Unbiased Black-Box Complexities of Jump Functions—How to Cross Large Plateaus

Benjamin Doerr
École Polytechnique,
Paris-Saclay, France

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

Timo Kötzing
Friedrich-Schiller-Universität
Jena, Germany

ABSTRACT

We analyze the unbiased black-box complexity of jump functions with large jump sizes.

Among other results, we show that when the jump size is $(1/2 - \varepsilon)n$, that is, only a small constant fraction of the fitness values is visible, then the unbiased black-box complexities for arities 3 and higher are of the same order as those for the simple OneMax function. Even for the extreme jump function, in which all but the two fitness values $n/2$ and n are blanked out, polynomial-time mutation-based (i.e., unary unbiased) black-box optimization algorithms exist. This is quite surprising given that for the extreme jump function almost the whole search space (all but a $\Theta(n^{-1/2})$ fraction) is a plateau of constant fitness.

To prove these results, we introduce new tools for the analysis of unbiased black-box complexities, for example, selecting the new parent individual not by comparing the fitnesses of the competing search points, but also by taking into account the (empirical) expected fitnesses of their offspring.

Categories and Subject Descriptors

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

Keywords

Black-Box Complexity; Theory; Run Time Analysis; Evolutionary Computation

1. INTRODUCTION

The analysis of black-box complexities in evolutionary computation aims in several complementary ways at supporting the development of superior evolutionary algorithms. By comparing the run time of currently used randomized search heuristics (RSHs) with the one of an optimal black-box algorithm, it allows a fair evaluation of how good today's RSH are. With specific black-box complexity notions, we can understand how algorithm components and parameters such as the population size, the selection rules, or the sampling procedures influence the run time of RSH. Finally, research in black-box complexity proved to be a source of inspiration for developing new algorithmic ideas that lead to the design of better search heuristics.

In this work, we analyze the unbiased black-box complexity of jump functions, which are observed as difficult for evolutionary approaches because of their large plateaus of constant (and low) fitness. Our results show that, surprisingly, even extreme jump functions that reveal only the three different fitness values 0, $n/2$, and n can be optimized by a mutation-based unbiased black-box algorithm in polynomial time. We introduce new methods that facilitate our analyses. The perhaps most interesting one is a routine that creates a number of samples from which it estimates the distance of the current search point to the fitness layer $n/2$. Our algorithm thus benefits significantly from analyzing some non-standard statistics of the fitness landscape. We believe this to be an interesting idea that should be investigated further. Our hope is that it can be used to design new search heuristics.

1.1 Black-Box Complexity

Black-box complexity studies how many function evaluations are needed in expectation by an optimal black-box algorithm until it queries for the first time an optimal solution for the problem at hand. Randomized search heuristics, like evolutionary algorithms, simulated annealing, and ant colony algorithms, are typical black-box optimizers: they are typically problem-independent and as such they learn about the problem to be solved only by generating and evaluating search points. The black-box complexity of a problem is thus a lower bound for the number of fitness evaluations needed by any search heuristic to solve it.

Several black-box complexity notions covering different aspects of randomized search heuristics exist, for example the unrestricted model [12], which does not restrict in any way the sampling or selection procedure of the algorithm, the ranking-based model [8, 21], in which the algorithms are required to base their selection only on relative and not on absolute fitness values, the memory-restricted model [9, 12], in which the algorithm can store only a limited number of search points and their corresponding fitness values, and the unbiased model [17], which requires the algorithms to treat the representation of the search points symmetrically. By comparing the respective black-box complexities of a problem, one learns how the run time of RSHs is influenced by

GECCO'14, July 12–16, Vancouver, BC, Canada.

ACM 978-1-4503-2662-9/14/07 ...$15.00.
http://dx.doi.org/10.1145/2576768.2598341.

Arity	Constant Jump $\ell = O(1)$		Short Jump $\ell = O(n^{1/2-\varepsilon})$		Long Jump $\ell = (1/2-\varepsilon)n$		Extreme Jump $\ell = n/2-1$	
$k = 1$	$\Theta(n \log n)$	[7]	$\Theta(n \log n)$	[here]	$O(n^2)$	[here]	$O(n^{9/2})$	[here]
$k = 2$	$O(n)$	[5, 7]	$O(n)$	[here]	$O(n \log n)$	[here]	$O(n \log n)$	[here]
$3 \leq k \leq \log n$	$O(n/k)$	[7, 10]	$O(n/k)$	[here]	$O(n/k)$	[here]	$\Theta(n)$	[here]

Table 1: Known results for the unbiased black-box complexity of Jump_ℓ.

certain algorithmic choices such as the population size, the use of crossover, the selection rules, etc.

For all existing black-box models, however, the typical optimal black-box algorithm is a highly problem-tailored algorithm that is not necessarily nature-inspired. Still we can learn from such "artificial" algorithms about RSH, as has been shown in [3]. In that work, a new genetic algorithm is presented that optimizes the OneMax function in run time $o(n \log n)$, thus showing that the $o(n \log n)$ bound for the 2-ary unbiased black-box complexity of OneMax found in [5] is not as unnatural as it might have seemed at first.

Here in this work we consider unbiased black-box complexities. The unbiased black-box model is one of the standard models for analyzing the influence of the arity on the performance of optimal black-box algorithms. It was originally defined by Lehre and Witt [17] for bit string representations and has later been generalized to domains different from bit strings [6, 19]. For bit string representations, the unbiased model requires the optimizing algorithms to treat different positions of the bit strings equally, similarly with the two different possible bit contents (thus the term "unbiased"). For example, unbiased algorithms are not allowed to explicitly write a 1 or a 0 at a specific position of a bit string to be evaluated; instead, such algorithms can either sample a bit string uniformly at random, or generate one from previously evaluated solutions via operators which are unbiased (i.e., treat positions and bit contents equally). See Section 2 for a detailed description of the model.

