Electronic Colloquium on Computational Complexity, Report No. 48 (2003)

Upper and Lower Bounds for Randomized Search Heuristics in Black-Box Optimization*

Stefan Droste Thomas Jansen
Ingo Wegener
FB Informatik, LS2, Univ. Dortmund, 44221 Dortmund, Germany
droste, jansen, wegener@ls2.cs.uni-dortmund.de

Abstract

Randomized search heuristics like local search, tabu search, simulated annealing or all kinds of evolutionary algorithms have many applications. However, for most problems the best worst-case expected run times are achieved by more problem-specific algorithms. This raises the question about the limits of general randomized search heuristics.

Here a framework called black-box optimization is developed. The essential issue is that the problem but not the problem instance is known to the algorithm which can collect information about the instance only by asking for the value of points in the search space. All known randomized search heuristics fit into this scenario. Lower bounds on the black-box complexity of problems are derived without complexity theoretical assumptions and are compared to upper bounds in this scenario.

1 Introduction

One of the best-studied areas in computer science is the design and analysis of algorithms for optimization problems. This holds for deterministic

*This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (SFB 531).

ISSN 1433-8092

algorithms as well as for randomized algorithms (see, e.g., Cormen, Leiserson, and Rivest (1990) and Motwani and Raghavan (1995)). The criterion of the analysis is the asymptotic (w.r.t. the problem dimension), worst-case (w.r.t. the problem instance) expected (w.r.t. the random bits used by the algorithm) run time of the algorithm. Large lower bounds need some complexity theoretical assumption like NP $\neq$ P or NP $\neq$ RP. For almost all well-known optimization problems the best algorithms in this scenario are problem-specific algorithms which use the structure of the problem and compute properties of the specific problem instance.

This implies that randomized search heuristics (local search, tabu search, simulated annealing, all kinds of evolutionary algorithms) are typically not considered in this context. They do not beat the highly specialized algorithms in their domain. Nevertheless, practitioners report surprisingly good results with these heuristics. Therefore, it makes sense to investigate these algorithms theoretically. There are theoretical results on local search (Papadimitriou, Schäffer, and Yannakakis (1990)). The analysis of the expected run time of the other search heuristics is difficult but there are some results (see, e.g., Glover and Laguna (1993) for tabu search, Kirkpatrick, Gelatt, and Vecchi (1983) and Sasaki and Hajek (1988) for simulated annealing, and Rabani, Rabinovich, and Sinclair (1998), Wegener (2001), Droste, Jansen, and Wegener (2002) and Giel and Wegener (2003) for evolutionary algorithms). Up to now, there is no "complexity theory for randomized search heuristics" which covers all randomized search heuristics and excludes highly specialized algorithms. Such an approach is presented in this paper.

Our approach follows the tradition in complexity theory to describe and analyze restricted scenarios. There are well-established computation models like, e.g., circuits or branching programs (also called binary decision diagrams or BDDs) where one is not able to prove large lower bounds for explicitly defined problems. Therefore, one has investigated restricted models like monotone circuits, constant-depth circuits with unbounded fan-in and different sets of available basis functions, length-restricted branching programs or OBDDs. In all cases, one was interested in these restricted models in order to develop lower-bound techniques which can be applied to less restricted models. In some cases, e.g., OBDDs, the restricted models have real applications. Most of the restrictions are restrictions of some resource like parallel time in constant-depth circuits or sequential time in length-restricted branching programs. Monotone circuits restrict the type of possible algorithms or circuits. In our case of black-box optimization the information about the input

is restricted. This reflects the way how randomized search heuristics work. Our framework allows to discuss optimal randomized search heuristics and difficult black-box optimization problems. Based on Yao's minimax principle (Yao (1977)) we can prove lower bounds without complexity theoretical assumptions.

In Section 2, the scenario of black-box optimization and black-box complexity is introduced. It is discussed why all well-known randomized search heuristics are indeed black-box algorithms. Moreover, it turns out that black-box algorithms can be described as randomized decision trees (a well-studied computational model in the context of boolean functions, see, e.g., Hajnal (1991), Heiman and Wigderson (1991), Heiman, Newman, and Wigderson (1993)).

In Section 3, it is shown that the model might be too generous since it does not restrict the resources for computing the next query in the decision tree. Therefore, some NP-hard problems have polynomial-time black-box complexity. It is possible to consider the restricted and realistic model where the time to compute the next query is polynomially bounded. Some lower bounds are even proved in the unrestricted model which makes the bounds only stronger. Upper bounds should be proved only in the time-restricted model. Moreover, a scenario is introduced where the information transfer is restricted. This is motivated by the fact that most randomized search heuristics are working with a limited information transfer. The restriction is realized by limiting the information which is transferred from round i to round $i + 1$ of the heuristic to a multiset of $s(n)$ search points with their function values.

In Section 4, we show that problems which are easy as optimization problems can have the largest possible black-box complexity. These easy results stress the differences between the usual and the black-box scenario.

Afterwards, we prove upper and lower bounds for several classes of problems. First, we investigate in Section 5 sorting as the minimization of unsortedness where unsortedness is measured by different measures known from the theory of adaptive sorting algorithms. Section 6 is motivated by problems which are typical examples in the community discussing evolutionary algorithms and Section 7 investigates the class of monotone pseudo-boolean polynomials. In Section 8, it is shown that the class of unimodal pseudo-boolean functions, i.e., functions where each point has a better Hamming neighbor or is globally optimal, is difficult in black-box optimization. In Section 9, we discuss black-box optimization for multi-objective optimiza-

tion problems and investigate the single-source-shortest-paths problem. We finish with some conclusions.

This paper is based on the conference paper by Droste, Jansen, Tinnefeld, and Wegener (2003) and contains several new results.

2 Black-Box Optimization, Randomized Search Heuristics, and Randomized Search Trees

The following meta-algorithm covers all randomized search heuristics working on the finite search space S. Functions to be optimized are functions $f\colon S \to \mathbb{R}$.

Algorithm 1 (*Black-box algorithm*)

1.) *Choose some probability distribution p on S and produce a random search point $x_1 \in S$ according to p. Compute $f(x_1)$.*

2.) *In Step t, stop if the considered stopping criterion is fulfilled. Otherwise, depending on $I(t) = (x_1, f(x_1), \ldots, x_{t-1}, f(x_{t-1}))$ choose some probability distribution $p_{I(t)}$ on S and produce a random search point $x_t \in S$ according to $p_{I(t)}$. Compute $f(x_t)$.*

All the randomized search heuristics mentioned in the introduction fit into this scenario. E.g., the famous 2-opt algorithm for TSP chooses the tour π_1 uniformly at random and computes its cost $C(\pi_1)$. The algorithm does not store the whole history $(\pi_1, C(\pi_1), \ldots, \pi_{t-1}, C(\pi_{t-1}))$ but only one tour π as current one. This is captured by Algorithm 1 if $p_{I(t)}$ depends essentially only on π. Then two non-neighbored edges of π are chosen and π_t results from π by cutting π into two pieces and pasting them together to obtain $\pi_t \neq \pi$. The interesting aspect is that the algorithm uses the parameters of the problem instance only for computing $f(x_t)$ and not for the choice of $p_{I(t)}$. Hence, it could work without knowing the problem instance if some black box produces $f(x_t)$ if the algorithm asks the query x_t. Generalizing this we obtain the following scenario called *black-box scenario*.

