the memory-restricted and the ranking-based black-box models with an enforced truncation selection. More precisely, the $(\mu+\lambda)$ *elitist black-box model* covers all algorithms that follow the pseudocode in Algorithm 3.7. We use here an adaptive initialization phase. A nonadaptive version, as in the $(\mu+\lambda)$ memory-restricted black-box model, can also be considered. This and other subtleties such as the tie-breaking rules for search points of equal function values can result in different black-box complexities. It is therefore important to make very precise the model with respect to which a bound is claimed or shown to hold.

The elitist black-box model covers, in particular, all $(\mu+\lambda)$ EAs, RLS, and other hill climbers. It does not cover algorithms using nonelitist selection rules such as Boltzmann selection, tournament selection, or fitness-proportional selection. Figure 3.7 illustrates the $(\mu+\lambda)$ elitist black-box model. As a seemingly subtle but possibly influential difference from the parallel black-box complexities introduced in Section 3.7.2, note that in the elitist black-box

Algorithm 3.7: The $(\mu+\lambda)$ elitist black-box algorithm for maximizing an unknown function $f : S \to \mathbb{R}$

1 **Initialization:**
2 $X \leftarrow \emptyset$;
3 **for** $i = 1, \dots, \mu$ **do**
4 Depending only on the multiset X and the ranking $\rho(X, f)$ of X induced by f, choose a probability distribution $D^{(i)}$ over S and sample from it $x^{(i)}$;
5 Set $X \leftarrow X \cup \{x^{(i)}\}$ and query the ranking $\rho(X, f)$ of X induced by f;
6 **Optimization: for** $t = 1, 2, 3, \dots$ **do**
7 Depending only on the multiset X and the ranking $\rho(X, f)$ of X induced by f choose a probability distribution $D^{(t)}$ on S^λ and sample from it $y^{(1)}, \dots, y^{(\lambda)} \in S$;
8 Set $X \leftarrow X \cup \{y^{(1)}, \dots, y^{(\lambda)}\}$ and query the ranking $\rho(X, f)$ of X induced by f;
9 **for** $i = 1, \dots, \lambda$ **do** Select $x \in \arg\min X$ and update $X \leftarrow X \setminus \{x\}$;

model the offspring sampled in the optimization phase do not need to be independent of each other. If, for example, an offspring x is created by crossover, in the $(\mu+\lambda)$ elitist black-box model with $\lambda \geq 2$ we allow another offspring y to be created from the same parents, whose entries y_i in those positions i in which the parents do not agree equal $1-x_i$. These two offspring are obviously not independent of each other. It is nevertheless required in the $(\mu+\lambda)$ elitist black-box model that the λ offspring are created *before* any evaluation of the offspring happens. That is, the k-th offspring may *not* depend on the ranking or fitness of the first $k-1$ offspring.

In addition to combining several features of previous black-box models, the elitist black-box model can be further restricted to cover only those elitist black-box algorithms that sample from unbiased distributions. For this *unbiased elitist black-box model*, we require that the distribution $p^{(t)}$ in line 7 of Algorithm 3.7 is unbiased (in the sense of Section 3.6). Some of the results mentioned below also hold for this more restrictive class.

3.7.4.1 Nonapplicability of Yao's Minimax Principle

An important difficulty in the analysis of elitist black-box complexities is the fact that Yao's minimax principle (Theorem 3.2.3) cannot be directly applied to the elitist black-box model, since in this model the previously exploited fact that randomized algorithms are convex combinations of deterministic ones does not apply; see [49, Section 2.2] for an illustrated discussion. As discussed in the previous sections, Yao's minimax principle is *the* most important tool for proving lower bounds in the black-box complexity context, and we can hardly do without it. A natural workaround that allows us to

nevertheless employ this technique is to extend the collection $\mathcal{A}$ of elitist black-box algorithms to some superset $\mathcal{A}'$ in which every randomized algorithm *can* be expressed as a probability distribution over deterministic ones. A lower bound shown for this broader class $\mathcal{A}'$ applies trivially to all elitist black-box algorithms. Finding extensions $\mathcal{A}'$ that do not decrease the lower bounds by too much is the main difficulty to be overcome in this strategy.

3.7.4.2 Exponential Gaps to Previous Models

In [49, Section 3], it was shown that even for quite simple function classes there can be an exponential gap between the efficiency of elitist and nonelitist black-box algorithms; and this applies even in the very restrictive (1+1) unary unbiased elitist black-box complexity model. This shows that heuristics can sometimes benefit quite crucially from eventually giving preference to search points of fitness inferior to that of the current best search points. The underlying intuition for these results is that elitist algorithms do not work very well if there are several local optima that the algorithm needs to explore in order to determine the best one of them.

3.7.4.3 The Elitist Black-Box Complexity of OneMax

As we have discussed in Sections 3.4 and 3.5, respectively, the (1+1) memory-restricted and the ranking-based black-box complexity of OneMax are only of order $n/\log n$. In contrast, it is easy to see that the combined (1+1) memory-restricted ranking-based black-box model does not allow algorithms that are faster than linear in n, as can easily be seen by standard information-theoretic considerations. In [47] (a journal version has appeared as [50]) it was shown that this linear bound is tight. Whether or not it applies to the (1+1) elitist model remains unsolved, but it was shown in [50] that the expected time needed to optimize OneMax with probability at least $1-\varepsilon$ is linear for every constant $\varepsilon > 0$. This is the so-called Monte Carlo black-box complexity, which we shall briefly discuss in Section 3.11. The following theorem summarizes the bounds presented in [50]. Without detailing this further, we note that [50, Section 9] also introduced and studied a comma-variant of the elitist black-box model.

Theorem 3.7.5 ([50]). *The (1+1) memory-restricted ranking-based black-box complexity of* OneMax *is* $\Theta(n)$.

For $1 < \lambda < 2^{n^{1-\varepsilon}}$, $\varepsilon > 0$ *being an arbitrary constant, the* $(1+\lambda)$ *memory-restricted ranking-based black-box complexity of* OneMax *is* $\Theta(n/\log\lambda)$ *(in terms of generations), while for* $\mu = \omega(\log^2(n)/\log\log n)$ *its* $(\mu+1)$ *memory-restricted ranking-based black-box complexity is* $\Theta(n/\log\mu)$.

