The Unrestricted Black-Box Complexity of Jump Functions

Maxim Buzdalov mbuzdalov@gmail.com
ITMO University, 49 Kpohbepkckuŭ npocnekt, Saint Petersburg, 197101, Russia

Benjamin Doerr doerr@lix.polytechnique.fr
LIX, École Polytechnique, 91126 Palaiseau Cedex, France

Mikhail Kever mikhail.kever@gmail.com
ITMO University, 49 Kpohbepkckuŭ npocnekt, Saint Petersburg, 197101, Russia

doi:10.1162/EVCO_a_00185

Abstract

We analyze the unrestricted black-box complexity of the JUMP function classes for different jump sizes. For upper bounds, we present three algorithms for small, medium, and extreme jump sizes. We prove a matrix lower bound theorem which is capable of giving better lower bounds than the classic information theory approach. Using this theorem, we prove lower bounds that almost match the upper bounds. For the case of extreme jump functions, which apart from the optimum reveal only the middle fitness value(s), we use an additional lower bound argument to show that any black-box algorithm does not gain significant insight about the problem instance from the first $\Omega(\sqrt{n})$ fitness evaluations. This, together with our upper bound, shows that the black-box complexity of extreme jump functions is $n + \Theta(\sqrt{n})$.

Keywords

Black-box complexity, jump functions, information theory.

1 Introduction

To understand how evolutionary algorithms (and other black-box optimizers as well) behave when optimizing certain functions, one proves upper bounds for the problem's difficulty (by constructing and studying various algorithms) and lower bounds (by studying how fast an algorithm can be in principle), which complement each other. Comparing these bounds helps researchers to evaluate how good today's heuristics are and sometimes to construct better algorithms (Doerr et al., 2015).

In this work, we study the second question, that is, how fast in principle a black-box optimization algorithm can solve certain optimization problems. Droste et al. (2003) were the first to ask this question in the context of evolutionary algorithms. In their seminal paper—see also Droste et al. (2006) for the journal version—they introduce the notion of black-box complexity as a measure of problem difficulty. In simple words, the black-box complexity of an optimization problem is the (expected) number of function evaluations that are needed by an optimal black-box algorithm until it queries an optimum for the first time. As many randomized search heuristics such as evolutionary algorithms, ant colony optimization, or simulated annealing are black-box optimizers, the black-box complexity of a problem gives a lower bound on performance of all these search heuristics.

Manuscript received: August 15, 2015; revised: March 10, 2016; accepted: May 5, 2016.

While dormant for several years, the area of black-box complexity became very active from 2009 on, possibly spurred by the remarkable works of Anil and Wiegand (2009) as well as of Lehre and Witt (2010). Since then, several deep and surprising results on black-box complexities were achieved, so that now we reasonably well understand the black-box complexities of classic test functions, for example, $\Theta(n/\log n)$ for OneMax (Droste et al., 2006; Anil and Wiegand, 2009) or $\Theta(n \log\log n)$ for LeadingOnes (Afshani et al., 2013) and of several combinatorial optimization problems like sorting, maximum clique, and the single-source shortest path problem (Droste et al., 2006), the minimum spanning tree problem (Doerr et al., 2013) and the partition problem (Doerr et al., 2014b). It was also observed that modified definitions of black-box complexity are able to study the influence of unbiasedness (Lehre and Witt, 2012; Rowe and Vose, 2011), ranking-basedness (Doerr and Winzen, 2014b), memory size (Doerr and Winzen, 2014a), parallel search (Badkobeh et al., 2014), or elitism (Doerr and Lengler, 2015).

In this article, we stay within the realm of classic black-box complexity of pseudo-Boolean functions; that is, we ask how many fitness evaluations an otherwise unrestricted algorithm needs to perform to find the optimum of a function $f : \{0,1\}^n \to \mathbb{R}$ (given in a black-box fashion) from a given problem class. With the OneMax test function class and the LeadingOnes class being studied, we turn to another important Jump test function class. These are test functions used as examples with scalable difficulty, because the fitness landscape has a large plateau of low fitness around the optimum. For a jump function with the jump size ℓ, this plateau consists of all search points with the Hamming distance from the optimum between 1 and ℓ. See Section 2 for a precise definition. Our motivation is both understanding the black-box complexity of this well-studied function class and using it as a trigger to develop a new method, in particular, to prove lower bounds for black-box complexities, where at the moment not much is known beyond the information theoretic argument of Droste et al. (2006).

Concerning the black-box complexities of jump functions, we observe that while jump functions tend to be difficult for many randomized search heuristics, their black-box complexity is not excessively large (for the restricted notion of unbiased black-box complexity, weaker and less precise results pointing in the same direction have been obtained in Doerr et al. (2014a) (see Section 2). We show that when the jump parameter ℓ satisfies $\ell < \frac{n}{2} - \sqrt{n}\log_2 n$, the black-box complexity satisfies the same upper bound of $(1+o(1))\frac{2n}{\log_2 n}$ that is the best known bound for the easy OneMax test function class. Note that $\ell = \frac{n}{2} - \sqrt{n}\log_2 n$ is actually quite large, meaning that all search points with distance between 1 and $\frac{n}{2} - \sqrt{n}\log_2 n - 1$ from the optimum lie on the plateau of low fitness, making this a plateau of size $2^{n(1-o(1))}$ and diameter $\Theta(n)$. For even larger jump sizes, that is, $\frac{n}{2} - \sqrt{n}\log_2 n \le \ell < \lfloor n/2 \rfloor - \omega(1)$, we show an upper bound of $(1+o(1))\frac{n}{\log_2(n-2\ell)}$, where the asymptotic notation refers to $n - 2\ell$ tending to infinity. This upper bound does not make precise the leading constant when $n - 2\ell$ is constant, in particular, not for the extreme case when the jump function has all fitness levels on the plateau except for the "middle level" (for even n) or except for the two middle levels $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ (for odd n). For such *extreme* jump functions (and thus also for all others), we show an upper bound of $n + \Theta(\sqrt{n})$.

These upper bounds are asymptotically of the right order of magnitude. This follows from the information theoretic argument (Theorem 2) in Droste et al. (2006): an optimization problem over a search space S such that each element of S is the unique solution to an instance of the problem has a black-box complexity of at least $\lceil \log_k |S| \rceil - 1$, where k is the maximum number of different answers a query can have. For jump functions with the jump size ℓ, this gives a lower bound of $\lceil \log_{n+1-2\ell}(2^n) \rceil - 1 =$

$(1 + o(1))n/\log_2(n - 2\ell)$ with the asymptotics being with respect to $n - 2\ell$ tending to infinity. Consequently, when $n^{0.5+o(1)} \leq \ell \leq o(n)$, then our upper bounds and this lower bound are identical up to the leading constant. For smaller ℓ, they differ by at most a factor of $2(1 + o(1))$. This is the same gap that exists for the black-box complexity of OneMax, which is an open problem pointed out (in a different context) already fifty years ago in the famous paper by Erdős and Rényi (1963). For constant-size values of $n - 2\ell$, we also see a substantial gap between our upper bounds and the information theoretic lower bound. For example, in the case of an extreme jump function for even n, we have the three different fitness values $0, n/2$, and n as possible answers to queries, and hence the lower bound is $(\log_3 2) \cdot n$. For odd n, we have the four fitness values $0, \lfloor n/2 \rfloor, \lceil n/2 \rceil$, and n, giving a lower bound of $(\log_4 2) \cdot n = 0.5n$ only. In both cases, our upper bound is $n + \Theta(\sqrt{n})$ and thus quite far away.

The reason is that the information theoretic argument pretends that at all times, all k answers may occur, and moreover, occur with similar frequency. This is clearly an overly optimistic view, as the following three examples show:

1. Once an optimal solution is found, the search is stopped. Hence such nodes of the decision tree have no children, even though some show up in a close distance from the root.
2. In an optimization problem with each search point being the unique optimum of exactly one instance, typically the different answers to a query do not occur with similar frequency as the optimal answer occurs at most once.
3. Correlations between the answers may also lead to a smaller information gain than assumed by the information theoretic lower bound. For simplicity, let us regard the OneMax problem, but the same effect exists for jump functions, most notable (as we will see) for extreme jump functions when n is odd. When the answer to the first query is received, we can predict the parity of answers for all subsequent queries, knowing only the parity of the number of one-bits in a query. Thus, when analyzing algorithms for solving this problem, we may safely assume that for every query (except for the first one), once the parity of one-bits is known, the number of different answers is reduced from $n + 1$ to at most $\lceil \frac{n+1}{2} \rceil$.

We significantly extend the information theoretic bound to allow taking care of such reasons for a smaller information gain, that is, to exploit these to prove stronger lower bounds. Our *matrix lower bound theorem* gives improved lower bounds for all black-box complexities of jump functions. In particular, for extreme jump functions, we raise the lower bounds from $(\log_3 2) \cdot n$ to $n - 1$ when n is even and from $0.5n$ to $n - 2$ when n is odd. Note that the larger number of fitness values for n odd does now have a much smaller influence on the result than when using the classic information theoretic bound. While these results do determine the leading constant, we could not prove with our general lower bound theorem that the $\sqrt{n}$ term in the upper bound $n + \Theta(\sqrt{n})$ is necessary. For this, we add an extra argument that shows that, as our upper bound suggests, indeed when optimizing an extreme jump function, the first $\Theta(\sqrt{n})$ fitness evaluations typically reduce the size of the solution space only by a constant factor. This argument together with the matrix lower bound proves that the black-box complexity of extreme jump function is indeed $n + \Theta(\sqrt{n})$.

The rest of the article is structured as follows. In Section 2, we make precise the definitions of black-box complexity and jump functions. We also summarize the state of the art concerning optimization and complexity of jump functions. Section 3 is dedicated

to upper bounds on JUMP which are proven by giving the corresponding algorithms and discussing their complexity. In Section 4, the matrix lower bound theorem is formulated and proven. It resembles Theorem 2 from Droste et al. (2006), but is able to give better lower bounds under certain conditions. We apply this theorem in Section 5 to the JUMP problem. Section 6 describes the refinement of the lower bound in the case of extreme jump sizes. Section 7 concludes the article, among others, with some open problems.

A preliminary version of a larger part of these results has appeared as a conference paper of Buzdalov et al. (2015). Apart from making a more detailed introduction, which was impossible in the conference paper format, we have refined the lower bound for extreme jump sizes up to $n + \Theta(\sqrt{n})$ to match the upper bound (see Section 6), which seems to be impossible by a straightforward application of the matrix lower bound theorem and required substantial additional effort.

2 Preliminaries

In this section, we define the notion of unrestricted black-box complexity, we make precise the definition of the jump functions we regard, and we review the existing results in runtime analysis and black-box complexity for jump functions.

2.1 Unrestricted Black-Box Complexity

The notion of black-box complexity was introduced by Droste et al. (2003) to investigate how difficult a problem is to be solved via general-purpose randomized search heuristics like evolutionary algorithms. In simple words, this black-box complexity (now sometimes called unrestricted black-box complexity to distinguish it from later developed more restricted variants) is the number of fitness evaluations needed to solve a problem. Let us make this notion precise.