1.2 Jump Functions

In this paper we are concerned with the optimization of functions mapping bit strings of fixed length (i.e., elements of the hypercube $\{0,1\}^n$) to real numbers; such functions are called *pseudo-Boolean*. A famous pseudo-Boolean function often considered as a test function for optimization is the OneMax function, mapping any $x \in \{0,1\}^n$ to the number of 1s in x (the *Hamming weight* of x).

Other popular test functions are the jump functions. For a non-negative integer ℓ, we define the Jump_ℓ as derived from OneMax by "blanking out" any useful information within the strict ℓ-neighborhood of the optimum (and the minimum) by giving all these search points a fitness value of 0. In other words, $\text{Jump}_\ell(x) = \text{OneMax}(x)$ if $\text{OneMax}(x) \in \{0\} \cup \{\ell+1, \dots, n-\ell-1\} \cup \{n\}$ and $\text{Jump}_\ell(x) = 0$ otherwise. This definition is mostly similar to the two, also not fully agreeing, definitions used in [11] and [15].

Jump functions are well-known test functions for randomized search heuristics. Droste, Jansen, and Wegener [11] analyzed the optimization time of the (1+1) evolutionary algorithm on Jump functions. From their work, it is easy to see that for our definition of Jump functions, a run time of $\Theta(n^{\ell+1})$ for the (1+1) evolutionary algorithm on Jump_ℓ follows for all $\ell \in \{1, \dots, \lfloor n/2 \rfloor - 1\}$. We are not aware of any natural mutation-based randomized search heuristic with significantly better performance (except for large ℓ, where simple random search with its $\Theta(2^n)$ run time becomes superior). For all ℓ, Jansen and Wegener [14] present a crossover-based algorithm for Jump_ℓ. With an optimal choice for the parameter involved, which in particular implies a very small crossover rate of $O(1/n)$, it has an optimization time of $O(n \log^3 n)$ for constant ℓ (this is mistakenly stated as $O(n \log^2 n \log \log n)$ on the last line of p. 60 of the paper, but it is clear from the proof that this is only a typo) and an optimization time of $O(n^{2c+1} \log n)$ for $\ell = \lceil c \log n \rceil$, c a constant.

Contrasting these results, it has been shown in [7] that for jump functions with constant ℓ, the k-ary unbiased black-box complexities are of the same order as those of the easy OneMax test function (which can be seen as a Jump function with parameter $\ell = 0$). The results in [7] are based on an intermediate result (Lemma 3 in [7]) which shows that a black-box algorithm having access to a jump function with constant ℓ can retrieve (with high probability) the true OneMax value of a search point using only a constant number of queries.

1.3 Our Results

In this work we greatly expand on the results obtained in [7] and give a much more complete picture of the unbiased black-box complexity of Jump functions.

In the regime of jump functions with super-constant jump size, we distinguish between *short*, *long*, and *extreme* jump functions. Short jump functions have ℓ values of $O(n^{1/2-\varepsilon})$; we show that the crucial Lemma 3 from [7], there only proven for constant ℓ, extends to all short jump functions. This implies that we get the same run time bounds for short jump functions as are known for OneMax.

A result like Lemma 3 in [7] is not to be expected to hold for larger values of ℓ. Nevertheless, we show that also long jump functions, where ℓ can be as large as $(1/2-\varepsilon)n$, have unbiased black-box complexities of the same asymptotic order as OneMax for arities $k \geq 3$. For $k = 2$ we get a bound of $O(n \log n)$ and for $k = 1$ we get $O(n^2)$, both surprisingly low black-box complexities. Even for the case of extreme jump functions, where $\ell = n/2-1$ and n even (a jump function revealing only the optimum and the fitness $n/2$), we are able to show polynomial unbiased black-box complexities for all arities $k \geq 1$.

Note that already for long jump functions, the fitness plateau that the algorithms have to cross has exponential size. For the extreme jump function, even all but a $\Theta(n^{-1/2})$ fraction of the search points form one single fitness plateau. This is the reason why none of the popular randomized search algorithms will find the optimum of long and extreme jump functions in subexponential time.

Table 1 summarizes the known black-box complexities of $\textsc{Jump}_\ell$ and our new results.

Note that, in Table 1, the bounds for the binary and k-ary black-box complexity of the short JUMP function differ from the ones presented in [7]. The reason is that the respective bounds for ONEMAX, to which the JUMP complexities have been reduced to in [7], have been improved recently in [10].

The table clearly indicates that even without the fitness function revealing useful information for search points close the optimum, efficient optimization is still possible in the framework of unbiased black-box algorithms.

1.4 Methods

In order to show the upper bounds on the black-box complexities we give efficient algorithms optimizing the different jump functions. For arity $k = 1$, these algorithms are based on iteratively getting closer to the optimum; however we do not (and in fact cannot) rely on fitness information about these closer search points: the fitness is 0 in almost all cases. Instead, we rely on the *empirical expected fitness* of offspring. For this we use mutation operators that have a good chance of sampling offspring with non-zero fitness. We show that already a polynomial number of samples suffices to distinguish search points whose fitness differs only minimally. In order to minimize the number of samples required, we choose this number *adaptively* depending on the estimated number of 1s in the search point to be evaluated; we also allow fairly frequent incorrect decisions, as long as the overall progress to the optimum is guaranteed.

In one of our proofs we make use of an additive Chernoff bound for negatively correlated variables. This bound is implicit in a paper by Panconesi and Srivnivasan [18] and is of independent interest.

Disclaimer: For reasons of space we can present only the main proof ideas. The full version of Sections 4 and 5 along with the claimed generalization of the bounds from [7] can be found in [4].