For every constant $0 < \varepsilon < 1$, there exists a (1+1) elitist black-box algorithm that finds the optimum of any OneMax *instance in time $O(n)$ with probability at least $1-\varepsilon$, and this running time is asymptotically optimal.*

For constant μ, the $(\mu+1)$ elitist black-box complexity of OneMax *is at most $n+1$.*

For $\delta > 0$, $C > 0$, $2 \leq \lambda < 2^{n^{1-\delta}}$, and a suitably chosen $\varepsilon = O(\log^2(n)\log\log(n)\log(\lambda)/n)$, there exists a $(1+\lambda)$ elitist black-box algorithm that needs at most $O(n/\log\lambda)$ generations to optimize OneMax *with probability at least $1-\varepsilon$.*

For $\mu = \omega(\log^2(n)/\log\log n) \cap O(n/\log n)$ and every constant $\varepsilon > 0$, there is a $(\mu+1)$ elitist black-box algorithm optimizing OneMax *in time $\Theta(n/\log\mu)$ with probability at least $1-\varepsilon$.*

There exists a constant $C > 1$ such that for $\mu \geq Cn/\log n$, the $(\mu+1)$ elitist black-box complexity is $\Theta(n/\log n)$.

3.7.4.4 The Elitist Black-Box Complexity of LeadingOnes

The (1+1) elitist black-box complexity of LeadingOnes was studied in [48] (a journal version has appeared as [51]). Using the approach sketched in Section 3.7.4.1, the following result was derived.

Theorem 3.7.6 (Theorem 1 in [51]). *The (1+1) elitist black-box complexity of* LeadingOnes *is $\Theta(n^2)$. This bound holds also in the case where the algorithms have access to (and can make use of) the absolute fitness values of the search points in the population, and not only their rankings, i.e., in the (1+1) memory-restricted black-box model with enforced truncation selection.*

The $(1+1)$ elitist black-box complexity of LeadingOnes is thus considerably larger than its unrestricted black-box complexity, which is known to be of order $n\log\log n$, as discussed in Theorem 3.3.12.

It is well known that the quadratic bound in Theorem 3.7.6 is matched by classical $(1+1)$-type algorithms such as the (1+1) EA, RLS, and others.

3.7.4.5 The Unbiased Elitist Black-Box Complexity of Jump Functions

Some shortcomings of previous models can be eliminated when they are combined with an elitist selection requirement. This was shown in [49] for the Jump_k function already discussed.

Theorem 3.7.7 (Theorem 9 in [49]). *For $k = 0$, the unary unbiased (1+1) elitist black-box complexity of Jump_k is $\Theta(n\log n)$. For $1 \leq k \leq \frac{n}{2} - 1$, it is of order $\binom{n}{k+1}$.*

Model	Lower bound	Upper bound
Unrestricted	$\Theta(n/\log n)$	
Unbiased, arity 1	$\Theta(n \log n)$	
Unbiased, arity $2 \leq k \leq \log n$	$\Omega(n/\log n)$	$O(n/k)$
Ranking-based (unrestricted)	$\Theta(n/\log n)$	
Ranking-based unbiased, arity 1	$\Theta(n \log n)$	
Ranking-based unbiased, arity $2 \leq k \leq n$	$\Omega(n/\log n)$	$O(n/\log k)$
(1+1) comparison-based	$\Theta(n)$	
(1+1) memory-restricted	$\Theta(n/\log n)$	
λ-parallel unbiased, arity 1	$\Theta\left(\frac{\lambda n}{\log(\lambda)} + n \log n\right)$	
(1+1) elitist Las Vegas	$\Omega(n)$	$O(n \log n)$
(1+1) elitist $\log n/n$-Monte Carlo	$\Theta(n)$	
(2+1) elitist Monte Carlo/Las Vegas	$\Theta(n)$	
(1+λ) elitist Monte Carlo (# generations)	$\Theta(n/\log \lambda)$	
(μ+1) elitist Monte Carlo	$\Theta(n/\log \mu)$	
$(1, \lambda)$ elitist Monte Carlo/Las Vegas (# generations)	$\Theta(n/\log \lambda)$	

Table 3.3 Summary of known black-box complexities of $\textsc{OneMax}_n$ in the different black-box complexity models

The bound in Theorem 3.7.7 is nonpolynomial for $k = \omega(1)$. This is in contrast to the unary unbiased black-box complexity of $\textsc{Jump}_k$, which, according to Theorem 3.6.13, is polynomial even for extreme values of k.

3.8 Summary of Known Black-Box Complexities for OneMax and LeadingOnes

For better identification of open problems concerning the black-box complexity of the two benchmark functions OneMax and LeadingOnes, we summarize the bounds that have been presented in previous sections.

Table 3.3 summarizes the known black-box complexities of $\textsc{OneMax}_n$ in the different models. The bound for the λ-parallel black-box model assumes $\lambda \leq e^{\sqrt{n}}$. The bounds for the $(1+\lambda)$ and the $(1, \lambda)$ elitist model assume $1 < \lambda < 2^{n^{1-\varepsilon}}$ for some $\varepsilon > 0$. Finally, the bound for the $(\mu+1)$ model assumes that $\mu = \omega(\log^2 n / \log\log n)$ and $\mu \leq n$.

Table 3.4 summarizes the known black-box complexities of $\textsc{LeadingOnes}_n$. The upper bounds for the unbiased black-box models also hold in the ranking-based variants.

Model	Lower bound	Upper bound
unrestricted	$\Theta(n \log \log n)$	
unbiased, arity 1	$\Theta(n^2)$	
unbiased, arity 2	$\Omega(n \log \log n)$	$O(n \log n)$
unbiased, arity ≥ 3	$\Omega(n \log \log n)$	$O(n \log(n)/\log \log n)$
λ-parallel unbiased, arity 1	$\Omega\left(\frac{\lambda n}{\log(\lambda/n)} + n^2\right)$	$O(\lambda n + n^2)$
(1+1) elitist	$\Omega(n^2)$	$O(n^2)$

Table 3.4 Summary of known black-box complexities of LeadingOnes$_n$ in the different black-box complexity models

3.9 From Black-Box Complexity to Algorithm Design

In the previous sections, the focus of our attention has been on computing performance limits for black-box optimization heuristics. In some cases, for example for OneMax in the unary unbiased black-box model and for LeadingOnes in the (1+1) elitist black-box model, we have obtained lower bounds that are matched by the performance of well-known standard heuristics such as RLS or the (1+1) EA. For several other models and problems, however, we have obtained black-box complexities that are much smaller than the expected running times of typical black-box optimization techniques. As discussed in the introduction, two possible reasons for this discrepancy exist. Either the respective black-box models do not capture very well the complexity of the problems for heuristic approaches, or there are ways to improve classical heuristics by novel design principles.

