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

Algorithmica (2018) 80:1579–1603
https://doi.org/10.1007/s00453-017-0304-6

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

Carola Doerr[1] · Johannes Lengler[2]

Received: 4 September 2016 / Accepted: 7 March 2017 / Published online: 21 March 2017

**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 to 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 and Witt in Algorithmica 64:623–642, 2012), which tells us that higher arity operators or biased sampling strategies are needed when trying to beat this bound. In lack of analyzing techniques, such non-trivial lower bounds are very rare in the existing literature on black-box optimization and therefore remain 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. in Lecture notes in computer science, vol 8066, pp 1–11. Springer, New York, 2013). The $\Omega(n^2)$ lower bound does not rely on the fact that elitist black-box algorithms are not allowed to make use of absolute fitness values. In contrast, we show that even if absolute fitness values are revealed to the otherwise elitist algorithm, it cannot significantly profit from

✉ Johannes Lengler
johannes.lengler@inf.ethz.ch

[1] CNRS and Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606, 4 place Jussieu, 75005 Paris, France

[2] Institute for Theoretical Computer Science, ETH Zürich, Zürich, Switzerland

this additional information. Our result thus shows that for LeadingOnes the memory-restriction, together with the selection requirement, has a substantial impact on the best possible performance.

**Keywords** Black-box complexity · Query complexity · LeadingOnes · Elitist algorithm · Memory restriction · Truncation selection · Evolutionary algorithms

## 1 Introduction

Randomized search heuristics such as evolutionary algorithms, simulated annealing, and randomized local search algorithms are so-called *black-box optimization algorithms*. That is, unlike their *white-box* counterparts that are typically regarded in algorithmics, these search heuristics do not have (or do not exploit) access to the problem instance other than by suggesting potential solution candidates and receiving (from an oracle/the black-box) information about the quality of these *search points* such as, for example, their function values. Based on this information, the black-box optimizers update the policy from which the next search points are sampled. This process is repeated until some stopping criterion is met. In discrete optimization, the most widely regarded performance measure for such black-box optimizers is the number of oracle/black-box queries that an algorithm needs to do until it evaluates for the first time an optimal solution candidate. This number is called the *runtime* of the algorithm. Runtime analysis is today one of the most prominent sub-areas in the theory of evolutionary computation. As is classical algorithmics, runtime analysis is complemented by a complexity theory. The *black-box complexity* of a problem $\mathcal{F}$, informally speaking, measures the minimum expected number of black-box queries that are needed to solve any problem instance $f \in \mathcal{F}$, where the minimum is taken over a class of algorithms $\mathcal{A}$. This is, since the seminal paper of Droste, Jansen, and Wegener [12], the most commonly regarded complexity measure for randomized search heuristics.

The original black-box model by Droste, Jansen, and Wegener regards the whole collection of possible black-box algorithms as $\mathcal{A}$. 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 co-exist 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 model are assumed to 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, for example, the population size, the variation operators in use, or the selection rules, influence the performance of the search heuristics.

At GECCO [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 $\mu$ best-so-far sampled solutions. In the next iteration, it is allowed to use only the relative (not absolute) function values of these $\mu$ points to create $\lambda$ new search points. A $(\mu + \lambda)$ elitist black-box algorithm is thus in particular a memory-restricted and ranking-based one in the sense of [6,7,12], respectively. In addition, it has to employ so-called *truncation selection* as replacement rule; that is, in each iteration, from the $\mu + \lambda$ parent and offspring search points only the $\mu$ best ones "survive", meaning that they form the parent population of the next iteration. Truncation selection is a very commonly applied selection rule in evolutionary computation. Note that this selection requirement implies in particular that an elitist algorithm has no influence on which of the $\mu + \lambda$ search points to keep in the population (only in case of ties we allow the algorithm to break these arbitrarily). This is different from all other existing black-box models where selection is not restricted and which therefore include algorithms eventually preferring search points that are not as good as the current-best ones.

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 the OneMax function class is of only linear order even for the most restrictive (1+1) setting. The class OneMax contains all functions $f : \{0,1\}^n \to \mathbb{R}$ with fitness landscape isomorphic to that of Om : $\{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^n x_i$.

Here in this work, we regard another classical problem, the class of all LeadingOnes functions; i.e., all functions that are isomorphic to the function Lo : $\{0,1\}^n \to \{0,1,\ldots,n\}, x \mapsto \max\{i \mid \forall j \leq i : x_j = 1\}$ which assigns to each string the length of the longest initial segment of ones (cf. Sect. 1.1 for some background on this function). We show that the $(1+1)$ elitist black-box complexity of this function class is $\Theta(n^2)$. It matches the performance of typical randomized search heuristics such as the $(1+1)$ Evolutionary Algorithm (EA) and Randomized Local Search. To prove our result, we develop tools for quantifying the amount of information that an elitist algorithm can collect about the problem instance.

An extended abstract of this paper has appeared at the 2016 edition of the ACM conference on Genetic and Evolutionary Computation (GECCO'16) [10]. The extended abstract does not include the formal proof of the main statement and only summarizes the proof approach. Moreover, the present version contains some additional discussion of the implications of our analysis (e.g., Remarks 2 and 3 in Sect. 3.6).

### 1.1 The LeadingOnes Problem

LeadingOnes (Lo for short) is among the best-studied 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 formally proven a bit later in [11]. Exact expressions for the expected runtime of the $(1+1)$ EA on Lo have been shown in [2,13,18].

The $(1+1)$ EA (with mutation probability $1/n$, the setting most commonly regarded in evolutionary computation) works in the following way. It starts with an initial solution that is drawn uniformly at random from $\{0,1\}^n$. In each iteration the search point ("individual") $x$ currently stored in the memory is *mutated* by flipping each bit independently with probability $1/n$. The so-created "offspring" $y$ replaces $x$ if and only if it is as good as $x$; that is, in the case of a maximization problem $x$ is replaced by $y$ if and only if $f(y) \geq f(x)$.