2. THE UNBIASED BLACK-BOX MODEL

The unbiased black-box model introduced in [16] is by now one of the standard complexity models in evolutionary computation. In particular the unary unbiased model gives a more realistic complexity estimate for a number of functions than the original unrestricted black-box model of Droste, Jansen, and Wegener [12]. An important advantage of the unbiased model is that it allows us to analyze the influence of the arity of the sampling operators in use. In addition, new search points can be sampled only either uniformly at random or from distributions that depend on previously generated search points in an *unbiased* way. In this section we briefly give a brief definition of the unbiased black-box model, pointing the interested reader to [16] and [10] for a more detailed introduction.

For all non-negative integers k, a *k-ary unbiased distribution* $\big(D(\cdot \mid y^{(1)}, \dots, y^{(k)})\big)_{y^{(1)},\dots,y^{(k)} \in \{0,1\}^n}$ is a family of probability distributions over $\{0,1\}^n$ such that for all *inputs* $y^{(1)}, \dots, y^{(k)} \in \{0,1\}^n$ the following two conditions hold.

1. [$\oplus$-invariance] $\forall x, z \in \{0,1\}^n$: $D(x \mid y^{(1)}, \dots, y^{(k)}) = D(x \oplus z \mid y^{(1)} \oplus z, \dots, y^{(k)} \oplus z)$;
2. [permutation-invariance] $\forall x \in \{0,1\}^n \, \forall \sigma \in S_n$: $D(x \mid y^{(1)}, \dots, y^{(k)}) = D(\sigma(x) \mid \sigma(y^{(1)}), \dots, \sigma(y^{(k)}))$,

Algorithm 1: Scheme of a k-ary unbiased black-box algorithm

Initialization: Sample $x^{(0)} \in \{0,1\}^n$ uniformly at random and query $f(x^{(0)})$.
Optimization: **for** $t = 1, 2, 3, \dots$ **until** *termination condition met* **do**
  Depending on $\big(f(x^{(0)}), \dots, f(x^{(t-1)})\big)$ choose up to k indices $i_1, \dots, i_k \in [0..t-1]$ and a k-ary unbiased distribution $D(\cdot \mid x^{(i_1)}, \dots, x^{(i_k)})$.
  Sample $x^{(t)}$ according to $D(\cdot \mid x^{(i_1)}, \dots, x^{(i_k)})$ and query $f(x^{(t)})$.

where $\oplus$ is the bitwise exclusive-OR, S_n the set of all permutations of the set $[n] := \{1, 2, \dots, n\}$, and $\sigma(x) := x_{\sigma(1)} \cdots x_{\sigma(n)}$ for $x = x_1 \cdots x_n \in \{0,1\}^n$.

An operator sampling from a k-ary unbiased distribution is called a *k-ary unbiased variation operator*.

A k-ary unbiased black-box algorithm is one that follows the scheme of Algorithm 1 (here and in the following with $[0..k]$ we abbreviate $[k] \cup \{0\}$). The *k-ary unbiased black-box complexity*, denoted $\mathrm{UBB}_k(\mathcal{F})$, of some class of functions $\mathcal{F}$ is the minimum complexity of $\mathcal{F}$ with respect to all k-ary unbiased black-box algorithms, where, naturally, the complexity of an algorithm A for $\mathcal{F}$ is the maximum expected number of black-box queries that A performs on a function $f \in \mathcal{F}$ until it queries for the first time a search point of maximal fitness. We let *$*$-ary unbiased black-box complexity* be based on the model in which operators of arbitrary arity are allowed.

The unbiased black-box model includes most of the commonly studied search heuristics, such as many $(\mu + \lambda)$ and (μ, λ) evolutionary algorithms (EAs), Simulated Annealing, the Metropolis algorithm, and the Randomized Local Search algorithm.

We recall a simple remark from [7] which helps us shortening some of the proofs in the subsequent sections.

REMARK 1. *Suppose for a problem P there exists a black-box algorithm A that, with constant success probability, solves P in s iterations (that is, queries an optimal solution within s queries). Then the black-box complexity of P is at most $O(s)$.*

A useful tool for proving lower bounds is Theorem 2. It formalizes the intuition that the black-box complexity of a function can only get harder if we "blank out" some of the fitness values. This is exactly the situation of the JUMP functions, whose definition we repeat here for the sake of completeness.

For all $\ell < n/2$, $\textsc{Jump}_\ell$ is the function that assigns to each $x \in \{0,1\}^n$ fitness

$$\textsc{Jump}_\ell(x) = \begin{cases} n, & \text{if } |x|_1 = n; \\ |x|_1, & \text{if } \ell < |x|_1 < n - \ell; \\ 0, & \text{otherwise,} \end{cases}$$

where $|x|_1 := \textsc{OneMax}(x) := \sum_{i=1}^n x_i$ denotes the number of 1s in x (also known as the *Hamming-weight of x*).

THEOREM 2. *For all sets of pseudo-Boolean functions C, all $k \in \mathbb{N}$, and all $f : \mathbb{R} \to \mathbb{R}$ such that $\forall g \in C$:*

$\{x \mid f(g(x)) \text{ optimal}\} \subseteq \{x \mid g(x) \text{ optimal}\}$, *we have* $\mathrm{UBB}_k(C) \leq \mathrm{UBB}_k(f(C))$.