By a *problem*, we shall always mean a set $\mathcal{F}$ of pseudo-Boolean functions $f : \{0,1\}^n \to \mathbb{R}$ (*problem instances*), with the implicit meaning that the task is to find a global maximum of such a function. A black-box algorithm A for $\mathcal{F}$ is a (possibly randomized) algorithm that takes as input a function $f \in \mathcal{F}$ and tries to maximize it in a black-box fashion. This means that the algorithm has no access to an explicit description of f, but can only evaluate f at any search point $x \in \{0,1\}^n$. In this unrestricted black-box optimization model, we do not make any other limiting assumption on the computational power of the algorithm. In particular, the algorithm may store all previously regarded search points and their fitnesses and may conduct arbitrary computations with these.

As a performance measure, similar as for evolutionary algorithms, we take the number of fitness evaluations performed by the algorithm. More precisely, the cost $T(A, f)$ of running A on f is the expected number of function evaluations performed until (and including this evaluation) for the first time a global maximum of f is evaluated. We take a worst-case view with respect to the problem instances and define the complexity of A by

$$T(A, \mathcal{F}) = \sup_{f \in \mathcal{F}} T(A, f).$$

The black-box complexity of the problem $\mathcal{F}$, as in classic algorithmics, now is the performance of the best possible algorithm for the problem, that is,

$$T(\mathcal{F}) = \inf_{A} T(A, \mathcal{F}),$$

where A runs over all black-box algorithms for $\mathcal{F}$.

Let us give a brief example to illustrate these definitions. The most classic test function in evolutionary computation is the so-called ONEMAX function $\textsc{OneMax} : \{0,1\}^n \to$

$\mathbb{R}; x \mapsto \sum_{i=1}^{n} x_i$, which simply counts the number of one-bits in x. To make this one instance a reasonable problem, in particular, to prevent discussing algorithms exploiting the obvious fact that OneMax has $x^* = (1, \dots, 1)$ as the unique optimum, we regard the problem consisting of all instances with fitness landscape isomorphic to OneMax. These are all functions $\text{OneMax}_{n,z}$, $z \in \{0,1\}^n$, which are defined by

$$\text{OneMax}_{n,z} : \{0,1\}^n \to \mathbb{R}; x \mapsto |\{i \in [1..n] \mid x_i = z_i\}|, \tag{1}$$

that is, $\text{OneMax}_{n,z}$ counts the number of bit positions in which x and z agree. Again, it is obvious that z is the unique optimum of $\text{OneMax}_{n,z}$; however; this is nontrivial to find out for an algorithm having no access to an explicit representation of $\text{OneMax}_{n,z}$ like Equation (1).

Let OneMax_n be the set of all $\text{OneMax}_{n,z}$, $z \in \{0,1\}^n$. Let A be the randomized local search heuristic (start with a random search point and then repeat flipping one bit, evaluating the new search point, and keeping the better of parent and offspring). It is an easy exercise to show that A finds the optimum of any function $\text{OneMax}_{n,z}$ in $O(n \log n)$ fitness evaluations. Hence $T(A, \text{OneMax}_{n,z}) = O(n \log n)$ for all z. Consequently, $T(A, \text{OneMax}_n) = O(n \log n)$ and thus $T(\text{OneMax}_n) = O(n \log n)$. It is much harder to prove that there is a better black-box algorithm A for OneMax_n that has a complexity $T(A, \text{OneMax}_n) = O(n/\log n)$, a result shown in the context of information theory (Erdős and Rényi, 1963), the context of the Mastermind guessing game (Chvátal, 1983), and in the evolutionary computation community (Anil and Wiegand, 2009). The first and second references as well as Droste et al. (2006) also prove that for any black-box algorithm for OneMax_n, there is a function $\text{OneMax}_{n,z}$ such that $T(A, \text{OneMax}_{n,z}) = \Omega(n/\log n)$. Consequently, $T(\text{OneMax}_n) = \Theta(n/\log n)$.

It is an ongoing discussion whether this is the best definition of a problem difficulty for randomized search of evolutionary computation and several alternative definitions have been defined, none of which, however, could clearly be shown to be more appropriate. We refer the reader to the works cited in the introduction or the tutorial (Doerr and Doerr, 2014).

2.2 Jump Functions

The Jump function is another popular test function. While OneMax is used to analyze how evolutionary algorithms cope with easy optimization problems, Jump is intended to be a difficult function where elitist optimization needs to flip several bits at once. There are several similar definitions of jump functions. They all have in common that they are a OneMax function modified by giving all search points within a certain radius ℓ from the optimum a low fitness. Consequently, typical hill-climbers find it easy to come close to the optimum (namely up to a search point with Hamming distance $\ell + 1$ from the optimum), but then have to struggle to jump over the valley of low fitness to the optimum.

For black-box complexity analyses, the following definition of jump functions was suggested in Doerr, Kötzing et al. (2011). For $\ell \in [1; \lfloor \frac{n}{2} \rfloor - 1]$ and $z \in \{0,1\}^n$, let

$$\text{Jump}_{n,\ell,z}(x) = \begin{cases} n & \text{if } \text{OneMax}_{n,z}(x) = n \\ \text{OneMax}_{n,z}(x) & \text{if } \ell < \text{OneMax}_{n,z}(x) < n - \ell \\ 0 & \text{otherwise} \end{cases}$$

for all $x \in \{0,1\}^n$. We define the class $\text{Jump}_{n,\ell}$ to consist of all function $\text{Jump}_{n,\ell,z}$ with $z \in \{0,1\}^n$. The special case of $\ell = \lfloor \frac{n}{2} \rfloor - 1$, which is the maximum possible ℓ that does not zero out the middle fitness values (thus giving the so-called Needle function), is called *extreme* Jump.

Our definition of the jump functions, which is also used in Doerr et al. (2014a, 2016) not only has the ℓ highest suboptimal fitness levels of the OneMax function blanked out, but also the fitness levels $1, \dots, \ell$. This is to avoid the trivial solution that a black-box algorithm first minimizes the jump function with a performance essentially equal to optimizing OneMax and, once a solution x with fitness 0 is found, inverts x and (with high probability, as the algorithm might have started in a blanked-out region with nonzero probability) finds the optimum. A very similar definition for jump functions, also in the context of black-box complexity, was used in Lehre and Witt (2010). The only difference there is that for a jump value of ℓ, the authors blank out only the highest $\ell - 1$ nonoptimal fitness values (but on the lower end, the fitness values $1, \dots, \ell$, that is, ℓ levels, are blanked out). For contexts outside of complexity theory, usually one does not care about the fitness levels at the low end, because a heuristic building on the hill-climbing paradigm will rarely encounter these levels, and if so, not profit from having the exact fitness available. Such a definition was used, for example, in Droste et al. (2002). Recently, Jansen (2015) introduced yet another type of jump function that, roughly speaking, agrees with the definition of Droste et al. (2002) except that the optimum is located at an unknown point in the plateau (or deceptive region) that is a Hamming ball of radius $\ell - 1$. Thus this function class contains $\binom{n}{\ell-1}$ nonisomorphic fitness landscapes (whereas the previous classes contain just a single isomorphism type). This function class has a black-box complexity of order $\Theta(n^{\ell-1})$ when ℓ is constant, very different from the previously discussed classes. It may be closer to what is the original intuition of the jump functions; however, several results other than black-box complexity results seem to fail for this class as well, for example, the proof that crossover can greatly speed up the optimization of jump functions (Jansen and Wegener, 2002).

2.3 Runtime Analysis and Black-Box Complexity of Jump Functions

Jump functions are generally difficult to optimize with classic randomized search heuristics. This was first shown by Droste et al. (2002) for their definition of jump functions and the $(1+1)$ evolutionary algorithm, but it is quite easy to see that also many other classic randomized search heuristics have an optimization time of $\Omega(n^{\ell+1})$ for any jump function definition that has the ℓ highest suboptimal fitness levels on OneMax reset to low fitness values. Jansen and Wegener (2002) showed that a steady state genetic algorithm using uniform crossover (however, only with rate $O(\frac{1}{n \log n})$) can optimize the jump functions defined in Droste et al. (2002) in time $O(n^2 \log n)$ when ℓ is constant. It is easy to see that this result also holds for all other definitions of jump functions discussed previously, with the exception of the definition of Jansen (2015). Here, in fact, his proof that the black-box complexity is $\Omega(n^{\ell-1})$ implies that the use of crossover cannot give similar advantages as for the other types of jump functions.

While there are some black-box complexity analyses for jump functions, surprisingly none of them uses the most classic unrestricted black-box complexity model; in fact, they all use the unbiased model introduced in Lehre and Witt (2012). Here, only the restricted class of unbiased black-box algorithms is regarded. While the precise definition is nontrivial, roughly speaking, a black-box algorithm is called unbiased if it satisfies three properties:

1. Each search point must be chosen uniformly at random or must be created from previous search points via a variation operator.

2. All variation operators must be unbiased, that is, treat the bit-values 0 and 1 in a symmetric way and treat the bit positions $1, \dots, n$ in a symmetric way.
3. All actions of the algorithm apart from what happens inside a variation operator may depend only on the search history and the fitness values observed, but not on the bit-string representation of the individuals.

This algorithm model includes many classic randomized search heuristics, except, for example, those using one-point crossovers.

For this unbiased black-box complexity model, Doerr et al. (2014a, 2016) show the following results. If $\ell \leq (0.5 - \varepsilon)n$, $\varepsilon > 0$ an arbitrary constant, then the black-box complexity is $\Theta(n/\log n)$. In the case of extreme jump functions for even n, that is $\ell = 0.5n - 1$, the black-box complexity is $\Theta(n)$. It is clear that these results are valid for the unrestricted black-box complexity as well; however, our results are stronger in that they provide much more precise bounds (making the leading constant precise and in the case of extreme jump functions also the lower order terms up to order $\sqrt{n}$) and regard wider ranges of ℓ (namely all values of ℓ).

3 Upper Bounds for the Black-Box Complexity of $\textsc{Jump}_{n,\ell}$

In this section, the upper bounds for JUMP are considered. To ease reading in the main subsections, in Section 3.1 we collect several useful technical results. The subsequent Sections 3.2–3.4 then treat separately small, large, and extreme jump sizes ℓ.