The algorithm knows that it has to optimize one function from a class $\mathcal{F}$ of functions $f\colon S \to \mathbb{R}$ on the same finite search space. In the case of

TSP, S consists of all tours on $\{1,\ldots,n\}$ and $\mathcal{F}$ contains all functions f_D, $D=(d_{ij})_{1\leq i,j\leq n}$, $d_{ij}\geq 0$, a distance matrix. Then $f_D(\pi)$ equals the cost of π with respect to the distance matrix D. The black box knows which function $f\in\mathcal{F}$ is considered (it knows D) while the algorithm does not have this knowledge. The algorithm is allowed to ask queries $x\in S$ to the black box and obtains the correct function values $f(x)$ as answers. This implies that, in Step t, the whole knowledge of the algorithm is the knowledge that the problem is described by $\mathcal{F}$ and the information contained in $I(t)$. Hence, the black-box scenario is an information-restricted scenario. The investigation of this scenario is motivated since randomized search heuristics work in this scenario.

We have not yet discussed the stopping criterion. Randomized search heuristics hopefully produce quickly good solutions but they do not prove that they are good. An exact branch-and-bound algorithm may produce an optimal solution in the first step and may need exponential time to prove that it is optimal. Randomized search heuristics are stopped without knowing whether the best search point produced is optimal. Therefore, we investigate the algorithms without any stopping criterion (as infinite stochastic processes) but we charge the algorithms only for the time until an optimal search point is chosen as query. The motivation for this is the observation that randomized search heuristics do not spend most of their time after having found an optimal search point. We remark that Lovász, Naor, Newman, and Wigderson (1991) have investigated search problems in the model of randomized decision trees. In their model the queries consider the value of input bits.

Finally, we have to decide how we charge the algorithm for the resources spent. For most optimization problems, the computation of $f(x)$ is easy (for the black box knowing f). Hence, we only count the number of queries. This allows arbitrary time for the choice of $p_{I(t)}$ but a polynomial number of queries with an exponential run time is not an efficient solution. In the time-restricted model, we therefore allow only polynomial time (w.r.t. the problem dimension $\lceil\log|S|\rceil$) for the choice of $p_{I(t)}$. Summarizing, the black-box complexity underestimates the run time since only queries are counted. Lower bounds on the black-box complexity describe limits for all randomized search heuristics. When proving upper bounds one should estimate the time for evaluating f and for the realization of the choice of the query points. Nevertheless, one has to keep in mind that an optimization problem and its black-box variant are different problems. The second one cannot be easier

since one has less information. The black-box complexity (the number of queries) can be smaller than the run time of the best-known algorithm for the optimization problem but this cannot hold for the overall run time of the black-box algorithm.

After having boiled down black-box optimization to a game of queries and answers we describe algorithms by decision trees. This makes it possible to apply the lower bound technique known as Yao's minimax principle. A *deterministic* black-box algorithm can be described as a decision tree T. The first query is computed deterministically and is represented by the root of the decision tree. Each vertex v of the tree describes a query x and has an outgoing edge for each possible answer $f(x)$, $f \in \mathcal{F}$. The history is described by the unique path from the root to v containing all earlier queries and answers. Only a subset $\mathcal{F}(v) \subseteq \mathcal{F}$ describes the problem instances which are consistent with all queries and answers in the history. Therefore, it is sufficient to consider all $f(x)$, $f \in \mathcal{F}(v)$. For each problem instance f the algorithm follows a unique path and the cost $\mathcal{C}(f, T)$ equals the number of nodes on this path until a node queries an f-optimal search point. In principle, these search trees may have infinite depth. When designing a good decision tree we can avoid to ask the same query twice. Then the depth of decision trees is limited by the finite number $|S|$. Nevertheless, we have to consider infinite trees. If we allow all integer-valued distance matrices for the TSP, the query π has infinitely many answers. In most cases, we can restrict the function values $f(x)$, $x \in S$, to a finite set. Yao's minimax principle can be applied only if the number of deterministic algorithms is finite. This assumption holds for deterministic trees of a depth bounded by $|S|$ and a finite number of answers for all queries.

A *randomized* black-box algorithm is a probability distribution on the set of all deterministic black-box algorithms and, therefore, a randomized decision tree. This is the most convenient definition when considering lower bounds. For the design of randomized black-box algorithms it is more convenient to define them as algorithms fitting into the framework of Algorithm 1. Both definitions are equivalent.

After having described randomized search heuristics in black-box optimization as randomized decision trees we recall Yao's minimax principle (Yao (1977), see also Motwani and Raghavan (1995)) which allows to prove lower bounds for *randomized* algorithms by proving lower bounds for the expected run time (w.r.t. a probability distribution on the problem instances) of *deterministic* algorithms.

Proposition 1 (*Yao's Minimax Principle*) *If a problem consists of a finite set of instances of a fixed size and allows a finite set of deterministic algorithms, the minimal worst-case instance expected optimization time of a randomized algorithm is lower bounded for each probability distribution on the instances by the expected optimization time of an optimal deterministic algorithm.*

3 NP-hard Problems with a Polynomial Black-Box-Complexity and Information-Restricted Black-Box Algorithms

The aim of this short section is to prove that the black-box scenario with unrestricted time for the computation of the queries is useful only for lower bounds. The reason is that algorithms may ask queries to get information about the problem instance. Then the optimal solution is computed by exhaustive search and, finally, presented as query.

The simplest example of this kind is the MAX-CLIQUE problem where the search space consists of all vertex sets $V' \subseteq \{1, \ldots, n\}$. Each graph G on $V = \{1, \ldots, n\}$ is a problem instance and the corresponding function f_G to be maximized is defined by $f_G(V') = |V'|$, if V' is a clique of G, and $f_G(V') = 0$, otherwise. A black-box algorithm may ask all sets V' where $|V'| = 2$ in order to get the information on the edge set of G. Afterwards, a maximum clique V_{opt} is computed and presented as query. This algorithm needs $\binom{n}{2} + 1$ queries. The overall run time of this algorithm is super-polynomial (assuming that NP $\neq$ P).

We have seen that NP-hard problems can have a polynomial black-box complexity but the corresponding black-box algorithm cannot be run efficiently. The general model of black-box optimization is too generous since the time for the computation of the queries is not limited. In the rest of this paper, we consider only upper bounds with polynomial-time algorithms to compute the queries. Even this restriction does not rule out black-box algorithms which first reconstruct the problem instance and then use a problem-specific algorithm to compute an optimal search point and present it as query. This is the case for the maximum matching problem (Giel and Wegener (2003)). Altogether, we can conclude that the class of black-box algorithms is not restricted enough.

A more reasonable class restricts the information about the history which is transferred to step t. Local search and simulated annealing only store one search point with its value and evolutionary algorithms only store a small number of search points with their values. The only information about the history consists of $s(n)$ search points and their values. The information package transferred to the next round has a size of $s(n)$. Hence, $s(n)$ will be called size bound. The corresponding meta algorithm can be described as follows.

Algorithm 2 (*Black-box algorithm with size bound* $s(n)$)

1. *Apply Algorithm* 1 *for* $s(n)$ *steps.*
2. *In Step* t*, stop if the considered stopping criterion is fulfilled. Otherwise, depending only on the multiset* I *consisting of* $(x_1, f(x_1))$, ..., $(x_{s(n)}, f(x_{s(n)}))$ *choose some probability distribution* p_I *on* S *and produce a random search point* $x \in S$ *according to* p_I*. Compute* $f(x)$*. Use a randomized algorithm to decide whether* $(x, f(x))$ *replaces some* $(x_i, f(x_i))$ *to update* I*.*

In this paper, we only prove lower bounds for the unrestricted case. The case of lower bounds for the information-restricted scenario is left for future research. We consider upper bounds for the unrestricted and the information-restricted case.

4 Simple Problems with Maximal Black-Box Complexity

The purpose of this section is to show that problems that are simple in the usual scenario can be very difficult in the black-box scenario. We start with a simple upper bound in order to see later that the difficult black-box problems have maximal complexity.

