Electronic Colloquium on Computational Complexity, Report No. 92 (2011)

Memory-Restricted Black-Box Complexity

Benjamin Doerr and Carola Winzen

Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract

We show that the black-box complexity with memory restriction one of the n-dimensional OneMax function class is at most $2n$. This disproves the $\Theta(n \log n)$ conjecture of Droste, Jansen, and Wegener (Theory of Computing Systems 39 (2006) 525–544).

Keywords: Algorithms; black-box complexity; query complexity; bounded memory.

1 Introduction

The black-box complexity of a set $\mathcal{F}$ of functions $\mathcal{S} \to \mathbb{R}$, roughly speaking, is the number of function evaluations necessary to find the maximum of any member of $\mathcal{F}$ which—apart from the points evaluated so far—is unknown. This and related notions are used to describe how difficult a problem is to be solved via general-purpose (randomized) search heuristics. We refer to the seminal paper by Droste, Jansen, and Wegener [1] for a more detailed discussion. In contexts not dealing with randomized search heuristics, the term *(randomized) query complexity* or *(randomized) decision tree complexity* is more common, cf. [2] for a survey.

It turns out that there exist problems with surprisingly low black-box complexities. Droste, Jansen, and Wegener [1] note that the black-box complexity of MaxClique is polynomial, when the search space $\mathcal{S}$ consists of all subsets V' of the vertex set V and the objective function equals 0 for all subsets V' that do not form a clique and is equal to the size $|V'|$ if V' forms one. The algorithm which first queries all $\binom{n}{2}$ possible edges and then computes a maximum clique from this information has a black-box complexity of $\binom{n}{2} + 1$.

Also the basic OneMax$_n$ function class has a surprisingly low black-box complexity. For every $z \in \{0,1\}^n$ we define the function $\textsc{Om}_z : \{0,1\}^n \to \{0,\dots,n\}, x \mapsto |\{j \in [n] \mid x_j = z_j\}|$, that is, $\textsc{Om}_z(x)$ equals the number of bit positions in which z and x agree. The problem class OneMax$_n$ is the set of all functions $\textsc{Om}_z, z \in \{0,1\}^n$. Optimizing OneMax$_n$ in a black-box fashion can be seen as a mastermind-like game. By guessing bit stings and learning in how many positions our guess agrees with the hidden string z, we try to find z.

ISSN 1433-8092

Anil and Wiegand [3] show that by sampling $O(n/\log n)$ random bit strings, it is possible to determine the optimum of any $f \in \textsc{OneMax}_n$ with $O(n/\log n)$ queries.

Of course, all classic randomized search heuristics are not able to solve $\textsc{MaxClique}$ with a polynomial number of queries or find an n-bit string with less than n queries. In fact, many standard heuristics like the Randomized Local Search algorithm and the $(1+1)$ Evolutionary Algorithm— confer the book [4] for definitions and results—need an expected number of $\Theta(n \log n)$ queries for maximizing any $f \in \textsc{OneMax}_n$.

For this reason, in [1] a more restrictive black-box model is suggested, in which the size of the memory is bounded (see [5, 6] for two other attempts to obtain more realistic black-box complexity notions). It is conjectured in [1] that any black-box optimization algorithm with a memory that can store only one solution candidate x and its objective value $f(x)$ needs an expected number of $\Omega(n \log n)$ queries to optimize a worst-case input function $f \in \textsc{OneMax}_n$.

The aim of this note is to disprove this conjecture and to show that there exists a black-box algorithm with memory size one that optimizes any function $\textsc{Om}_z \in \textsc{OneMax}_n$ in $O(n)$ queries.

2 The Black-Box Model with Bounded Memory

Our original motivation for studying query complexity models is the following. Let us assume that we aim at maximizing some function $f : \mathcal{S} \to \mathbb{R}$. General purpose algorithms (also referred to as search heuristics) learn about the objective function f only by evaluating solution candidates $x \in \mathcal{S}$. That is, the objective function $f : \mathcal{S} \to \mathbb{R}$ is given as a black-box and the algorithm may optimize it by iteratively evaluating the objective value of search points $s \in \mathcal{S}$. We are typically interested in expected first hitting times, i.e., the expected number of function evaluations until an optimal search point is queried for the first time. The general scheme of a black-box algorithm is given by Algorithm 1. Note that this algorithm runs forever. Since our performance measure is the expected number of iterations needed until for the first time an optimal search point is queried, we do not specify a termination criterion for black-box algorithms here.