It is easily seen that the $(1+1)$ EA qualifies as a $(1+1)$ elitist algorithm in the sense of [8]. Furthermore, its runtime is identical on any of the above-mentioned generalized LO functions regarded here in this work (see Sect. 2 for precise definitions). The $O(n^2)$ runtime bound for the $(1+1)$ EA shown in [11] therefore implies that the $(1+1)$ elitist black-box complexity of LO is of order at most $n^2$. Our main result, Theorem 1, shows that this 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 before, 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), a search heuristic which differs from the $(1+1)$ EA only in the mutation step. In RLS, exactly one bit—chosen uniformly at random among the $n$ positions—of the current best solution $x$ is changed in each iteration. Our result thus implies 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. The most relevant ones 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 et al. [1], the $O(n \log n)$ bound for the binary unbiased black-box complexity by Doerr et al. [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

**Table 1** Known black-box complexities of LEADINGONES

| 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] |
| Unbiased, arity 2 | $\Omega(n \log\log n)$ | | $O(n \log n)$ | [4] |
| Unbiased, arity $\geq 3$ | $\Omega(n \log\log n)$ | | $O(n \log(n)/\log\log n)$ | [5] |
| Ranking-based unbiased, arity $\geq 3$ | $\Omega(n \log\log n)$ | | $O(n \log(n)/\log\log n)$ | [5] |
| Elitist, arity 1 | $\Omega(n^2)$ | (Here) | $O(n^2)$ | [11] |

The lower bounds for higher arities follow from the lower bound in the unrestricted model. Our contribution is the $\Omega(n^2)$ lower bound for the unrestricted model. We recall that both the $(1+1)$ EA [17] as well as RLS have an expected runtime of $O(n^2)$ on LEADINGONES problem (cf. [1] for an implementation of the binary search strategy). This shows that indeed (some of) the restrictions of the $(1+1)$ elitist black-box model are needed to achieve the quadratic lower bound.

Our result is not the first lower bound of quadratic order for the LO problem. 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 the identification of 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 remind 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 information 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 extend the class of elitist algorithms such that the resulting model allows for the application of the information-theoretic tool. This is a delicate task since extending the class of algorithms may decrease the black-box complexity. So we need to devise an extension that does not decrease the black-box complexity by too much.

## 2 Formal Definitions

### 2.1 Black-Box Complexity

A $(\mu + \lambda)$ elitist black-box algorithm is a (possibly randomized) algorithm that can be described by the framework of Algorithm 1. That is, the algorithm tries to optimize a given pseudo-Boolean function $f : \{0,1\}^n \rightarrow \mathbb{R}$, the *fitness function*, as follows. It maintains a multiset $X$ of $\mu$ search points, which is called the *population*. The population is initialized by sampling iteratively $\mu$ 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 $\lambda$ 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 $\mu$ 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-Boolean 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 $A$ for 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 for the class $\mathcal{F}$.

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

```
Initialization:
  $X \leftarrow \emptyset$;
  for $i = 1, \ldots, \mu$ 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 $t = 1, 2, 3, \ldots$ 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 $(y^{(1)}, \ldots, y^{(\lambda)})$ according to $p^{(t)}$;
    Set $X \leftarrow X \cup \{y^{(1)}, \ldots, y^{(\lambda)}\}$;
    for $i = 1, \ldots, \lambda$ do Select $x \in \arg\min X$ and update $X \leftarrow X \setminus \{x\}$
```

*Remark 1* 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 minimum 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 2 below (page 22), which is a more technical version of Theorem 1.

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

### 2.2 LeadingOnes

As mentioned in Sect. 1.1 the original LO function assigns to each bit string $x$ the length of the longest prefix of $x$ that consists completely of ones. Formally, $\text{LO} : \{0,1\}^n \to \mathbb{R}, x \mapsto \max\{i \in \{0,1,\ldots,n\} \mid \forall j \le i : x_j = 1\}$. This function is generalized to a permutation- and bit-invariant version in the following way. Let $S_n$ denote the set of permutations of $[n] := \{1,\ldots,n\}$, and let $[0..n] := \{0,1,\ldots,n\}$. For $z \in \{0,1\}^n$ and $\sigma \in S_n$, we consider the function

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

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 $\sigma$. For $i < j$ we say that $\sigma(i)$ is *more significant* than $\sigma(j)$. In particular, $\sigma(1),\ldots,\sigma(k)$ are the $k$ most significant bits, and $\sigma(n),\ldots,\sigma(n-k+1)$ 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.

### 2.3 Other Notation and Definitions

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 $\text{LO}_{z,\sigma}$ of course 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 algorithms have strictly more power than in the original elitist model (Sects. 3.3–3.5).

We then show the $\Omega(n^2)$ lower bound for this more generous model in Sect. 3.6. Before we start with the technical details of the alternative model, we explain in Sect. 3.1 our motivation for introducing it. We present a high-level overview of the proof in Sect. 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 strictly 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 $\varepsilon$ 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 1** (Yao's Principle [15,19]) *Consider 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 $p$, $A_q$ denotes 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 1 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} = $ Lo, let $p$ be the uniform distribution over the inputs $\mathcal{I}$, and let $I_p$ be defined as in Lemma 1. Then any deterministic algorithm $A$ has a positive probability of getting stuck during the optimization of $I_p$. 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, so it 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 Sect. 1.1).

To resolve this apparent discrepancy, note that Lemma 1 makes a statement about algorithms that can be formally seen as probability distributions over deterministic ones (such algorithms are thus *convex combinations* of deterministic ones). For typical classes of algorithms this describes exactly the class of all randomized algorithms, 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 probability distributions over 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 probability distribution over 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.

