The (1+1) Elitist Black-Box Complexity of LeadingOnes

Carola Doerr
CNRS & Univ. Pierre et Marie Curie
Paris, France

Johannes Lengler
ETH Zürich
Zürich, Switzerland

ABSTRACT

One important goal of black-box complexity theory is the development of complexity models allowing to derive meaningful lower bounds for whole classes of randomized search heuristics. Complementing classical runtime analysis, black-box models help us understand how algorithmic choices such as the population size, the variation operators, or the selection rules influence the optimization time. One example for such a result is the $\Omega(n \log n)$ lower bound for unary unbiased algorithms on functions with a unique global optimum [Lehre/Witt, GECCO 2010], which tells us that higher arity operators or biased sampling strategies are needed when trying to beat this bound. In lack of analyzing techniques, almost no non-trivial bounds are known for other restricted models. Proving such bounds therefore remains to be one of the main challenges in black-box complexity theory.

With this paper we contribute to our technical toolbox for lower bound computations by proposing a new type of information-theoretic argument. We regard the permutation- and bit-invariant version of LeadingOnes and prove that its (1+1) elitist black-box complexity is $\Omega(n^2)$, a bound that is matched by (1+1)-type evolutionary algorithms. The (1+1) elitist complexity of LeadingOnes is thus considerably larger than its unrestricted one, which is known to be of order $n \log \log n$ [Afshani et al., 2013].

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*

Keywords

Black-Box Complexity; Selection; Information; Entropy

1. INTRODUCTION

Since the seminal paper of Droste, Jansen, and Wegener [12] *black-box complexity* is the most accepted complexity measure for black-box search heuristics such as evolutionary algorithms (EAs). Informally, the black-box complexity of a set $\mathcal{F}$ of functions is the minimum expected number of function evaluations that are needed to solve any problem instance $f \in \mathcal{F}$, where the minimum is taken over a class of algorithms $\mathcal{A}$. The original black-box model by Droste, Jansen, and Wegener regards as $\mathcal{A}$ the whole collection of possible black-box algorithms. It is therefore called the *unrestricted black-box model*. However, unlike in classical complexity theory where a widely accepted complexity notion is used, several black-box complexity models coexist in the theory of randomized search heuristics. Each model regards a different collection $\mathcal{A}$ of algorithms (e.g., the memory-restricted model regards only such algorithms that keep in the memory only a limited number of previously sampled search points while in contrast the algorithms in the unrestricted black-box complexity have full access to all previous function evaluations). When we compare the black-box complexity of a problem in the different models, we thus learn how certain algorithmic choices such as the population size, the variation operators in use, or the selection rules influence the performance of the search heuristics.

At GECCO 2015 [8] we have presented a new black-box model that combines several of the previously regarded restrictions such as the size of the memory and the selection rules. More precisely, we have defined a collection of $(\mu+\lambda)$ *elitist black-box models*, where a $(\mu+\lambda)$ *elitist algorithm* is one that keeps at any point in time the μ best-so-far sampled solutions. In the next iteration it is allowed to use only the relative (not absolute) function values of these μ points to create some λ new search points. Truncation selection is then used to select from these $\mu+\lambda$ points the μ surviving ones that form the parent population of the next iteration. A $(\mu+\lambda)$ black-box algorithm is thus in particular a memory-restricted and ranking-based one in the sense of [6, 12] and [7], respectively. In addition, it has to employ truncation selection as replacement rule.

In [8] examples are presented for which the elitist black-box complexity is much larger than in any of the previously existing black-box models, while in [9] it is shown that the complexity of OneMax is of linear order only even for the most restrictive (1+1) setting. Here in this work we regard another classical problem, a generalized version of the LeadingOnes function. We show that its (1+1) elitist black-box complexity matches the performance of typical evolutionary algorithms.

1.1 The LeadingOnes Problem

LeadingOnes (Lo for short) is among the best-studied

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GECCO '16, July 20 - 24, 2016, Denver, CO, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-4206-3/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2908812.2908922

Model	Lower Bound		Upper Bound	
unrestricted	$\Omega(n \log\log n)$	[1]	$O(n \log\log n)$	[1]
unbiased, arity 1	$\Omega(n^2)$	[14]	$O(n^2)$	[17] for (1+1) EA
unbiased, arity 2	$\Omega(n \log\log n)$		$O(n \log n)$	[4]
unbiased, arity ≥ 3	$\Omega(n \log\log n)$		$O(n \log(n)/\log\log n)$	[5]
ranking-based unbiased, arity ≥ 3	$\Omega(n \log\log n)$		$O(n \log(n)/\log\log n)$	[5]
elitist, arity 1	$\Omega(n^2)$	this paper	$O(n^2)$	[11]

Table 1: Known black-box complexities of LeadingOnes. The lower bounds for higher arities follow from the lower bound in the unrestricted model.

functions in the theory of evolutionary computation. It was originally designed in [17] to disprove the conjecture of Mühlenbein [16] that the expected runtime of the $(1 + 1)$ EA on every unimodal function is $O(n \log n)$. While Rudolph showed experimentally that its expected runtime is $\Theta(n^2)$, this bound was proved a bit later in [11]. Quite exact bounds for the expected runtime of the $(1+1)$ EA on Lo have been shown in [2, 13, 18].

As argued in [5], most EAs behave symmetrically with respect to function representation, and it is therefore natural that the proven performance guarantees extend to the *class of generalized* Lo *functions* which contains all pseudo-Boolean functions $f : \{0, 1\}^n \to \mathbb{R}$ having a fitness landscape that is isomorphic to that of the original Lo function (which assigns to each bit string the number of initial ones, cf. Section 2 for precise definitions).

The (1+1) EA qualifies as a (1+1) elitist algorithm in the sense of [8]. The above-mentioned runtime bounds therefore imply that the (1+1) elitist black-box complexity of Lo is of order at most n^2. We show that this is bound is tight.

1.2 Discussion of Our Result

Our main result is summarized by the following statement.

Theorem 1. *The (1+1) elitist black-box complexity of* Lo *is* $\Theta(n^2)$.

As mentioned, this bound is matched by the average performance of the (1+1) EA. It is also matched by the expected runtime of Randomized Local Search (RLS), thus showing that these two simple strategies are asymptotically optimal for Lo among all (1+1)-type elitist algorithms. Our result also shows that an algorithm trying to beat the $\Omega(n^2)$ bound (and such algorithms exist, cf. below) has to use larger population sizes or non-elitist selection strategies.