PROOF. Let C, k, and f be as in the statement of the theorem. Let A be any k-ary unbiased black-box algorithm for $f(C)$. We derive from this a k-ary unbiased black-box algorithm for C by using queries to $g \in C$ and then mapping the resulting objective value with f. Clearly, A' finds an optimum of $g \in C$ after no more expected queries than A for $f \circ g$, using the condition on the set of optimal points. Thus, the theorem follows. □

From Theorem 2 we immediately obtain a lower bound of $\Omega(n/\log n)$ for the unbiased black-box complexities of jump functions. The theorem implies that the k-ary unbiased black-box complexity of OneMax is a lower bound of that of any jump function. In general, the k-ary unbiased black-box complexity of any pseudo-Boolean function f is at least the unrestricted black-box complexity of the class of functions obtained from f by first applying an automorphism of the hypercube $\{0,1\}^n$. That the latter for OneMax is $\Omega(n/\log n)$ was shown independently in [12] and [13]. A similar line of arguments will prove the lower bound for extreme jump functions in Section 5.

3. SHORT JUMP FUNCTIONS

The key idea in [7] for dealing with jump functions of constant gap is to revert the problem to optimizing a OneMax function. The following lemma is a generalization of Lemma 3 in [7], extending it from constant ℓ to all $\ell \in O(n^{1/2-\varepsilon})$.

LEMMA 3. *For all constants ε and c and all $\ell \in O(n^{1/2-\varepsilon})$, there is a unary unbiased subroutine s using $O(1)$ queries to Jump_ℓ such that, for all bit strings x, $s(x) = \mathrm{OneMax}(x)$ with probability $1 - O(n^{-c})$.*

With Lemma 3 at hand, the results stated in Table 1 follow easily from the respective OneMax bounds proven in [10, 12, 16], cf. [7] for a detailed proof.

4. LONG JUMP FUNCTIONS

In this section we give bounds on long jump functions; we start with a bound on the ternary black-box complexity, followed by a bound on the unary black-box complexity. Note that the bound on the binary unbiased black-box complexity of long jump follows from the same bound on extreme jump.

4.1 Ternary Unbiased Optimization of Long Jump Functions

We show that ternary operators allow for solving the problem independently in different parts of the bit string, and then combining the partial solutions. This has the advantage that, as done in [7], we can revert to optimizing OneMax, and the missing fitness values will not show in any of the partial problems.

We start with a lemma regarding the possibility of simulating unbiased algorithms for OneMax on subsets of the bits.

LEMMA 4. *For all bit strings $x, y \in \{0,1\}^n$ we let $[x,y] = \{z \in \{0,1\}^n \mid \forall i \leq n : x_i = y_i \Rightarrow x_i = z_i\}$ (this set is isomorphic to a hypercube). Let A be a k-ary unbiased black-box algorithm optimizing OneMax with constant probability in time at most $t(n)$. Then there is a $(k+2)$-ary unbiased black-box subroutine* `simulateOnSubcube` *as follows.*

- *Inputs to* `simulateOnSubcube` *are $x, y \in \{0,1\}^n$ and the Hamming distance a of x and y; x and y are accessible as search points sampled previous to the call of the subroutine.*
- `simulateOnSubcube` *has access to an oracle returning $\mathrm{OneMax}(z)$ for all $z \in [x,y]$.*
- *After at most $t(a)$ queries* `simulateOnSubcube` *has found the $z \in [x,y]$ with maximal OneMax value with constant probability.*

This subroutine can be used to optimize Jump_ℓ blockwise. Each block is optimized by itself while we have to make sure that the correct OneMax value is available as long as only bits within the block are modified. Afterwards, the different optimized blocks are merged to obtain the optimum. As the subroutine requires an increase of the arity by two, this idea yields the following result.

THEOREM 5. *Let $\ell \leq (1/2 - \varepsilon)n$. For all $k \geq 3$, the k-ary unbiased black-box complexity of Jump_ℓ is $O(\mathrm{UBB}_{k-2}(\mathrm{OneMax}))$.*

From Theorem 5 we immediately get the following corollary, using the known run time bounds for OneMax from [10].

COROLLARY 6. *Let $\ell \leq (1/2 - \varepsilon)n$. Then the unbiased black-box complexity of Jump_ℓ is*

- *$O(n \log n)$, for ternary variation operators;*
- *$O(n/k)$, for k-ary variation operators with $4 \leq k \leq \log n$;*
- *$\Theta(n/\log n)$, for k-ary variation operators with $k \geq \log n$ or unbounded arity.*

Note the upper bound of $O(n \log n)$ for the ternary unbiased black-box-complexity which we will improve in Section 5 to $O(n)$. For all higher arities, the theorem presented in this section gives the best known bound.

4.2 Unary Unbiased Optimization of Long Jump Functions

When optimizing a Jump_ℓ function via unary unbiased operators, the only way to estimate the OneMax-value of a search point x (equivalently, its Hamming distance $H(x, \mathbf{1}_n)$ from the optimum), is by sampling suitable offspring that have a non-zero fitness. When ℓ is small, that is, many OneMax-values can be derived straight from the fitness, we can simply flip ℓ bits and hope that the retrieved fitness value is by ℓ smaller than $\mathrm{OneMax}(x)$. This is the main idea in [7].

When ℓ is larger, this does not work anymore, simply because the chance that we only flip 1-bits to zero is too small. Therefore, in this section, we resort to a sampling approach that, via strong concentration results, learns the expected fitness of the sampled offspring of x, and from this the OneMax-value of the parent x. This will lead to a unary unbiased black-box complexity of $O(n^2)$ for all jump functions Jump_ℓ with $\ell \leq n/2 - \varepsilon n$.

THEOREM 7. *Let $\ell \leq (1/2 - \varepsilon)n$. The unary unbiased black-box complexity of Jump_ℓ is $O(n^2)$.*

Proof outline and methods.

Since we aim at an asymptotic statement, let us assume that n is sufficiently large and even. Also, since we shall not elaborate on the influence of the constant $\varepsilon > 0$, we may assume (by replacing ε by a minimally smaller value) that ε is such that εn is even.