**Observation 1** *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 $q_1, \dots, q_\ell$ of $A$ reduce the number of possible problem instances to some set $\mathcal{C}$. Then* $\Pr[i=c \mid q_1,\dots,q_\ell] = \Pr[i=c \mid i \in \mathcal{C}] = 1/|\mathcal{C}|$ *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 algorithms for which Yao's principle allows us to restrict to deterministic algorithms. The details are spelled out in Sect. 3.3.

The algorithm may be lucky and skip a fitness level, because 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 5 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 Sect. 3.3. In Sect. 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 Sect. 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 6 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 7. 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 Sect. 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 $\max_{i<t}\{\text{Lo}(x^{(i)})\} = k-1$ and $\text{Lo}(x^{(t)}) \geq k$. We then 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 $\text{Lo}(x^{(t)}) \geq k$, 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 $(z, \sigma)$ 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 $[n]\backslash\{\sigma(1), \ldots, \sigma(k)\}$, thus creating a new search point $\tilde{x}^{(t)}$. This revision does not cost anything in terms of runtime.

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 $(z, \sigma)$. A discussion of this tool has been given in Sect. 3.1. We will show next that, unlike for the original elitist model, in $\mathcal{A}_{\mathcal{M}}$ every (reasonable) randomized algorithm is a probability distribution over deterministic ones, the crucial difference to the original elitist model being that in $\mathcal{A}_{\mathcal{M}}$ the algorithms may remember the search points of the current fitness level. In particular, a reasonable deterministic algorithm therefore never gets stuck. Formally, we get the following statement.

**Lemma 2** *Assume that $A \in \mathcal{A}_{\mathcal{M}}$ is a randomized algorithm that never samples a search point twice on the same fitness level. Then $A$ is a probability distributions over deterministic algorithms in $\mathcal{A}_{\mathcal{M}}$.*

Before we prove Lemma 2, we remind that a (deterministic or randomized) algorithm $A \in \mathcal{A}_{\mathcal{M}}$ is allowed to remember all previous queries on the same fitness level, so it will know whether it is about to sample the same search point twice on the same fitness level. Its runtime can only increase by sampling the same search point twice, so for proving lower bounds we can restrict to algorithms $A$ as in the lemma. The reason why we actually need this restriction is that there is only a finite number of possible executions (i.e., of sequences of queries) of $A$ if we forbid multiple sampling. Otherwise the number of possible executions would be uncountable, which causes some problems that we want to avoid.

*Proof of Lemma 2* Assume we have a randomized algorithm $A \in \mathcal{A}_{\mathcal{M}}$. All random decisions of $A$ can be based on a sequence of random coin flips.[1]

For simplicity, we will first argue that for every $N > 0$, the algorithm $A$ in the phase before the $N$-th coin flip can be obtained as a probability distribution over deterministic algorithms. In other words, if we only consider the execution of $A$ up to the $N$-th coin flip, then the randomized algorithm is just a randomly chosen deterministic one. In the following paragraph we will thus only regard the execution of $A$ until it requests the $N$-th coin flip.

We can emulate $A$ as follows. At the very beginning (before actually running $A$), we flip $n \cdot N$ coins, which we denote by $F_{i,j}$, where $1 \leq i \leq n$ runs through all fitness levels, and $1 \leq j \leq N$. We now run $A$ as follows. When it is on the $i$-th fitness level and it asks for the $j$-th random bit on this fitness level, then we feed it the bit $F_{i,j}$. Note that although there are $n \cdot N$ random bits available in total, $A$ will use at most $N$ of them (in the part of the execution that we consider).

Now that the $F_{i,j}$ are fixed, the algorithm $A$ is just a deterministic algorithm. The subtle point here is that we can also realize it as a deterministic algorithm $A'$ in the class $\mathcal{A}_M$. To understand why this might be a problem, note that we can describe any deterministic black-box algorithm (restricted or not) as a function $\alpha$ from the set of possible memory states into the set of search points, since in every state the algorithm selects one search point to be evaluated next. Conversely, every such function defines a deterministic algorithm. So how do we emulate $A$ by an algorithm $A'$ defined by a function $\alpha$? If $A$ samples $s^{(i)}$ in step $i$, and if the memory state of $A'$ before step $i$ is $S_i$, then we only need to ensure that $\alpha(S_i) = s^{(i)}$ for all $i \geq 0$. This may serve as a definition if we can argue that all the states $S_i$ are different. Note that here lies the crucial difference to the elitist model, in which the state space is so small that $A'$ may come to the same state twice.

So we need to argue that the algorithm $A'$ is never in the same state twice. This is because whenever $A'$ is in a specific state (i.e, it has a specific current search point with specified fitness, and it has some content in the memory that is allowed for the class $\mathcal{A}_M$), then the available information suffices to determine the number of queries

[1] We remark without proof that the same argument also works in a different model of computation where the algorithm is allowed to generate random real number in the interval [0, 1].

in this fitness level. In particular, regardless of what $A'$ has done in previous steps, the current state is different from all previous states. Therefore, once the $F_{i,j}$ are fixed we can properly define a function $\alpha$ and thus select a deterministic algorithm $A'$ from $\mathcal{A}_{\mathcal{M}}$ that behaves like $A$ for the first $N$ coin flips.

For unlimited $N$, we use the same argument as before, only that we flip an infinite sequence of coins for each fitness level. Still, every outcome $(F_{i,j})_{1 \le i \le n, j \ge 1}$ corresponds to exactly one deterministic algorithm, which also does not query the same search point twice on the same fitness level. Hence, every such deterministic algorithm is chosen with some probability, where the probabilities add up to 1. The argument now runs as before. □

We show that there is a constant $\varepsilon > 0$ such that for all $1 < m \le \varepsilon n$ 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 $\varepsilon n$. This yields the desired $\Omega(n^2)$ lower bound. By the following lemma, this bound also holds for all elitist $(1+1)$ algorithms.