In the case of the restrictive models discussed in Sections 3.4-3.7, we have seen that there is some truth in the first possibility. For several optimization problems, we have seen that their complexity increases if the class of black-box algorithms is restricted to subclasses of heuristics that all share some properties that are commonly found in state-of-the-art optimization heuristics. Here, in this section, we shall demonstrate that this is nevertheless not the end of the story. We discuss two examples where a discrepancy between black-box complexity and the performance of classical heuristics can be observed, and we show how the analysis of typically rather artificial problem-tailored algorithms can inspire the design of new heuristics.

3.9.1 The $(1+(\lambda,\lambda))$ Genetic Algorithm

Our first example is a binary unbiased algorithm, which optimizes OneMax more efficiently than any classical unbiased heuristic, and provably faster than any unary unbiased black-box optimizer.

We recall from Theorems 3.6.7 and 3.6.8 that the unary unbiased black-box complexity of $\textsc{OneMax}_n$ is $\Theta(n \log n)$, while its binary unbiased black-box complexity is only $O(n)$. The linear-time algorithm flips one bit at a time, and uses a simple but clever encoding to store which bits have been flipped already. In this way, it is a rather problem-specific algorithm, since it "knows" that a bit that has been tested already does not need to be tested again. The algorithm is therefore not very suitable for nonseparable problems, where the influence of an individual bit depends on the value of several or all other bits.

Until recently, all existing running-time results have indicated that general-purpose unbiased heuristics need $\Omega(n \log n)$ function evaluations to optimize OneMax, so that the question of whether the binary unbiased black-box model is too "generous" arose. In [21, 25] this question was answered negatively, through the presentation of a novel binary unbiased black-box heuristic that optimizes OneMax in expected linear time. This algorithm is the $(1+(\lambda,\lambda))$ GA. Since the algorithm itself will be discussed in more detail in Chapter 5.6, we present here only the main ideas behind it.

Disregarding some technical subtleties, one observation that we can make when considering the linear-time binary unbiased algorithm for OneMax is that when it test the value of a bit, the amount of information that it obtains is the same whether or not the offspring has a better function value. In other words, the algorithm benefits equally from offspring that are better or worse than the previously best. A similar observation applies to all of the $O(n/\log n)$ algorithms for OneMax discussed in Sections 3.3-3.7. These algorithms do not strive to sample search points of large objective value, but rather aim at maximizing the amount of information that they can learn about the problem instance at hand. This way, they benefit substantially also from those search points that are (much) worse than other ones already evaluated.

Most classical black-box heuristics are different. They store only the best solutions so far, or use inferior search points only to create diversity in the population. Thus, in general, they are not very efficient in learning from "bad" samples (where we consider a search point to be "bad" if it has a small function value). When a heuristic is close to a local or a global optimum (in the sense that it has identified search points that are not far from these optima), it samples, in expectation, a fairly large number of search points that are wore than the current best solutions. Not learning from these offspring results in a significant number of "wasted" iterations, from which the heuristic does not benefit. This observation was the starting point for the development of the $(1+(\lambda,\lambda))$ GA.

Since the unary unbiased black-box complexity of OneMax is $\Omega(n \log n)$, it was clear in the development of the $(1+(\lambda,\lambda))$ GA that an $o(n \log n)$ unbiased algorithm must be at least binary. This led to the question of how recombination can be used to learn from inferior search points. The following idea emerged. For illustration purposes, assume that we have identified a search point x of function value $\mathrm{OM}_z(x) = n-1$. From the function value, we know that there exists exactly one bit that we need to flip in order to obtain

the global optimum z. Since we want to be unbiased, the best mutation seems to be a random 1-bit flip. This has a probability $1/n$ of returning z. If we were to do this until we identified z, the expected number of samples would be n, and even if we stored which bits had been flipped already, we would need $n/2$ samples on average.

Assume now that, in the same situation, we flip $\ell > 1$ bits of x. Then, with a probability that depends on ℓ, we may have only flipped already optimized bits (i.e., bits in positions i for which $x_i = z_i$) to $1 - z_i$, thus resulting in an offspring of function value $n - 1 - \ell$. However, the probability that the position j in which x and z differ is among the ℓ positions is ℓ/n. If we repeat this experiment some λ times, independently of each other and always starting with x as the "parent," then the probability that j has been flipped in at least one of the offspring is $1 - (1 - \ell/n)^\lambda$. For moderately large ℓ and λ, this probability is sufficiently large for us to assume that among the λ offspring there is at least one in which j has been flipped. Such an offspring is distinguished from the others by a function value of $n - 1 - (\ell - 1) + 1 = n - \ell + 1$ instead of $n - \ell - 1$. Assume that there is one such offspring x' among the λ independent samples created from x. When we compare x' with x, they differ in exactly ℓ positions. In $\ell - 1$ of these, the entry of x equals that of z. Only in the j-th position is the situation reversed: $x'_j = z_j \neq x_j$. We would therefore like to identify this position j, and to incorporate the bit value x'_j into x.

So far, we have used only mutation, which is a unary unbiased operation. At this point, we want to compare and merge two search points, which is one of the driving motivations behind *crossover*. Since x clearly has more "good" bits than x', a uniform crossover, which takes for each position i its entry uniformly at random from either of its two parents, does not seem to be a good choice. We would like to add some bias to the decision-making process, in favor of choosing the entries of x. This yields a biased crossover, which takes for each position i its entry y_i from x' with some probability $p < 1/2$, and from x otherwise. The hope is to choose p in such a way that in the end only good bits are chosen. Where x and x' are identical, there is nothing to worry about, as these positions are correct already (and, in general, we have no indication to flip the entry in this position). So, we only need to look at those ℓ positions in which x and x' differ. The probability of making only good choices, i.e., of selecting $\ell - 1$ times the entry from x and, only for the j-th position, the entry from x' equals $p(1-p)^{\ell-1}$. This probability may not be very large, but when we do λ independent trials again, the probability of having created z in at least one of the trials equals $1 - (1 - p(1-p)^{\ell-1})^\lambda$. As we shall see, for suitable values of the parameters p, λ, and ℓ, this expression is sufficiently large to gain over the $O(n)$ strategies discussed above. Since we want to sample exactly one out of the ℓ bits in which x and x' differ, it seems intuitive to set $p = 1/\ell$; see the discussion in [25, Section 2.1].