In our proof we will not make use of the fact that elitist algorithms have to be ranking-based; that is, the (1+1) elitist black-box complexity of Lo remains $\Theta(n^2)$ even if the algorithms have access to the absolute fitness value of the current search point.

We summarize all known black-box complexities of Lo in Table 1. Relevant for our contribution are in particular the tight $\Theta(n \log\log n)$ bound for the unrestricted black-box complexity of Lo obtained by Afshani, Agrawal, Doerr, Doerr, Larsen, and Mehlhorn [1], the $O(n \log n)$ bound for the binary unbiased black-box complexity by Doerr, Johannsen, Kötzing, Lehre, Wagner, and Winzen, [4], and the $\Omega(n^2)$ bound for the unary unbiased model by Lehre and Witt [14]. Note also that already simple binary search exhibits a complexity of only $O(n \log n)$ on the Lo problem (cf. [1] for an implementation of the binary search strategy). This is why it is remarkable that (1+1) elitist algorithms have such a rather weak performance on Lo.

Our result is not the first lower bound of quadratic order for the Lo problem. In fact, Lehre and Witt proved in [14] that all unary unbiased search strategies, i.e., intuitively speaking, all black-box algorithms using only mutation as variation operators, need $\Omega(n^2)$ function evaluations on average to optimize this problem. Combining our result with theirs, we see that even if we replace the mutation operator in RLS (which flips exactly one bit, chosen uniformly at random) or the (1+1) EA (which flips independently each bit with probability $1/n$) by a—possibly strongly—biased one, the resulting algorithm would still need time $\Omega(n^2)$ on average. This shows that not only unbiased sampling, but also the population structure of the algorithm and the selection strategies determine the comparatively slow convergence of these two well-known search heuristics.

In addition to identifying such structural bottlenecks, our result is also—and this is in fact the main motivation for our studies—interesting from a purely mathematical point of view, as we need to develop some new tools for the lower bound proof. Specifically, we use some information-theoretic arguments, utilizing that the amount of information that the algorithm has at any given point is not sufficient to make substantial progress. Such information-theoretic arguments are notoriously hard to formulate rigorously, and are even harder to employ in a non-trivial situation like ours.

What complicates our analysis is the fact that the Lo functions, in principle, allow for a rather huge storage. Indeed, when the fitness of an individual is k for some $k < n$, then all but the $k + 1$ bits determining the fitness of the search point can be used for storing information about previous samples, the number of iterations elapsed, or any other information gathered during the optimization process. We recall that such strategies of constantly storing information about previous samples are at the heart of many upper bounds in black-box complexity, cf., for example, the proofs of the $O(n/\log n)$ bound for the (1+1) memory-restricted [6], the ranking-based [7] or the (1+1) elitist Monte Carlo [9] black-box complexity of OneMax. We therefore need to show that although changing the $n-(k+1)$ *irrelevant bits* has no influence on the fitness, the algorithm cannot make effective use of this storage space.

The intuitive reason why the given storage is not large enough is that since the Lo problem is permutation-invariant, the black-box algorithms do not know where the irrelevant bits are located, and storing this information would require more bits than available. However, the discrepancy is rather small: in most parts of the process, if the number of bits of the storage space was larger by just a constant factor, then this would trivially allow for efficient use of the storage, and the lower bounds would break down. It is thus essential to find a good measure for the informa-

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

Initialization: $X \leftarrow \emptyset$;
  for $i = 1, \dots, \mu$ **do**
    Depending only on the multiset X and the ranking $\rho(X, f)$ of X induced by f, choose a probability distribution $p^{(i)}$ over $\{0,1\}^n$ and sample $x^{(i)}$ according to $p^{(i)}$;
    $X \leftarrow X \cup \{x^{(i)}\}$;
Optimization: **for** $t = 1, 2, 3, \dots$ **do**
  Depending only on the multiset X and the ranking $\rho(X, f)$ of X induced by f choose a probability distribution $p^{(t)}$ on $(\{0,1\}^n)_{i=1}^{\lambda}$ and sample $(y^{(1)}, \dots, y^{(\lambda)})$ according to $p^{(t)}$;
  Set $X \leftarrow X \cup \{y^{(1)}, \dots, y^{(\lambda)}\}$;
  for $i = 1, \dots, \lambda$ **do** Select $x \in \arg\min X$ and update $X \leftarrow X \setminus \{x\}$;

tion that an algorithm can possibly encode in its queries. We develop a precise notion that bounds the amount of information that the algorithm has about the function $\text{Lo}_{z,\sigma}$ at any given state. We can use this notion to estimate the gain that the algorithm can draw from any given amount of information. We consider one of the main contributions of our paper to make these intuitive concepts precise and utilizable in proofs. Although our definitions are adapted to the Lo problem, we are optimistic that the developed techniques are applicable also to other black-box settings and, of course, also to other problem classes.

A second difficulty that we face in our proof is that the the MiniMax principle of Yao [19]—which is the foremost technique in black-box complexity theory to prove lower bounds—cannot be applied to the elitist model, as discussed in [8]. We therefore need to find an extension of the elitist model that is generous enough to allow for the application of the information-theoretic tool but that does, at the same time, not decrease the black-box complexity by too much.

Proofs. Proofs that are omitted for reasons of space can be found in the full version of this paper [10].

2. FORMAL DEFINITIONS

Black-box Complexity. We come to the formal definitions. A $(\mu+\lambda)$ elitist black-box algorithm is a (possibly randomized) algorithm that can be described by the framework of Algorithm 1. I.e., assume that a pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$ is given, the *fitness function*. Then the algorithm maintains a multiset X of μ search points, which is called the *population*. It is initialized by sampling iteratively μ search points, which may only depend on the previous search points and the ranking induced on them by the fitness function f. Afterwards, in each round it samples λ new search points (*offspring*) based only on the search points in X and their ranking with respect to f. It then forms the new population by choosing the μ search points from the old population and the offspring that have the largest fitness, where it may break ties arbitrarily. This process is repeated until a maximum of f is sampled.

The *runtime* of a $(\mu + \lambda)$ elitist black-box algorithm A on a pseudo-Booloean function f is the number of search points (*queries*) that A samples before it samples for the first time a maximum of f. The expected runtime of A on a class $\mathcal{F}$ of pseudo-Boolean functions is the maximum expected runtime of A on f, where f runs through $\mathcal{F}$. The $(\mu+\lambda)$ elitist black-box complexity of $\mathcal{F}$ is the smallest expected runtime of a $(\mu + \lambda)$ elitist black-box algorithm on $\mathcal{F}$.