**Lemma 3** *Every elitist* $(1 + 1)$ *algorithm is also in* $\mathcal{A}_{\mathcal{M}}$. *This holds even if the algorithm receives exact fitness information, i.e., every* $(1 + 1)$ *algorithm that uses truncation selection is also in* $\mathcal{A}_{\mathcal{M}}$.

*Proof* This follows rather trivially from the definition of $\mathcal{A}_{\mathcal{M}}$. A $(1 + 1)$ elitist 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 that an algorithm in $\mathcal{A}_{\mathcal{M}}$ has.

In the same way, the algorithm in $\mathcal{A}_{\mathcal{M}}$ can simulate a $(1+1)$ algorithm $A$ that uses truncation selection. Recall that an algorithm in $\mathcal{A}_{\mathcal{M}}$ is always aware of the current fitness level, but it does not generally learn the fitness of the offspring. There are three cases: if the offspring has the same fitness as the parent then the algorithm in $\mathcal{A}_{\mathcal{M}}$ can trivially deduce the fitness of the offspring. If the offspring has strictly larger fitness, then the algorithm in $\mathcal{A}_{\mathcal{M}}$ eventually learns the fitness of the offspring since it waives all options to revise the offspring (i.e., since it always chooses $\tilde{x}^{(t)} := x^{(t)}$). Finally, if the offspring has strictly smaller fitness than the current search point, then the algorithm in $\mathcal{A}_{\mathcal{M}}$ does not learn the fitness of the offspring, but it can still predict that $A$ will reject the offspring. Thus, in all cases the algorithm in $\mathcal{A}_{\mathcal{M}}$ can predict the next state of $A$, i.e., it can simulate the algorithm $A$. □

### 3.4 It Suffices to Study One-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 4** *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 1 from $y^{(0)}$.*

*Proof* 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 $\ell_i$ bits. In the $i$-th *step*, which may consist of several queries, $A'$ goes through these $\ell_i$ bits, flipping them one by one and querying the resulting strings until it finds a string that has fitness smaller than $k$, or until it has exhausted the $\ell_i$ bits. Note that $A'$ does *not* apply the one-bit flips iteratively, but rather each of the $\ell_i$ strings is created by a one-bit flip applied to $y^{(0)}$. 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.

We need to show two things: (i) after each step, $A'$ has at least as much information as $A$ has, so that it knows which query $y^{(i+1)}$ algorithm $A$ will choose next, and (ii) $A'$ uses at most $s + m$ queries, where $m = n - k$. In fact, we show that each of the $m$ insignificant bits causes at most one additional query for $A'$. Assume first that $A'$ exhausts the $\ell_i$ bits in step $i$, i.e., that all the $\ell_i$ corresponding strings have fitness at least $k$. In this case $A$ learns that the fitness of $y^{(i)}$ is at least $k$ (and possibly that it is strictly larger than $k$), and $A'$ learns the same. Moreover, $A$ has done one query, while $A'$ has done one query for each of the at most $\ell_i$ insignificant bits that have not been evaluated before.

Next assume that in the $i$-th step $A'$ finds a string that has fitness smaller than $k$. Say it is the $\ell'$-th string that $A'$ queries in the $i$-th step. Then $A$ only learns that among the $\ell_i$ flipped bits there is at least one significant bit. $A'$ learns this, too, but it also learns specifically the position of such a bit. Algorithm $A$ uses one query, while $A'$ uses $\ell' \leq \ell_i$ queries, spending again one query for each insignificant bit that it discovers.

In both cases, algorithm $A'$ learns at least all the information that $A$ learns, and the total number of queries done by $A'$ is at most $s + m$ since it spends at most one query for each insignificant bit that it tests. □

It is now easy to argue that with Lemma 4 at hand we need to consider only algorithms that do one-bit flips. We will 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 $k$. Assume for the sake of contradiction that there was any other algorithm in $\mathcal{A}_{\mathcal{M}}$ (possibly doing multiple bit flips) which needs less than $s := \varepsilon n$ queries in expectation to leave level $k$, provided that it visits this level. Then by Lemma 4, there would also be an algorithm using one-bit flips which needs less than $s + n - k = m + \varepsilon n$ queries, which is a contradiction. Therefore, every algorithm in $\mathcal{A}_{\mathcal{M}}$ needs at least $\varepsilon n$ queries in expectation to leave level $k$. (The formal proof can be found in Sect. 3.6.) Hence, it suffices to analyze algorithms that do one-bit flips.

### 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 $(z, \sigma)$ 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 $(z, \sigma)$ 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 Sect. 3.3, we reveal to the algorithm the value $k$ of the fitness level, the position of the $k$ significant bits $\sigma(1), \ldots, \sigma(k)$, and the values $z_{\sigma(1)}, \ldots, z_{\sigma(k)}$ 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 set $P \subseteq [n]$ of size $k$ and a bit string $u \in \{0, 1\}^k$ of length $k$. We interpret $(P, u)$ as the set $\{\sigma(1), \ldots, \sigma(k)\}$ of the first $k$ significant bit positions, together with the values $z_{\sigma(1)}, \ldots, z_{\sigma(k)}$ 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 $f(y^{(0)}) = k$, with the fact that the algorithm has chosen $y^{(0)}$, and with the oracle's answers to $y^{(1)}, \ldots, y^{(i)}$, for some $i \geq 0$. Note that $C$ will in general be larger than one, since the algorithm does not have perfect information about the positions and bit values of the first $k$ significant bits—it does briefly know this information while choosing $y^{(0)}$, but it forgets the information afterwards. However, the algorithm is able to limit the size of $C$ by reverse-engineering its configuration-to-query mapping since it *does* still know the string $y^{(0)}$. Also note that Observation 1 applies, i.e., all $k$-configurations contributing to $C$ are equally likely to occur as the problem instance is drawn uniformly at random.