Proposition 2 *If the black-box problem is defined on the search space* S*, the black-box complexity is bounded above by* $(|S| + 1)/2$.

Proof We create uniformly at random a permutation of S and query the search points in this random order. For each $x \in S$, the expected time until it is queried equals $(|S| + 1)/2$. □

The following problem is known as "needle in the haystack" in the area of evolutionary computation. The class of functions consists of all N_a, $a \in S$, where $N_a(a) = 1$ and $N_a(x) = 0$, if $x \neq a$. We investigate maximization problems if nothing else is mentioned.

Theorem 1 *The black-box complexity of the needle-in-the-haystack problem equals* $(|S| + 1)/2$.

Proof The upper bound is contained in Proposition 2. The lower bound follows by a simple application of Yao's minimax principle. We consider the uniform distribution on all N_a, $a \in S$. After having queried m search points without finding the optimal one, all other $|S| - m$ search points have the same probability of being optimal. Hence, each deterministic search strategy queries on average $(|S| + 1)/2$ different search points. □

Random search does not transfer information, i.e., $s(n) = 0$, and it needs in this case $|S|$ queries on the average and is almost optimal. In the usual optimization scenario, the problem instance N_a and, therefore, a has to be specified and it is trivial to compute the optimal solution a. Why are we interested in such a class of functions? The reason is that it contains the interesting single-source-shortest-paths problem SSSP if we model that problem in the following way. The search space consists of all trees T rooted at the source s. The cost function $C_D(T)$ with respect to a distance matrix $D = (d_{ij})$, $d_{ij} > 0$, equals the sum of the cost of all s-i-paths in T and the aim is minimization. If we restrict the problem to those distance matrices where the connections contained in some tree T^* have cost 1 and all other connections have cost ∞, we are in a needle-in-the-haystack scenario. The tree T^* has finite cost and all other trees have infinite cost. Hence, the important SSSP has the largest-possible black-box complexity in this description of the problem. In Section 9, we describe SSSP as a multi-objective problem such that its black-box complexity is linear.

The following problem is known as "trap" in the area of evolutionary computation. Let $S = \{0, 1\}^n$, let $T_a(x)$ equal the number of ones in x, if $x \neq a$, and let $T_a(a) = 2n$. The name "trap" describes the fact that typical randomized search heuristics get trapped in the local optimum 1^n (if a does not contain many ones). Their expected optimization time is much larger than the bound of Proposition 2 or the bound 2^n for random search. In the same way as Theorem 5 we obtain the following result.

Proposition 3 *The black-box complexity of the trap problem equals* $(2^n + 1)/2$.

This result has been mentioned since trap functions are bimodal. For all a, there are at most two local optima, i.e., points without a better Hamming neighbor. Hence, the class of bimodal functions has the maximal black-box complexity. Unimodal functions have exactly one local optimum which necessarily is globally optimal. Many people believe that unimodal functions are simple for randomized search heuristics. This is disproved in Section 8.

5 Sorting as Black-Box Optimization Problem

There seems to be no computer science problem which is the subject of more publications than sorting. This motivates the investigation of sorting when considering new algorithmic aspects. Here it is necessary to describe sorting as optimization problem, i.e., the problem to minimize the unsortedness. Measures of unsortedness or presortedness have been considered in the theory of adaptive sorting algorithms (see Petersson and Moffat (1995)). We investigate the five best-known measures for the unsortedness of permutations π on $\{1, \ldots, n\}$ with respect to an optimal permutation π'. Because of symmetry it is sufficient to consider the case where $\pi' = id$ is the identity.

- INV(π) equals the number of inversions, i.e., the number of pairs (i, j), where $i < j$ and $\pi(i) > \pi(j)$.
- RUN(π) equals the number of runs, i.e., the number of maximal-length sorted subblocks.
- REM(π) equals the number of removals, i.e., the minimal number of elements which have to be deleted from $\pi(1), \ldots, \pi(n)$ in order to obtain a sorted subsequence. It is known that REM(π) equals the minimal number of jumps to sort the sequence. A jump is an operation where one element is removed from $\pi(1), \ldots, \pi(n)$ and inserted again somewhere.
- EXC(π) equals the minimal number of exchanges of two elements to sort the sequence. It is known that a permutation π with k cycles has an EXC-value of $n - k$.

– HAM(π) is the "Hamming distance" to the sorted sequence, i.e., it counts the number of positions with a wrong element.

All the measures lead to minimization problems. Scharnow, Tinnefeld and Wegener (2002) have investigated an evolutionary algorithm with size bound 1 for these five optimization problems. The algorithm has an expected optimization time of $O(n^2 \log n)$ in all cases with the exception of RUN where it needs exponential time.

In our framework, the search space is the set Σ_n of all permutations on $\{1, \ldots, n\}$ and we have for each $\pi' \in \Sigma_n$ a problem instance $f_{\pi'} \colon \Sigma_n \to \mathbb{R}$ where $f_{\pi'}(\pi)$ measures the unsortedness of π with respect to the optimal permutation π' and the measure of unsortedness characterizing the problem.

Our lower bounds are corollaries to a more general result.

Theorem 2 *Let S be the search space of an optimization problem. If for each $s \in S$ there is an instance such that s is the unique optimum and if each query has at most $k \geq 2$ possible answers, then the black-box complexity is bounded below by $\lceil \log_k |S| \rceil - 1$.*

Proof We apply Yao's minimax principle and choose for each $s \in S$ a problem instance $I(s)$ such that s is the unique optimum for $I(s)$. Then we investigate the uniform distribution on these instances. For each deterministic search strategy we obtain a decision tree T whose outdegree is bounded by k and which has to contain a node with the query s for each $s \in S$. For each s we consider an s-node with minimal depth. The average number of queries is at least by 1 larger than the average depth of all chosen s-nodes and, therefore, at least $\lceil \log_k |S| \rceil - 1$. □

We have $|S| = n!$ for sorting problems. The parameter k is bounded above by $\binom{n}{2} + 1$ for INV, by n for RUN, REM, EXC, and HAM. ($0 \leq \mathrm{HAM}(\pi) \leq n$ but the value 1 is impossible.) Since $\log(n!) = n \log n - O(n)$ we get the following corollary.

Corollary 1 *The black-box complexity of the sorting problem with respect to INV is bounded below by $n/2 - o(n)$ and with respect to RUN, REM, EXC, or HAM it is bounded below by $n - o(n)$.*

In the following, we prove upper bounds.

Theorem 3 *The following upper bounds hold for the black-box complexity of the different sorting problems:*

- *INV:* $n+1$ *in the unrestricted case and* $2n-1$ *for the size bound* 3*,*
- *RUN:* $2n \log n + O(n)$*,*
- *HAM:* $O(n \log n)$*.*

Proof For INV, we ask the queries $a_k := (k, k+1, \ldots, n, 1, \ldots, k-1)$, $1 \leq k \leq n$. Using the answers to these queries we are able to compute the rank of each element in the sorted sequence. We show this for item k. Let b be the number of inversions of a_k and c the corresponding number of a_{k+1} (or a_1, if $k = n$). Let r be the unknown rank of k. Each of the items smaller than k is counted in a_k and not in a_{k+1} while each of the items larger than k is counted in a_{k+1} and not in a_k. All other pairs are counted both in a_k and a_{k+1} or neither in a_k nor in a_{k+1}. Hence,

$$b - c = r - 1 - (n - r) = 2r - n - 1$$

and r can be computed knowing b and c. Finally, we know the rank of each item and can present the sorted list as query $n+1$.