Remark 2. *In [8] we distinguished between two different notions of runtime, which in turn lead to different notions of (elitist) black-box complexity: the* Las Vegas *runtime of a black-box algorithm A on a function f is the expected number of steps until A finds the optimum of f, while the* p-Monte Carlo *runtime is the minimal number of steps A needs in order to find the optimum of f with probability at least $1-p$. It was shown that there may be an exponential gap between the resulting elitist black-box complexities. However, this is not the case for* Lo*. Formally, we show for* Lo *that the (1+1) elitist Las Vegas black box complexity is $\Omega(n^2)$, and for every constant $p > 0$ the (1+1) elitist p-Monte Carlo black box complexity is $\Omega(n^2)$. Both statements are immediate consequences of Theorem 11.*

In this paper, black-box complexity refers to the Las Vegas version unless specified otherwise.

LeadingOnes. The original Lo function is defined via $\text{Lo} : \{0,1\}^n \to \mathbb{R}, x \mapsto \max\{i \in \{0,1,\dots,n\} \mid \forall j \leq i : x_j = 1\}$. This is generalized to a permutation- and bit-invariant version in the following way. Let S_n denote the set of permutations of $[n] := \{1, \dots, n\}$ and let $[0..n] := \{0, 1, \dots, n\}$. For $z \in \{0,1\}^n$ and $\sigma \in S_n$ we consider the function $\text{Lo}_{z,\sigma} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : z_{\sigma(j)} = x_{\sigma(j)}\}$, so $\text{Lo}_{z,\sigma}(x)$ is the length of the longest common prefix of the search point x and the *target string* z in the order σ. For $i < j$ we say that $\sigma(i)$ is *more significant* than $\sigma(j)$. In particular, $\sigma(1), \dots, \sigma(k)$ are the k most significant bits, and $\sigma(n), \dots, \sigma(n-k+1)$ are the k least significant bits. The Lo problem is the problem of optimizing an unknown member of the class $\text{Lo} := \{\text{Lo}_{z,\sigma} \mid z \in \{0,1\}^n, \sigma \in S_n\}$. Clearly, the unique global optimum of $\text{Lo}_{z,\sigma}$ is z. For ease of notation we drop the subscript $\{z, \sigma\}$ in the following.

Notation. Throughout this work, we will write $\log x$ for the binary logarithm $\log_2 x$ and $\ln x$ for the natural logarithm of x. We say that an event $\mathcal{E}$ holds *with high probability* if $\Pr[\mathcal{E}] \to 1$ for $n \to \infty$.

A *fitness level* of a function f is a maximal set of search points which have the same fitness. In particular, every function $\text{Lo}_{z,\sigma}$ has exactly n different fitness levels.

3. PROOF OF THE LOWER BOUND

To prove the desired $\Omega(n^2)$ bound we introduce a more generous model in which the algorithms have strictly more power than in the original elitist model (Sections 3.3–3.5). We then show the $\Omega(n^2)$ lower bound for this more generous model in Section 3.6. Before we start with the technical details of the alternative model, we explain in Section 3.1 our motivation for introducing it. We present a high-level overview of the proof in Section 3.2.

3.1 Challenges in Proving the Lower Bound

Learning from search points with inferior fitness: One may be tempted to believe that in the (1+1) elitist model we cannot learn information from search points that

have stricty lower fitness than that of the current best one. This is indeed a tantalizing thought as such search points have to be discarded immediately and can therefore not influence the sampling distribution of the next query. However, one has to be very careful with such arguments. To illustrate why it fails in general, consider the following setting: assume that there is a search point x from which we sample search point y_1 with some very small probability ε and search point y_2 otherwise; i.e., we sample y_2 with probability $1-\varepsilon$. For the sake of the argument assume further that for all search points $z \neq x$ the probability to sample y_1 or y_2 is zero. If at some stage of the algorithm we happen to have y_1 in the memory we may then conclude that we must have been at x in the previous step. Moreover, if $f(y_2) > f(x)$ then with probability $1-\varepsilon$ we would have proceeded to y_2, and thus we would never have visited y_1 (as we cannot return to x from a fitter search point). Therefore, by Bayes' theorem $f(y_2) \leq f(x)$ with probability at least $1-\varepsilon$. Summarizing, although we have not visited y_2, we can deduce information about its fitness.

Application of Yao's Principle: A tool that has proven to be extremely helpful in deriving lower bounds for black-box complexities is the so-called MiniMax Principle of Yao [19]. All lower bounds that we are aware of directly or indirectly use (the easy direction of) this tool. In simple words, Yao's Principle allows us to restrict our attention to the performance of a best-possible deterministic algorithm on a random input. This is a lower bound for the expected performance of a best possible randomized algorithm for this problem. This principle is typically used with a very simple distribution p. Indeed, in most proofs p can be chosen to be the uniform distribution.

LEMMA 3 (YAO'S PRINCIPLE [15, 19]). *Let Π be a problem with a finite set $\mathcal{I}$ of input instances (of a fixed size) permitting a finite set $\mathcal{A}$ of deterministic algorithms. Let p be a probability distribution over $\mathcal{I}$ and q be a probability distribution over $\mathcal{A}$. Then,*

$$\min_{A \in \mathcal{A}} \mathbb{E}[T(I_p, A)] \leq \max_{I \in \mathcal{I}} \mathbb{E}[T(I, A_q)], \tag{1}$$

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

As was pointed out in [8], the informal interpretation of Yao's principle as stated before Lemma 3 does not apply to elitist algorithms. Since this is a crucial difficulty in our proofs, we explain this apparent contradiction in detail, even though a very similar example was given in [8].

Let $\mathcal{I} = \textsc{Lo}$, let p be the uniform distribution over the inputs $\mathcal{I}$, and let I_p as in Lemma 3. Then any deterministic algorithm A has a positive probability during the optimization of I_p of getting stuck. Assume x and y are the first two search points that A queries, and note that since A is an elitist black-box algorithm, the choice of y does not depend on the fitness of x (although the example could easily be adapted to cover this case as well). If the fitness of y is strictly smaller than that of x, y has to be discarded immediately and the algorithm is in exactly the same state as before, and will continue sampling and discarding y. Therefore, the expected runtime of A on this fitness function is infinite. Since this holds for every deterministic algorithm (that is, every deterministic (1+1) elitist algorithm has an infinite expected runtime on a uniformly chosen LO instance), the lower bound in (1) is infinite, too. However, of course there are randomized search strategies with finite expected runtime, e.g., RLS and the $(1+1)$ EA (see Section 1.1).