A first idea to optimize $\textsc{Jump}_\ell$ with $\ell = n/2 - \varepsilon n/2$ could be to flip each bit of the parent x with probability $1/2 - \varepsilon/2$. Such an offspring u has an expected fitness of $n/2 - \varepsilon n/2 + \varepsilon\textsc{OneMax}(x)$. If ε is constant, then by Chernoff bounds $O(n \log n)$ samples are enough to ensure that the average observed fitness $n/2 - \varepsilon n/2 + \varepsilon v$ satisfies $v = \textsc{OneMax}(x)$ with probability $1 - n^{-c}$, c an arbitrary constant. This is enough to build a unary unbiased algorithm using $O(n^2 \log^2(n))$ fitness evaluations.

We improve this first approach via two ideas. The more important one is to not flip an expected number of $n/2-\varepsilon n/2$ bits independently, but to flip exactly that many bits (randomly chosen). By this, we avoid adding extra variation via the mutation operator. This pays off when x already has many ones—if $\textsc{OneMax}(x) = n - a$, then we will observe that only $O(a \log n)$ samples suffice to estimate the $\textsc{OneMax}$-value of x precisely (allowing a failure probability of n^{-c} as before).

The price for not flipping bits independently (but flipping a fixed number of bits) is that we have do deal with hypergeometric distributions, and when sampling repeatedly, with sums of these. The convenient way of handling such sums is to rewrite them as sums of negatively correlated random variables and then argue that Chernoff bounds also hold for these. This has been stated explicitly in [2] for multiplicative Chernoff bounds, but not for additive ones. Since for our purposes an additive Chernoff bound is more convenient, we extract such a bound from the original paper [18].

The second improvement stems from allowing a larger failure probability. This will occasionally lead to wrong estimates of $\textsc{OneMax}(x)$, and consequently to wrong decisions on whether to accept x or not, but as long as this does not happen too often, we will still expect to make progress towards the optimum. To analyze this, we formulate the progress of the distance to the optimum as a random walk and use the gambler's ruin theorem to show that the expected number of visits to each state is constant.

One key notion used in the proof of Theorem 7 is that of a p-estimator. It is used to estimate the number of zeroes in a bit string. Since the idea underlying this notion will be re-used in Theorem 14, we briefly present its definition.

Definition 8. *Let f be a pseudo-Boolean function and let p be a function that maps non-negative integers to non-negative integers. Let g be an algorithm which takes as input a bit string x and a natural number α and uses $O(p(n)\alpha \log(2 + n/\alpha))$ unary unbiased queries to f. We call g a* p-estimator using f *if, for all bit strings x, $a = n - \textsc{OneMax}(x)$, and for all $\alpha \in [a/2, 3a/2]$ we have*

- $P(g(x, \alpha) \neq a) \leq \frac{a}{16n}$;
- $P(g(x, \alpha) \notin [a/2, 3a/2]) \in O(1/(p(n)n^3))$.

To prove Theorem 7 we show that there exists a p-estimator for $\textsc{Jump}_{n/2-\varepsilon n}$ with $p(n)$ being a sufficiently large constant. The result then follows from the following lemma.

Lemma 9. *Let f be a pseudo-Boolean function such that, for some p, there is a p-estimator using f. Then the unary unbiased black-box complexity of f is $O(p(n)n^2)$.*

5. EXTREME JUMP FUNCTIONS

In this section, we regard the most extreme case of jump functions where all search points have fitness zero, except for the optimum and search points having exactly $n/2$ ones. Surprisingly, despite some additional difficulties, we still find polynomial-time black-box algorithms.

Throughout this section, let n be even. We call a jump function $\textsc{Jump}_\ell$ an *extreme jump function* if $\ell = n/2 - 1$. Consequently, this functions is zero except for the optimum (where it has the value n) and for bit-string having $n/2$ ones (where it has the value $n/2$).

The information-theoretic argument of [12] immediately gives a lower bound of $\Omega(n)$ for the unbiased black-box complexities of extreme jump functions. The intuitive argument is that an unrestricted black-box algorithm needs to learn n bits of information, but receives only a constant amount of information per query.

Lemma 10. *For all arities k, the k-ary unbiased black-box complexity of an extreme jump function is $\Omega(n)$.*

Proof. Since an extreme jump function takes only three values, Theorem 2 in [12] gives a lower bound of $\Omega(n)$ for the unrestricted black-box complexity of the set of all extreme jump functions. The latter is a lower bound for the unbiased black-box complexity of a single extreme jump function (cf. the end of Section 2). □

5.1 The Upper Bounds on Extreme Jump Functions

In the following three subsections, we shall derive several upper bounds for the black-box complexities of extreme jump functions.

Notice that, for an extreme jump function, we cannot distinguish between having a $\textsc{OneMax}$ value of $n/2 + k$ and $n/2-k$ until we have encountered the optimum or its inverse. More precisely, let $x^{(1)}, x^{(2)}, \ldots$ be a finite sequence of search points not containing the all-ones and all-zeroes string. Define $y^{(i)}$ to be the inverse of $x^{(i)}$ for all i, i.e., $y^{(i)} = x^{(i)} \oplus \mathbf{1}_n$ where $\mathbf{1}_n$ denotes the all-ones string of length n. Then both these sequences of search points yield exactly the same fitness values. Hence the only way we could find out on which side of the symmetry point $n/2$ we are would be by querying a search point having no or n ones. However, if we know such a search point, we are done anyway.

Despite these difficulties, we will develop a linear time ternary unbiased black-box algorithm in the following section. In Section 5.3, we show that restricting ourselves to binary variation operators increases the black-box complexity by at most a logarithmic factor. For unary operators, the good news derived in the final subsection of this section is that polynomial-time optimization of extreme jump functions is still possible, though the best complexity we find is only $O(n^{9/2})$.