If $s(n) = 3$, we ask the following queries: a_1, a_2, a_1^*, a_3, $a_2^*, \ldots, a_n, a_{n-1}^*$. The a_i-queries are the same as above. The search point a_i^* should have the items $1, \ldots, i$ at their correct positions and should be different from all a_j. The aim is to store a_i, a_{i-1}^*, and a_{i+1} after having queried a_{i+1}, and to store a_i, a_{i+1}, and a_i^* after having queried a_i^*. First, we query a_1 and a_2 and compute some a_1^* with the proposed properties. From a_i, a_{i+1}, and a_i^*, we can decide which is the a^*-query. Then we can compute from a_i and a_{i+1} the value of i and a_{i+2}. This is the next query. Knowing the answer we forget a_i and its value. From a_i, a_{i-1}^*, and a_{i+1}, we can compute which are the search points a_i and a_{i+1}. Then we compute the rank of item i and some a_i^* with the proposed properties. As long as $i \leq n-3$, we have three free items. If we rank them in decreasing order, the search point is different from all a-queries. If $i = n-2$, the items $1, \ldots, n-2$ have to be at their correct positions. There is always one choice to place $n-1$ and n such that we fulfil the properties. Then we query a_i^* and forget a_{i-1}^* and its value. If $i = n-1$, the search point where $1, \ldots, n-1$ are at their correct positions is unique and optimal. If it equals some a_i, it has been queried earlier and the search has been finished earlier.

For the scenario described by RUN, we can simulate binary comparisons and, therefore, sorting algorithms with a small worst-case number of comparisons. Let a and b be two items we want to compare and let $c = c_3, c_4, \ldots, c_n$ be the other items. Let r be the unknown number of runs of $(c_3, \ldots, c_n)$. We ask for the number of runs of $\pi_1 = (a, b, c_3, \ldots, c_n)$ and $\pi_2 = (b, a, c_3, \ldots, c_n)$. The results are summarized in the following table for all six complete orderings of a, b, and c.

	$\mathrm{RUN}(\pi_1)$	$\mathrm{RUN}(\pi_2)$
$a < b < c$	r	$r+1$
$c < a < b$	$r+1$	$r+2$
$b < a < c$	$r+1$	r
$c < b < a$	$r+2$	$r+1$
$a < c < b$	$r+1$	$r+1$
$b < c < a$	$r+1$	$r+1$

If $\mathrm{RUN}(\pi_1) < \mathrm{RUN}(\pi_2)$, we conclude that $a < b$. If $\mathrm{RUN}(\pi_2) < \mathrm{RUN}(\pi_1)$, we conclude that $b < a$. Otherwise, $\min\{a, b\} < c < \max\{a, b\}$. Then we use the same approach to compare a and c and put item b at position 3. Then we know whether $a < c$ (implying that $a < b$) or $c < a$ (implying that $b < a$). Hence, a binary comparison can be simulated by four queries. With at most $4(n-1)$ queries we determine the maximal item which then is used as c-element. Then two queries are sufficient to simulate a query and the total number of queries can be bounded by $2n \log n + O(n)$.

For HAM, we use a randomized sampling strategy to collect information. Moreover, we use the power of negative thinking. If $\mathrm{HAM}(\pi) = n$, we obtain for each item the information that its position in π is wrong. If we know $n-1$ wrong positions for each item, we know the correct position of each item and can compute the optimum search point. If π is chosen uniformly at random, the probability that $\mathrm{HAM}(\pi) = n$ is at least $1/e - O(1/(n!))$ (see Graham, Knuth, and Patashnik (1994)). We investigate a sequence of $cn \log n$ queries chosen independently and uniformly at random. The constant c is chosen large enough. By Chernoff bounds, with overwhelming probability we have at least $c'n \log n$ permutations π such that $\mathrm{HAM}(\pi) = n$, e.g., for $c' = c/3$. By the coupon collector's theorem (see Motwani and Raghavan (1995)), the probability that item i does not take all its $n-1$ wrong positions in these permutations is less than $1/(2n)$, if $c' = 1+\varepsilon$. Hence, the probability that we do not know all wrong positions for all items is less than $1/2$. Altogether, the

expected number of queries until we can compute the optimal permutation is bounded by $O(n \log n)$. □

It is easy to prove $O(n^2)$ bounds for REM and EXC. These bounds are not presented in detail, since they are not significantly better than the $O(n^2 \log n)$ bounds mentioned earlier which even hold for the size bound 1. The upper and lower bounds for INV differ only by a factor of 2. They differ since the lower bound works with the maximal possible number of $\binom{n}{2} + 1$ different answers to a query. The search strategy for the upper bound allows only up to n different answers for each query with the only exception of the first one. A similar reason holds for the difference in the lower and upper bound for RUN. The lower bound is based on the fact that $\mathrm{RUN}(\pi)$ can take n different values but we ask some queries to get the information of a binary comparison. The situation for HAM is different. Each query can have n different answers but we only use the information whether $\mathrm{HAM}(\pi) = n$ or $\mathrm{HAM}(\pi) < n$. This implies that this strategy needs $\Theta(n \log n)$ queries. Altogether, we have solved the problem completely only for INV. For the unrestricted case and EXC and REM one may hope to find better upper bounds. For the case of a constant size bound, it would be interesting to look for better lower bounds.

6 Classes of Simple Functions

Since this paper has been motivated by problems on the analysis of evolutionary algorithms, we now discuss classes of functions which are investigated as typical examples in the literature on evolutionary algorithms.

A function is called separable if some subblocks of the input vector can be optimized independently. The class LIN of all linear functions on $\{0,1\}^n$, namely all

$$f(x) = w_1 x_1 + \cdots + w_n x_n,$$

contains those functions where each bit can be optimized independently from the others. Evolutionary algorithms on linear functions have been investigated in many papers, before Droste, Jansen, and Wegener (2002) have proved that a simple evolutionary algorithm with size bound 1 has an expected optimization time of $\Theta(n \log n)$ on LIN. The special function where $w_1 = \cdots = w_n = 1$ is known as ONEMAX. The idea is that all variables have the same influence, i.e., the same absolute weight. To obtain a class of functions, we define ONEMAX as the class of linear functions where

$w_i \in \{-1, +1\}$ for all i. The special function where $w_i = 2^{n-i}$ is known as BV (it interprets $(x_1, \ldots, x_n)$ as a binary representation and computes its binary value). The idea is that x_i has more influence than $x_{i+1}, \ldots, x_n$ altogether. We define BV as the class of linear functions where $w_i \in \{-2^{n-i}, +2^{n-i}\}$. Finally, we consider the non-linear function known as LO (leading ones). The value of $\mathrm{LO}(x)$ is the length of the longest prefix of x consisting of ones only. Again, x_i has more influence than $x_{i+1}, \ldots, x_n$ altogether. Moreover, as long as $(x_1, \ldots, x_i)$ does not have the right value, $x_{i+1}, \ldots, x_n$ have no influence at all. Droste, Jansen, and Wegener (2002) have proved that the already mentioned evolutionary algorithm has an expected optimization time of $\Theta(n^2)$ on LO. We define LO as the class of all functions $f_a, a \in \{0,1\}^n$, where $f_a(x)$ is the length of the longest prefix of x which equals the corresponding prefix of a.

Theorem 4 *The black-box complexity of LIN is bounded above by $n+1$ in the unrestricted case and by $(3/2)n + 1/2$ for the size bound 2, already for ONEMAX it is bounded below by $n/\log(2n+1) - 1$. The black-box complexity of BV equals $2 - 2^{-n}$.*

Proof Let $e_i = 0^{i-1}10^{n-i}$. Knowing all $f(e_i)$ for $f \in \mathrm{LIN}$ is equivalent to knowing all w_i. Then query $n+1$ can be chosen as optimal search point.