2.1 Black-Box Complexity

Let $\mathcal{A}$ be a class of algorithms and $\mathcal{F}$ be a class of functions. For every $A \in \mathcal{A}$ and $f \in \mathcal{F}$ let $T(A,f) \in \mathbb{R} \cup \{\infty\}$ be the expected number of function evaluations until A queries for the first time some $x \in \arg\max f$. We call $T(A,f)$ the *runtime of A for f*. The *A-black-box complexity of $\mathcal{F}$* is $T(A,\mathcal{F}) := \sup_{f \in \mathcal{F}} T(A,f)$, the worst-case runtime of A on $\mathcal{F}$. The *$\mathcal{A}$-black-box complexity of $\mathcal{F}$* is $\inf_{A \in \mathcal{A}} T(A,\mathcal{F})$. We drop the "$\mathcal{A}$"-prefix when referring to all black-box algorithms (that is, if we consider the class of all algorithms that can be expressed via the scheme of Algorithm 1).

```
1 Initialization:
2     Sample $x^{(0)}$ according to some probability distribution $p^{(0)}$ on $\mathcal{S}$;
3     Query $f(x^{(0)})$;
4 Optimization:
5     for $t = 1, 2, 3, \ldots$ do
6         Depending on $\big((x^{(0)}, f(x^{(0)})), \ldots, (x^{(t-1)}, f(x^{(t-1)}))\big)$ choose a
          probability distribution $p^{(t)}$ on $\mathcal{S}$;
7         Sample $x^{(t)}$ according to $p^{(t)}$;
8         Query $f(x^{(t)})$;
```

Algorithm 1: Scheme of a black-box algorithm for optimizing $f : \mathcal{S} \to \mathbb{R}$

2.2 Black-Box Algorithms with Memory of Size One

As discussed in the introduction, the black-box complexity can be surprisingly low even for NP-hard optimization problems. To obtain more realistic estimates on problem difficulties, Droste, Jansen, and Wegener [1] suggested to restrict the algorithms by allowing only a limited memory. This is inspired by the fact that many heuristics do not take advantage of knowing the full history of the search points queried so far.

Let $f : \mathcal{S} \to \mathbb{R}$ be the objective function to be maximized. A *black-box algorithm with bounded memory of size one* can be described as follows. In each iteration the algorithm has access to at most one previously queried search point x and its objective value $f(x)$. Depending on the content $(x, f(x))$ of his *memory*, it chooses a probability distribution on the search space $\mathcal{S}$, samples from it a new search point $y \in \mathcal{S}$ and queries its objective value $f(y)$. In the *selection step*, based only on $(x, f(x))$ and $(y, f(y))$, it decides which of the pairs $(x, f(x)), (y, f(y))$ to store in the memory (it "forgets" the other). This is formalized in Algorithm 2.

It is important to note that a black-box algorithm with bounded memory is not allowed to access any other information than the one $(x, f(x))$ stored currently in the memory and, in the selection step, also the information provided by $(y, f(y))$. In particular, the algorithm does not have access to an iteration counter.

Let $\mathcal{A}_1$ be the class of all black-box algorithms with bounded memory of size one. In what follows, the $\mathcal{A}_1$-black-box complexity of $\mathcal{F}$ is called the *memory*-1 *black-box complexity of* $\mathcal{F}$.

3 The Memory-1 Black-Box Complexity of OneMax$_n$

In this section, we prove that the memory-1 black-box complexity of OneMax$_n$ is at most linear in n.