3.1 Preliminaries

LEMMA 1: *For sufficiently large n, for all $t \geq (1 + \frac{4\log_2 \log_2 n}{\log_2 n})\frac{2n}{\log_2 n}$ and the cases of all even $d \in [2; n]$, it holds that $\binom{n}{d}(\binom{d}{d/2}2^{-d})^t \leq 2^{-3t/4}$.*

PROOF: This is proven in Doerr, Johannsen, et al. (2011) as Statement 8. □

LEMMA 2: *For sufficiently large n, for a fixed $z \in \{0,1\}^n$, for $\ell < n/2 - \sqrt{n}\log_2 n$ and for $x \in \{0,1\}^n$ taken uniformly at random, the probability for $\textsc{Jump}_{n,\ell,z}(x)$ to be zero is at most $2e^{-2(\log_2 n)^2}$.*

PROOF: The value of $\textsc{OneMax}_{n,z}(x)$ for a random x has a binomial distribution with parameters n and $p = 1/2$. From Hoeffding's inequality (see Hoeffding, 1963 or Theorem 1.11 in Doerr, 2011), for $k \leq np$, the distribution function for binomial distribution $F_{n,p}(k)$ is bound from above by $e^{-2\frac{(np-k)^2}{n}}$. As a consequence, the probability for $\textsc{Jump}_{n,\ell,z}(x)$ to be zero is at most $2F_{n,1/2}(\ell) \leq 2e^{-2\frac{(n/2-\ell)^2}{n}} \leq 2e^{-2\frac{(\sqrt{n}\log_2 n)^2}{n}} = 2e^{-2(\log_2 n)^2}$. □

The following result extends the black-box complexity analysis of Doerr, Johannsen, et al. (2011) for ONEMAX to JUMP functions.

THEOREM 1: *Assume that n is sufficiently large and $\ell < n/2 - \sqrt{n}\log_2 n$. Let $z \in \{0,1\}^n$. Let X be a (multi-)set of $t \geq (1 + \frac{4\log_2 \log_2 n}{\log_2 n})\frac{2n}{\log_2 n}$ elements from $\{0,1\}^n$ chosen randomly using uniform distribution and mutually independently. Then the probability that there exists a $y \in \{0,1\}^n$ such that $y \neq z$ and $\textsc{Jump}_{n,\ell,z}(x) = \textsc{Jump}_{n,\ell,y}(x)$ for all $x \in X$, is at most $2^{-t/4}$.*

PROOF: We define A_d as a set of points which differ from z in exactly d positions, where $0 \leq d \leq n$.

We say that a point $y \in \{0,1\}^n$ *agrees* with $x \in X$ if $\textsc{Jump}_{n,\ell,z}(x) = \textsc{Jump}_{n,\ell,y}(x)$. This means that $\textsc{Jump}_{n,\ell,z}(x) = \textsc{Jump}_{n,\ell,y} = 0$ or $\textsc{OneMax}_{n,y}(x) = \textsc{OneMax}_{n,z}(x)$. The probability of the former does not exceed $2e^{-2\left(\log_2 n\right)^2}$ by Lemma 2. The latter holds if and only if x and y, as well as x and z, differ in exactly half of the bits in which y and z differ. To sum up, if $y \in A_d$, the probability for y to agree with a random x is at most $2e^{-2\left(\log_2 n\right)^2}$ for an odd d and at most $2e^{-2\left(\log_2 n\right)^2} + \binom{d}{d/2}2^{-d}$ for an even d. As for large enough n it holds that $2e^{-2\left(\log_2 n\right)^2} \le \left(2^{1/4} - 1\right)\binom{d}{d/2}2^{-d}$, the latter is at most $2^{1/4}\binom{d}{d/2}2^{-d}$.

Let p be the probability that there exists an $y \in \{0,1\}^n \setminus \{z\}$ such that y agrees with all $x \in X$. Then

$$
\begin{aligned}
p &= \Pr\left(\bigcup_{y\in\{0,1\}^n\setminus\{z\}} \bigcap_{x\in X} (y \text{ agrees with } x)\right) \\
&\le \sum_{y\in\{0,1\}^n\setminus\{z\}} \Pr\left(\bigcap_{x\in X} (y \text{ agrees with } x)\right) \\
&= \sum_{d=1}^{n} \sum_{y\in A_d} \prod_{x\in X} \Pr(y \text{ agrees with } x) \\
&\le \sum_{d \text{ even}} \binom{n}{d} \left(2^{1/4}\binom{d}{d/2}2^{-d}\right)^t + \sum_{d \text{ odd}} \binom{n}{d} \left(2e^{-2\left(\log_2 n\right)^2}\right)^t \\
&= \sum_{d \text{ even}} \binom{n}{d} \left(2^{1/4}\binom{d}{d/2}2^{-d}\right)^t + 2^{n-1} \left(2e^{-2\left(\log_2 n\right)^2}\right)^t .
\end{aligned}
$$

After applying Lemma 1, we obtain

$$
p \le \frac{n+1}{2} 2^{t/4} 2^{-3t/4} + 2^{n-1+t} e^{-2t\left(\log_2 n\right)^2},
$$

which is less than $2^{-t/4}$ for sufficiently large n. □

3.2 Upper Bound for Smaller ℓ

Using Theorem 1, we easily find the following upper bound for the unrestricted black-box complexity of jump functions with small jump size $\ell < n/2 - \sqrt{n}\log_2 n$. This bound is asymptotically equal to the best known upper bound for $\textsc{OneMax}_n$. Also note that, trivially, any lower bound for the black-box complexity of $\textsc{OneMax}_n$ also holds for any jump function class with problem size n. Consequently, the best known lower bound for $\textsc{OneMax}_n$, which is a factor of $2(1 + o(1))$ below these upper bounds, also holds for the jump functions regarded in this subsection. In a sense, these results show that for solving OneMax, only the inner $2\sqrt{n}\log_2 n$ fitness levels are sufficient.

THEOREM 2: *If $\ell < n/2 - \sqrt{n}\log_2 n$, the unrestricted black-box complexity of $\textsc{Jump}_{n,\ell}$ is at most $(1 + o(1))\frac{2n}{\log_2 n}$, where $o(1)$ refers to $n \to \infty$.*

PROOF: We use the same algorithm which is used in Doerr, Johannsen, et al. (2011) for proving the upper bound for OneMax. We select randomly and independently t queries such that $t \ge (1 + \frac{4\log_2\log_2 n}{\log_2 n})\frac{2n}{\log_2 n}$ and check if there exists a single optimum z which agrees with all these queries (a query q with an answer a agrees with an

```
1: function LargeJump(n, ℓ, f ∈ Jump_{n,ℓ})
2:     k ← ⌊n/2⌋ − ℓ − 1
3:     s ← max{w | √w log_2 w < k; w is even}
4:     τ ← ⌈n/s⌉
5:     repeat
6:         x ← Uniform()
7:     until f(x) = ⌊n/2⌋
8:     B ← [1; n]
9:     for i ∈ [1; τ) do
10:        repeat
11:            b_i ← ChooseSubsetRandomlyUniformly(B, s)
12:        until f(FlipBits(x, b_i)) = ⌊n/2⌋
13:        B ← B \ b_i
14:    end for
15:    b_τ ← B
16:    ω_0 ← x
17:    for i ∈ [1; τ] do
18:        α_i ← SmallJumpProjection(b_i, x)
19:        ω_i ← SetAtPositions(b_i, ω_{i−1}, α_i)
20:    end for
21:    return ω_τ
22: end function
```

Figure 1: Algorithm for $\textsc{Jump}_{n,\ell}$ with $\frac{n}{2} - \sqrt{n}\log_2 n \le \ell < \lfloor \frac{n}{2} \rfloor - \omega(1)$.

optimum z if $\textsc{Jump}_{n,\ell,z}(q) = a$). If there is more than one possible z, then we repeat the whole procedure. The complexity of one invocation of this procedure equals t. The probability of not finding a unique optimum is at most $2^{-t/4}$ by Theorem 1. Thus the complexity of the algorithm is at most $\frac{t}{1-2^{-t/4}} = (1 + o(1))\frac{2n}{\log_2 n}$. □

It is not difficult to see that, with slightly more care, Theorems 1 and 2 can be shown to hold also for larger values of ℓ, however, not for $\ell = \frac{n}{2} - \Theta(\sqrt{n})$. For this reason, we will not follow this way. Instead, in the next subsection we propose an algorithm that works for even larger ℓ, but gives the same results that an extension of the previous theorem would have given in the small range above $\ell = \frac{n}{2} - \sqrt{n}\log_2 n$, where such an extension would have been possible.

3.3 Upper Bound for Larger ℓ

For bigger ℓ, finding an optimum for $\textsc{Jump}_{n,\ell}$ can be reduced to finding optima of several jump functions with smaller dimensions and jump sizes, for which the algorithm from the previous section suffices.

Theorem 3: *For $\frac{n}{2} - \sqrt{n}\log_2 n \le \ell < \lfloor \frac{n}{2} \rfloor - \omega(1)$, the unrestricted black-box complexity of $\textsc{Jump}_{n,\ell}$ is at most $(1 + o(1))\frac{n}{\log_2(n-2\ell)}$, where $o(1)$ and $\omega(1)$ refer to $(n - 2\ell) \to \infty$.*

Proof: Let $k := \lfloor \frac{n}{2} \rfloor - \ell - 1 \neq 0$. We reduce our problem to the one of $\textsc{Jump}_{s,\frac{s}{2}-k-1}$, where s is chosen such that $\sqrt{s}\log_2 s < k$. The algorithm is outlined in Figure 1.

First, the algorithm finds a maximum even s such that $\sqrt{s}\log_2 s < k$, which would allow applying Theorem 2 for solving $\textsc{Jump}_{s,\frac{s}{2}-k-1}$. After that, the algorithm finds a

string $x \in \{0,1\}^n$ with exactly $\lfloor \frac{n}{2} \rfloor$ correct bits using random queries. The probability that $\textsc{Jump}_{n,\ell}$ is equal to $\lfloor \frac{n}{2} \rfloor$ for a random query is $2^{-n}\binom{n}{\lfloor n/2 \rfloor}$ which is $\Theta\left(\frac{1}{\sqrt{n}}\right)$ by the Stirling's formula. This means that the string x can be found in $\Theta(\sqrt{n})$ queries.

After finding x, the algorithm splits all bit indices into sets of size s in such a way that x and the answer agree in exactly half of the bits. To be precise, the last such set may have $s' < s$ indices, in which case x and the answer agree in $\left\lfloor \frac{s'}{2} \right\rfloor$ of its positions. This is done in lines 8–15 in Figure 1, where b_i is the i-th such set. B, the set of yet undistributed bits, is always such that x and the answer agree in exactly $\lfloor |B|/2 \rfloor$ indices of B.

To do that, the algorithm generates random subsets of size s and checks if they contain exactly $\frac{s}{2}$ correct bits, which is done by flipping the bits from the chosen subset and checking whether the fitness remains equal $\lfloor \frac{n}{2} \rfloor$. If $|B| = m$, the probability of choosing such a subset is

$$
\begin{aligned}
p &= \binom{\lfloor m/2 \rfloor}{s/2}\binom{\lceil m/2 \rceil}{s/2}\binom{m}{s}^{-1} = \frac{\lfloor m/2 \rfloor!\,\lceil m/2 \rceil!\, s!\,(m-s)!}{\lfloor (m-s)/2 \rfloor!\,\lceil (m-s)/2 \rceil!\,(s/2)!} \\
&= \binom{s}{s/2}\binom{m-s}{\lfloor (m-s)/2 \rfloor}\binom{m}{\lfloor m/2 \rfloor}^{-1} = \Theta\left(\frac{\frac{2^s}{\sqrt{s}}\frac{2^{m-s}}{\sqrt{m-s}}}{\frac{2^m}{\sqrt{m}}}\right) \\
&= \Theta\left(\frac{\sqrt{m}}{\sqrt{s}\sqrt{m-s}}\right) = \Omega\left(\frac{1}{\sqrt{s}}\right).
\end{aligned}
$$

This gives an $O(\sqrt{s})$ bound for one subset selection and an $O(n/\sqrt{s})$ bound on entire process of finding subsets.