In the case of a size bound of 2 the idea is to store $(e_i, f(e_i))$ and $(a_i, f(a_i))$ after round i where a_i contains the optimal prefix of length i followed by $n-i$ zeros. Remember that $(e_i, f(e_i))$ reveals the optimal bit at position i. Hence, the first round is obvious. In general, the storage contains some e_i (always together with $f(e_i)$) and some a with an optimal prefix whose length is at least $i-1$. From $(e_i, f(e_i))$ we can decide whether a contains the optimal prefix of length i. In the positive case, the next query is e_{i+1} and, in the negative case, the next query is a' obtained from a by flipping the ith bit. It is obvious that this strategy uses at most $2n$ queries. We can improve this by using random bits at the positions $2, \ldots, n$ of a_1. Then the expected number of wrong bits equals $(n-1)/2$ and only for wrong bits we have to ask the a-queries.

The lower bound for ONEMAX (and, therefore, also LIN) follows from the fact that $f(x) \in \{-n, \ldots, n\}$ for $f \in \mathrm{ONEMAX}$ and an application of Theorem 2.

The upper bound for BV follows by asking at first a random query a which is successful with probability 2^{-n}. The value $f(a), f \in \mathrm{BV}$, reveals

the full information on all w_i and the second query can be chosen as optimal search point. The lower bound is easy to obtain. □

The results on LIN are interesting since we conjecture a lower bound (even for ONEMAX) of $\Omega(n \log n)$ for the size bound 1. In the case of BV, $f(a)$ contains the full information on f. The main idea behind BV is also contained in the class MBV of all functions $g \circ f$ where $f \in$ BV and $g\colon \mathbb{Z} \to \mathbb{R}$ is monotone. The class MBV contains also nonlinear functions and has infinite size. It is interesting to see how Yao's minimax principle can nevertheless be applied to get lower bounds on the black-box complexity of MBV.

Theorem 5 *The black-box complexity of MBV is bounded above by $n + 2$ and bounded below by $\Omega(n/\log n)$.*

Proof The upper bound follows as the upper bound for LIN in Theorem 4. We only have to add the query point $e_0 = 0^n$ to get a reference point.

For the lower bound, we first assume a less powerful black box. It does not produce the correct $g \circ f$-value for the query but only the complete order of the $g \circ f$-values of all queries asked so far. For the $(t+1)$th query point there are only $t+1$ places in the complete order of the previous t queries where the new query point can be inserted. The number of deterministic algorithms is finite and we can apply Yao's minimax principle for the uniform distribution on all 2^n possible $f \in$ BV. The number of nodes in the decision tree on the first t levels is bounded above by $t!$. Hence, the average depth of 2^n nodes is $\Omega(n/\log n)$.

In the following, we prove that we cannot do better than this lower bound if we know the exact values of $g \circ f(a)$. Let $s = (s_1, \ldots, s_n)$ be the sign vector of f, i.e., $s_i = +1$ if $w_i = 2^{n-i}$, and $s_i = -1$, otherwise. After having queried $a_1, \ldots, a_m$ the set of still possible s-vectors is independent of the fact whether we get only the order of the $g \circ f$-values or their exact values. In the original black-box scenario, the decision on the next query can depend on the exact values of the function. Let us consider two situations where the same queries have been asked with the same order of the function values but with different function values. It makes no sense to use the knowledge of the function values. We have a worst-case input scenario and the adversary is still free to choose the monotone function g in such a way that it transforms one vector of function values into the other one. □

Finally, we prove quite precise bounds on the black-box complexity of LO.

Theorem 6 *The black-box complexity of LO is bounded above by* $n/2+o(n)$ *for the size bound* 1 *and bounded below by* $n/2-o(n)$ *in the unrestricted case.*

Proof The upper bound follows by the following simple strategy. The first query is chosen uniformly at random. If the storage contains the search point b and $f_a(b) = i$, we know that $(a_1, \ldots, a_i) = (b_1, \ldots, b_i)$ and $a_{i+1} = 1 - b_{i+1}$. The next query b' starts with $(a_1, \ldots, a_{i+1})$ followed by $n - i - 1$ random bits. In any case b' replaces b in the storage. Then $f_a(b') = j \geq i + 1$ and the random number N_i of new correct bits equals $j - i$. We have $\text{Prob}(N_i = k) = 2^{-k}$, if $1 \leq k \leq n - i - 1$, and $\text{Prob}(N_i = n - i) = 2^{-(n-i-1)}$. This corresponds to a geometrically distributed random variable with parameter $p = 1/2$ where values larger than $n - i$ are replaced by $n - i$. Therefore,

$$2 - 2^{-(n-i-1)} \leq E(N_i) \leq 2 \qquad \text{and } V(N_i) \leq 3.$$

We partition the run of the algorithm into two phases where the first one ends when we know at least $n - \lceil n^{1/2} \rceil$ a-bits. The length of the second phase is bounded by $\lceil n^{1/2} \rceil = o(n)$. We try to estimate the probability q that the length of the first phase is not bounded above by $t := n/2 + n^{2/3} = n/2 + o(n)$. This equals the probability that the sum S of t independent random variables $N_{i(j)}$, $0 \leq i(j) < n - \lceil n^{1/2} \rceil$ is smaller than $n - \lceil n^{1/2} \rceil$. Then $E(S) \geq (2 - 2^{-(n-\lceil n^{1/2} \rceil)})t$ and $V(S) \leq 3t$. It follows from Tschebyscheff's inequality that $q = o(1)$. Since n is a trivial upper bound on the length of the first phase, its expected length is bounded by $(1 - q)(n/2 + n^{2/3}) + q \cdot n = n/2 + o(n)$.

For the lower bound, we apply Yao's minimax principle for the uniform distribution on all $a \in \{0, 1\}^n$. Note that Theorem 2 gives only a lower bound of order $n/\log n$. In the scenario of LO it is easy to evaluate precisely our knowledge after having asked t queries $b(1), \ldots, b(t)$. If $m = f_a(b(j))$ is the maximal answer to the queries, we know precisely the first $m + 1$ bits of a and the a-posteriori distribution for the $n - m - 1$ suffix bits is the uniform distribution on $\{0, 1\}^{n-m-1}$. Hence, a query not starting with the correct $m + 1$ bits reveals no information. Each query starting with the correct $m + 1$ bits reveals the random number of N_{m+1} new correct bits (where N_{m+1} is the random variable defined in the proof of the upper bound). Now, Tschebyscheff's inequality implies that the probability that $n/2 - n^{2/3}$ steps are enough to find the optimum is bounded above by $o(1)$. This implies the theorem. □

7 Monomials and Monotone Polynomials

Each pseudo-boolean function $f\colon \{0,1\}^n \to \mathbb{R}$ can be written uniquely as a polynomial

$$f(x) = \sum_{A \subseteq \{1,\ldots,n\}} w_A \cdot \prod_{i \in A} x_i.$$

Its degree d is the largest size of a set A where $w_A \neq 0$. The function f is monotone increasing if $w_A \geq 0$ for all A. Let $z_i \in \{x_i, 1 - x_i\}$. Then we get also a unique representation (of the same degree) with respect to $z_1, \ldots, z_n$ (variables and negated variables). The function f is monotone if it is monotone increasing for some choice of $z_1, \ldots, z_n$. It is well known that the optimization of polynomials of degree 2 is NP-hard. Therefore, we consider the class MP(d) of all monotone pseudo-boolean functions whose degree is bounded by d. Wegener and Witt (2003) have proved that a randomized local search heuristic with size bound 1 has a worst-case expected optimization time of $\Theta(2^d \cdot \frac{n}{d} \cdot \log \frac{n}{d})$ on MP(d) and it is interesting to investigate the black-box complexity of MP(d). Our results will show that randomized local search is not far from optimal. Our lower bound holds even for the class MON(d) of monotone monomials (polynomials with one term).