To resolve this apparent discrepancy, note that Lemma 3 makes a statement about algorithms that are a convex combination of deterministic ones. For typical classes of algorithms this describes exactly the class of randomized algorithm, since we can emulate every randomized algorithm by making all random coin flips in advance, and then choosing the deterministic algorithm whose decisions in each step agree with these coin flips. However, this only works if the algorithm is free to make a new decision in each step, e.g., if the algorithm may base its decision on the number of previous steps. However, (1+1) black-box algorithms (elitist or not) may not do so since they are *memory-restricted*. Therefore, randomized (1+1) black-box algorithms cannot be in general written as convex combinations of deterministic (1+1) black-box algorithms.

These observations have a quite severe effect on our ability to prove lower bounds in elitist black-box models. Indeed, the only way we currently know is the approach taken below, where we consider a superset $\mathcal{A}_{\mathcal{M}}$ of algorithms such that every randomized algorithm in $\mathcal{A}_{\mathcal{M}}$ can be expressed as a convex combination of deterministic ones. A lower bound shown for this broader class trivially applies to all elitist black-box algorithms. In our case, we achieve the class $\mathcal{A}_{\mathcal{M}}$ by giving the algorithms access to enough memory to determine the current step (within a certain phase).

If applicable, Yao's principle allows us to restrict ourselves to deterministic algorithms, which are usually easier to analyze. In particular, we may use the following observation.

REMARK 4. *Assume that we run a deterministic algorithm A on a problem instance i that we have taken from the set of instances $\mathcal{I}$ uniformly at random, so $\Pr[i = c] = 1/|\mathcal{I}|$ for every $c \in \mathcal{I}$. Assume further that the first queries $q_1, \ldots, q_\ell$ of A reduce the number of possible problem instances to some set $\mathcal{C}$. Then $\Pr[i = c \mid q_1, \ldots, q_\ell] = \Pr[i = c \mid i \in \mathcal{C}] = 1/|\mathcal{C}|$ for all $c \in \mathcal{C}$. In particular, each $c \in \mathcal{C}$ is equally likely to be the secret instance i.*

3.2 High-Level Ideas of the Proof

As explained above, we cannot use Yao's principle directly for the set of all elitist black-box algorithms. Instead, we use a larger class $\mathcal{A}_{\mathcal{M}}$ of algorithms, the definition of which is adapted to the special structure of the LO problem. In this model, whenever an algorithm reaches fitness level k for the first time, we reveal for a brief moment the position of the k most significant bits. Note that by symmetry of the LO function, an algorithm cannot discriminate between the less significant bits from its previous samples. Therefore, everything that the algorithm has learned previously is covered by this piece of information. Based on this information, we allow the algorithm to "store" whatever it wants in the $m = n - k$ least significant bits. After that, we occlude the information about the k significant bits again, and the algorithm may only use its storage of size m. However, until the algorithm finds the next fitness level we allow it to keep track of all the search points that it visits on the current fitness level. In this way we create a class of algorithm for which Yao's principle allows us to restrict to deterministic algorithms. The details are spelled out in Section 3.3.

The algorithm may be lucky and skip a fitness level, be-

cause the $(k+1)$-st significant bit in its search point is correct. However, this only happens with probability $1/2$. Otherwise, the algorithm can only carry over m bits of information to the next level, cf. Lemma 7 for a precise statement. Crucially, if $m = \delta n$ for some small $\delta > 0$, then this information is not enough to encode the positions of the k most significant bits, which would require $\log \binom{n}{k} = \log \binom{n}{m} \approx m(1 + \log(1/\delta))$ bits. One strategy of the algorithm might be to store as many of the insignificant bit positions as possible, so that it can test quickly whether one of these candidates is the next significant bit. However, whenever the algorithm wants to be certain (or "rather certain") that a specific bit b is insignificant, then this decreases the available information about the remaining bits. This strategy might pay off if b is the next significant bit, because then the algorithm reaches a new fitness level immediately. However, with a too high probability b is not the next significant bit, and the algorithm is left with an even worse situation.

The key to the proof lies in the exact definition of the class $\mathcal{A}_{\mathcal{M}}$, and in a precise way to capture the rather vague notation of information used above. We start by defining $\mathcal{A}_{\mathcal{M}}$ in Section 3.3. In Section 3.4 we show that by restricting to one-bit flips, we extend the runtime of an algorithm on the k-th fitness level by at most $m = n - k$. In Section 3.5 we first give a precise definition of what we mean by information, and we define the quantity $\Phi(k, m, B)$ to be the minimum expected number of queries that an algorithm in $\mathcal{A}_{\mathcal{M}}$ using one-bit flips needs to advance a fitness level, if there are k significant and m insignificant bits and if the available information is B. In Lemma 9 we then give a rather straight-forward recursion for the function $\Phi(k, m, B)$. Once the recursion is established, it is purely a matter of (somewhat tedious) algebra to derive the lower bound $\Phi(k, m, B) \geq \varepsilon(k + m)(1 - (\log B)/(2m))$ in Lemma 10. Since the starting amount of information is $B \approx 2^m$, each algorithm using one-bits flips needs to spend expected time $\Phi(k, m, 2^m) \approx \varepsilon(k + m)/2$ on the corresponding fitness level (provided that it visits this level at all). Since using multi-bit flips can save us at most m queries, a general algorithm in $\mathcal{A}_{\mathcal{M}}$ spends at least time $\approx \varepsilon(k + m)/2 - m$ on this level, which is $\Omega(n)$ if $m \leq \delta n$ for a sufficiently small $\delta > 0$. Thus there is a linear number of fitness levels, such that the algorithm spends an expected linear time on each of them, showing the $\Omega(n^2)$ runtime. The details of this concluding argument are found in Section 3.6.