To ease the language, let us denote by $d(x) := |\textsc{OneMax}(x) - n/2|$ a *symmetricized version of* $\textsc{OneMax}$ taking into account this difficulty. Also, let us define the *sign* $\operatorname{sgn}(x)$ of x to be -1, if $\textsc{OneMax}(x) < n/2$, $\operatorname{sgn}(x) := 0$, if $\textsc{OneMax}(x) = n/2$, and $\operatorname{sgn}(x) = +1$, if $\textsc{OneMax}(x) > n/2$. In other words, $\operatorname{sgn}(x)$ is the sign of $\textsc{OneMax}(x)-n/2$.

5.2 Ternary Unbiased Optimization of Extreme Jump Functions

When ternary operators are allowed, we quite easily obtain an unbiased black-box complexity of $O(n)$, which is best possible by Lemma 10. The reason for this fast optimization progress is that, as explained next, we may test individual bits. Assume that we have a search point u with OneMax-value $n/2+1$. If we flip a certain bit in u, then from the fitness of this offspring, we learn the value of this bit. If the new fitness is $n/2$, then the OneMax-value is $n/2$ as well and the bit originally had the value one. If the new fitness is zero, then the new OneMax-value is $n/2+2$ and the original bit was set to one. We can thus first learn the correct bit values and then copy them one by one into a bit string, creating the optimum (this is the only part of the algorithm which requires a ternary operator).

One difficulty to overcome, as sketched in Section 5.1, is that we will never have a search point where we know that its OneMax-value is $n/2+1$. We overcome this by generating a search point with fitness $n/2$ and flipping a single bit. This yields a search point with OneMax-value either $n/2+1$ or $n/2-1$. Implementing the above strategy in a sufficiently symmetric way, we end up with a search point having OneMax-value either n or 0 and in the latter case output its complement.

Theorem 11. *For $k \geq 3$, the k-ary unbiased black-box complexity of extreme jump functions is $\Theta(n)$.*

5.3 Binary Unbiased Optimization of the Extreme Jump Function

In this section, we prove that the unbiased 2-ary black-box complexity of extreme jump functions is $O(n \log n)$. With 2-ary operators only, it seems impossible to copy bits one by one, which was crucial to the strategy used in the previous subsection.

To overcome this difficulty, we follow a hill-climbing approach. We first find a search point m with d-value 0 by repeated sampling. We copy this into our "current-best" search point x and try to improve x to a new search point x' by flipping a random bit in which x and m are equal (this needs a 2-ary operation), hoping to gain a search point with d-value equal to $d(x)+1$. The main difficulty is to estimate the d-value of x', which is necessary to decide whether we keep this solution as new current-best or whether we try again.

Using binary operators, we can exploit the fact that $H(x,m) = d(x)$. For example, we can flip $d(x)-1$ of the $d(x)+1$ bits in which x' and m differ. If this yields an individual with fitness $n/2$, then clearly x' has not the targeted d-value of $d(x)+1$. Unfortunately, we detect this shortcoming only when the bit that marks the difference of x and x' is not among the $d(x)-1$ bits flipped. This happens only with probability $2/(d(x)+1)$. Consequently, this approach may take $\Theta(n)$ iterations to decide between the cases $d(x') = d(x)+1$ and $d(x') = d(x)-1$.

We can reduce this time to logarithmic using the following trick. Recall that the main reason for the slow decision procedure above is that the probability of not flipping the newly created bit is so small. This is due to the fact that the only way to gain information about x' is by flipping almost all bits so as to possibly reach a fitness of $n/2$. We overcome this difficulty by in parallel keeping a second search point y that has the same d-value as x, but is "on the other side" of m. To ease the language in this overview, let us assume that OneMax$(x) > n/2$. Let $k := d(x)$ and $H(m,x) = k$. Then we aim at keeping a y such that $d(y) = k$, $H(m,y) = k$, $H(x,y) = 2k$, and OneMax$(y) = n/2-k$. With this at hand, we can easily evaluate the d-value of x'. Assume that x' was created by flipping exactly one of the bits in which x and y agree. Let u be created by flipping in x' exactly $k-1$ of the bits in which x' and y differ. If $d(x') = k+1$, then surely $d(u) = 2$, and thus $f(u) = 0$. If $d(x') = k-1$, then with probability $(k+2)/(2k+1) \geq 1/2$ the bit in which x' and x differ is not flipped, leading to OneMax$(u) = n/2$, visible from a fitness equal to $n/2$. Hence, with probability at least $1/2$, we detect the undesired outcome $d(x') = k-1$. Unfortunately, there is no comparably simple certificate for "$d(x') = k+1$", so we have to repeat the previous test $2\log n$ times to be sufficiently sure (in the case no failure is detected) that $d(x') = k+1$. Overall, this leads to an almost linear complexity of $O(n \log n)$.

This finishes a rough outline of the proof for Theorem 12. The algorithm verifying Theorem 12 is given in Algorithm 2. It uses the following unbiased operators. The first, `uniform()`, is the (0-ary unbiased) operator that samples a bit string $x \in \{0,1\}^n$ uniformly at random. $\texttt{flipWhereEqual}_1(\cdot,\cdot)$, takes two arguments and returns a bit string which is like the first argument, except that exactly one of the bits in which the first argument agrees with the second one is flipped, this bit is chosen uniformly at random; if the two strings are complements of each other, then no bits are flipped and the operator returns the first argument. The operator $\texttt{mix}(\cdot,\cdot)$ also takes two arguments; if these two arguments differ at exactly two positions, the output is exactly like the input strings where the two inputs are equal, and inherits one bit from each argument on the two positions where they differ, but if the two arguments