Theorem 7 *The black-box complexity of MON(d) is bounded above by 2^d for the size bound 0 and bounded below by $2^{d-1} + 1/2$. The black-box complexity of MP(d) is bounded above by $O(2^d \log n + n^2)$ for the size bound 3.*

Proof The results on MON(d) are easy. The upper bound follows by asking random queries and the lower bound by Theorem 2 considering the 2^d monomials $z_1 \cdots z_d, z_i \in \{x_i, 1 - x_i\}$.

For the upper bound on MP(d) we start with the following approach which works in the unrestricted scenario. The ith bit of a is called essential if $f(a^i) < f(a)$ where a^i is obtained from a by flipping the ith bit. If a is the search point with the largest f-value (at first some arbitrary a), we determine the essential bits with the queries $a^1, \ldots, a^n$. Afterwards, we ask random queries b where b agrees with a in its essential bits and is chosen uniformly at random at the other positions. If a is not optimal, the expected number of queries to find a better search point b is bounded by $O(2^d)$ since it suffices to activate one further monomial of f. Altogether, there are at most n phases until all bits f depends on essentially are essential. Each phase has an expected length of $O(2^d + n)$ leading to a time bound of $O(2^d n + n^2)$.

The first term can be improved. For the analysis, we may choose at most n monomials containing all variables f essentially depends on. In each step of the search for better search points each of these monomials has a probability of at least 2^{-d} of being activated. Hence, the probability of not activating at least one of the chosen monomials in $c \cdot 2^d \cdot \log n$ queries is bounded above by $1/2$ for some constant c large enough. This implies that the expected number of queries for searching for better search points is even bounded by $O(2^d \log n)$.

Now we show how we can perform the search within a size bound of 3. Our aim is to store the search point a and "as an indicator" the search point b such that $a_i = b_i$ for all essential bits of a and $a_i \neq b_i$, otherwise. Then we can perform the random search fixing all essential bits of a. This search can be successful immediately if $f(b) > f(a)$. We still have to describe how to construct b with limited storage space. First, we only store a and its bitwise complement c. Then we ask the queries $a^1, \ldots, a^n$ only storing the last query in order to identify the index i. Whenever we find a new essential correct bit, e. g., bit i, we ask c^i and replace c by c^i. Finally, $c = b$. We store a, c, and the last query of type a^i. Since a and a^i differ only at position i but a and c differ at least at the positions $i+1, \ldots, n$ we can compute the "role" of each search point if $i < n$. For $i = n$, either a and c differ at one of the positions $1, \ldots, n-1$ or $c = a^n$. In any case, the algorithm works with a size bound of 3. □

8 Unimodal Functions

Typical difficult optimization problems have many local optima which are not globally optimal. Here we investigate the class of unimodal functions $f : \{0,1\}^n \to \mathbb{R}$, i.e., functions where each $a \in \{0,1\}^n$ is globally optimal or has a Hamming neighbor with a better function value. Better means larger since we consider maximization problems. Let $U(b)$ be the class of unimodal functions f which take at most b different function values. There is an obvious $O(nb)$ bound on the black-box complexity of $U(b)$. This bound holds for the following search strategy with size bound 1 (and also for simple evolutionary algorithms). The first search point is chosen uniformly at random. Afterwards, we query a random Hamming neighbor a' of the search point a in the storage. The new search point a' replaces a if $f(a') \geq f(a)$.

Horn, Goldberg, an Deb (1994) have designed unimodal functions where

$b = b(n)$ grows exponentially and where local search heuristics including the one discussed above have an expected optimization time of $\Theta(nb)$. Rudolph (1997) has proved that the function introduced by Horn et al. can be optimized in polynomial expected time by a non-local randomized search strategy. He defined unimodal functions which have been shown to be difficult for mutation-based evolutionary algorithms by Droste, Jansen, and Wegener (1998). Here we investigate the more general scenario of black-box optimization and prove that the black-box complexity of the class of unimodal functions grows exponentially. This extends a similar bound for deterministic strategies proved by Llewellyn, Tovey, and Trick (1989) with an adversary argument. We are also interested in the dependence of the black-box complexity of $U(b)$ on $b = b(n)$.

The proof idea is again an application of Yao's minimax principle. For $U(b)$ we consider functions $f : \{0,1\}^n \to \{0,1,\ldots,b-1\}$. It is difficult to investigate the uniform distribution on the set of these unimodal functions. Therefore, we describe another probability distribution on $U(b)$. The idea is to create a random simple path $P = (p_0,\ldots,p_l)$ where $p_0 = 1^n$, p_{i+1} is a Hamming neighbor of p_i, and $l \le b-n-1$. Then

$$f_P(a) = \begin{cases} n+i & \text{if } a = p_i \\ \text{ONEMAX}(a) & \text{otherwise} \end{cases}$$

is a unimodal function. If a search strategy tries to follow the path, we expect a search time of $\Omega(b-n)$. If $b = 2^{o(n)}$, the set of path points is a sparse set in $\{0,1\}^n$ and it seems to be difficult to find shortcuts.

Simple paths are free of cycles. It is easier to investigate random paths which are not necessarily simple. Let $R = R(L)$ be the following random path $R = (r_0,\ldots,r_L)$. Let $r_0 = 1^n$ and let r_{i+1} be a Hamming neighbor of r_i chosen uniformly at random. The corresponding random simple path $P = P(R)$ starts at $p_0 = r_0$. After having reached $p_i = r_j$ let $k \ge j$ be the largest index where $r_j = r_k$. Then $p_{i+1} := r_{k+1}$. With large probability, R has only short cycles and the Hamming distance of r_i and r_j is quite large if $j-i$ is not too small. These ideas are made precise in the following lemma.

Lemma 1 *For $L(n) = 2^{o(n)}$, $d(n) \le \min\{n, L(n)\}$ and each constant $\beta > 0$ there exists some $\alpha = \alpha(\beta) > 0$ such that*

$$Prob(H(r_i, r_j) \le \alpha \cdot d(n)) = 2^{-\Omega(d(n))}$$

for the random path $R = R(L(n))$ and each (i,j) where $j \ge i + \beta \cdot d(n)$.

Proof Since R is defined by a time-homogeneous Markov chain, it is sufficient to consider the case $i = 0$. Let $H_t := H(r_0, r_t)$. Then $H_{t+1} = H_t + 1$ iff one of the $n - H_t$ bits where r_0 and r_t are equal flips. Hence,

$$\text{Prob}(H_{t+1} = H_t + 1) = 1 - H_t/n$$

and

$$\text{Prob}(H_{t+1} = H_t - 1) = H_t/n.$$

Let $\gamma := \min\{\beta, 1/10\}$ and $\alpha := \gamma/6$. We investigate the subpath $R' = (r_k, \ldots, r_j)$ of R where $k = j - \lfloor \gamma \cdot d(n) \rfloor$.

If $H_k \geq (3/10) \cdot n - \lfloor \gamma \cdot d(n) \rfloor$, then $H_j \geq (3/10) \cdot n - 2\lfloor \gamma \cdot d(n) \rfloor$, since the Hamming distance can decrease at most by 1 per step. By definition, $2 \cdot \gamma \cdot d(n) \leq d(n)/5 \leq n/5$ and $H_j \geq n/10 \geq \alpha \cdot d(n)$.