Before we summarize the main findings, let us briefly reflect on the structure of the algorithm. In the *mutation step*, we have created λ offspring from x,

by a mutation operator that flips ℓ random bits in x. This is a unary unbiased operation. From these λ offspring, we have selected one offspring x' with the largest function value among all offspring (with ties broken at random). In the *crossover phase*, we have then created λ offspring again, by recombining x and x' using a biased crossover. This biased crossover is a binary unbiased variation operator. The algorithm now chooses from these λ recombined offspring one that has the largest function value (for OneMax, ties can again be broken at random, but for other problems it can be better to favor individuals that are different from x; see [25, Section 4.3]). This selected offspring y replaces x if it is at least as good as x, i.e., if $f(y) \geq f(x)$.

We see that we have employed only unbiased operations, and that the largest arity in use is two. Both of the variation operators, mutation and biased crossover, are standard operators in the evolutionary computation literature. What is novel is that crossover is used as a *repair mechanism*, and after the mutation step.

We also see that this algorithm is ranking-based, and even comparison-based in the sense that it can be implemented in a way in which, instead of querying absolute function values, only a comparison of the function values of two search points is asked for. Using information-theoretic arguments as described in Section 3.2.2, it is then not difficult to show that for any (adaptive or nonadaptive) parameter setting the best expected performance of the $(1+(\lambda,\lambda))$ GA on OneMax_n is at least linear in the problem dimension n.

The following theorem summarizes some of the results on the expected running time of the $(1+(\lambda,\lambda))$ GA on OneMax. An exhaustive discussion of these results can be found in [24]. The fitness-dependent and self-adjusting choice of the parameters will also be discussed in Section 6.5.2.1 in this book.

Theorem 3.9.1 (from [19, 21, 22, 25]). *The $(1+(\lambda,\lambda))$ GA is a binary unbiased black-box algorithm. For a mutation strength ℓ sampled from the binomial distribution $Bin(n,k/n)$ and a crossover bias $p = 1/k$, the following holds:*

- *For $k = \lambda = \Theta(\sqrt{\log(n)\log\log(n)/\log\log\log(n)})$ the expected optimization time of the $(1+(\lambda,\lambda))$ GA on OneMax_n is $O(n\sqrt{\log(n)\log\log\log(n)/\log\log(n)})$.*
- *No static parameter choice of $\lambda \in [n]$, $k \in [0..n]$, and $p \in [0,1]$ can give a better expected running time.*
- *There exists a fitness-dependent choice of λ and $k = \lambda$ such that the $(1+(\lambda,\lambda))$ GA has a linear expected running time on OneMax.*
- *A linear expected running time can also be achieved by a self-adjusting choice of $k = \lambda$.*

Note that these results answer one of the most prominent long-standing open problems in evolutionary computation: the usefulness of crossover in an optimization context. Previous and other recent examples exist where crossover has been shown to be beneficial [16, 18, 33, 36, 40, 56, 64, 83, 84],

but in all of these publications, either nonstandard problems or operators were considered or the results hold only for uncommon parameter settings, or substantial additional mechanisms such as diversity-preserving selection schemes are needed to make crossover really work. To our knowledge, Theorem 3.9.1 is thus the first example that proves advantages of crossover in a natural algorithmic setting for a simple hill-climbing problem.

Without going into detail, we mention that the $(1+(\lambda,\lambda))$ GA has also been analyzed on a number of other benchmark problems, both by theoretical [8] and by empirical [25, 58, 76] means. These results indicate that the concept of using crossover as a repair mechanism can be useful far beyond OneMax.

3.9.2 Randomized Local Search with Self-Adjusting Mutation Strengths

Another example highlighting the impact that black-box complexity studies can have on the design of heuristic optimization techniques was presented in [30]. This work built on [31], where the tight bound for the unary unbiased black-box complexity of OneMax stated in Theorem 3.6.7 was presented. This bound is attained, up to an additive difference that is sublinear in n, by a variant of RLS that in each iteration chooses a value r that depends on the function value $\mathrm{OM}_z(x)$ of the current best search point x and then uses the flip_r variation operator introduced in Definition 3.6.3 to create an offspring y. The offspring y replaces x if and only if $\mathrm{OM}_z(y) \geq \mathrm{OM}_z(x)$. The dependence of r on the function value OM_z is rather complex and difficult to compute directly; see the discussion in [31]. Quite surprisingly, a self-adjusting choice of r is capable of identifying the optimal mutation strengths r in all but a small fraction of the iterations. This way, RLS with this self-adjusting parameter choice achieves an expected running time on OneMax that is only worse by an additive $o(n)$ term than that of the theoretically optimal unary unbiased black-box algorithm.

The algorithm in [30] will be discussed in Section 5.6 of this book. In the context of black-box optimization, it is interesting to note that the idea of taking a closer look at self-adjusting parameter choices, as well as our ability to investigate the optimality of such nonstatic parameter choices, is deeply rooted in the study of black-box complexities.

3.10 From Black-Box Complexity to Mastermind

In [29], the black-box complexity studies for OneMax were extended to the following generalization of OneMax to functions over an alphabet of size k.

For a given string $z \in [0..k-1]^n$, the *Mastermind* function f_z assigns to each search point $x \in [0..k-1]^n$ the number of positions in which x and z agree. Thus, formally,

$$f_z : [0..k-1]^n \to \mathbb{R}, x \mapsto |\{i \in [n] \mid x_i = z_i\}|.$$

The collection $\{f_z \mid z \in [0..k-1]^n\}$ of all such Mastermind functions forms the Mastermind problem of n *positions* and k *colors.*

The Mastermind problem models the homonymous board game, which was very popular in North America and in the western parts of Europe in the 1970s and 1980s. More precisely, it models a variant of this game, as in the original Mastermind game information is provided also about *colors* x_i that appear in z but not in the same position i; see [29] for details and results about this Mastermind variant using *black and white pegs.*

Mastermind and similar *guessing games* were studied in the computer science literature long before the release of Mastermind as a commercial board game. As we have discussed in Section 3.3, the case of $k = 2$ colors (this is the OneMax problem) had already been considered by Erdős and Rényi and several other authors in the early 1960s. These authors were mostly interested in the complexity- and information-theoretic aspects of this problem, and/or its cryptographic nature. The playful character of the problem, in turn, was the motivation of Knuth [67], who computed an optimal strategy that solves any Mastermind instance with $n = 4$ positions and $k = 6$ colors in at most five guesses.