As mentioned in the introduction, the two standard search heuristics Ran-

```
Initialization:
    Sample $x$ according to some probability distribution $p$ on $\mathcal{S}$;
    Query $f(x)$;
Optimization:
    for $t = 1, 2, \ldots$ do
        Variation step:
            Depending only on $(x, f(x))$ choose a probability distribution $p$
            on $\mathcal{S}$;
            Sample $y \in \mathcal{S}$ according to $p$;
            Query $f(y)$;
        Selection step:
            Depending only on $(x, f(x))$ and $(y, f(y))$, decide whether or not
            to update $(x, f(x)) \leftarrow (y, f(y))$;
```

Algorithm 2: Scheme of a black-box algorithm with memory of size one for optimizing $f : \mathcal{S} \to \mathbb{R}$

domized Local Search and the $(1+1)$ Evolutionary Algorithm need an expected number of $\Theta(n \log n)$ queries to optimize any $\textsc{Om}_z \in \textsc{OneMax}_n$, that is, to find $z = \arg\max \textsc{Om}_z$, cf. [7]. Since both algorithms are black-box algorithms with bounded memory of size one, this shows that the memory-1 black-box complexity of $\textsc{OneMax}_n$ is $O(n \log n)$. Droste, Jansen, and Wegener [1, Section 6] conjectured that this bound is tight, i.e., they conjectured that the memory-1 black-box complexity of $\textsc{OneMax}_n$ is $\Theta(n \log n)$. We show that this is not the case.

Theorem 1. *The memory-1 black-box complexity of* $\textsc{OneMax}_n$ *is at most* $2n$*.*

Before we give the proof, let us briefly fix some notation. For all positive integers $k \in \mathbb{N}$ we abbreviate $[k] := \{1, \ldots, k\}$. By e_k^n we denote the k-th unit vector $(0, \ldots, 0, 1, 0, \ldots, 0)$ of length n. The bitwise exclusive-or is denoted by $\oplus$.

Proof of Theorem 1. We give an algorithm that finds the unknown string $z \in \{0,1\}^n$ using an expected number of at most $2n$ queries, given that it has access to an $\textsc{Om}_z$-oracle.

The basic idea of our algorithm (Algorithm 3) is to learn z bit by bit from left to right, i.e, from z_1 to z_n. Since the algorithm does not have access to an iteration counter or to any other storage, we need to encode all necessary information in the string stored in the memory.

This we do as follows. Let $(x, \textsc{Om}_z(x))$ denote the current search point and its corresponding objective value. Set $s(x) := \min\{i \in [n] \mid \forall j \geq i : x_j = x_n\}$, the smallest index such that all bits in the *tail* $(x_{s(x)}, \ldots, x_n)$ are identical. Throughout the run of the algorithm we shall ensure that in the memory we only store such x for which all bits which are not in the tail are already optimal, that is, we ensure that $x_i = z_i$ for all $i < s(x)$. The key idea is that we can mark a newly learned bit as optimal by suitably setting the remaining tail to all zeros or all ones. Since memory is sparse, we have to guess the correct value of the bit and accept only if our guess is right. For the details refer to Algorithm 3.

```
Initialization:
  Initialize $x \leftarrow (0, \ldots, 0)$;
  Query $\textsc{Om}_z(x)$;
Optimization:
  while true do
    if $s(x) = n$ and $\textsc{Om}_z(x) = n-1$ then
      $y \leftarrow x \oplus e^n_n$;
      Query $\textsc{Om}_z(y)$; // optimum necessarily found
    else
      Sample $y \in \{x \oplus e^n_{s(x)}, x \oplus e^n_{s(x)+1} \oplus \ldots \oplus e^n_n\}$ uniformly at random;
      Query $\textsc{Om}_z(y)$;
      if $y = x \oplus e^n_{s(x)}$ then
        if $\textsc{Om}_z(y) > \textsc{Om}_z(x)$ then $(x, \textsc{Om}_z(x)) \leftarrow (y, \textsc{Om}_z(y))$;
      else
        if $\textsc{Om}_z(x) + \textsc{Om}_z(y) = n + s(x)$ then
          $(x, \textsc{Om}_z(x)) \leftarrow (y, \textsc{Om}_z(y))$;
```

Algorithm 3: A black-box algorithm with memory of size one for maximizing $\textsc{Om}_z \in \textsc{OneMax}_n$. Recall that we have defined $s(x) := \min\{i \leq n \mid \forall\, j \geq i : x_j = x_n\}$.