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 $z_{\sigma(1)}, \ldots, z_{\sigma(k)}$ (these bits were not changed when creating $\tilde{x}^{(t)} = y^{(0)}$). Since the algorithm has access to $y^{(0)}$ anyway, a compatible configuration can be described by the $k$ significant positions. There are $\binom{n}{k} = \binom{k+m}{m}$ sets of size $k$ in $[n]$, so it is convenient to normalize by this factor. We thus define

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

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 5** *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 $\{0,1\}^n \backslash \mathcal{C}$. 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 $\text{Lo}_{z,\sigma}(y^{(0)}) = k$, use only one-bit flips of $y^{(0)}$ until they have found a string $y^{(s)}$ with $\text{Lo}_{z,\sigma}(y^{(s)}) > k$.

**Definition 1** For any $k \geq 0$, $m \geq 1$, and $B \geq 1$, we define $\Phi := \Phi(k, m, B)$ to be the minimum expected number of fitness evaluations that an algorithm $A \in \mathcal{A}_{\text{one}}$ needs in order to find the next fitness level on a string with $k$ significant and $m$ insignificant bits if the instance $(z, \sigma)$ is chosen uniformly at random among all instances whose $k$-configuration is contained in a set $\mathcal{C}$. Here the minimum is taken over all algorithms $A \in \mathcal{A}_{\text{one}}$ and all sets $\mathcal{C}$ of $k$-configurations 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 $\text{Lo}_{z,\sigma}(\tilde{x}^{(t)}) = k$ 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 5, 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 $\text{Lo}_{z,\sigma}(\tilde{x}^{(t)}) = k$. In particular, with probability at least $1/4$, we have both $\text{Lo}_{z,\sigma}(\tilde{x}^{(t)}) = k$ and $B \leq 2^{m+1}$. In this case, by Lemma 4, 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 $\Phi$.

**Lemma 6** *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*

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

*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 at random 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 configurations 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 $B_{\text{new}}$ 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 $B_{\text{new}}$ 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 $\frac{m-1}{m}\Phi\left(k, m-1, \frac{B}{p}\cdot\frac{m}{k+m}\right)$.

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 $B_{\text{new}}$ such that $\binom{k+m-1}{m}/B_{\text{new}} \leq (1-p)|\mathcal{C}|$. We may choose $B_{\text{new}}$ 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 6 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 7** *There exists a constant $\varepsilon > 0$ such that for all $k \geq 0$, $m \geq 1$ and $B \geq 1$,*

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

*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 Observation 1 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 $\varepsilon \leq 1/8$.

Now we come to the inductive step, where by the paragraph above we may assume that $m \geq 2$. Furthermore, we may also assume that $\log B < 2m$ since the statement is trivial otherwise. By Lemma 6 we have

$$\begin{aligned}\Phi(k, m, B) \geq 1 + \min_{p\in[p_{\min}, p_{\max}]} \Bigg\{ & p\frac{m-1}{m}\Phi\left(k, m-1, \frac{B}{p}\cdot\frac{m}{k+m}\right) \\ & + (1-p)\cdot\Phi\left(k-1, m, \frac{B}{1-p}\cdot\frac{k}{k+m}\right)\Bigg\}.\end{aligned} \tag{5}$$

By the induction hypothesis we may thus conclude from inequality (5) that

$$\begin{aligned}\Phi(k, m, B) &\geq 1 + \min_{p\in[p_{\min}, p_{\max}]} \Bigg\{ p\frac{m-1}{m}\varepsilon(k+m-1)\left(1 - \frac{\log\left(\frac{B}{p}\cdot\frac{m}{k+m}\right)}{2(m-1)}\right) \\ &\quad + (1-p)\varepsilon(k+m-1)\left(1 - \frac{\log\left(\frac{B}{1-p}\cdot\frac{k}{k+m}\right)}{2m}\right)\Bigg\} \\ &= 1 + \varepsilon(k+m)\left(1 - \frac{\log B}{2m}\right) + \min_{p\in[p_{\min}, p_{\max}]}\{-R(p)\},\end{aligned}$$

where $R(p)$ equals

$$\begin{aligned}
R(p) = \tfrac{1}{m} p\varepsilon(k+m-1)\left(1 - \frac{\log\left(\frac{B}{p}\cdot\frac{m}{k+m}\right)}{2(m-1)}\right) && =: \mathbf{X}_1 \\
+ p\varepsilon\left(1 - \frac{\log\left(\frac{B}{p}\cdot\frac{m}{k+m}\right)}{2(m-1)}\right) && =: \mathbf{A}_1 \\
+ p\varepsilon(k+m)\frac{\log\left(\frac{1}{p}\cdot\frac{m}{k+m}\right)}{2m} && =: \mathbf{Y}_1 \\
+ p\varepsilon(k+m)\frac{\log\left(\frac{1}{p}\cdot\frac{m}{k+m}\right)}{2m(m-1)} && =: \mathbf{Z} \\
+ p\varepsilon(k+m)\frac{\log B}{2m(m-1)} && =: \mathbf{X}_2 \\
+ (1-p)\varepsilon\left(1 - \frac{\log\left(\frac{B}{1-p}\cdot\frac{k}{k+m}\right)}{2m}\right) && =: \mathbf{A}_2 \\
+ (1-p)\varepsilon(k+m)\frac{\log(\frac{1}{1-p}\cdot\frac{k}{k+m})}{2m} && =: \mathbf{Y}_2
\end{aligned}$$

The above expression for $R(p)$ can be verified by elementary calculations, using only that $\log(Bx) = \log B + \log x$ for $x = 1/p \cdot m/(k+m)$ and for $x = 1/(1-p) \cdot k/(k+m)$. It thus suffices to show that

$$R(p) \le 1 \text{ for all } p \in [p_{\min}, p_{\max}]. \tag{6}$$

To this end we first observe that $\frac{B}{p} \cdot \frac{m}{k+m} \ge 1$ and $\frac{B}{1-p} \cdot \frac{k}{k+m} \ge 1$ by definition of $p_{\min}$ and $p_{\max}$. Since these expressions occur in $A_1$ and $A_2$, we get

$$A_1 + A_2 \le \varepsilon. \tag{7}$$

For the same reason, $X_1 \le p\varepsilon(k+m-1)/m$. Recalling $\log B < 2m$ and using $m-1 \ge m/2$, we thus get

$$X_1 + X_2 \le \frac{3p\varepsilon(k+m)}{m} =: X \tag{8}$$

In the following we will show that $Y_1 + Y_2 + X + Z \le 1/2$ if $\varepsilon$ is sufficiently small. Together with (7) and (8), this will complete the inductive step.