The first to study the general case with arbitrary values of k was Chvátal [14].

Theorem 3.10.1 (Theorem 1 in [14]). *For every $k \geq 2$ the unrestricted black-box complexity of the Mastermind game with n positions and k colors is $\Omega(n \log k / \log n)$. For $\varepsilon > 0$ and $k \leq n^{1-\varepsilon}$, it is at most $(2+\varepsilon)n(1 + 2\log k)/\log(n/k)$.*

Note that for $k \leq n^{1-\varepsilon}$, $\varepsilon > 0$ being a constant, Theorem 3.10.1 gives an asymptotically tight bound of $\Theta(n \log k / \log n)$ for the k-color, n-position Mastermind game. Similarly to the random guessing strategy of Erdős and Rényi, it is sufficient to perform this many *random* queries, chosen independently and uniformly at random from $[0..k-1]^n$. That is, no adaptation is needed for such combinations of n and k to *learn* the secret target vector z.

The situation changes for the regime around $k = n$, which was the focus of several subsequent publications [13, 59, 60]. These publications all showed bounds of order $n \log n$ for the $k = n$ Mastermind problem. Originally motivated by the study of black-box complexities for randomized black-box heuristics, these bounds were improved to $O(n \log \log n)$ in [29].

Theorem 3.10.2 (Theorem 2.1 in [29]). *For Mastermind with n positions and $k = \Omega(n)$ colors, the unrestricted black-box complexity of the n-*

position, k-color Mastermind game is $O(n \log\log n + k)$. *For* $k = o(n)$, *it is* $O\left(n \log\left(\frac{\log n}{\log(n/k)}\right)\right)$.

Like the $O(n/\log n)$ bound for the case $k = 2$, the bounds in Theorem 3.10.2 can be achieved by *deterministic* black-box algorithms [29, Theorem 2.3]. On the other hand, and unlike the situation considered in Theorem 3.10.1, it can be shown that any (deterministic or randomized) $o(n \log n)$ algorithm for the Mastermind game with $k = \Theta(n)$ colors has to be *adaptive,* showing that in this regime adaptive strategies are indeed more powerful than nonadaptive ones.

Theorem 3.10.3 (Theorem 4.1 and Lemma 4.2 in [29]). *The nonadaptive unrestricted black-box complexity of the Mastermind problem with n positions and k colors is* $\Omega\left(\frac{n \log k}{\max\{\log(n/k), 1\}}\right)$. *For* $k = n$, *this bound is tight, i.e., the nonadaptive unrestricted black-box complexity of the Mastermind problem with n positions and n colors is* $\Theta(n \log n)$.

Whether or not the $O(n \log\log n)$ upper bound in Theorem 3.10.2 can be further improved remains a – seemingly quite challenging – open problem. To date, the best known lower bound is the linear one reported in [14]. Some numerical results for different values of $k = n$ can be found in [7], but extending these numbers to asymptotic results may require a substantially new idea or technique for proving lower bounds in the unrestricted black-box complexity model.

3.11 Conclusion and Selected Open Problems

In this chapter we have surveyed theory-driven approaches that shed light on the performance limits of black-box optimization techniques such as local search strategies, nature-inspired heuristics, and pure random search. We have presented a detailed discussion of existing results for these black-box complexity measures. We now highlight a few avenues for future work in this young research discipline.

3.11.1 Extension to Other Optimization Problems

In line with the existing literature, our focus has been on classes of classical benchmark problems such as the OneMax, LeadingOnes, Jump, MST, and SSSP problems, since for these problems we can compare the black-box complexity results with known running-time results for well-understood heuristics. As with running-time analysis, it would be highly desirable to extend these results to other problem classes.

3.11.2 Systematic Investigation of Combined Black-Box Models

In the years before around 2013, most research on black-box complexity was centered around the question of how individual characteristics of state-of-the-art heuristics influence their performance. With this aim in mind, various black-box models have been developed that each restrict the algorithms with respect to some specific property, for example their memory size, or the properties of their variation operators or of the selection mechanisms in use. Since 2013 we have observed an increasing interest in combining two or more such restrictions to obtain a better picture of what is needed to design algorithms that significantly excel over existing approaches. A systematic investigation of such combined black-box models constitutes one of the most promising avenues for future research.

3.11.3 Tools to Derive Lower Bounds

To date, the most powerful technique to prove lower bounds on the black-box complexity of a problem is the information-theoretic approach, most notably in the form of Yao's minimax principle, and the simple information-theoretic lower bound presented in Theorem 3.2.4. Refined variants of this theorem have been designed to capture the situation in which the number of possible function values depends on the state of the optimization process, or where the probabilities for different objective values are nonhomogeneous. Unfortunately, either the verification that the conditions under which these theorems apply or the computation of a closed expression that summarizes the resulting bounds is often very tedious, making these extensions rather difficult to apply. Alternative tools for the derivation of lower bounds in black-box complexity contexts form another of the most desirable directions for future work.

In particular, for the k-ary unbiased black-box complexity with arities $k \geq 2$, we do not have any model-specific lower bounds. We therefore do not know, for example, if the linear bound on the binary unbiased black-box complexity of $\textsc{OneMax}_n$ or the $O(n \log n)$ bound on the binary unbiased black-box complexity of $\textsc{LeadingOnes}_n$ is tight, or whether the power of recombination is even larger than what these bounds, in comparison with the unary unbiased black-box complexities, indicate.

Another specifically interesting problem is raised by the $\Omega(n^2)$ lower bound on the (1+1) elitist black-box complexity of $\textsc{LeadingOnes}_n$ presented in Theorem 3.7.6. It has been conjectured in [51, Section 4] that this bound holds even for the (1+1) memory-restricted setting. A more systematic investigation of lower bounds for memory-restricted black-box models would

help us to understand better the role of large populations in evolutionary computation, a question that is not very well understood to date.

3.11.4 Beyond Worst-Case Expected Optimization Time as Unique Performance Measure

Black-box complexity, as introduced in this chapter, takes the *worst-case expected optimization time* as the performance measure. This measure reduces the whole optimization procedure to one single number. This, naturally, has several disadvantages. The same critique applies to running-time analysis in general, which is very much centered around this single performance measure. Complementary performance indicators such as *fixed-budget* (see [65]) and *fixed-target* (see [11]) results have been proposed in the literature, but unfortunately have not yet attracted significant attention. Since these measure give a better picture of the *anytime behavior* of black-box optimization techniques, we believe that an extension of existing black-box complexity results to such anytime statements would make it easier to communicate and to discuss the results with practitioners, for whom the anytime performance is often at least as important as the expected optimization time.