Algorithm 2: A 2-ary unbiased black-box algorithm that for any extreme jump function f with high probability finds the optimum in $O(n \log n)$ iterations.

```
repeat
    m ← uniform();
until f(m) = n/2;
status ← failure;
while status = failure do
    x ← flipWhereEqual_1(m, m);
    y ← flipWhereEqual_1(m, x);
    status ← success;
    i ← 0;
    while (status ≠ failure) and (i ≤ √n) do
        i ← i + 1;
        u ← mix(x, y);
        if f(u) ≠ n/2 then status ← failure;
for k = 1 to n/2 − 1 do
    x' ← movefirst_k(x, y);
    y' ← movefirst_k(y, x);
    (x, y) ← (x', y');
if f(x) = n then
    return x;
else
    return y;
```

do not differ at exactly two positions, a uniformly random bit string is returned. Finally, the algorithm also uses an unbiased subroutine $\texttt{movefirst}_k(\cdot,\cdot)$ which takes two arguments x and y and assumes $d(x) = k = d(y)$ as well as $H(x,y) = 2d$. The subroutine returns (with sufficient probability and only using unbiased operators) a bit string x' with $d(x') = k+1$ and $d(x',y) = 2k+1$.

THEOREM 12. *There is a 2-ary unbiased black-box algorithm solving the "extreme jump functions" problem in an expected number of $O(n \log n)$ queries.*

5.4 Unary Unbiased Optimization of Extreme Jump Functions

With the next theorem we show that, surprisingly, even the *unary* unbiased black-box complexity of extreme jump functions is still polynomially bounded. Note that now we cannot learn the ONEMAX-value of a search point x by repeatedly flipping a certain number of bits and observing the average objective value. Since $n/2$ is the only non-trivial objective value, any such average will necessarily be $n/2$ (except in the unlikely event that we encountered the optimum). The solution is to flip, depending on parity reasonings, exactly $n/2-1$ or $n/2$ bits in x and note that the probability p_a of receiving a search point with (visible) fitness $n/2$ depends on the distance $a := a(x) = \min\{|x|_1, |x|_0 := n - |x|_1\}$ of x to the optimum or its opposite. We roughly have $p_a \in \Theta(a^{-1/2})$ and $p_{a-2} - p_a \in \Theta(a^{-3/2}n^{-1}(n-2a))$. These small numbers lead to the fact that for $a \in n/2 - \Theta(1)$, we will need $\Theta(n^{9/2})$ samples to estimate the a-value of x with constant probability. Since estimating becomes easier for smaller a, we are able to construct a unary unbiased black-box algorithm finding the optimum of an extreme jump function in $O(n^{9/2})$ expected fitness evaluations.

Key to proving Theorem 14 is Lemma 13, formalizing that it is possible to distinguish, in a unary unbiased manner, whether a search point x has distance $a(x) = a-1$ or $a(x) = a+1$ to the optimum (or its complement).

LEMMA 13. *There exists a unary unbiased procedure* `estimate` *that has the following properties. On arbitrary inputs $y \in \{0,1\}^n$ and $a \in [1\,..\,n/2-1]$, it performs $\Theta(a^{5/2}n^2(n-2a)^{-3/2})$ fitness evaluations. If $a(y) \in \{a-1, a+1\}$, then with probability at least $1 - \exp(-\Theta((n-2a)^{1/2}))$ the true value of $a(y)$ is returned. By choosing the implicit constant in the first statement sufficiently large, this success probability can be made arbitrary close to one.*

The algorithm certifying the desired black-box complexity stated in Table 1 is given by Algorithm 3. In this pseudocode, `uniform`() is as before, $\texttt{flip}_1(.)$ is the unary unbiased operator that, given an argument x flips in x exactly one bit, and `complement`(.) is the unary unbiased operator which outputs the bitwise complement of its argument, i.e., $\texttt{complement}(x) := x \oplus \mathbf{1}_n$ for all x.

THEOREM 14. *Let n be even and $\ell = n/2 - 1$. Then the unary unbiased black-box complexity of JUMP_ℓ is $O(n^{9/2})$. This is witnessed by Algorithm 3, which with constant probability finds the optimum of an extreme jump function using $O(n^{9/2})$ fitness evaluations.*

PROOF. The first sentence of the theorem follows from the second and Remark 1.

Algorithm 3: A unary unbiased black-box algorithm that for any extreme jump function f with constant probability finds the optimum in $O(n^{9/2})$ fitness evaluations.

```
repeat
    m ← uniform();
until f(m) = n/2;
x ← flip_1(m);
for a = n/2 − 1 DownTo 1 do
    status ← failure;
    while status = failure do
        y ← flip_1(x);
        if estimate(y, a) = a − 1 then
            x ← y;
            status ← success;
Sample complement(x);
```

In the analysis of Algorithm 3, let us first assume that all $\texttt{estimate}(y,a)$ calls with $a(y) \in \{a-1, a+1\}$ return $a(y)$ correctly.

Assume that we start the while-loop in Algorithm 3 with an x such that a is equal to $a(x)$. Since y is a Hamming neighbor of x, we have $a(y) \in \{a-1, a+1\}$. If $a(y) = a+1$, nothing changes. If $a(y) = a-1$, this is again correctly detected in the if-clause, and the while-loop is left with $x \leftarrow y$. Consequently, we start the following iteration of the for-loop again with $a(x) = a$. The probability of generating a y with $a(y) = a-1$ is a/n. Consequently, the while-loop is left after an expected number of n/a iterations.