3.3 A More Generous Model

We consider the set $\mathcal{A}_{\mathcal{M}}$ of all algorithms that can be implemented in the following model $\mathcal{M}$:

Assume that the algorithm has queried some search points $x^{(1)}, \ldots, x^{(t)}$ with $\max_{i<t}\{\textsc{Lo}(x^{(i)})\} = k-1$ and $\textsc{Lo}(x^{(t)}) \geq k$. We say that the algorithm *reaches a new fitness level* with the t-th query. In addition to letting the algorithm know that the fitness value of $x^{(t)}$ is strictly larger than the previous best search point, we reveal to the algorithm the first k *significant* positions $\sigma(1), \ldots, \sigma(k)$ and the corresponding bit values $z_{\sigma(1)}, \ldots, z_{\sigma(k)}$ of the target string. Note that from this information the algorithm can in particular infer that $\textsc{Lo}(x^{(t)}) \geq k$, but it does not learn the precise function value of $x^{(t)}$. Furthermore, it is not difficult to see that the information revealed to the algorithm contains everything about the unknown instance (z, σ) that the algorithm could have collect so far (and typically it reveals much more information about the target instance than the information currently present to the algorithm). We now allow the algorithm to "revise" its choice of the *insignificant* $m := n - k$ bits of $x^{(t)}$. That is, the algorithm may opt to change the entries in the positions $[n] \setminus \{\sigma(1), \ldots, \sigma(k)\}$, thus creating a new search point $\tilde{x}^{(t)}$.

By construction, the fitness of $\tilde{x}^{(t)}$ is at least k. It is possibly strictly greater than k in which case the algorithm may again revise the entries of the insignificant bits, now based on the first $k+1$ positions. This process continues until the fitness of the revised search point equals the number of significant bit positions that the algorithm has seen when creating it. For ease of notation, let us assume that $f(\tilde{x}^{(t)}) = k$. For clarity we emphasize that the algorithm never learns about the exact fitness value of its original choice $x^{(t)}$.

Starting from $y^{(0)} := \tilde{x}^{(t)}$, until it reaches a new fitness level $> k$ we allow the algorithm to remember all queries $y^{(1)}, \ldots, y^{(s)}$, and for each of them whether $f(y^{(i)})$ is smaller or whether it is equal to $f(y^{(0)})$ (if $f(y^{(i)})$ is larger than $f(y^{(0)})$, a new fitness level is reached). Moreover, we allow it to remember the value k. The algorithm may thus choose $y^{(s+1)}$ depending on k, $y^{(0)}, \ldots y^{(s)}$, and on the information which of the $y^{(i)}$ have smaller fitness than $y^{(0)}$. Crucially, note that in this phase the algorithm does no longer have access to the positions $\sigma(1), \ldots, \sigma(k)$ of the first k significant bits, unless it has somehow encoded this information implicitly in $y^{(0)}$.

We want to bound from below the expected number of queries that are needed to reach a new fitness level, that is, the expected number T_m of queries before the algorithm queries a search point of fitness strictly larger than $k = n-m$. Note that this number may be zero if $f(\tilde{x}^{(t)}) > k$. We shall apply Yao's MiniMax Principle with the uniform distribution over the possible Lo instances (z, σ). A discussion of this tool has been given in Section 3.1. Note that, unlike for the original elitist model, in $\mathcal{A}_{\mathcal{M}}$ every randomized algorithm is a convex combination of deterministic ones, the crucial difference to the original model being that in $\mathcal{A}_{\mathcal{M}}$ the algorithms may remember the search points on the current fitness level. In particular, a reasonable deterministic algorithm never "gets stuck". We omit the formal proof.

For deterministic algorithms in $\mathcal{A}_{\mathcal{M}}$ we will show that there is a constant $\varepsilon > 0$ such that for all $1 < m \leq \varepsilon n$ and all deterministic algorithms in $\mathcal{A}_{\mathcal{M}}$ the expected number of queries that the algorithm spends on fitness level $k = n - m$ is at least εn. This yields the desired $\Omega(n^2)$ lower bound. By the following lemma, this bound also holds for all elitist $(1 + 1)$ algorithms.

LEMMA 5. *Every elitist $(1 + 1)$ algorithm is also in $\mathcal{A}_{\mathcal{M}}$.*

PROOF. This follows rather trivially from the definition of $\mathcal{A}_{\mathcal{M}}$. An elitist $(1 + 1)$ algorithm can be simulated by an algorithm in $\mathcal{A}_{\mathcal{M}}$ by choosing $\tilde{x}^{(t)} := x^{(t)}$, and by ignoring all information except for the current search point. Note that for a (1+1) elitist algorithm it suffices that the oracle tells it which of the two search points it compares is the better one, or whether they are of equal fitness. The (1+1) elitist algorithm will thus not know more about the search points $y^{(i)}$ than whether the fitness is worse, equal, or better than the fitness of $y^{(0)}$. Thus the (1+1) elitist algorithm has always at most the information of an algorithm in $\mathcal{A}_{\mathcal{M}}$. □

3.4 It Suffices to Study Single Bit-Flips

One technical challenge in bounding the complexity of LO with respect to $\mathcal{A}_{\mathcal{M}}$ is the question of how to deal with multiple bit flips. The following lemma tells us that we do not give away much if we restrict ourselves to algorithms that only use one-bit flips. This observation simplifies the upcoming computations significantly.

LEMMA 6. *For every deterministic algorithm $A \in \mathcal{A}_{\mathcal{M}}$ there exists a deterministic algorithm $A' \in \mathcal{A}_{\mathcal{M}}$ such that the following holds. If for some instance algorithm A uses s queries to leave fitness level k, then A' uses at most $s + n - k$ queries on fitness level k. Moreover, all the search points $y'^{(1)}, \dots, y'^{(r)}$ that A' uses on this fitness level have Hamming distance one from $y^{(0)}$.*

PROOF. For reasons of space, we can only sketch the main idea. The detailed proof can be found in [10]. Let $A \in \mathcal{A}_{\mathcal{M}}$ be deterministic, and assume it queries $y^{(0)}, y^{(1)}, \dots, y^{(s)}$ on fitness level k. The algorithm A' also starts with $y^{(0)}$. For $i \in [s]$ let $\ell_i \in [0..n]$ be such that $y^{(i)}$ differs from $y^{(0)}$ in ℓ_i bits. In the i-th *step*, which may consist of several queries, A' goes through these ℓ_i bits, flipping them one by one (always starting with $y^{(0)}$) and querying the resulting strings until it finds a string that has fitness smaller than k, or until it has exhausted the ℓ_i bits. If A' creates a string that it queried in one of the previous $i-1$ steps, it does not query this string again. Note that this is possible in the model $\mathcal{A}_{\mathcal{M}}$, but would not be possible in the (1+1) elitist model. □

It is now easy to argue that with Lemma 6 at hand we need to consider only algorithms that do single bit-flips. We show below that, for m being within a suitable range of linear size, every such algorithm in expectation needs at least $m + \varepsilon n$ queries to leave fitness level $k := n - m$, provided that it started with a search point $y^{(0)}$ of fitness exactly k. Lemma 6 implies that every other algorithm (possibly doing multiple bit-flips) needs in expectation at least εn queries to leave level k, provided that it visits this level.