In the same context, one may ask if the *expected* optimization time should be the only measure considered. Clearly, when the optimization time $T(A, f)$ of an algorithm A on a function f is highly concentrated, its expectation is often very similar to its median, and is in particular of the same or similar asymptotic order. Such concentration can be observed for the running time of classical heuristics on most of the benchmark problems considered in this chapter. At the same time, it is also not very difficult to construct problems for which such a concentration provably does not hold. In particular, for multimodal problems, in which two or more local optima exist, the running time is often not concentrated. In [49, Section 3] examples were presented for which the probability of finding a solution within a small polynomial given bound is rather large, but where – owing to excessive running times in the remaining cases – the expected optimization time is very large. This motivated the authors of [49] to introduce the concept of *p-Monte Carlo black-box complexity*. The *p-Monte Carlo black-box complexity* of a class $\mathcal{F}$ of functions measures the time it takes to optimize any problem $f \in \mathcal{F}$ with failure probability at most p. It was shown that even for small p, the p-Monte Carlo black-box complexity of a function class $\mathcal{F}$ can be smaller by an exponential factor than its traditional (expected) black-box complexity, which is referred to as the *Las Vegas black-box complexity* in [49].

Acknowledgements This work was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH, in a

joint call with the Gaspard Monge Program for optimization, operations research, and their interactions with data sciences.

References

[1] Afshani, P., Agrawal, M., Doerr, B., Doerr, C., Larsen, K.G., Mehlhorn, K.: The query complexity of finding a hidden permutation. In: Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday, *Lecture Notes in Computer Science*, vol. 8066, pp. 1–11. Springer (2013)

[2] Anil, G., Wiegand, R.P.: Black-box search by elimination of fitness functions. In: Proc. of Foundations of Genetic Algorithms (FOGA'09), pp. 67–78. ACM (2009)

[3] Antipov, D., Doerr, B.: Precise runtime analysis for plateaus. In: Proc. of Parallel Problem Solving from Nature (PPSN'18), *Lecture Notes in Computer Science*, vol. 11102, pp. 117–128. Springer (2018)

[4] Badkobeh, G., Lehre, P.K., Sudholt, D.: Unbiased black-box complexity of parallel search. In: Proc. of Parallel Problem Solving from Nature (PPSN'14), *Lecture Notes in Computer Science*, vol. 8672, pp. 892–901. Springer (2014)

[5] Badkobeh, G., Lehre, P.K., Sudholt, D.: Black-box complexity of parallel search with distributed populations. In: Proc. of Foundations of Genetic Algorithms (FOGA'15), pp. 3–15. ACM (2015)

[6] Bshouty, N.H.: Optimal algorithms for the coin weighing problem with a spring scale. In: Proc. of the 22nd Conference on Learning Theory (COLT'09). Omnipress (2009)

[7] Buzdalov, M.: An algorithm for computing lower bounds for unrestricted black-box complexities. In: Companion Material for Proc. of Genetic and Evolutionary Computation Conference (GECCO'16), pp. 147–148. ACM (2016)

[8] Buzdalov, M., Doerr, B.: Runtime analysis of the $(1+(\lambda,\lambda))$ Genetic Algorithm on random satisfiable 3-CNF formulas. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'17), pp. 1343–1350. ACM (2017)

[9] Buzdalov, M., Doerr, B., Kever, M.: The unrestricted black-box complexity of jump functions. Evolutionary Computation **24**(4), 719–744 (2016). DOI 10.1162/EVCO_a_00185. URL https://doi.org/10.1162/EVCO_a_00185

[10] Cantor, D.G., Mills, W.H.: Determining a subset from certain combinatorial properties. Canadian Journal of Mathematics **18**, 42–48 (1966)

[11] Carvalho Pinto, E., Doerr, C.: Discussion of a more practice-aware runtime analysis for evolutionary algorithms. In: Proc. of Artificial Evo-

lution (EA'17), pp. 298–305 (2017). URL https://ea2017.inria.fr//EA2017_Proceedings_web_ISBN_978-2-9539267-7-4.pdf
[12] Cathabard, S., Lehre, P.K., Yao, X.: Non-uniform mutation rates for problems with unknown solution lengths. In: Proc. of Foundations of Genetic Algorithms (FOGA'11), pp. 173–180. ACM (2011)
[13] Chen, Z., Cunha, C., Homer, S.: Finding a hidden code by asking questions. In: Proc. of the 2nd Annual International Conference on Computing and Combinatorics (COCOON'96), pp. 50–55. Springer (1996)
[14] Chvátal, V.: Mastermind. Combinatorica **3**, 325–329 (1983)
[15] Corus, D., He, J., Jansen, T., Oliveto, P.S., Sudholt, D., Zarges, C.: On easiest functions for mutation operators in bio-inspired optimisation. Algorithmica **78**, 714–740 (2017). DOI 10.1007/s00453-016-0201-4. URL https://doi.org/10.1007/s00453-016-0201-4
[16] Dang, D., Friedrich, T., Kötzing, T., Krejca, M.S., Lehre, P.K., Oliveto, P.S., Sudholt, D., Sutton, A.M.: Escaping local optima with diversity mechanisms and crossover. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'16), pp. 645–652. ACM (2016)
[17] Dang, D., Lehre, P.K.: Runtime analysis of non-elitist populations: From classical optimisation to partial information. Algorithmica **75**, 428–461 (2016). DOI 10.1007/s00453-015-0103-x. URL https://doi.org/10.1007/s00453-015-0103-x
[18] Dang, D.C., Friedrich, T., Kötzing, T., Krejca, M.S., Lehre, P.K., Oliveto, P.S., Sudholt, D., Sutton, A.M.: Escaping local optima using crossover with emergent diversity. IEEE Transactions on Evolutionary Computation **22**(3), 484–497 (2018)
[19] Doerr, B.: Optimal parameter settings for the $(1+(\lambda,\lambda))$ genetic algorithm. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'16), pp. 1107–1114. ACM (2016)
[20] Doerr, B., Doerr, C.: Black-box complexity: from complexity theory to playing Mastermind. In: Companion Material for Proc. of Genetic and Evolutionary Computation Conference (GECCO'14), pp. 623–646. ACM (2014). URL http://doi.acm.org/10.1145/2598394.2605352
[21] Doerr, B., Doerr, C.: Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'15), pp. 1335–1342. ACM (2015)
[22] Doerr, B., Doerr, C.: A tight runtime analysis of the $(1+(\lambda,\lambda))$ genetic algorithm on OneMax. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'15), pp. 1423–1430. ACM (2015)
[23] Doerr, B., Doerr, C.: The impact of random initialization on the runtime of randomized search heuristics. Algorithmica **75**, 529–553 (2016). URL https://doi.org/10.1007/s00453-015-0019-5
[24] Doerr, B., Doerr, C.: Optimal static and self-adjusting parameter choices for the $(1+(\lambda,\lambda))$ genetic algorithm. Algorithmica **80**, 1658–1709 (2018)