Next, the algorithm separately optimizes bits from each of the subsets b_i using the algorithm for small $\textsc{Jump}$ from Theorem 2 (lines 17–20 in Figure 1). If every query for a subproblem on bits from b_i is forwarded to the main function f with all bits not from b_i taken from x, the resulting subproblem becomes exactly a $\textsc{Jump}_{|b_i|,\frac{|b_i|}{2}-k-1}$ problem with the following corrections:

- from all nonzero answers, a value of $\left\lfloor \frac{n-|b_i|}{2} \right\rfloor$ needs to be subtracted;
- at the optimum of the subproblem, zero will be returned.

The latter correction, however, does not change the algorithm very much, because the algorithm from Theorem 2 doesn't actually query the optimum point. Line 19 in Figure 1 collects the partial answers one by one: it sets the bits of a_i at the corresponding positions from b_i to the previous partial answer ω_{i-1} and returns the updated value.

Assuming $n = qs + r$, $0 < r \le s$, the complexity of the algorithm is at most:

$$
O(\sqrt{n}) + O\left(\frac{n}{\sqrt{s}}\right) + q(2+o_s(1))\frac{s}{\log_2 s} + (2+o_r(1))\frac{r}{\log_2 r} = \frac{(2+o_s(1))n}{\log_2 s}.
$$

However, due to the choice of s, it holds that $\log_2 s = (2+o_k(1))\log_2 k$, which finally results in $\frac{(1+o_{n-2\ell}(1))n}{\log_2(n-2\ell)}$. □

3.4 Upper Bound for Extreme Jump

The algorithm from Theorem 3 cannot be applied to the case of extreme $\textsc{Jump}$ function, because then k would be zero and the subproblems are extreme $\textsc{Jump}$ problems as well.

In this case we have to use another algorithm, which will be given in the proof of the following theorem.

THEOREM 4: *The unrestricted black-box complexity of an extreme* JUMP *problem is at most* $n + \Theta(\sqrt{n})$.

PROOF: As described in the proof of Theorem 3, one can find a point x such that $f(x) = \lfloor \frac{n}{2} \rfloor$ in $\Theta(\sqrt{n})$ queries. After that, if one flips two bits, the value of f remains the same if and only if one of these bits was correct and the other was not.

We denote by $a \oplus b$ the bitwise exclusive OR of the bit strings a and b having equal lengths. The algorithm queries $f(x \oplus 10^{i-2}10^{n-i})$ for all $i \in [2; n]$, and if it equals $\lfloor \frac{n}{2} \rfloor$, the value of b_i is set to one, otherwise to zero. This results in $n - 1$ queries. After that, if the first bit is correct, then $x \oplus 0b_2 \dots b_n$ is the answer, otherwise its inverse is the answer. One has to make a single query, $f(x \oplus 0b_2 \dots b_n)$, to determine which one is true. The complexity of this algorithm is $n + \Theta(\sqrt{n})$. □

4 The Matrix Lower Bound Theorem

In this section we present a new theorem which is similar to Theorem 2 from Droste et al. (2006) except that the nodes corresponding to queries are required to be split in several types.

THEOREM 5: *Let S be the search space of an optimization problem, and for each $s \in S$ there exists an instance such that s is a unique optimum. Let each query have one of T types, such that for any query q of the i-th type the following holds:*

- *there is exactly one answer to the query q which means that q is an optimum;*
- *there are at most $A_{i,j}$ answers such that the next query after such answer belongs to the j-th type.*

Define $B_{i,j}$, $1 \le i, j \le T + 2$, to be a matrix such that:

- $B_{i,j} = A_{j,i}$ *for* $1 \le i, j \le T$ *(note the transposition);*
- $B_{T+1,j} = 1$ *for* $1 \le j \le T + 1$*;*
- $B_{T+2,j} = 1$ *for* $1 \le j \le T + 2$*;*
- $B_{i,j} = 0$ *otherwise.*

Let the first ever query in the optimization process be of type 1. Define $V(d) = B^d \cdot (1, 0, \dots, 0)^T$ to be a vector, $C(d) = V(d)_{T+1}$, $S(d) = V(d)_{T+2}$. Then the following statements are true:

1. *$C(d)$ is the maximum total number of possible queries with depth in $[1; d]$, where depth of a root is equal to one.*
2. *The lower bound on the average depth of N nodes is $d + 1 - \frac{S(d)}{N}$ where d is an integer such that $C(d) \le N \le C(d + 1)$.*
3. *The unrestricted black-box complexity of the considered optimization problem is not less than the lower bound on average depth of $|S|$ nodes.*

PROOF: According to the Yao's minimax principle (Yao, 1977), the expected runtime of a randomized algorithm on any input is not less than the average runtime of the best deterministic algorithm over all possible inputs. Thus we construct a lower bound on the complexity of a randomized algorithm by constructing a lower bound on the average performance of any deterministic algorithm over all possible inputs. A deterministic algorithm can be represented as a (rooted) decision tree with nodes corresponding to queries and arcs going downwards corresponding to answers to these queries. A total lower bound on the average performance of deterministic algorithms, just as in Droste et al. (2006), is done by assigning $|S|$ different queries to different nodes of a tree such that their average depth is minimized, and then by considering all such trees and taking a minimum over them.

It should be noted that, if a (fixed) set of queries is to be assigned to nodes of a (fixed) rooted tree such that the average depth of these queries is minimized, an optimal assignment can be constructed in a greedy way: each query should be assigned to a free node with the minimum possible depth. Assume that an optimal assignment does not use at least one node a with depth d while using at least one node b with depth $d' > d$. Then one can move a query from the node b to the node a, which decreases the average depth, so the initial assignment is, in fact, not optimal.

Next, we show that, in order to minimize the average depth, one needs to consider only the complete tree, that is, a tree where for any query of the i-th type, for any j there are exactly $A_{i,j}$ answers, each leading to a query of the j-th type. Indeed, if an optimal assignment can be done for an incomplete tree, it can be done for the complete tree as well, because all the nodes of any incomplete tree are preserved in the complete tree.

For a complete tree with the constraints determined by the matrix A (as specified in the theorem's statement) and with the root vertex of type 1, the number of vertices of type i and depth d (the root has the depth equal to 1) is exactly $((A^T)^{d-1} \cdot (1, 0, \ldots, 0)^T)_i$. In the matrix B, the next-to-last row is designed to collect the sum of all numbers of vertices at all previous depths (which is exactly how $C(d)$ is defined), and the last row, in a similar manner, collects $S(d)$—the sum of $C(i)$'s for all $1 \le i \le d$. In a more explicit way, $S(d)$ can be expressed as:

$$S(d) = \sum_{i=1}^{d} L(i) \cdot (d + 1 - i),$$

where $L(i)$ is the number of vertices of all types residing at the depth i, so the expression $C(d) \cdot (d + 1) - S(d)$ is actually the sum of depths of all vertices up to the depth d:

$$C(d) \cdot (d + 1) - S(d) = \sum_{i=1}^{d} L(i) \cdot i,$$

and the expression $d + 1 - \frac{S(d)}{C(d)}$ is thus exactly the average depth of all such vertices.

If we consider arbitrary integer N, we can find an integer d such that $C(d) \le N \le C(d + 1)$. In this case, the total sum of depths of the first $C(d)$ vertices is $C(d) \cdot (d + 1) - S(d)$, and the next $N - C(d)$ vertices have the depth of $d + 1$. The average depth is thus:

$$d_{\text{avg}}(N) = \frac{C(d) \cdot (d + 1) - S(d) + (d + 1) \cdot (N - C(d))}{N} = d + 1 - \frac{S(d)}{N}.$$

□

It is difficult to use this theorem straightaway, because the lower bound on the average depth of N vertices is not defined only in terms of N and the matrix A, but

additionally requires to find which depth d fulfils $C(d) \le N \le C(d+1)$. However, for several common usages it is possible to make it more convenient.

THEOREM 6: *If there is only one type of query in Theorem 5, and $A_{1,1} = k$ such that $k \ge 2$, then for the search space S the lower bound on the average depth is at least $\left\lfloor log_k(1 + |S|(k-1)) \right\rfloor - \frac{1}{k-1}$.*

PROOF: The value of $B^d \cdot (1,0,0)^T$ yields the following result (intermediate computations omitted):

$$\begin{pmatrix} k & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}^d \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} k^d \\ \frac{k^d-1}{k-1} \\ \frac{k^{d+1}-k-d(k-1)}{(k-1)^2} \end{pmatrix}.$$

One can see that $C(d) = \frac{k^d-1}{k-1}$ and $S(d) = \frac{k^{d+1}-k-d(k-1)}{(k-1)^2}$.

Consider an equality $N = C(d) = \frac{k^d-1}{k-1}$. It follows that:

$$d(N) = \log_k(1 + N(k-1)).$$

As for a given N, we need to find an integer d such that $C(d) \le N < C(d+1)$; we need to round it down: $d = \lfloor d(N) \rfloor$.

Note that, if $d \ge 1$ and $k \ge 1$, $S(d)$ grows when d grows, as $S(d)' > 0$.

The expression for a lower bound on the average depth of N queries is at most:

$$\begin{aligned} d_{\text{avg}}(N) &= \lfloor d(N) \rfloor + 1 - \frac{S(\lfloor d(N) \rfloor)}{N} \ge \lfloor d(N) \rfloor + 1 - \frac{S(d(N))}{N} \\ &\ge \left\lfloor \log_k(1 + N(k-1)) \right\rfloor - \frac{1}{k-1}. \end{aligned}$$

□

Note that the classical result from Droste et al. (2006), the $\lceil \log_{k+1} N \rceil - 1$ lower bound, is actually not greater than the given bound (note that $k+1$ differs from k used in the definition of Droste et al., 2006 because in the current context k does not include the answer corresponding to the optimum). Indeed, for $k \ge 2$:

$$\begin{aligned} &\log_k(1 + N(k-1)) - \log_{k+1} N > \log_k(N(k-1)) - \log_{k+1} N \\ &= \log_k N - \log_{k+1} N + \log_k(k-1) > \log_k N - \log_{k+1} N > 0. \end{aligned}$$

For the case of $k = 1$, the lower bound is even stronger.

THEOREM 7: *If there is only one type of query in Theorem 5, and $A_{1,1} = 1$, then for the search space S the lower bound on the average depth is at least $(|S|+1)/2$.*

PROOF: In this case one can show that $C(d) = d$ and $S(d) = \frac{d^2+d}{2}$. The average depth for N is $N + 1 - \frac{N^2+N}{2N} = N + 1 - (N+1)/2 = (N+1)/2$. □

5 Lower Bounds for $\textsc{Jump}_{n,\ell}$

First, let's apply Theorem 6 immediately to the JUMP problem.