Note that Algorithm 3 is indeed a black-box algorithm with bounded memory of size one: If the memory is empty, we do a trivial variation and selection step filling it with $\big((0,\ldots,0), \textsc{Om}_z((0,\ldots,0))\big)$. If the memory is not empty, the (random) choice of the new search point y depends solely on the memory content $(x, \textsc{Om}_z(x))$. Lastly, the selection step is only based on the content $(x, \textsc{Om}_z(x))$ of the memory and the newly created candidate solution $(y, \textsc{Om}_z(y))$.

We claim that this algorithm queries the optimum z within an expected number of at most $2n$ queries. To this end, let us first show that indeed we have $x_i = z_i$ for all bits which are not in the tail, i.e, for all $i < s(x)$. This is trivially satisfied after initialization. We show the claimed via induction over $s(x)$. To this end, let $x \in \{0,1\}^n$ be a string with $s(x) < n$ and $x_i = z_i$ for all $i < s(x)$. Let y be created from x as in the algorithm above. It is immediate that $s(y) = s(x) + 1$ for all outcomes of the random y. Thus, all we need to show is that the algorithm updates its memory to $(y, \textsc{Om}_z(y))$ if and only if $y_{s(x)} = z_{s(x)}$.

Let us first consider the case $y = x \oplus e^n_{s(x)}$, i.e., y is created from x by flipping the $s(x)$-th bit of x. Clearly, $\textsc{Om}_z(y) > \textsc{Om}_z(x)$ if and only if $y_{s(x)} = z_{s(x)}$.

The case $y = x \oplus e^n_{s(x)+1} \oplus \ldots \oplus e^n_n$ is more involved. Assume first that $y_{s(x)} = z_{s(x)}$. Then we also have $x_{s(x)} = y_{s(x)} = z_{s(x)}$. By the induction hypothesis also $y_i = x_i = z_i$ holds for all $i < s(x)$. Consequently, the two substrings $(y_1, \ldots, y_{s(x)})$ and $(x_1, \ldots, x_{s(x)})$ contribute $2s(x)$ to $\textsc{Om}_z(x) + \textsc{Om}_z(y)$. For all indices $i > s(x)$ we either have $y_i = z_i$ or $x_i = z_i$. Therefore, the contribution of the substrings $(y_{s(x)+1}, \ldots, y_n)$ and $(x_{s(x)+1}, \ldots, x_n)$ to $\textsc{Om}_z(x) + \textsc{Om}_z(y)$ is exactly $n - s(x)$. Altogether we have shown that $\textsc{Om}_z(x) + \textsc{Om}_z(y) = 2s(x) + n - s(x) = n + s(x)$.

If $y_{s(x)} \neq z_{s(x)}$, then by similar arguments as above we obtain $\textsc{Om}_z(x) + \textsc{Om}_z(x \oplus y) = 2(s(x) - 1) + n - s(x) \neq n + s(x)$.

Since y is only accepted if $\textsc{Om}_z(x) + \textsc{Om}_z(y) = n + s(x)$, this happens if and only if $y_{s(x)} = z_{s(x)}$ as claimed.

Since the algorithm samples $y \in \{x \oplus e^n_{s(x)+1} \oplus \ldots \oplus e^n_n, x \oplus e^n_{s(x)}\}$ uniformly at random, it takes in expectation two queries to increase $s(x)$ (i.e., to shorten the length of the tail) by one. After an expected number of $2(n-1)$ iterations, we end up with an x in the memory satisfying $s(x) = n$. By the above, this implies $\textsc{Om}_z(x) \geq n - 1$. If $\textsc{Om}_z(x) = n$, we are done. Otherwise, in lines 7 and 8 of Algorithm 3 the last missing bit of x is flipped and $y = z$ is created. Together with the query of the initial string $x = (0, \ldots, 0)$, the expected number of queries needed to find z is at most $2(n-1) + 1 + 1 = 2n$. $\square$

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

References

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

[2] H. Buhrman, R. de Wolf, Complexity measures and decision tree complexity: a survey, Theoretical Computer Science 288 (2002) 21–43.

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

[4] A. Auger, B. Doerr, Theory of Randomized Search Heuristics, World Scientific, 2011.

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

[6] B. Doerr, C. Winzen, Towards a complexity theory of randomized search heuristics: Ranking-based black-box complexity, To appear in Proc. of Computer Science Symposium in Russia (CSR'11), Springer. Available also as ArXiv e-prints 1102.1140.

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