[25] Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing new genetic algorithms. Theoretical Computer Science **567**, 87–104 (2015)
[26] Doerr, B., Doerr, C., Kötzing, T.: The unbiased black-box complexity of partition is polynomial. Artificial Intelligence **216**, 275–286 (2014). URL https://doi.org/10.1016/j.artint.2014.07.009
[27] Doerr, B., Doerr, C., Kötzing, T.: Solving problems with unknown solution length at (almost) no extra cost. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'15), pp. 831–838. ACM (2015). URL http://doi.acm.org/10.1145/2739480.2754681
[28] Doerr, B., Doerr, C., Kötzing, T.: Unbiased black-box complexities of jump functions. Evolutionary Computation **23**, 641–670 (2015). URL https://doi.org/10.1162/EVCO_a_00158
[29] Doerr, B., Doerr, C., Spöhel, R., Thomas, H.: Playing Mastermind with many colors. Journal of the ACM **63**, 42:1–42:23 (2016). URL http://dl.acm.org/citation.cfm?id=2987372
[30] Doerr, B., Doerr, C., Yang, J.: k-bit mutation with self-adjusting k outperforms standard bit mutation. In: Proc. of Parallel Problem Solving from Nature (PPSN'16), *Lecture Notes in Computer Science*, vol. 9921, pp. 824–834. Springer (2016)
[31] Doerr, B., Doerr, C., Yang, J.: Optimal parameter choices via precise black-box analysis. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'16), pp. 1123–1130. ACM (2016)
[32] Doerr, B., Fouz, M., Witt, C.: Quasirandom evolutionary algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'10), pp. 1457–1464. ACM (2010). DOI 10.1145/1830483.1830749. URL http://doi.acm.org/10.1145/1830483.1830749
[33] Doerr, B., Happ, E., Klein, C.: Crossover can provably be useful in evolutionary computation. Theoretical Computer Science **425**, 17–33 (2012)
[34] Doerr, B., Jansen, T., Witt, C., Zarges, C.: A method to derive fixed budget results from expected optimisation times. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'13), pp. 1581–1588. ACM (2013). DOI 10.1145/2463372.2463565. URL http://doi.acm.org/10.1145/2463372.2463565
[35] Doerr, B., Johannsen, D., Kötzing, T., Lehre, P.K., Wagner, M., Winzen, C.: Faster black-box algorithms through higher arity operators. In: Proc. of Foundations of Genetic Algorithms (FOGA'11), pp. 163–172. ACM (2011)
[36] Doerr, B., Johannsen, D., Kötzing, T., Neumann, F., Theile, M.: More effective crossover operators for the all-pairs shortest path problem. Theoretical Computer Science **471**, 12–26 (2013)
[37] Doerr, B., Kötzing, T., Lengler, J., Winzen, C.: Black-box complexities of combinatorial problems. Theoretical Computer Science **471**, 84–106 (2013)

[38] Doerr, B., Kötzing, T., Winzen, C.: Too fast unbiased black-box algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'11), pp. 2043–2050. ACM (2011)
[39] Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'17), pp. 777–784. ACM (2017). DOI 10.1145/3071178.3071301. URL http://doi.acm.org/10.1145/3071178.3071301
[40] Doerr, B., Theile, M.: Improved analysis methods for crossover-based algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'09), pp. 247–254. ACM (2009)
[41] Doerr, B., Winzen, C.: Black-box complexity: Breaking the $O(n \log n)$ barrier of LeadingOnes. In: Artificial Evolution (EA'11), Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 7401, pp. 205–216. Springer (2012)
[42] Doerr, B., Winzen, C.: Memory-restricted black-box complexity of OneMax. Information Processing Letters **112**(1-2), 32–34 (2012). URL https://doi.org/10.1016/j.ipl.2011.10.004
[43] Doerr, B., Winzen, C.: Playing Mastermind with constant-size memory. Theory of Computing Systems **55**, 658–684 (2014). URL https://doi.org/10.1007/s00224-012-9438-8
[44] Doerr, B., Winzen, C.: Ranking-based black-box complexity. Algorithmica **68**, 571–609 (2014). URL https://doi.org/10.1007/s00453-012-9684-9
[45] Doerr, B., Winzen, C.: Reducing the arity in unbiased black-box complexity. Theoretical Computer Science **545**, 108–121 (2014). URL https://doi.org/10.1016/j.tcs.2013.05.004
[46] Doerr, C., Lengler, J.: Elitist black-box models: Analyzing the impact of elitist selection on the performance of evolutionary algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'15), pp. 839–846. ACM (2015). URL http://doi.acm.org/10.1145/2739480.2754654
[47] Doerr, C., Lengler, J.: OneMax in black-box models with several restrictions. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'15), pp. 1431–1438. ACM (2015)
[48] Doerr, C., Lengler, J.: The (1+1) elitist black-box complexity of LeadingOnes. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'16), pp. 1131–1138. ACM (2016). URL http://doi.acm.org/10.1145/2908812.2908922
[49] Doerr, C., Lengler, J.: Introducing elitist black-box models: When does elitist behavior weaken the performance of evolutionary algorithms? Evolutionary Computation **25** (2017). DOI 10.1162/evco_a_00195. URL https://doi.org/10.1162/evco_a_00195
[50] Doerr, C., Lengler, J.: OneMax in black-box models with several restrictions. Algorithmica **78**, 610–640 (2017). URL https://doi.org/10.1007/s00453-016-0168-1