THEOREM 8: *For any n and $\ell < n/2$, the unrestricted black-box complexity of $\textsc{Jump}_{n,\ell}$ is at least $\left\lfloor log_{n-2\ell}(1 + 2^n(n - 2\ell - 1)) \right\rfloor - \frac{1}{n-2\ell-1}$.*

PROOF: In $\textsc{Jump}_{n,\ell}$, the search space has a size of 2^n. There are $n - 2\ell + 1$ possible answers to a query, but one of them terminates the search process immediately, so $k = n - 2\ell$. The result follows straightaway from Theorem 6. □

THEOREM 9: *The unrestricted black-box complexity of extreme* JUMP *for even n is at least* $n - 1$.

PROOF: It follows from Theorem 8 by assuming $n - 2\ell = 2$. □

The presented bounds are already an improvement over the currently known bounds (say, $\frac{n}{\log_2 3}$ for extreme JUMP and even n, as follows from Droste et al., 2006). However, for odd n Theorem 8 reports $\lfloor \log_3(1 + 2^{n+1}) \rfloor - 1/2$, which is still quite far away from the best known algorithms. Fortunately, the JUMP problem possesses a particular property, which can be used to refine the lower bounds using Theorem 5 with *two types* of queries.

THEOREM 10: *For* $\text{JUMP}_{n,\ell}$, *define an answer to the query to be nontrivial if it is neither 0 nor n. After receiving the first nontrivial answer for every subsequent query it is possible to determine a priori the parity of any nontrivial answer.*

PROOF: Consider the optimum and a query. We introduce the following values:

- q_{00}: number of positions with zeros in both the optimum and the query;
- q_{01}: number of positions with zeros in the optimum and ones in the query;
- q_{10}: number of positions with ones in the optimum and zeros in the query;
- q_{11}: number of positions with ones in both the optimum and the query.

The number of zeros in the optimum modulo 2, which is $q_{00} \oplus q_{01}$, is fixed. The number of ones in the query modulo 2 is $q_{01} \oplus q_{11}$, and the answer to the query modulo 2 is $q_{00} \oplus q_{11}$. The following equality holds:

$$(q_{01} \oplus q_{11}) \oplus (q_{00} \oplus q_{11}) = q_{00} \oplus q_{01},$$

which means that the parity of the nontrivial answer is uniquely determined by the parity of the number of ones in the query.

As a result, if an algorithm receives the first nontrivial answer, all subsequent queries will probably have fewer possible answers. □

Using Theorem 10, we can define two types of queries to use with Theorem 5, namely, the queries that happened before and after a nontrivial answer.

THEOREM 11: *The unrestricted black-box complexity of* $\text{JUMP}_{n,\ell}$ *for odd n is at least:*

$$\left\lfloor \log_{\frac{n-2\ell+1}{2}} \left(2^{n-2}(n - 2\ell - 1) + 1\right) \right\rfloor - \frac{2}{n - 2\ell - 1}.$$

PROOF: For odd n there are $n - 2\ell + 1 = 2k + 2$ possible answers: one answer equal to 0, one answer equal to n, and k pairs of nontrivial answers. For Theorem 5, the first type of query has $2k + 1$ nonterminating answers, and the second type of query, which occurs after one of $2k$ nontrivial answers is received from a query of the first type, has only $k + 1$ nonterminating answers. The value of $B^d \cdot (1, 0, 0, 0)^T$ is thus:

$$B^d \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2k & k+1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}^d \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 2\left((k+1)^d - 1\right) \\ \frac{2(k+1)^d - dk - 2}{k} \\ \frac{2(k+1)^{d+1} - \frac{(dk+2)^2 + dk^2 + 4k}{2}}{k^2} \end{pmatrix}.$$

A problem of defining d in terms of N is more difficult this time: as $C(d) = \frac{2(k+1)^d - dk - 2}{k}$, the equality $N = C(d)$ cannot be easily solved in terms of d. Instead, we introduce a function $d(N)$ such that the following equality holds:

$$N = \frac{2(k+1)^{d(N)} - d(N)k - 2}{k}.$$

We find the lower bound on the average depth $d_{\text{avg}}(N)$, keeping in mind that $S(d)$ grows as d grows and that $d(N) \geq 1$ for $N \geq 1$:

$$\begin{aligned}
d_{\text{avg}}(N) &= \lfloor d(N) \rfloor + 1 - \frac{S(\lfloor d(N) \rfloor)}{N} \geq \lfloor d(N) \rfloor + 1 - \frac{S(d(N))}{N} \\
&= \lfloor d(N) \rfloor + 1 - \frac{\frac{2(k+1)^{1+d(N)} - \frac{(d(N)k+2)^2 + d(N)k^2 + 4k}{2}}{k^2}}{\frac{2(k+1)^{d(N)} - d(N)k - 2}{k}} \\
&= \lfloor d(N) \rfloor + 1 - \frac{\frac{2(k+1)^{1+d(N)} - d(N)k^2 - d(N)k - 2k - 2 - \frac{d(N)k^2(d(N)-1)}{2}}{k}}{\frac{2(k+1)^{1+d(N)} - d(N)k^2 - d(N)k - 2k - 2}{k+1}} \\
&\geq \lfloor d(N) \rfloor + 1 - \frac{k+1}{k} = \lfloor d(N) \rfloor - \frac{1}{k}.
\end{aligned}$$

We can also obtain a good lower bound on $d(N)$ by throwing out the $d(N)k$ part in the definition of $d(N)$ above, which leads to $d(N) > \log_{k+1}\left(\frac{Nk}{2} + 1\right)$. Together, $d_{\text{avg}}(N) \geq \left\lfloor \log_{k+1}\left(\frac{Nk}{2} + 1\right) \right\rfloor - \frac{1}{k}$. For $\text{JUMP}_{n,\ell}$, it holds that $N = 2^n$ and $2k + 2 = n - 2\ell + 1$, which constitutes:

$$\left\lfloor \log_{\frac{n-2\ell+1}{2}}\left(2^{n-2}(n - 2\ell - 1) + 1\right) \right\rfloor - \frac{2}{n - 2\ell - 1}.$$

□

THEOREM 12: *The unrestricted black-box complexity of extreme* JUMP *for odd n is at least* $n - 2$.

PROOF: For extreme JUMP and odd n, $n - 2\ell + 1 = 4$. Then from Theorem 11 it follows that the lower bound is at least:

$$\left\lfloor \log_2\left(2^{n-2} \cdot 2 + 1\right) \right\rfloor - \frac{2}{2} = \left\lfloor \log_2\left(2^{n-1} + 1\right) \right\rfloor - 1 \geq n - 2.$$

□

THEOREM 13: *The unrestricted black-box complexity of* $\text{JUMP}_{n,\ell}$ *for even n is at least:*

$$\left\lfloor \log_{\frac{n-2\ell+2}{2}}\left(1 + 2^{n-1}\frac{(n-2\ell)^2}{n - 2\ell - 1}\right) \right\rfloor - \frac{2}{n - 2\ell}.$$

PROOF: For even n there are $n - 2\ell + 1 = 2k + 3$ possible answers ($k \geq 0$): one answer equal to 0, one answer equal to n, one answer equal to $n/2$, and k more pairs of nontrivial answers. For Theorem 5, the first type of query has $2k + 2$ nonterminating answers, and the second type of query can have either $k + 1$ or k nonterminating answers, depending on the parity of the number of ones in a query. As we cannot predict the parity for all possible algorithms, the maximum number of queries is limited to $k + 1$. The matrix B has the following form:

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2k+1 & k+2 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

We omit the intermediate computations and just state that:

$$C(d) = \frac{(2k+1)(k+2)^d - dk^2 - dk - 2k - 1}{(k+1)^2}$$

$$S(d) = \frac{(k+2)^d(2k^2+5k+2) - \frac{d^2k^3+dk^3+2d^2k^2+6dk^2+4k^2+2d+d^2k+7dk+10k+4}{2}}{(k+1)^3}.$$

Following the same approach as in the proof of Theorem 11, we define $d(N)$ such that $C(d(N)) = N$ and produce the following lower bound:

$$d_{\text{avg}}(N) \geq \lfloor d(N) \rfloor - \frac{1}{k+1}.$$

The lower bound on $d(N)$ can be achieved from the value of $C(d)$ by throwing out the $dk^2 + dk$ part, which yields:

$$d(N) \geq \log_{k+2}\left(1 + \frac{(k+1)^2 N}{2k+1}\right)$$

and, together:

$$d_{\text{avg}}(N) \geq \left\lfloor \log_{k+2}\left(1 + \frac{(k+1)^2 N}{2k+1}\right)\right\rfloor - \frac{1}{k+1}.$$

Substitution of N with 2^n and $2k+2$ with $n - 2\ell$ proves the theorem. □

Note that Theorem 13 does not improve the bound for extreme JUMP and even n—it remains equal to $n-1$ when one sets $k = 0$—because in this case the number of possible answers does not change after receiving the first nontrivial answer.

6 Refining the Lower Bound for Extreme Jump Sizes

For extreme jump functions, we gave a black-box algorithm finding the optimum in time $n + \Theta(\sqrt{n})$, whereas the lower bound stemming from the matrix lower bound theorem proposed in Section 4 was $n - O(1)$. The gap between these two bounds is relatively small compared to most black-box complexity results—recall that, for example, for the ONEMAX problem not even the leading constant in the $\Theta(n/\log n)$ complexity is known. Nevertheless, it is an interesting question whether the $\sqrt{n}$ term in the upper bound is necessary or only stems from insufficient proof methods. So far, the upper bound proof suggests that black-box algorithms optimizing extreme jump functions have an initial phase of $\Theta(\sqrt{n})$ rounds in which they gain relatively little information about the optimum. Only once they have found a search point with nonzero fitness, they become more efficient and solve the problem in roughly n additional iterations.

In this section, we show that the $\sqrt{n}$ term is indeed necessary; that is, the black-box complexity of the extreme jump functions is $n + \Omega(\sqrt{n})$. Our proof also shows that it indeed cannot be avoided that the first $\Omega(\sqrt{n})$ fitness evaluations do not reveal much information about the optimum.

The remainder of this section is organized as follows. In Section 6.1, we reduce the problem of finding a lower bound for extreme JUMP to a minimization problem over decision trees having a particular structure. This is where we exploit particular properties of optimizing extreme jump functions. In Section 6.2, we solve this minimization problem in a general form via a recursive argument. In Section 6.3, we minimize the obtained lower bound by choosing the best possible value for the parameter t of the minimization problem. Finally, in Section 6.4, we derive from the results of Section 6.3 our improved lower bounds for the black-box complexity of extreme jump functions.