If $H_k < (3/10) \cdot n - \lfloor \gamma \cdot d(n) \rfloor$, then $H_t < (3/10) \cdot n$ for all $t \in \{k, \ldots, j\}$. Hence, we have $\lfloor \gamma \cdot d(n) \rfloor$ independent steps where the Hamming distance increases with a probability of at least 7/10. By Chernoff bounds, the probability of less than $(3/5) \cdot \lfloor \gamma \cdot d(n) \rfloor$ distance increasing and then more than $(2/5) \cdot \lfloor \gamma \cdot d(n) \rfloor$ distance decreasing steps is bounded above by $2^{-\Omega(d(n))}$. Otherwise, the Hamming distance increases by at least $(1/5) \cdot \lfloor \gamma \cdot d(n) \rfloor > \alpha \cdot d(n)$ for large n. □

Now we are prepared to prove a lower bound on the black-box complexity of $U(b(n)+n)$. We describe the bound with respect to functions with at most $b(n)+n$ different function values since our construction uses n function values for the search points outside the path.

Theorem 8 *The black-box complexity of the class $U(b(n) + n)$ of unimodal functions is bounded below by $\Omega(b(n)/\log^2 b(n))$ if $b(n) = 2^{o(n)}$.*

Proof We apply Yao's minimax principle for the probability distribution on all f_P defined by the experiment described before Lemma 1 where $L(n) = b(n)$. First, we have to investigate the length $l(n)$ of the simple path P based on the random path R of length $L(n)$. Let $d(n) = c \cdot \log b(n)$ for some constant c. Lemma 1 for $\beta = 1$ implies for each i that the probability of the event $r_k = r_i$ for some k where $k - i \geq d(n)$ is bounded above by $b(n) \cdot 2^{-\Omega(d(n))}$. Hence, the probability of a cycle whose length is at least $d(n)$ is bounded above by $b(n)^2 \cdot 2^{-\Omega(d(n))}$ which equals $2^{-\Omega(d(n))}$ if c is chosen large enough. With probability $1 - 2^{-\Omega(d(n))}$, the length of P is at least $b(n)/d(n)$. For all other cases we estimate the search time below by 0.

We investigate a scenario which provides the search process with some additional information and prove the lower bound for this scenario. The knowledge of the heuristic is described by

- the index i such that the prefix $(p_0, \ldots, p_i)$ of P but no further P-point is known and
- the set N of points known to lie outside P.

Initially, $i = 0$ and $N = \emptyset$. The search heuristic asks a query point a. We define the search to be successful if $a = p_k$ and $k \geq i + d(n)$. If the search is not successful, i is replaced by $i + d(n)$ (the points $p_{i+1}, \ldots, p_{i+d(n)}$ are made public) and N is replaced by $N \cup \{a\}$ if a does not belong to P. We prove the theorem by proving that the success probability of each of the first $b(n)/d(n)^2$ steps is bounded by $2^{-\Omega(d(n))}$. If c is chosen large enough, this bounds the success probability of all $b(n)/d(n)^2$ steps by $b(n) \cdot 2^{-\Omega(d(n))} = 2^{-\Omega(d(n))}$, if c is chosen large enough.

The initial situation where $i = 0$ and $N = \emptyset$ is quite simple. Lemma 1 for $\beta = 1$ implies that a search point a where $H(p_0, a) \leq \alpha(1) \cdot d(n)$ has a success probability of $2^{-\Omega(d(n))}$. The proof of Lemma 1 shows the same for a query point a where $H(p_0, a) > \alpha(1) \cdot d(n)$. Even the probability of reaching the Hamming ball with radius $\alpha(1) \cdot d(n)$ around a after at least $d(n)$ steps is bounded by $2^{-\Omega(d(n))}$. This is important later since the knowledge $a \in N$ should not be a useful information to guide the search.

After m unsuccessful queries the heuristic knows the first $m \cdot d(n) + 1$ points on P and knows for the at most m points in N that they are outside P. Let M be the set containing those points where it is known whether they belong to P or not. Let y be the last point of P known to lie on P. We partition M into the set M' of points which are far from y and the set $M'' = M - M'$ of points which are close to y. The point z is far from y if $H(y, z) > \alpha(1) \cdot d(n)$.

First, we analyze the success probability of the next query assuming the following event E. The path starts at y and does not reach a point in M'. By the same arguments as above, $\text{Prob}(E) = 1 - 2^{-\Omega(d(n))}$. For a query point a let A be the event that the search is finished successfully. Then

$$\begin{aligned} \text{Prob}(A|E) &= \text{Prob}(A \cap E)/\text{Prob}(E) \\ &\leq \text{Prob}(A)/\text{Prob}(E) \\ &= \text{Prob}(A) \cdot (1 + 2^{-\Omega(d(n))}). \end{aligned}$$

Hence, the success probability of each query point is still $2^{-\Omega(d(n))}$.

We have to take into account the points in M''. Now we apply Lemma 1 for $\beta = 1/2$. Let z be the point reached after the first $d(n)/2$ steps of P starting at y. The probability that some point from M has a Hamming distance of at most $\alpha(1/2) \cdot d(n)$ from z is bounded by $2^{-\Omega(d(n))}$. Otherwise, all M-points are far from z if far is now defined by a Hamming distance of at least $\alpha(1/2) \cdot d(n)$. Hence, we can apply the above arguments for the following $d(n)/2$ steps. This still gives a bound of $2^{-\Omega(d(n))}$ on the success probability. □

If $b(n) - n = \Omega(n)$, the upper and lower bound on the black-box complexity of $U(b(n))$ differ by a factor of order $n \cdot \log^2 b(n)$. For the case of exponentially increasing $b(n)$ the bounds are quite tight. For small $b(n)$ it is possible to save one $(\log b(n))$-factor. We conjecture that the upper bound $O(n \cdot b(n))$ is optimal, at least for small $b(n)$. In particular, for $b(n) = (3/2)n$, we can consider a random path of length $\lfloor n/2 \rfloor$ starting in $p_0 = 1^n$ by flipping a randomly chosen 1-bit in each step. We conjecture a bound of $\Theta(n^2)$ for black-box algorithms in this scenario.

In many papers and textbooks it is stated that unimodal functions are "easy" for evolutionary algorithms or randomized local search. Our results show that this holds if the size of the image space is bounded but the statement does not hold in the general setting.

9 Black-Box Complexity and Multi-Objective Optimization

In black-box optimization, the modeling of the problem is essential. It has influence on the information revealed by queries. The SSSP problem leads to a needle-in-the-haystack problem, if it is considered as a single-objective optimization problem (see Section 4). The real objective is to compute $n-1$ shortest paths although these paths can be described by one tree. Hence, it seems to be more appropriate to consider the problem as a minimization problem with $n-1$ objectives. More precisely, we consider $f_D \colon \mathcal{T} \to \mathbb{R}^{n-1}$ where $\mathcal{T}$ contains all trees on $V = \{1, \ldots, n\}$ rooted at $s := n$ and $f_D(T)$ equals the vector $(d_1^T, \ldots, d_{n-1}^T)$ such that d_i^T is the cost of the unique s-i-path in T. Scharnow, Tinnefeld, and Wegener (2002) have analyzed a simple

evolutionary algorithm in this scenario. It has an expected optimization time of $O(n^3)$ for the size bound 1.

Theorem 9 *The black-box complexity of the SSSP in the description as multi-objective optimization problem is at least $n/2$ and at most $2n-3$. The upper bound can be obtained by a black-box algorithm with size bound 2.*

Proof For the lower bound, we consider the uniform distribution on all distance matrices where $d_{i,i-1} = 1$, if $3 \leq i \leq n$, $d_{k1} = 1$ for some $k \in \{2, \ldots, n\}$ and $d_{i,j} = \infty$, otherwise. For each search point, i.e., each tree T rooted at s and each $i \in \{2, \ldots, n-1\}$, we have $d_i^T = n - i$, if T contains the path $s, n-1, \ldots, i$, and $d_i^T = \infty$, otherwise. These values do not contain any information on k. Moreover, $d_1^T = n - k + 1$, if T contains the path $s, n-1, \ldots, k, 1$, and $d_1^T = \infty$, otherwise. Hence, we are in the situation where we search for the value of k which is chosen uniformly at random from $\{2, \ldots, n\}$ and where we may try one value per step. This is a needle in a small haystack of size $n-1$ and we get the lower bound $n/2$ from Theorem 1.