Let $p_0 := m/(k+m)$, and write $p = \gamma p_0$ for some (not necessarily constant) $\gamma \ge 0$. Note that $X = 3\gamma\varepsilon$. We first consider the case $\gamma \le 2$. In this case, $X \le 6\varepsilon$. Moreover, $Z$ is either negative (for $\gamma > 1$) or upper bounded by $-\varepsilon\gamma\log\gamma \le \varepsilon/(e\ln 2)$, since the function $-\gamma\log\gamma$ has a unique maximum $1/(e\ln 2)$ in the interval $0 < \gamma \le 1$. Either way, we can choose $\varepsilon$ so small that $X + Z \le 1/2$. In the following we will prove that $Y_1 + Y_2 \le 0$, which will settle the case $\gamma \le 2$.

To prove that $Y_1 + Y_2$ is non-positive, we may just as well consider the sign of the function

$$f(p) := \frac{2m}{(k+m)\varepsilon}(Y_1 + Y_2) = p\log\left(\frac{p_0}{p}\right) + (1-p)\log\left(\frac{1-p_0}{1-p}\right).$$

The function has derivatives

$$f'(p) = \log\left(\frac{p_0}{p}\right) - \log\left(\frac{1-p_0}{1-p}\right)$$

and

$$f''(p) = -\frac{1}{p(1-p)\ln(2)} < 0.$$

Since the second derivative is negative, the function $f$ is concave. We further observe that $f(p_0) = f'(p_0) = 0$. Thus $f$ has a unique maximum at $p = p_0$. Therefore, $f(p) \leq f(p_0) = 0$ for all $0 \leq p \leq 1$. Hence, we have proven $Y_1 + Y_2 \leq 0$, and this concludes the case $\gamma \leq 2$.

For the case $\gamma > 2$, we first show that $Y_1/2 + X + Z \leq 1/2$. Clearly we have $Z \leq 0$. For sufficiently small $\varepsilon > 0$ we obtain

$$\frac{Y_1}{2} + X = -\frac{\varepsilon\gamma\log\gamma}{4} + 3\varepsilon\gamma = \varepsilon\gamma\left(3 - \frac{\log\gamma}{4}\right) \leq \frac{1024\varepsilon}{e\ln 2} \leq \frac{1}{2},$$

where the second to last inequality can be easily checked by taking the derivative of the function and observing that is has a global maximum for $\gamma = 4096/e$.

Next we show that $Y_1/2 + Y_2 \leq 0$ for all $\gamma > 2$. Similar as before, we may consider the sign of the function

$$\tilde{f}(p) := \frac{2m}{(k+m)\varepsilon}\left(\frac{Y_1}{2} + Y_2\right) = \frac{p}{2}\log\left(\frac{p_0}{p}\right) + (1-p)\log\left(\frac{1-p_0}{1-p}\right)$$

under the additional constraint $2p_0 < p \leq 1$. Similar as above, the second derivative of $\tilde{f}$ for $0 < p < 1$ is

$$\tilde{f}''(p) = -\frac{(p+1)}{2\ln(2)p(1-p)} < 0.$$

Thus $\tilde{f}$ is concave for $0 \leq p \leq 1$. Recall that $\log(1+x) < x$ for all $x > 0$. Therefore,

$$\tilde{f}(2p_0) = -p_0\log(2) + (1-2p_0)\log\left(1 + \frac{p_0}{1-2p_0}\right) < -p_0 + p_0 = 0.$$

On the other hand, $\tilde{f}(p_0) = 0 > \tilde{f}(2p_0)$. Since $\tilde{f}$ is concave, this implies $\tilde{f}(p) < 0$ for all $2p_0 < p < 1$. This shows that $Y_1/2 + Y_2 \leq 0$ for all $\gamma > 2$.

Summarizing, we have shown that $Y_1/2 + Y_2 \leq 0$ and $Y_1/2 + X + Z \leq 1/2$ for all $\gamma > 2$, so it follows that $Y_1 + Y_2 + X + Z \leq 1/2$. Together with (7) and (8), this concludes the inductive step and the proof. □

### 3.6 Putting Everything Together

As outlined in the high-level overview, Lemma 4, 5, and 7 together imply runtime $\Omega(n^2)$ on the LO problem for any algorithm in $\mathcal{A}_{\mathcal{M}}$. More precisely, we obtain the following theorem.

**Theorem 2** *With $\varepsilon$ as in Lemma 7 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 1 that each deterministic 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 5), and conditioned on both these events, by Lemma 4, $A$ spends at least expected time $\Phi(k, m, 2^{m+1}) - m$ on the $k$-th fitness level. By Lemma 2 we may apply Yao's principle to deduce that the same bound also holds for every randomized algorithm $A \in \mathcal{A}_{\mathcal{M}}$.

The lower bound follows immediately from Lemma 7 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 that 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.

The lower bound $\Omega(n^2)$ on the expected runtime follows immediately. It remains to prove that the same bound holds with high probability. We will show that for each of the last $\varepsilon n/8$ levels, the algorithm has probability $\Omega(1)$ to spend at least linear time on this level, regardless of the behavior on previous levels. Note that this implies the statement, since by the Chernoff bound, with high probability every algorithm spends at least linear time on a linear number of levels. The argument is formal but somewhat tricky, so we elaborate on it.