[51] Doerr, C., Lengler, J.: The (1+1) elitist black-box complexity of LeadingOnes. Algorithmica **80**, 1579–1603 (2018). DOI 10.1007/s00453-017-0304-6. URL https://doi.org/10.1007/s00453-017-0304-6
[52] Dorigo, M., Stützle, T.: Ant colony optimization. MIT Press (2004)
[53] Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science **276**, 51–81 (2002)
[54] Droste, S., Jansen, T., Wegener, I.: Upper and lower bounds for randomized search heuristics in black-box optimization. Theory of Computing Systems **39**, 525–544 (2006)
[55] Erdős, P., Rényi, A.: On two problems of information theory. Magyar Tudományos Akadémia Matematikai Kutató Intézet Közleményei **8**, 229–243 (1963)
[56] Fischer, S., Wegener, I.: The Ising model on the ring: Mutation versus recombination. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'04), *Lecture Notes in Computer Science*, vol. 3102, pp. 1113–1124. Springer (2004)
[57] Fournier, H., Teytaud, O.: Lower bounds for comparison based evolution strategies using VC-dimension and sign patterns. Algorithmica **59**, 387–408 (2011)
[58] Goldman, B.W., Punch, W.F.: Fast and efficient black box optimization using the parameter-less population pyramid. Evolutionary Computation **23**, 451–479 (2015)
[59] Goodrich, M.T.: On the algorithmic complexity of the Mastermind game with black-peg results. Information Processing Letters **109**, 675–678 (2009)
[60] Jäger, G., Peczarski, M.: The number of pessimistic guesses in generalized black-peg Mastermind. Information Processing Letters **111**, 933–940 (2011)
[61] Jansen, T.: Black-box complexity for bounding the performance of randomized search heuristics. In: Y. Borenstein, A. Moraglio (eds.) Theory and Principled Methods for the Design of Metaheuristics, Natural Computing Series, pp. 85–110. Springer (2014). DOI 10.1007/978-3-642-33206-7_5. URL https://doi.org/10.1007/978-3-642-33206-7_5
[62] Jansen, T.: On the black-box complexity of example functions: The real jump function. In: Proc. of Foundations of Genetic Algorithms (FOGA'15), pp. 16–24. ACM (2015)
[63] Jansen, T., Sudholt, D.: Analysis of an asymmetric mutation operator. Evolutionary Computation **18**, 1–26 (2010). DOI 10.1162/evco.2010.18.1.18101. URL https://doi.org/10.1162/evco.2010.18.1.18101
[64] Jansen, T., Wegener, I.: The analysis of evolutionary algorithms - a proof that crossover really can help. Algorithmica **34**, 47–66 (2002)

[65] Jansen, T., Zarges, C.: Performance analysis of randomised search heuristics operating with a fixed budget. Theoretical Computer Science **545**, 39–58 (2014). URL https://doi.org/10.1016/j.tcs.2013.06.007

[66] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)

[67] Knuth, D.E.: The computer as Master Mind. Journal of Recreational Mathematics **9**, 1–5 (1977)

[68] Kötzing, T., Neumann, F., Sudholt, D., Wagner, M.: Simple max-min ant systems and the optimization of linear pseudo-boolean functions. In: Proc. of Foundations of Genetic Algorithms (FOGA'11), pp. 209–218. ACM (2011). DOI 10.1145/1967654.1967673. URL http://doi.acm.org/10.1145/1967654.1967673

[69] de Perthuis de Laillevault, A., Doerr, B., Doerr, C.: Money for nothing: Speeding up evolutionary algorithms through better initialization. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'15), pp. 815–822. ACM (2015). URL http://doi.acm.org/10.1145/2739480.2754760

[70] Lässig, J., Sudholt, D.: Analysis of speedups in parallel evolutionary algorithms and (1+λ) EAs for combinatorial optimization. Theoretical Computer Science **551**, 66–83 (2014). DOI 10.1016/j.tcs.2014.06.037. URL https://doi.org/10.1016/j.tcs.2014.06.037

[71] Lehre, P.K., Witt, C.: Black-box search by unbiased variation. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'10), pp. 1441–1448. ACM (2010)

[72] Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**, 623–642 (2012)

[73] Lindström, B.: On a combinatory detection problem i. Mathematical Institute of the Hungarian Academy of Science **9**, 195–207 (1964)

[74] Lindström, B.: On a combinatorial problem in number theory. Canadian Mathematical Bulletin **8**, 477–490 (1965)

[75] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. The Journal of Chemical Physics **21**, 1087–1092 (1953)

[76] Mironovich, V., Buzdalov, M.: Hard test generation for maximum flow algorithms with the fast crossover-based evolutionary algorithm. In: Companion Material for Proc. of Genetic and Evolutionary Computation Conference (GECCO'15), pp. 1229–1232. ACM (2015)

[77] Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)

[78] Neumann, F., Witt, C.: Runtime analysis of a simple ant colony optimization algorithm. Algorithmica **54**, 243–255 (2009)

[79] Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity. Springer (2010)

[80] Paixão, T., Pérez Heredia, J., Sudholt, D., Trubenová, B.: Towards a runtime comparison of natural and artificial evolution. Algorithmica **78**, 681–713 (2017)
[81] Rowe, J., Vose, M.: Unbiased black box search algorithms. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'11), pp. 2035–2042. ACM (2011)
[82] Storch, T.: Black-box complexity: Advantages of memory usage. Information Processing Letters **116**(6), 428–432 (2016). DOI 10.1016/j.ipl.2016.01.009. URL https://doi.org/10.1016/j.ipl.2016.01.009
[83] Sudholt, D.: Crossover is provably essential for the Ising model on trees. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'05), pp. 1161–1167. ACM Press (2005)
[84] Sudholt, D.: Crossover speeds up building-block assembly. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO'12), pp. 689–702. ACM (2012)
[85] Sudholt, D.: A new method for lower bounds on the running time of evolutionary algorithms. IEEE Transactions on Evolutionary Computation **17**, 418–435 (2013)
[86] Teytaud, O., Gelly, S.: General lower bounds for evolutionary algorithms. In: Proc. of Parallel Problem Solving from Nature (PPSN 2006), *Lecture Notes in Computer Science*, vol. 4193, pp. 21–31. Springer (2006)
[87] Witt, C.: Tight bounds on the optimization time of a randomized search heuristic on linear functions. Combinatorics, Probability & Computing **22**, 294–318 (2013)
[88] Yao, A.C.C.: Probabilistic computations: Toward a unified measure of complexity. In: Proc. of Foundations of Computer Science (FOCS'77), pp. 222–227. IEEE (1977)