3.5 Evolution of the Available Information

In this section we study how the amount of information that a deterministic algorithm has about the problem instance (z, σ) changes over time, in particular while the algorithm stays on one fitness level.

Let us consider first how much information the algorithm has when entering a new fitness level $k = n - m$. Recall that we consider a problem instance (z, σ) that is taken from all LO functions uniformly at random. Recall also that in our model, i.e., model $\mathcal{A}_{\mathcal{M}}$ described in Section 3.3, we reveal to the algorithm the value k of the fitness level, the position of the k significant bits $\sigma(1), \dots, \sigma(k)$, and the values $z_{\sigma(1)}, \dots, z_{\sigma(k)}$ of the corresponding bits. In our model $\mathcal{A}_{\mathcal{M}}$ we allow the algorithm to change the entries in the m insignificant positions. Intuitively, we thus implicitly grant it m bits for storing information about the problem instance. In the following, we make this intuition precise.

Let a *k-configuration* be a pair (P, u) of a subset P of $[n]$ of size k and a $\{0,1\}$-valued string u of length k. We interpret (P, u) as the set $\{\sigma(1), \dots, \sigma(k)\}$ of the first k significant bit positions, together with the values $z_{\sigma(1)}, \dots, z_{\sigma(k)}$ of these bits in the optimum. Thus a k-configuration (together with the value of k) describes exactly the information the algorithm has before choosing the revised search point $\tilde{x}^{(t)}$, when it leaves the $(k-1)$-st fitness level. The (deterministic) algorithm maps each such possible k-configuration (P, u) to a bit string $\tilde{x}^{(t)}$ of length n. However, since there are $2^k \binom{n}{k}$ k-configurations and only 2^n bit strings, there are on average at least $2^{k-n}\binom{n}{k} = 2^{-m}\binom{k+m}{m}$ different k-configurations that are matched to the same string $\tilde{x}^{(t)}$. In the following, we will track the number C of k-configurations that are still compatible with the history of the algorithm on level k, i.e., that are compatible with the fact that $f(y^{(0)}) = k$, with the fact that the algorithm has chosen $y^{(0)}$, and with the oracle's answers to $y^{(1)}, \dots, y^{(i)}$, for some $i \geq 0$. Note that Remark 4 applies, i.e., all these k-configurations are equally likely to be the problem instance.

Assume that the algorithm starts the k-th fitness level with some string $y^{(0)}$ $(= \tilde{x}^{(t)})$. Then the compatible k-configurations (P, u) are determined by $y^{(0)}$ and the set P. This follows from the fact that by construction the entries in the k significant positions in the string $y^{(0)}$ coincide with the optimal ones $z_{\sigma(1)}, \dots, z_{\sigma(k)}$ (these bits were not changed when creating $\tilde{x}^{(t)} = y^{(0)}$). Since the algorithm has access to $y^{(0)}$ anyway, a compatible configurations can be described by the k significant positions. There are $\binom{n}{k} = \binom{k+m}{m}$ sets of size k in $[n]$. We thus define

$$B := B(k, m, C) = \frac{\binom{k+m}{m}}{C} \tag{2}$$

as the factor by which the number of possible target configurations has been reduced already. We call B the *available information* after querying $y^{(i)}$. Note that always $B \geq 1$. We remark that information is often measured in bits, which would correspond to $\log B$. However, for our purposes it is more convenient to work with B rather than $\log B$.

LEMMA 7. *With probability at least $1/2$, the available information after querying $y^{(0)}$ is at most 2^{m+1}.*

PROOF. Recall that the algorithm matches on average $2^{-m}\binom{k+m}{m}$ different k-configurations to each string $\tilde{x}^{(t)}$. Let $\mathcal{C}$ be the set of all strings $\tilde{x}^{(t)}$ which correspond to at most $2^{-m-1}\binom{k+m}{m}$ k-configurations. These strings together cover at most $2^{n-m-1}\binom{k+m}{m} = 2^{k-1}\binom{k+m}{m}$ k-configurations, i.e., at most half of all k-configurations. Since the k-configuration of the problem instance is drawn uniformly at random, with probability at least $1/2$ we draw a configuration that belongs to a string in $\{0,1\}^n \setminus \mathcal{C}$. Thus, with probability at least $1/2$ we hit a string which is mapped to a $\tilde{x}^{(t)}$ that is compatible with more than $2^{-m-1}\binom{k+m}{m}$ k-configurations. This proves the claim. □

Let us now consider how the information evolves with the queries on the k-th fitness level. Let $\mathcal{A}_{\text{one}}$ be the set of all deterministic algorithms that, starting from an n-bit string $y^{(0)}$ with $\text{LO}_{z,\sigma}(y^{(0)}) = k$, query only search points in Hamming distance one from $y^{(0)}$ until they have found a string $y^{(s)}$ with $\text{LO}_{z,\sigma}(y^{(s)}) > k$.

DEFINITION 8. *For any $k \geq 0$, $m \geq 1$, and $B \geq 1$ we define $\Phi := \Phi(k, m, B)$ to be the minimal expected number of fitness evaluations that an algorithm $A \in \mathcal{A}_{one}$ needs in order to find the next fitness level on a string with k significant and m insignificant bits if the instance (z, σ) is chosen uniformly at random among a set $\mathcal{C}$ of k-configurations. Here the minimum is taken over all algorithms $A \in \mathcal{A}_{one}$ and all*

sets $\mathcal{C}$ with $|\mathcal{C}| \geq C(k,m,B) := \binom{k+m}{m}/B$. For convenience we set $\Phi(k,0,B) := 0$ for all k and B.

Note that $\Phi(k,m,B)$ is a decreasing function in B. Before we study $\Phi(k,m,B)$ in detail, let us first compare $\Phi(k,m,B)$ with the expected time needed by *any* algorithm in $\mathcal{A}_{\mathcal{M}}$ (i.e., not necessarily based on single bit-flips) to reach a new fitness level. When an algorithm $A \in \mathcal{A}_{\mathcal{M}}$ exceeds fitness $k-1$ and chooses $\tilde{x}^{(t)}$, with probability $1/2$ it holds that $\textsc{Lo}_{z,\sigma}(\tilde{x}^{(t)}) = k$ and with probability $1/2$ the function value of $\tilde{x}^{(t)}$ is strictly larger than k. This is by the uniform choice of the problem instance. Moreover, by Lemma 7, with probability at least $1/2$ the available information is at most $B \leq 2^{m+1}$, where $m = n-k$, and this event is independent of whether $\textsc{Lo}_{z,\sigma}(\tilde{x}^{(t)}) = k$. In particular, with probability at least $1/4$, we have both $\textsc{Lo}_{z,\sigma}(\tilde{x}^{(t)}) = k$ and $B \leq 2^{m+1}$. In this case, by Lemma 6, the expected time that A spends on the k-th fitness level is at least $\Phi(k,m,2^{m+1}) - m$. Thus, our aim will be to show that $\Phi(k,m,2^{m+1}) - m = \Omega(n)$ for a linear number of values of m. We start our investigations with a recursive formula for Φ.