We want to show that there are constants $c, p > 0$ such that for sufficiently large $n$, every algorithm in $\mathcal{A}_{\mathcal{M}}$ spends at least time $cn$ with probability at least $p$ on level $k$. Assume otherwise (for the sake of contradiction), i.e., assume that for every constant $c, p > 0$ there are arbitrarily large $n$ and algorithms $A = A(n, c, p) \in \mathcal{A}_{\mathcal{M}}$ such that $A$ spends at least time $cn$ with probability at most $p$. Then as a formal consequence[2] the

[2] For every $i$ we fix $n_i$ arbitrarily for which the statement holds when $p = c = 1/i$. Without loss of generality, we may assume that the $n_i$ are growing (by assumption, for every $i$ there are arbitrary large values for $n$ for which some algorithm $A$ spends at least time $cn$ with probability at most $p$). Then we may

same holds for $c, p = o(1)$, so there are functions $c = c(n) = o(1)$ and $p = p(n) = o(1)$ such that there are arbitrarily large $n$ and algorithms $A = A(n, c(n), p(n)) \in \mathcal{A}_{\mathcal{M}}$ such that $A$ spends at least time $cn$ with probability at most $p$. Fix such $c(n), p(n)$, and such an algorithm $A = A(n, c(n), p(n))$, and consider the algorithm $A' \in \mathcal{A}_{\mathcal{M}}$ that behaves like $A$ for the first $cn$ queries, and afterwards just does random single bit flips. Note that the latter strategy takes expected time at most $n$ to leave fitness level $k$. Therefore, $A'$ needs in expectation at most $cn + pn = o(n)$ queries to leave level $k$ (for arbitrarily large values of $n$), contradicting the fact that every algorithm in $A$ needs expected time $\Omega(n)$. This concludes the formal argument, and thus the proof. □

*Remark 2* Theorem 2 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 $\varepsilon'$ is sufficiently small, the algorithm still has expected runtime $\Omega(n^2)$.

On the other hand, if the algorithm has in addition $n + O(\log n)$ bits of memory that it may use without restriction, then it is possible to achieve a runtime of $O(n \log n)$. Hence, if the algorithm has access to $cn$ bits of unrestricted memory, then it depends on the constant $c$ whether runtime $o(n^2)$ is possible or not.

*Proof* The proof of the first statement is identical to the proof of Theorem 2, 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 $\varepsilon n/12 \leq m \leq \varepsilon n/8$.

The second statement holds because 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. The details are as follows.

We split the additional memory into a block $B_1$ of length $n$, and the remaining part $B_2$. Let $x$ denote the current search point, and let $\ell$ be the current number of one-bits in $B_1$. We maintain the invariant that every one-bit in $B_1$ is at the position of one of the $f(x)$ leading bits. Note that this implies that every such position must be correct in $x$. At start we choose $B_1$ to be the all-zero string.

We iteratively proceed as follows. If $\ell = f(x)$, then the bits in $B_1$ correspond exactly to the first $f(x)$ leading bits. In this case we flip all positions in $x$ that correspond to zero-bits in $B_1$, and this operation improves the fitness of $x$ by at least one. In the same operation, we (re-)set $B_2$ to zero. On the other hand, assume $\ell \neq f(x)$. Then our invariant implies $\ell < f(x)$. Let us call $P$ the set of all bit positions that are correct in $x$ but are not one-bits in $B_1$. Our aim is to identify one of the $f(x) - \ell$ positions in $P$. Let $P_0$ be the set of positions of zero-bits in $B_1$. In our strategy $P_0$ will serve as a set of candidate positions, which we will shrink iteratively. We create a search point $x'$ by flipping the first half of the positions $P_0$ in $x$, and query $f(x')$. If $f(x') > f(x)$ then we (have to) accept the new search point, and we reset all of $B_2$

Footnote 2 continued
choose the functions $p(n), c(n) = o(1)$ so slowly decreasing that $p(n_i), c(n_i) \geq 1/i$. For these functions $p(n), c(n)$, there are still arbitrarily large $n$ for which the statement holds, since it holds for all triples $(n_i, p(n_i), c(n_i))$.

to zero. If $f(x') < f(x)$ then we know that at least one of the positions in $P$ lies in the first half of $P_0$, so we replace $P_0$ by its first half, and encode this result by a single bit in $B_2$. Finally, if $f(x') = f(x)$ then at least one (in fact, all) of the positions in $P$ lie in the second half of $P_0$, so we replace $P_0$ by its second half, and also encode this result by a single bit in $B_2$. Repeating this step, we iteratively decrease the size of $P_0$ by a factor of 2, or find a better search point. We continue this procedure for at most $\lceil \log n \rceil$ rounds, after which we have either found a better search point, or reduced $P_0$ to a single position, which then must be a position in $P$. In the latter case, we flip the corresponding bit in $B_1$ to one, reset $B_2$ to zero, and continue.

In this way, in $\log n$ steps, we can increase either the fitness of $x$, or the number of ones in $B_1$. Since we never flip any bit in $B_1$ to zero, after $O(n \log n)$ steps, either $f(x) = n$ or $B_1$ is the all-one string. By the invariant the latter also implies $f(x) = n$, so the algorithm finds the optimum after $O(n \log n)$ steps. □

*Remark 3* The proof of Lemma 7 also allows an interesting interpretation. We have defined $B := \binom{k+m}{m}/C$, where $C$ denotes the number of compatible $k$-configurations, i.e., loosely speaking, the number of possible positions for the first $k$ significant bits which are compatible with the current state of the algorithm. Interestingly, the notion of compatible $k$-configurations is applicable to any algorithm, also algorithms with unrestricted memory. Thus, the quantity $B$ is also well-defined for any algorithm. So we can regard $B$ as a universal measure on how accurate the algorithm has encoded (in its current state) the positions of the first $k$ significant bits. Of course, for most settings the current state is just determined by the current content of the memory.