For the upper bound, we simulate Dijkstra's famous algorithm. This algorithm works in n rounds. Let $d_{\text{opt}}(i)$ be the length of a shortest s-i-path. Dijkstra's algorithm finds a vertex i_k in round k such that $d_{\text{opt}}(i_1) \leq d_{\text{opt}}(i_2) \leq \cdots \leq d_{\text{opt}}(i_n)$. We can implement Dijkstra's algorithm in such a way that $d_{\text{opt}}(i_j) = d_{\text{opt}}(i_{j+1})$ implies $i_j < i_{j+1}$. After round k, we like to know a tree $T(k)$ containing s-i-paths of minimal length under the restriction that only $i_1, \ldots, i_k$ are allowed as intermediate vertices. Such a tree allows the computation of $i_1, \ldots, i_{k+1}$ and the computation of $T'(k)$ containing the same s-i-paths for all $i \in \{i_1, \ldots, i_{k+1}\}$ and the edges (i_{k+1}, i) for all $i \notin \{i_1, \ldots, i_{k+1}\}$. Comparing $T(k)$ and $T'(k)$ and the corresponding $d^{T(k)}$- and $d^{T'(k)}$-values, we can simulate Dijkstra's algorithm to compute $T(k+1)$. Moreover, $T(1)$ consists of all edges (s, i), $i \neq s$. The number of queries is bounded above by $2n-3$, since $T(n-1)$ is optimal. It is sufficient to store $T(k)$ and $T'(k)$ or $T'(k)$ and $T(k+1)$. Then we have the problem of recomputing k and i_{k+1}. The only vertex which is a leaf in $T(k)$ and an inner node in $T'(k)$ is i_{k+1}. Knowing this node and its rank in the list of path lengths we can compute k. This is enough to compute $T(k+1)$. Then we store $T'(k)$ and $T(k+1)$. If $T'(k) \neq T(k+1)$, there is a vertex i with different s-i-path lengths in $T'(k)$ and $T(k+1)$. Its path length in $T(k+1)$ is smaller. Hence, we can identify $T'(k)$ and the predecessor of i in $T'(k)$ is the special vertex i_{k+1}. We can identify k by considering the position of

$d^{T'(k)}(i)$ among all $d^{T'(k)}$-values. This information is enough to determine i_{k+2} and to compute $T'(k+1)$. Since we can distinguish $T'(k)$ and $T(k+1)$, we are able to store $T(k+1)$ and $T'(k+1)$ after having queried $T'(k+1)$. If $T'(k) = T(k+1)$, we do not query $T(k+1)$. We know $T(k)$, $T'(k)$, and, therefore, also k. Since we also know $T(k+1) = T'(k)$, we directly compute $T'(k+1)$ and store afterwards $T(k+1)$ and $T'(k+1)$. □

10 Conclusions

We have identified randomized search heuristics as black-box algorithms since they do not use the information about the problem instance for their computations. All popular randomized search heuristics work moreover with a limited information transfer between the rounds of the algorithm. This leads to the scenario of black-box problems. We have proved lower bounds in the unrestricted black-box scenario. In some cases, these bounds are strong since they are close to upper bounds for well-known search heuristics. In other cases, the bounds are close to optimal with respect to upper bounds in the unrestriced scenario. There are several problems where we conjecture that the corresponding bounds in the scenario with size bounds like a constant or linear are much larger. Such bounds can reveal more on the limits of popular randomized search heuristics. The proof of these bounds is open for future research.

References

[1] Cormen, T.H., Leiserson, C.E., and Rivest, R.L. (1990). *Introduction to Algorithms.* MIT Press.

[2] Droste, S., Jansen, T., Tinnefeld, K., and Wegener, I. (2003). A new framework for the valuation of algorithms for black-box optimization. In K. A. de Jong, R. Poli, and J. E. Rowe (Eds.): *Foundations of Genetic Algorithms* 7 (FOGA), 253–270, Morgan Kaufmann, San Francisco.

[3] Droste, S., Jansen, T., and Wegener, I. (1998). On the optimization of unimodal functions with the (1+1) evolutionary algorithm. Proc. of Parallel Problem Solving from Nature (PPSN V), LNCS 1498, 47–56.

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

[5] Giel, O. and Wegener, I. (2003). Evolutionary algorithms and the maximum matching problem. Proc. of 20th Symp. on Theoretical Aspects of Computer Science (STACS), LNCS 2607, 415–426.

[6] Glover, F. and Laguna, M. (1993). Tabu search. In C.R. Reeves (Ed.): *Modern Heuristic Techniques for Combinatorial Problems,* 70–150, Blackwell, Oxford.

[7] Graham, R.L., Knuth, D.E., and Patashnik, O. (1994). *Concrete Mathematics.* Addison-Wesley.

[8] Hajnal, P. (1991). An $\Omega(n^{4/3})$ lower bound on the randomized complexity of graph properties. Combinatorica 11, 131–143.

[9] Heiman, R., Newman, I., and Wigderson, A. (1993). On read-once threshold formulae and their randomized decision tree complexity. Theoretical Computer Science 107, 63–76.

[10] Heiman, R. and Wigderson, A. (1991). Randomized vs. deterministic decision tree complexity for read-once boolean functions. Computational Complexity 1, 311–329.

[11] Horn, J., Goldberg, D. E., and Deb, K. (1994). Long path problems. Proc. of Parallel Problem Solving from Nature (PPSN III), LNCS 866, 149–158.

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

[13] Llewellyn, D.C., Tovey, C., and Trick, M. (1989). Local optimzation on graphs. Discrete Applied Mathematics 23, 157–178.

[14] Lovász, L., Naor, M., Newman, I., and Wigderson, A. (1991). Search problems in the decision tree model. Proc. of 32nd IEEE Symp. on Foundations of Computer Science (FOCS), 576–585.

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

[16] Papadimitriou, C.H., Schäffer, A.A., and Yannakakis, M. (1990). On the complexity of local search. Proc. of 22nd ACM Symp. of Theory of Computing (STOC), 438–445.

[17] Petterson, O. and Moffat, A. (1995). A framework for adaptive sorting. Discrete Applied Mathematics 59, 153–179.

[18] Rabani, Y., Rabinovich, Y., and Sinclair, A. (1998). A computational view of population genetics. Random Structures and Algorithms 12, 314–330.

[19] Rudolph, G. (1997). How mutation and selection solve long-path problems in polynomial expected time. Evolutionary Computation 4, 195–205.

[20] Sasaki, G. and Hajek, B. (1988). The time complexity of maximum matching by simulated annealing. Journal of the ACM 35, 387–403, 1988.

[21] Scharnow, J., Tinnefeld, K., and Wegener, I. (2002). Fitness landscapes based on sorting and shortest paths problems. Proc. of Parallel Problem Solving from Nature (PPSN VII), LNCS 2439, 54–63.

[22] Wegener, I. (2001). Theoretical aspects of evolutionary algorithms. Proc. of 28th Int. Colloquium on Automata, Languages and Programming (ICALP), LNCS 2076, 64–78.

[23] Wegener, I. and Witt, C. (2003). On the optimization of monotone polynomials by simple randomized search heuristics. To appear in Combinatorics, Probability and Computing.

[24] Yao, A.C. (1977). Probabilistic computations: Towards a unified measure of complexity. Proc. of 17th IEEE Symp. on Foundations of Computer Science (FOCS), 222–227.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'