LEMMA 9. *Let $k \geq 0$, $m \geq 1$, and $B \geq 1$. Then $\Phi(k,m,B) \geq \frac{m+1}{2}$. Furthermore, for $p_{\min} := \max\{0, 1 - Bk/(k+m)\}$ and $p_{\max} := \min\{1, Bm/(k+m)\}$ it holds that*

$$\Phi(k,m,B) \geq 1 + \min_{p \in [p_{\min}, p_{\max}]} \left\{ p\frac{m-1}{m}\Phi\left(k, m-1, \frac{B}{p}\cdot\frac{m}{k+m}\right) + (1-p)\Phi\left(k-1, m, \frac{B}{1-p}\cdot\frac{k}{k+m}\right)\right\}, \quad (3)$$

where we use the convention that for $p=0$ ($p=1$) the first (second) summand of the minimum evaluates to zero.

PROOF. For the first formula, simply observe that even if the algorithm knows the configuration exactly, it still needs to test the m insignificant bits one by one (by definition of $\mathcal{A}_{\text{one}}$) until it finds the next significant one, i.e., until the fitness improves. Recalling that the position of the next significant bit is uniformly among the insignificant ones, the expected number of steps that it takes the algorithm to find it is $(m+1)/2$.

To verify (3), let $A \in \mathcal{A}_{\text{one}}$, and assume that the set $\mathcal{C}$ of configuration compatible with the algorithm's choice of $y^{(0)}$ satisfies $|\mathcal{C}| \geq \binom{k+m}{m}/B$. We need to show that for each such A and $\mathcal{C}$ the expected number of remaining fitness evaluations to find the next fitness level is at least the right hand side of (3). Assume further that in its next query, A flips the bit b_i, yielding a search point $y^{(1)}$, and let $p \in [0,1]$ be such that exactly $p|\mathcal{C}|$ configurations are compatible with the event $f(y^{(1)}) \geq f(y^{(0)})$. Since all configurations are equally likely, p is also the probability that $f(y^{(1)}) \geq f(y^{(0)})$. Moreover, the number of such configurations is at most $\binom{k+m-1}{m-1}$, since b_i has to be one of the insignificant bit positions. This shows that p is bounded by the inequality $p|\mathcal{C}| \leq \binom{k+m-1}{m-1}$. Using $|\mathcal{C}| \geq \binom{k+m}{m}/B$ this implies $p \leq Bm/(k+m)$. Similarly, $(1-p)|\mathcal{C}| \leq \binom{k+m-1}{m}$, which implies $p \geq 1 - Bk/(k+m)$.

Assume that the event $f(y^{(1)}) \geq f(y^{(0)})$ happens. Then with probability $1/m$ the algorithm leaves the k-th fitness level (since all m insignificant bits have the same probability of being the next significant bit). Otherwise, that is, with probability $(m-1)/m$, the algorithm learns that b_i is not the next significant bit, and it will not query b_i again on this level. Therefore, we may just exclude it from our considerations, and replace m by $m-1$. Since the number of remaining compatible configurations is $p|\mathcal{C}|$, we need to find B_{new} such that $\binom{k+m-1}{m-1}/B_{\text{new}} \leq p|\mathcal{C}|$. Since $p|\mathcal{C}| \geq p\binom{k+m}{m}/B$, we may choose B_{new} to satisfy $\binom{k+m-1}{m-1}/B_{\text{new}} = p\binom{k+m}{m}/B$, or equivalently $B_{\text{new}} = (B/p)\cdot(m/(k+m))$. So if $f(y^{(1)}) \geq f(y^{(0)})$ then the algorithm needs at least an expected additional time of $\frac{m-1}{m}\Phi(k,m-1,\frac{B}{p}\cdot\frac{m}{k+m})$.

Now we consider the case $f(y^{(1)}) < f(y^{(0)})$, which happens with probability $1-p$. Then the algorithm learns that b_i is significant, and it will not query b_i again on this level. Thus we can exclude it from our considerations and replace k by $k-1$. Similar as before, the number of compatible configurations drops to $(1-p)|\mathcal{C}|$, so we need to find B_{new} such that $\binom{k+m-1}{m}/B_{\text{new}} \leq (1-p)|\mathcal{C}|$. We may choose B_{new} according to the equation $\binom{k+m-1}{m}/B_{\text{new}} = (1-p)\binom{k+m}{m}/B$ and obtain $B_{\text{new}} = (B/(1-p))\cdot(k/(k+m))$. So if $f(y^{(1)}) < f(y^{(0)})$ then the algorithm needs at least an expected additional time of $\Phi(k-1,m,B/(1-p)\cdot k/(k+m))$. This proves (3). □

We use Lemma 9 to show the following lower bound for $\Phi(k,m,B)$. Once this bound is proven, we have everything together to prove the claimed $\Omega(n^2)$ bound for the (1+1) elitist black-box complexity of LO.

LEMMA 10. *There exists a constant $\varepsilon > 0$ such that for all $k \geq 0, m \geq 1$ and $B \geq 1$,*

$$\Phi(k,m,B) \geq \varepsilon(k+m)\left(1 - \frac{\log B}{2m}\right). \quad (4)$$

PROOF. We use induction on $k+m$. First we show the statement for the case $m=1$ and arbitrary k. If $m=1$ and $B \geq 4$, the lower bound is at most zero, and thus trivial. If $m=1$ and $B<4$, by (2), the number of compatible configurations is at least $(k+1)/B > (k+1)/4$, and in each of these configurations there is exactly one insignificant bit. Since these bits are all different from each other, there are at least $(k+1)/4$ positions at which the insignificant bit might be, and by Remark 4 all these positions are equally likely. Since the algorithm is by definition only allowed to make one-bit flips, it needs in expectation at least $(k+1)/8$ steps. Thus, the statement is satisfied for all $\varepsilon \leq 1/8$.