Moreover, in the proof of Lemma 7 the only restriction on the algorithm that we use is that it is restricted to one-bit flips. Thus Lemma 7 can be rephrased as follows: "*Every* algorithm using only one-bit flips starting with a search point of fitness $k$ and with information at most $B$ needs at least $\varepsilon(k+m)(1-\log B/(2m))$ queries in expectation to find a search point of larger fitness." This formulation is of little interest in general due to the technical restriction to one-bit flips. However, we believe that the structure of the statement is quite interesting, since we have shown (under the restriction to one-bit flips) that every algorithm that has not—directly or indirectly—encoded the positions of the first $k$ significant positions accurately enough has a bad performance.

We find this interpretation remarkable because it is very common to use the concept of encoding in a positive sense ("Algorithm $A$ has encoded the positions by…"), but it has rarely been used in a negative sense ("Algorithm $A$ has neither directly nor indirectly encoded the positions of the first $k$ significant bits."). In fact, we believe that it is not a priori obvious what the latter statement means, and we provide a formal definition here ("quantity $B$ is small") that is applicable in the general case and that proved useful in our specific situation. Of course, we could only obtain a general result because in our case the algorithms had small available memory, which necessarily leads to poor encoding of the positions. But note that there may also be algorithms with large memory which nevertheless have a poor encoding of the positions. In this sense our result is not about the quantity of information ("how large is the memory"), but rather about the quality ("how good are the positions encoded"), and we merely use the fact that a small quantity automatically implies poor quality.

## 4 Discussion

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 elitist selection 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 4, i.e., the statement that it suffices to consider one-bit flips, seems tricky.

Related to this, another interesting question arising from our proofs is the question of whether or not a memory-restricted black-box algorithm can benefit from trading-off a search point of better fitness value for one possibly containing more information (but having lower fitness). For LEADINGONES, we believe that this is not true; that is, we conjecture that for every (1 + 1) black-box algorithm eventually accepting search points of smaller fitness there exists an elitist one with the same or better expected runtime. Interestingly, this does not seem to be true for ONEMAX. In fact, the (1 + 1) black-box algorithm for ONEMAX with runtime $O(n/\log n)$ presented in [6] always selects the most recently queried search point, regardless of its fitness. This is another extreme selection criterion.

The proof methods employed in this work are adapted to the LEADINGONES problem, and it is probably non-trivial to transfer them to other problems. Nevertheless, we believe that the general idea of *tracking the amount of information* that a black-box algorithm has been able to gather about the optimization problem at hand can be used to derive lower bounds for other black-box complexity models as well as for other function classes. Beyond a purely mathematical interest, we are optimistic that such considerations can also serve as a source of inspiration for the design of new search heuristics—just as the idea of using crossover as a repair mechanism triggered the development of the $(1 + (\lambda, \lambda))$ GA presented in [3].

**Acknowledgements** 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 EDF (Électricité de France).

## References

1. Afshani, P., Agrawal, M., Doerr, B., Doerr, C., Larsen, K.G., Mehlhorn, K.: The query complexity of finding a hidden permutation. 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, New York (2013)
2. Böttcher, S., Doerr, B., Neumann, F.: Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In: Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN'10), pp. 1–10. Springer (2010)
3. Doerr, B., Doerr, C., Ebel, F.: Lessons from the black-box: Fast crossover-based genetic algorithms. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO'13), pp. 781–788. ACM (2013)

4. Doerr, B., Johannsen, D., Kötzing, T., Lehre, P.K., Wagner, M., Winzen, C.: Faster black-box algorithms through higher arity operators. In: Proceedings of Foundations of Genetic Algorithms (FOGA'11), pp. 163–172. ACM (2011)
5. Doerr, B., Winzen, C.: Black-box complexity: breaking the $O(n \log n)$ barrier of LeadingOnes. Artificial Evolution (EA'11), Revised Selected Papers. Lecture Notes in Computer Science, vol. 7401, pp. 205–216. Springer, New York (2012)
6. Doerr, B., Winzen, C.: Playing mastermind with constant-size memory. Theory Comput. Syst. **55**, 658–684 (2014)
7. Doerr, B., Winzen, C.: Ranking-based black-box complexity. Algorithmica **68**, 571–609 (2014)
8. Doerr, C., Lengler, J.: Elitist black-box models: analyzing the impact of elitist selection on the performance of evolutionary algorithms. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO'15), pp. 839–846. ACM (2015)
9. Doerr, C., Lengler, J.: OneMax in black-box models with several restrictions. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO'15), pp. 1431–1438. ACM (2015)
10. Doerr, C., Lengler, J.: The $(1 + 1)$ elitist black-box complexity of LeadingOnes. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO'16), pp. 1131–1138. ACM (2016)
11. Droste, S., Jansen, T., Wegener, I.: On the analysis of the $(1 + 1)$ evolutionary algorithm. Theor. Comput. Sci. **276**, 51–81 (2002)
12. Droste, S., Jansen, T., Wegener, I.: Upper and lower bounds for randomized search heuristics in black-box optimization. Theory Comput. Syst. **39**, 525–544 (2006)
13. Ladret, V.: Asymptotic hitting time for a simple evolutionary model of protein folding. J. Appl. Prob. **42**, 39–51 (2005)
14. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**, 623–642 (2012)
15. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
16. Mühlenbein, H.: How genetic algorithms really work: mutation and hillclimbing. In: Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature (PPSN'92), pp. 15–26. Elsevier (1992)
17. Rudolph, G.: Convergence Properties of Evolutionary Algorithms. Verlag Dr. Kovač, Hamburg (1997)
18. Sudholt, D.: A new method for lower bounds on the running time of evolutionary algorithms. IEEE Trans. Evol. Comput. **17**, 418–435 (2013)
19. Yao, A.C.C.: Probabilistic computations: toward a unified measure of complexity. In: Proceedings of Foundations of Computer Science (FOCS'77), pp. 222–227. IEEE (1977)