The expected total number of fitness evaluations in the for-loop is now easily computed as

$$\sum_{a=1}^{n/2-1} (n/a) O(a^{5/2}n^2/(n-2a)^{3/2}) \in O(n^{9/2}) \sum_{a=1}^{n/2-1} (n-2a)^{-3/2} = O(n^{9/2}).$$

The above is true if we assume that none of the exceptional events ("failures") of Lemma 13 occurs. We now argue that in fact with constant probability, none of them occurs. To this aim, we estimate the expected number of first failures in a typical run of the algorithm (a first failure is one where all previous calls of the `estimate` procedure did not fail). Consider one iteration of the while loop. If $a \neq a(x)$, then a failure must have occurred before, hence the probability now for a first failure is zero. If $a = a(x)$, we can invoke Lemma 13 and deduce that this iteration has a failure probability of at most $\exp(-k(n-2a)^{1/2})$, where k is a sufficiently large absolute constant.

We may further assume, for the sake of this argument, that a failure is immediately corrected by some external authority. Note that this only changes the run of the algorithm after the occurrence of the first failure. So in particular, it does not change the expected number of first failures. By this, however, we may assume that the expected number of iterations done with x having a certain a-value is exactly n/a. Consequently, the expected number of first failures is at most $\sum_{a=1}^{n/2-1}(n/a)\exp(-k(n-2a)^{1/2}) = \sum_{b=1}^{n/2-1} n(n/2-b)^{-1}\exp(-k\sqrt{b}) \in O(1)$, where the implicit constant can be made arbitrarily small by the appropriate

choice of k. Hence, with constant probability there is no first failure, and thus, also no failure at all. □

6. SUMMARY AND OUTLOOK

We have analyzed the unbiased black-box complexity of short, long, and extreme jump functions. Along the way, we have introduced new tools for such analyses. Our work raises a number of interesting questions for future research.

Since our focus was on deriving new ideas for the design of new search heuristics, we did not undertake in this work a complete investigation of all possible combinations of arity and jump size, but rather highlighted prominent complexity behaviors and prototypical algorithmic ideas. Still, it would be interesting to have a more complete picture than Table 1, in particular, making clear how far certain algorithmic ideas take and where certain regimes change.

Another interesting line of research would be results that are more precise than just the asymptotic order. For example, it seems reasonable that for ℓ small enough, the unary unbiased black-box complexity of $\textsc{Jump}_\ell$ is not only of the same order as the one of OneMax, but equal apart from lower order terms (which might actually be surprisingly small). Note that such precise analyses for run times of given algorithms recently attracted quite some interest, see [1, 17, 20, 22] and the references therein.

Furthermore, we are optimistic that some of the algorithmic ideas developed in the previous sections can be used to design new search heuristics.

Acknowledgments

Parts of this work have been done during the Dagstuhl seminar 10361 "Theory of Evolutionary Algorithms".

7. REFERENCES

[1] S. Böttcher, B. Doerr, and F. Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Proc. of the 11th International Conference on Parallel Problem Solving from Nature (PPSN'10)*, pages 1–10. Springer, 2010.

[2] B. Doerr. Analyzing randomized search heuristics: Tools from probability theory. In A. Auger and B. Doerr, editors, *Theory of Randomized Search Heuristics*, pages 1–20. World Scientific Publishing, 2011.

[3] B. Doerr, C. Doerr, and F. Ebel. Lessons from the black-box: Fast crossover-based genetic algorithms. In *Proc. of the 15th Annual Genetic and Evolutionary Computation Conference (GECCO'13)*, pages 781–788. ACM, 2013.

[4] B. Doerr, C. Doerr, and T. Kötzing. Unbiased black-box complexities of jump functions—how to cross large plateaus. *CoRR*, abs/1403.7806, 2014. Available online at http://arxiv.org/abs/1403.7806.

[5] B. Doerr, D. Johannsen, T. Kötzing, P. K. Lehre, M. Wagner, and C. Winzen. Faster black-box algorithms through higher arity operators. In *Proc. of the 11th ACM Workshop on Foundations of Genetic Algorithms (FOGA'11)*, pages 163–172. ACM, 2011.

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

[7] B. Doerr, T. Kötzing, and C. Winzen. Too fast unbiased black-box algorithms. In *Proc. of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 2043–2050. ACM, 2011.

[8] B. Doerr and C. Winzen. Ranking-based black-box complexity. *Algorithmica*. To appear. DOI: 10.1007/s00453-012-9684-9.

[9] B. Doerr and C. Winzen. Playing Mastermind with constant-size memory. In *Proc. of the Symposium on Theoretical Aspects of Computer Science (STACS'12)*, pages 441–452. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[10] B. Doerr and C. Winzen. Reducing the arity in unbiased black-box complexity. In *Proc. of the 14th Annual Genetic and Evolutionary Computation Conference (GECCO'12)*, pages 1309–1316. ACM, 2012.

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

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

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

[14] T. Jansen and I. Wegener. The analysis of evolutionary algorithms - a proof that crossover really can help. *Algorithmica*, 34:47–66, 2002.

[15] P. K. Lehre and C. Witt. Black-box search by unbiased variation. In *Proc. of the 12th Annual Genetic and Evolutionary Computation Conference (GECCO'10)*, pages 1441–1448. ACM, 2010.

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

[17] P. K. Lehre and C. Witt. General drift analysis with tail bounds. *CoRR*, abs/1307.2559, 2013. Available online at http://arxiv.org/abs/1307.2559.

[18] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. *SIAM Journal on Computing*, 26:350–368, 1997.

[19] J. Rowe and M. Vose. Unbiased black box search algorithms. In *Proc. of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO'11)*, pages 2035–2042. ACM, 2011.

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

[21] O. Teytaud and S. Gelly. General lower bounds for evolutionary algorithms. In *Proc. of the 9th International Conference on Parallel Problem Solving from Nature - PPSN IX (PPSN'06)*, pages 21–31. Springer, 2006.

[22] C. Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing*, 22:294–318, 2013.