6.1 Representing a Deterministic Algorithm for Extreme JUMP

Consider an extreme jump problem, for simplicity here for even n.[1] As we have argued in Section 4 already, a lower bound for the black-box complexity can be obtained by regarding the best average performance a deterministic black-box algorithm can have (where the average is taken over all instances, here over all extreme jump functions over $\{0,1\}^n$). Hence, let us consider a deterministic black-box algorithm for the extreme jump problem (and argue that it cannot have a too good average performance).

Before starting this argument, we remind the reader that the extreme jump functions problem has the properties that (i) for each point of the search space $\{0,1\}^n$ there is exactly one problem instance having this as optimal solution and (ii) all instances have unique optimal solutions. Consequently, we can (and will) identify the problem instances with their unique optima.

A deterministic black-box algorithm gives rise (and in fact is equivalent) to the following type of decision tree. In a *decision tree* for a search space S (recall that we use $S = \{0,1\}^n$ as representatives of the (unique) extreme jump functions with these optima), each node is labeled with a subset $S_v \subseteq S$ ("remaining search space"); the root is labeled with S. Each internal node v is also labeled with a query $q_v \in \{0,1\}^n$. If q_v has the answer i to at least one $s \in S_v$, then v has an outgoing edge (v,w) to a node w labeled with all $s \in S_v$ having the answer i to the query q_v. Consequently, the labels S_w of the children w of v form a partition of S_v. A node with ingoing edge labeled with an optimal answer (here n) has no children. These are the only nodes without children; this last requirement stems from the fact that we do not only require the black-box algorithm to "know" the optimal solution, but also to query it.

It is clear that each deterministic black-box algorithm gives rise to a decision tree and that each decision tree describes a black-box algorithm. By our observation that there is a bijection between the extreme jump functions and their unique optima, we see that at each leaf the set of possible solutions contains just a single element and that the average performance of the algorithm on a random instance is exactly the average depth of the leaves of the tree. Consequently, the black-box complexity of the extreme jump functions is the smallest possible average depth of the leaves of a decision tree.

By analyzing the structure of the decision trees for the extreme jump problem, we shall show improved lower bounds for their black-box complexity. To this aim, note first that if a query q gets the answer $\frac{n}{2}$, one knows that the remaining search space contains only the binary strings of length n which have exactly $\frac{n}{2}$ common bits with the query q. There are at most $\binom{n}{n/2}$ such strings.[2] From this innocent observation, we derive that the decision tree cannot be very balanced. The crucial property to look at is the length of the maximal path starting at the root of the tree with all edges labeled with the answer 0. Figure 2 shows a decision tree for extreme JUMP with this path highlighted as the trunk of the tree. Note that this trunk is formed by queries which are made before any nonzero answer is received. The branches which are above the trunk in Figure 2 are used when the answer of n is received, in these cases the algorithm immediately stops. The branches below the trunk correspond to the cases when the answer of $\frac{n}{2}$ is received.[3] The values of s_i are the remaining search space sizes for the corresponding branches. These

[1] In this section, the footnotes will explain how the proof is adapted to odd n where necessary.

[2] For odd n, there are two such answers, $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$, each leaving the room for $\binom{n}{\lfloor n/2 \rfloor}$ binary strings.

[3] For odd n, each vertex has two children below the trunk corresponding to the answers $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$.

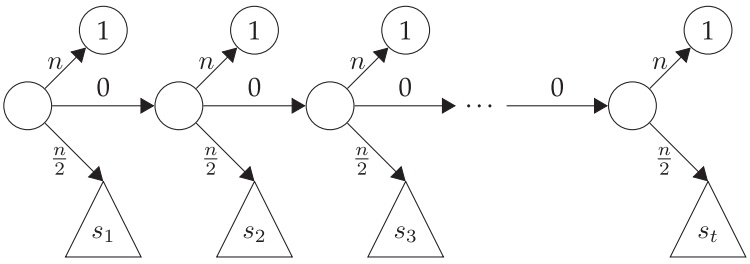

Figure 2: A decision tree of a deterministic algorithm to solve extreme JUMP for even n. For odd n, the branches below the trunk appear in pairs for answers $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$.

values satisfy $1 \le s_i \le \binom{n}{n/2}$ as follows from above. The value of t is the total number of queries in the trunk, that is, the t-th query in the trunk cannot have an answer of 0.

Let the total search space size be s, then $\sum_{i=1}^{t} s_i + t = s$. If we fix both s and t, any lower bound $l(s, t)$ on the average depth of the decision tree of an *optimal* deterministic algorithm with the fixed t will be a lower bound on the expected runtime of any black-box search algorithm with the fixed t. After that, we can optimize $l(s, t)$ for fixed s by varying t to find the lower bound for any black-box search algorithm for solving the extreme JUMP.

The average depth of a subtree of a size s_i is at least $\lfloor \log_2(s_i + 1) \rfloor - 1$, which follows from Theorem 6. To obtain the desired lower bound, we minimize the average depth of the entire tree by finding the proper values for s_i. In a general form, this minimization problem is formalized and solved in Section 6.2.

6.2 Lower Bound for Fixed s and t in General Form

In this section, we construct a lower bound for a function $f(s, t)$, which requires that $0 \le s - t \le tQ$ and integer $t \ge 1$, where Q is a certain positive value. This function is defined as follows:

$$f(s,t) = \begin{cases} 1 + (s-1)\log_2 s & \text{if } t = 1 \\ s + \min\limits_{u \in U(s,t,Q)} \left(f(s-u-1, t-1) + u \log_2 (u+1) \right) & \text{if } t > 1, \end{cases}$$

where $U(s, t, Q) = [\max(0, s - t - (t-1)Q); \min(Q, s - t)]$. The aim of this function is to represent the sum of depths of all queries for the given s and t in a tree from Figure 2, provided that s_i are assigned optimally. The bounds for the parameter u in the minimization clause come from the domain of $f(s - u - 1, t - 1)$ and the fact that u is actually the value for s_1, which cannot be negative and cannot exceed $\binom{n}{n/2}$, the latter[4] is rewritten as Q.

For the sake of brevity, we define a function $\mathrm{lbp}(x, p) = x \log_2 (2^p \cdot x + 1)$ and a function $\mathrm{lbp}^+(x, p) = x \log_2 (2^p \cdot x + 2)$. Note that $\mathrm{lbp}^+(x, p + 1) = x + \mathrm{lbp}(x, p)$ and $\mathrm{lbp}^+(x, p) > \mathrm{lbp}(x, p)$. As $\mathrm{lbp}(x, p)$ is convex downwards for every fixed p, the following lemma holds.

[4]For odd n, the corresponding Q is $2\binom{n}{\lfloor n/2 \rfloor}$. For the sake of the current bound, the two branches for the answers $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$, each with the same maximum size $\binom{n}{\lfloor n/2 \rfloor}$, can be safely glued together.

LEMMA 3: *For $0 \le u \le x$ and $p \le 0$, the following inequality holds:*

$$lbp(x-u,\,p) + lbp(u,\,0) \ge \frac{2^p+1}{2^p} lbp\left(\frac{2^p}{2^p+1}x,\,0\right) = x\,log_2\left(\frac{2^p}{2^p+1}x+1\right)$$

which turns to equality at $u = \frac{2^p}{2^p+1}x$.

PROOF: We rewrite $\mathrm{lbp}(x-u,\,p)$ as $2^{-p}\cdot\mathrm{lbp}(2^p(x-u),0)$. By Jensen's inequality,

$$\begin{aligned}
2^{-p}\mathrm{lbp}(2^p(x-u),0) + \mathrm{lbp}(u,0) &= \frac{2^p+1}{2^p}\left(\frac{\mathrm{lbp}(2^p(x-u),0)}{2^p+1} + \frac{2^p\mathrm{lbp}(u,1)}{2^p+1}\right)\\
&\ge \frac{2^p+1}{2^p}\mathrm{lbp}\left(\frac{2^p(x-u)}{2^p+1} + \frac{2^p}{2^p+1}u,\,0\right)\\
&= \frac{2^p+1}{2^p}\mathrm{lbp}\left(\frac{2^p}{2^p+1}x,\,0\right).
\end{aligned}$$

□

The following theorem gives a lower bound for $f(s,t)$ which we use later.

THEOREM 14: *For $t \ge 2$, the following lower bound holds:*

$$f(s,t) \ge \frac{t(t+1)}{2} + \begin{cases} lbp(s-t,\,0) & \text{if } 0 \le s-t \le 2Q \\ \sum_{i=1}^{x} lbp(Q,i) + lbp(s-t-xQ,\,x) & \text{if } (x+1)Q \le s-t \le (x+2)Q. \end{cases}$$

PROOF: We prove this theorem using induction by t.

Induction base: $t = 2$. By definition:

$$\begin{aligned}
f(s,2) &= s + \min_{u\in[\max(0,s-2-Q);\min(Q,s-2)]}\big(f(s-u-1,0) + \mathrm{lbp}(u,0)\big)\\
&= s+1+\min_{u\in\ldots}\big(\mathrm{lbp}(s-u-2,0) + \mathrm{lbp}(u,0)\big).
\end{aligned}$$

By Lemma 3, the minimization clause gets minimum at $u = \frac{s-2}{2}$. As $0 \le s-2 \le 2Q$ by definition of f, u is a feasible minimum point, so:

$$f(s,2) \ge s+1+(s-2)\log_2\left(\frac{s-2}{2}+1\right) = 3+(s-2)\log_2 s > 3+\mathrm{lbp}(s-2,0).$$

Induction step: $t > 2$. Figure 3 shows the feasible region for u and the regions corresponding to the same pieces from the definition of $f(s-u-1,t-1)$ on the Cartesian plane with axes of $\frac{s-t}{Q}$ and $\frac{u}{Q}$.

We prove the induction step by parts which correspond to parts from the definition of $f(s,t)$.

1. For $0 \le s-t \le 2Q$ the proof follows exactly the same scheme as in the induction base.
2. Consider $(x+1)Q \le s-t \le (x+2)Q$ and $s-t-(x+1)Q \le u \le Q$. These constraints correspond to lower triangles in Figure 3, examples of which are shown as triangle A and triangle C. The triangle C corresponds to the situation when $x > 1$ and $x-1$ is used as x for $f(s-u-1,t-1)$. If $x = 1$, which corresponds to the triangle A, the first clause from the definition of $f(s-u-1,t-1)$ should be used; however, in this case it is precisely equal to what would happen if $x = 0$ is substituted to the second clause. Hence we can evaluate both cases (triangles A and C) simultaneously.

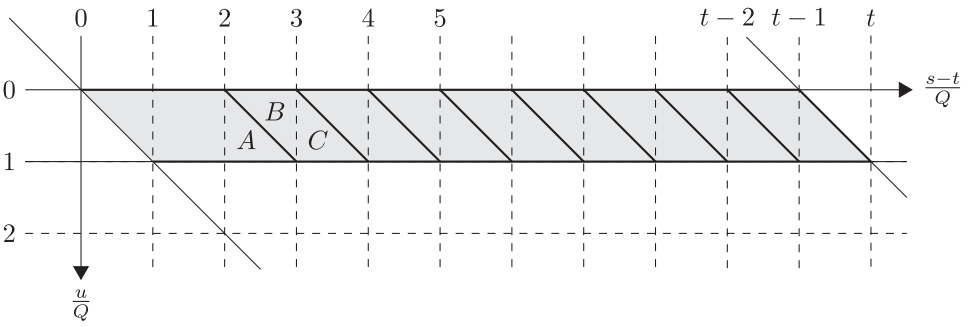

Figure 3: Feasible region for u and pieces from the definition of $f(s-u-1, t-1)$.