The inductive step follows from the recursive formula in Lemma 9. Some extensive algebraic manipulations are required, but no further knowledge about the process. We omit the details due to space restriction. □

3.6 Putting Everything Together

THEOREM 11. *With ε as in Lemma 10 and k satisfying $1 < n-k \leq \varepsilon n/8$, every algorithm $A \in \mathcal{A}_{\mathcal{M}}$ spends in expectation at least $\varepsilon n/32$ function evaluations on level k, and this lower bound holds independently of the time spent on previous levels. In particular, every algorithm in $\mathcal{A}_{\mathcal{M}}$ (and thus, every elitist (1+1) black-box algorithm) needs at least time $\Omega(n^2)$ in expectation and with high probability.*

PROOF. We have already argued above after Definition 8 that each algorithm $A \in \mathcal{A}_{\mathcal{M}}$ spends in expectation at least time $(\Phi(k,m,2^{m+1}) - m)/4$ on fitness level $k = n-m$, since with probability $1/2$ the algorithm does not skip the level,

with probability 1/2 the available information is at most 2^{m+1} (Lemma 7), and conditioned on both these events, by Lemma 6, A spends at least expected time $\Phi(k, m, 2^{m+1}) - m$ on the k-th fitness level.

The lower bound follows immediately from Lemma 10, which for k and m satisfying $1 < m = n - k \leq \varepsilon n/8$ yields $\Phi(k, m, 2^{m+1}) - m \geq \varepsilon n \left(1 - \frac{m+1}{2m}\right) - m \geq \frac{\varepsilon n}{4} - \frac{\varepsilon n}{8} = \frac{\varepsilon n}{8}$. Note that this lower bound holds independently of the time spent on other fitness levels because in the model $\mathcal{M}$ every algorithm which enters the k-th fitness level is in exactly the same state, i.e., it has access to exactly the same information, independent of its history. Since the lower bound holds for all algorithms in $\mathcal{A}_{\mathcal{M}}$, it still holds if we condition on the history of the algorithm, or specifically on the time spent on previous fitness levels. This immediately implies the statement on the expectation. The "with high probability"-statement follows by similar arguments, which we omit. □

REMARK 12. *Theorem 11 can be strengthened in the following way. Assume that an elitist* $(1 + 1)$ *algorithm has an additional memory of size* $\varepsilon' n$ *which it may use without restrictions. Then if* ε' *is sufficiently small, the algorithm still has expected runtime* $\Omega(n^2)$*. The proof is identical to the proof of Theorem 11, except that we increase the available information* B *by a factor* $2^{\varepsilon' n}$*. E.g., for* $\varepsilon' = \varepsilon/16$ *the algorithm still spends an expected linear time on all levels with* $\varepsilon n/12 \leq m \leq \varepsilon n/8$*.*

On the other hand, if the algorithm has in addition $n + O(\log n)$ *bits of memory that it may use without restriction (which is only a constant factor more), then it is not hard to achieve a runtime of* $O(n \log n)$*. In particular, it is possible to store all* k *significant bits in* n *bits of the additional memory, and the remaining* $O(\log n)$ *bits may serve as a counter. Then, whenever the fitness increases, the algorithm performs* $O(\log n)$ *steps to determine (one of) the position(s) which was responsible for the improvement. We omit the details.*

4. CONCLUSIONS AND OUTLOOK

We have shown that the (1+1) elitist black-box complexity of LO is $\Theta(n^2)$. This is in contrast to the situation for the ONEMAX function, where an enforced elitist behavior does not substantially harm the runtime [9]. Given the much smaller complexity of LO in many other models, this sheds some light on the cost of elitism. In fact, our proof suggests that the reason for the large complexity is rather the memory restriction than the selection strategy. We thus conjecture that the lower bound in Theorem 1 holds already for (1+1) memory-restricted algorithms, but we do not see at the moment a feasible proof for this claim. In particular the generalization of Lemma 6, i.e., the statement that it suffices to consider one-bit flips, seems tricky.

Our methods are adapted to the LO problem, and it is probably non-trivial to transfer them to other problems. Nevertheless, we are optimistic that similar approaches can work both for other black-box complexities and other function classes.

Finally, in light of [3] we are interested to use the insights from our investigations for the design of new search heuristics.

Acknowledgments. This research benefited from the support of the "FMJH Program Gaspard Monge in optimization and operation research", and from the support to this program from Électricité de France.

5. REFERENCES

[1] P. Afshani, M. Agrawal, B. Doerr, C. Doerr, K. G. Larsen, and K. Mehlhorn. The query complexity of finding a hidden permutation. In *LNCS 8066*, pages 1–11. Springer, 2013.

[2] S. Böttcher, B. Doerr, and F. Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *PPSN'10*, pages 1–10. Springer, 2010.

[3] B. Doerr, C. Doerr, and F. Ebel. Lessons from the black-box: Fast crossover-based genetic algorithms. In *GECCO'13*, pages 781–788. ACM, 2013.

[4] B. Doerr, D. Johannsen, T. Kötzing, P. K. Lehre, M. Wagner, and C. Winzen. Faster black-box algorithms through higher arity operators. In *FOGA'11*, pages 163–172. ACM, 2011.

[5] B. Doerr and C. Winzen. Black-box complexity: Breaking the $O(n \log n)$ barrier of LeadingOnes. In *EA'11, LNCS 7401*, pages 205–216. Springer, 2012.

[6] B. Doerr and C. Winzen. Playing Mastermind with constant-size memory. *Theory of Computing Systems*, 55:658–684, 2014.

[7] B. Doerr and C. Winzen. Ranking-based black-box complexity. *Algorithmica*, 68:571–609, 2014.

[8] C. Doerr and J. Lengler. Elitist black-box models: Analyzing the impact of elitist selection on the performance of evolutionary algorithms. In *GECCO'15*, pages 839–846, 2015.

[9] C. Doerr and J. Lengler. OneMax in black-box models with several restrictions. In *GECCO'15*, pages 1431–1438. ACM, 2015.

[10] C. Doerr and J. Lengler. The (1+1) elitist black-box complexity of LeadingOnes. volume abs/1604.02355, 2016.

[11] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

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

[13] V. Ladret. Asymptotic hitting time for a simple evolutionary model of protein folding. *Journal of Applied Probability*, 42:39–51, 2005.

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

[15] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[16] H. Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In *PPSN'92*, pages 15–26. Elsevier, 1992.

[17] G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovac, 1997.

[18] D. Sudholt. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE TEC*, 17:418–435, 2013.

[19] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *FOCS'77*, pages 222–227. IEEE, 1977.