Denote the interval $[s-t-(x+1)Q; Q]$ as $\triangle$. By definition:

$$\begin{aligned} f(s,t) &= s + \min_{u \in \triangle}(f(s-u-1, t-1) + \mathrm{lbp}(u, 0)) \\ &\geq s + \frac{t(t-1)}{2} + \sum_{i=1}^{x-1} \mathrm{lbp}(Q, i) \\ &\quad + \min_{u \in \triangle}(\mathrm{lbp}(s-t-u-(x-1)Q, x-1) + \mathrm{lbp}(u, 0)). \end{aligned}$$

The minimum point for the expression in the minimization clause is, by Lemma 3, $u = \frac{2^{x-1}}{2^{x-1}+1}((s-t)-(x-1)Q) \geq \frac{2^{x-1}}{2^{x-1}+1}(2Q) \geq Q$. This value is infeasible as $u \leq Q$, and all feasible values of u are at the same side of the minimum point, so the closest one should be used: $u = Q$. This results in the following bound:

$$\begin{aligned} f(s,t) &\geq s + \frac{t(t-1)}{2} + \sum_{i=1}^{x-1} \mathrm{lbp}(Q, i) + \mathrm{lbp}(Q, 0) + \mathrm{lbp}(s-t-xQ, x-1) \\ &= \frac{t(t+1)}{2} + \sum_{i=0}^{x-1} \mathrm{lbp}^+(Q, i+1) + \mathrm{lbp}^+(s-t-xQ, x) \\ &> \frac{t(t+1)}{2} + \sum_{i=1}^{x} \mathrm{lbp}(Q, i) + \mathrm{lbp}(s-t-xQ, x). \end{aligned}$$

3. Consider $(x+1)Q \leq s-t \leq (x+2)Q$ and $0 \leq u \leq s-t-(x+1)Q$. These constraints correspond to upper triangles in Figure 3, of which the triangle B is an example. We denote the interval $[0; s-t-(x+1)Q]$ as $\triangledown$. By definition:

$$\begin{aligned} f(s,t) &= s + \min_{u \in \triangledown}(f(s, u-1, t-1) + \mathrm{lbp}(u, 0)) \\ &\geq s + \frac{t(t-1)}{2} + \sum_{i=1}^{x} \mathrm{lbp}(Q, i) + \min_{u \in \triangledown}(\mathrm{lbp}(s-t-u-xQ, x) + \mathrm{lbp}(u, 0)). \end{aligned}$$

The minimum point for the expression in the minimization clause is, by Lemma 3, $u = \frac{2^x}{2^x+1}(s-t-xQ) = \frac{2^x}{2^x+1}(s-t-(x+1)Q+Q) \geq \frac{2^{x+1}}{2^x+1}(s-t-(x+1)Q)$. As $u \leq s-t-(x+1)Q$ and $x \geq 1$, this minimum point is infeasible. As all the feasible values of u are at the same side of the minimum point, the closest one should

be used: $u = s - t - (x+1)Q$. This results in the following bound:

$$f(s,t) \geq s + \frac{t(t-1)}{2} + \sum_{i=1}^{x} \mathrm{lbp}(Q,i) + \mathrm{lbp}(Q,x) + \mathrm{lbp}(s-t-(x+1)Q, 0)$$

$$= \frac{t(t+1)}{2} + \sum_{i=2}^{x+1} \mathrm{lbp}^{+}(Q,i) + \mathrm{lbp}^{+}(Q,x+1) + \mathrm{lbp}^{+}(s-t-(x+1)Q, 1)$$

$$> \frac{t(t+1)}{2} + \sum_{i=2}^{x+1} \mathrm{lbp}(Q,i) + \mathrm{lbp}(Q,x+1) + \mathrm{lbp}(s-t-(x+1)Q, 1).$$

4. As parts 2 and 3 correspond to the same values of x, but to different ranges of u, we need to take a minimum of them. We denote as $f_{ac}(s,t)$ the lower bound from part 2 and as $f_b(s,t)$ the lower bound from part 3. To prove that the first one is always the minimum of two, we should prove that $f_b(s,t) - f_{ac}(s,t) \geq 0$. Here we denote $s - t - (x+1)Q$ as z, such that $0 \leq z \leq Q$. Thus we have:

$$f_b(s,t) - f_{ac}(s,t) = 2\mathrm{lbp}(Q,x+1) + \mathrm{lbp}(z,1) - \mathrm{lbp}(Q,1) - \mathrm{lbp}(z+Q,x)$$
$$= (\mathrm{lbp}(2Q,x) + \mathrm{lbp}(z,1)) - (\mathrm{lbp}(z+Q,x) + \mathrm{lbp}(Q,1)).$$

   Both parentheses blocks in the latter expression are actually instantiations of a function $g(j) = \mathrm{lbp}(2Q-j,x) + \mathrm{lbp}(j,1)$, the first block has $j = 0$, the second block has $j = Q - z$. The function $g(j)$, by Lemma 3, has a minimum point at $j = \frac{2^{x-1}}{2^{x-1}+1}2Q \geq Q$, so it decreases for $0 \leq j \leq Q$, which means that $f_b(s,t) - f_{ac}(s,t) = g(0) - g(Q-z) \geq 0$.

5. As follows from the results of the part 4, the lower bound for $f(s,t)$ at $x \geq 1$ is always the expression $f_{ac}(s,t)$ from the part 2. As for the last definition piece, $x = t-2$, the expression $f_{ac}(s,t)$ is the only lower bound to consider, this proves the induction step and the whole theorem as well. □

6.3 Minimizing the General Form by t

In this section, we use the lower bounds for $f(s,t)$ from Theorem 14 and try to find a minimum of them by t. For the case $0 \leq s - t \leq 2Q$ we relax the lower bound: $f(s,t) \geq \frac{t(t+1)}{2} \geq \frac{(s-2Q)(s-2Q+1)}{2}$.

Consider the case $2Q \leq s - t \leq tQ$. We replace x by $\lfloor \frac{s-t}{Q} \rfloor - 1$ as x was defined by $(x+1)Q \leq s - t \leq (x+2)Q$. Thus:

$$f(s,t) \geq \frac{t(t+1)}{2} + \sum_{i=1}^{\left\lfloor \frac{s-t}{Q} \right\rfloor - 1} \mathrm{lbp}(Q,i) + \mathrm{lbp}\left(s - t - Q\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right), \left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right).$$

The two last addends together can be simplified as:

$$\sum_{i=1}^{\left\lfloor \frac{s-t}{Q} \right\rfloor - 1} \mathrm{lbp}(Q,i) + \mathrm{lbp}\left(s - t - Q\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right), \left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right)$$

$$\geq \sum_{i=1}^{\left\lfloor \frac{s-t}{Q} \right\rfloor - 1} \mathrm{lbp}(Q,i) + \left(s - t - Q\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right)\right) \log_2\left(2^{\left\lfloor \frac{s-t}{Q} \right\rfloor - 1} Q + 1\right)$$

$$
\begin{aligned}
&> Q \sum_{i=1}^{\left\lfloor \frac{s-t}{Q} \right\rfloor - 1} (\log_2 Q + i) + \left(s - t - Q\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right)\right)\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1 + \log_2 Q\right) \\
&> Q \sum_{i=1}^{\left\lfloor \frac{s-t}{Q} \right\rfloor - 1} (\log_2 Q + i) + \left(s - t - Q\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right)\right)\log_2 Q + Q\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right) \\
&= \log_2 Q \cdot \left(Q\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right) + s - t - Q\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right)\right) \\
&+ Q \sum_{i=1}^{\left\lfloor \frac{s-t}{Q} \right\rfloor - 1} i + Q\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right) \\
&= \log_2 Q \cdot (s-t) + Q\frac{\left(\left\lfloor \frac{s-t}{Q} \right\rfloor - 1\right)\left(\left\lfloor \frac{s-t}{Q} \right\rfloor + 2\right)}{2} \\
&\geq \log_2 Q \cdot (s-t) + \frac{(s-t-Q)(s-t+2Q)}{2Q}.
\end{aligned}
$$

Now, we have to minimize by t the following expression for the lower bound:

$$\frac{t(t+1)}{2} + \log_2 Q \cdot (s-t) + \frac{(s-t-Q)(s-t+2Q)}{2Q}.$$

This is a polynomial of degree two, which has its minimum at $t_{\min} = \frac{s+Q\log_2 Q - Q}{1+Q}$. Note that $\frac{s+Q\log_2 Q - Q}{1+Q} > \frac{s}{1+Q}$; that is, $t_{\min}$ is feasible. We get the following expression by substituting $t_{\min}$:

$$f(s,t) \geq \frac{Q\log_2 Q \cdot (s+1-\log_2 Q)}{1+Q} + \frac{(s-2Q)(s+Q)+s-3Q}{2(1+Q)}.$$

6.4 Applying the Lower Bounds to Extreme Jump

To apply the results of previous sections to the extreme Jump problem, we first need to note that, while Sections 6.2–6.3 use $u\log_2(u+1)$ as the minimum sum depth of all queries in a subtree of size u, the real value is $u\left(\lfloor \log_2(u+1) \rfloor - 1\right)$. However, $u\left(\lfloor \log_2(u+1) \rfloor - 1\right) \geq u(\log_2(u+1) - 2) = u\log_2(u+1) - 2u$. We can estimate the lower bound on the black-box complexity of the extreme Jump as follows:

$$T_{\text{JUMP}}(n) \geq \frac{f(s,t) - 2s}{s} = \frac{f(s,t)}{s} - 2.$$

The search space size for extreme Jump is $s = 2^n$. For even n, $Q = \binom{n}{n/2}$. Note that $\frac{s}{Q} = 2^n / \binom{n}{n/2} = \Theta(\sqrt{n})$, also note that $\log_2 Q = \log_2 s - \log_2 \frac{s}{Q} = n - \Theta(\log\sqrt{n}) = n - \Theta(\log n)$.

We get the following expression for even n:

$$
\begin{aligned}
T_{\text{JUMP}}(n) &\geq \frac{Q\log_2 Q \cdot (s+1-\log_2 Q)}{(1+Q)s} + \frac{(s-2Q)(s+Q)+s-3Q}{2s(1+Q)} - 2 \\
&\geq \left(1 - \frac{1}{1+Q}\right)\left(1 - \frac{\log_2 Q - 1}{s}\right)\log_2 Q + \frac{s}{2(1+Q)}\left(1 - \frac{2Q}{s}\right)\left(1 + \frac{Q}{s}\right) - 2
\end{aligned}
$$

$$= (n - \Theta(\log n))(1 - o(n^{-1}))^2 + \frac{s}{2(1+Q)}(1 \pm O(n^{-0.5}))^2 - O(1)$$
$$= n + \frac{s}{2(1+Q)} - O(\log n) = n + \Theta(\sqrt{n}).$$

For odd n, $Q = 2\binom{n}{\lfloor n/2 \rfloor}$ as noticed before, which brings the same asymptotic.

Note that in both the odd and the even cases, the value for Q plugged in the expression for $t_{\min}$ gives $t_{\min} = \Omega(\sqrt{n})$. Hence, when running any deterministic black-box algorithm for extreme jump functions, it cannot be avoided that, on average, only 0-answers are received for the first $\Omega(\sqrt{n})$ iterations, and consequently, the size of the remaining search space is still a constant fraction of the initial search space.

7 Conclusion

New black-box algorithms for solving $\textsc{Jump}_{n,\ell}$ problem are presented, giving the following upper bounds:

- for $\ell < n/2 - \sqrt{n}\log_2 n$: $\frac{2n(1+o(1))}{\log_2 n}$, where $o(1)$ is measured when $n \to \infty$;
- for $n/2 - \sqrt{n}\log_2 n \le \ell < \lfloor \frac{n}{2} \rfloor - \omega(1)$: $\frac{n(1+o(1))}{\log_2(n-2\ell)}$, where $o(1)$ and $\omega(1)$ are measured when $n - 2\ell \to \infty$;
- for $\ell = \lfloor \frac{n}{2} \rfloor - 1$: $n + \Theta(\sqrt{n})$.

A new theorem for constructing lower bounds on unrestricted black-box complexity of problems is proposed. The underlying idea is that influence of particular answers to queries to all subsequent queries can be formalized by assigning a type to each query and writing the relations in a form of a matrix. Several following steps for constructing the lower bounds are automated and can be performed using tools like Wolfram Alpha. We hope that this theorem can be used to obtain better lower bounds in other problems.

Using the proposed theorem, the lower bounds for $\textsc{Jump}_{n,\ell}$ are updated:

- for even n: $\left\lfloor \log_{\frac{n-2\ell+2}{2}} \left(1 + 2^{n-1}\frac{(n-2\ell)^2}{n-2\ell-1}\right) \right\rfloor - \frac{2}{n-2\ell} \ge \frac{n}{\log_2 \frac{n-2\ell+2}{2}} - 1$;
- for odd n: $\left\lfloor \log_{\frac{n-2\ell+1}{2}} \left(1 + 2^{n-2}(n-2\ell-1)\right) \right\rfloor - \frac{2}{n-2\ell-1} \ge \frac{n-1}{\log_2 \frac{n-2\ell+1}{2}} - 1$.

For extreme $\textsc{Jump}$, the lower bounds produced by the matrix theorem are $n-1$ for even n and $n-2$ for odd n. We applied additional knowledge about the jump functions, namely, the fact that the search space size decreases significantly when the algorithm receives an answer equal to $\lfloor \frac{n}{2} \rfloor$ or $\lceil \frac{n}{2} \rceil$. The lower bounds for extreme $\textsc{Jump}$ were proven to be $n + \Theta(\sqrt{n})$, which matches the upper bound. More precisely, the lower bound equals $n + \frac{2^n}{2(Q+1)} + O(\log n)$, while the upper bound equals $n + \frac{2^n}{Q+1}$, where $Q = \binom{n}{n/2}$ for even n and $Q = 2\binom{n}{\lfloor n/2 \rfloor}$ for odd n.

In the case of large, but not extreme ℓ, the quotients at the highest terms coincide as well (when $n - 2\ell$ tends to infinity), because the ratio of the upper bound to the lower bound approaches:

$$1 \le \frac{\log_2(n-2\ell)}{\log_2\left(\frac{n-2\ell+1}{2}\right)} \le \frac{\log_2(n-2\ell)}{\log_2\left(\frac{n-2\ell}{2}\right)} = \frac{\log_2(n-2\ell)}{\log_2(n-2\ell) - 1} = \frac{1}{1 - \frac{1}{\log_2(n-2\ell)}} = 1 + o(1).$$

These new methods to prove lower bounds are interesting beyond the particular application to jump functions, since there are very few lower bound proofs in black-box complexity theory beyond the information theoretic argument from Droste et al. (2006), while there are many challenging problems still open. Let us point out two.

For the black-box complexity of the $\textsc{OneMax}_n$ class, we know that it is between $(1 + o(1))n/\log_2 n$ and $(1 + o(1))2n/\log_2 n$. It is not known what is the truth. This problem was already raised by Erdős and Rényi (1963). The lower bound stems from the information theoretic argument, building on the fact that each query has at most $n + 1$ different answers. For a single query regarded independently, the information gain is much lower. In fact, for each query the vast majority of the instances have the answer lying in an interval of size $\Theta(\sqrt{n})$ around $n/2$. For this reason, nonadaptive black-box algorithms such as asking random queries (but also all other algorithms asking queries independent of the previous answers) necessarily have a complexity of at least $(1 + o(1))2n/\log_2 n$; see again Erdős and Rényi (1963). To prove that this is also a lower bound for any black-box algorithm, one would need a lower bound technique that goes beyond the argument of counting the number of different answers.

A second challenging lower bound problem is the unbiased k-ary black-box complexity of the $\textsc{OneMax}_n$ function class. The k-ary unbiased black-box complexity is the unbiased black-box complexity defined in Section 2.3 with the additional restriction that the unbiased variation operators take at most k previous search points as arguments. There are upper bounds for this black-box complexity decreasing with k (Doerr, Johannsen, et al., 2011; Doerr and Winzen, 2014c), which indicates a possible stronger power of higher arity operators. Unfortunately, no matching lower bounds confirming this exist. Worse, there are no lower bounds for k-ary black-box complexities of $\textsc{OneMax}_n$ for $k \geq 2$ at all that are stronger than the information theoretic $\Omega(n/\log n)$. For $k = 1$, Lehre and Witt (2012) show a lower bound of $\Omega(n \log n)$ by, very roughly speaking, showing that a unary unbiased black-box algorithm for $\textsc{OneMax}$ cannot do much better than taking the best-so-far search point and mutating it by flipping a certain number of bits.

While it is not clear how to use our matrix lower bound theorem for either of these two problems, we hope that our work does give some motivation to push the lower bound question beyond the classic information theory argument.

Acknowledgments

This work was partially financially supported by the Government of Russian Federation, Grant 074-U01. This research also benefited from the support of the FMJH Program Gaspard Monge in Optimization and Operation Research, and from the support to this program from EDF.

References

Afshani, P., Agrawal, M., Doerr, B., Doerr, C., Larsen, K. G., and Mehlhorn, K. (2013). The query complexity of finding a hidden permutation. In *Space-Efficient Data Structures, Streams, and Algorithms*, pp. 1–11. Lecture Notes in Computer Science, Vol. 8066. Berlin: Springer.

Anil, G., and Wiegand, R. P. (2009). Black-box search by elimination of fitness functions. In *Proceedings of Foundations of Genetic Algorithms*, pp. 67–78.

Badkobeh, G., Lehre, P. K., and Sudholt, D. (2014). Unbiased black-box complexity of parallel search. In *Parallel Problem Solving from Nature XIII*, pp. 892–901. Lecture Notes in Computer Science, Vol. 8672. Berlin: Springer.

Buzdalov, M., Kever, M., and Doerr, B. (2015). Upper and lower bounds on unrestricted black-box complexity of $\textsc{Jump}_{n,\ell}$. In *Evolutionary Computation in Combinatorial Optimization*, pp. 209–221. Lecture Notes in Computer Science, Vol. 9026.

Chvátal, V. (1983). Mastermind. *Combinatorica*, 3(3): 325–329.

Doerr, B. (2011). Analyzing randomized search heuristics: Tools from probability theory. In *Theory of Randomized Search Heuristics*, pp. 1–20.

Doerr, B., and Doerr, C. (2014). Black-box complexity: From complexity theory to playing mastermind. In *Proceedings of Genetic and Evolutionary Computation Conference Companion*, pp. 623–646.

Doerr, B., Doerr, C., and Ebel, F. (2015). From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567: 87–104.

Doerr, B., Doerr, C., and Kötzing, T. (2014a). Unbiased black-box complexities of jump functions: How to cross large plateaus. In *Proceedings of Genetic and Evolutionary Computation Conference*, pp. 769–776.

Doerr, B., Doerr, C., and Kötzing, T. (2014b). The unbiased black-box complexity of partition is polynomial. *Artificial Intelligence*, 216: 275–286.

Doerr, B., Doerr, C., and Kötzing, T. (2016). Unbiased black-box complexities of jump functions. *Evolutionary Computation*. Accepted for publication.

Doerr, B., Johannsen, D., Kötzing, T., Lehre, P. K., Wagner, M., and Winzen, C. (2011). Faster black-box algorithms through higher arity operators. In *Proceedings of Foundations of Genetic Algorithms*, pp. 163–172.

Doerr, B., Kötzing, T., Lengler, J., and Winzen, C. (2013). Black-box complexities of combinatorial problems. *Theoretical Computer Science*, 471: 84–106.

Doerr, B., Kötzing, T., and Winzen, C. (2011). Too fast unbiased black-box algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference*, pp. 2043–2050.

Doerr, B., and Winzen, C. (2014a). Playing Mastermind with constant-size memory. *Theory of Computing Systems*, 55(4): 658–684.

Doerr, B., and Winzen, C. (2014b). Ranking-based black-box complexity. *Algorithmica*, 68(3): 571–609.

Doerr, B., and Winzen, C. (2014c). Reducing the arity in unbiased black-box complexity. *Theoretical Computer Science*, 545: 108–121.

Doerr, C., and Lengler, J. (2015). Elitist black-box models: Analyzing the impact of elitist selection on the performance of evolutionary algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference*, pp. 839–846.

Droste, S., Jansen, T., Tinnefeld, K., and Wegener, I. (2003). A new framework for the valuation of algorithms for black-box optimization. In *Proceedings of Foundations of Genetic Algorithms*, pp. 253–270.

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

Droste, S., Jansen, T., and Wegener, I. (2006). Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39(4): 525–544.

Erdős, P., and Rényi, A. (1963). On two problems of information theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézet Közleményei*, 8: 229–243.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301): 13–30.

Jansen, T. (2015). On the black-box complexity of example functions: The real Jump function. In *Proceedings of Foundations of Genetic Algorithms*, pp. 16–24.

Jansen, T., and Wegener, I. (2002). The analysis of evolutionary algorithms—A proof that crossover really can help. *Algorithmica*, 34(1): 47–66.

Lehre, P. K., and Witt, C. (2010). Black-box search by unbiased variation. In *Proceedings of Genetic and Evolutionary Computation Conference*, pp. 1441–1448.

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

Rowe, J., and Vose, M. (2011). Unbiased black box search algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference*, pp. 2035–2042.

Yao, A. C.-C. (1977). Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science*, pp. 222–227.