
Unbiased Black-Box Complexities of Jump Functions

Benjamin Doerr

École Polytechnique, Palaiseau, France

Carola Doerr

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

Timo Kötzing

Friedrich-Schiller-Universität Jena, Germany

doi:10.1162/EVCO_a_00158

Abstract

We analyze the unbiased black-box complexities of jump functions with small, medium, and large sizes of the fitness plateau surrounding the optimal solution. Among other results, we show that when the jump size is $(1/2 - \varepsilon)n$, that is, when 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 only by comparing the fitnesses of the competing search points but also by taking into account the (empirical) expected fitnesses of their offspring.

Keywords

Black-box complexity, theory, runtime analysis.

1 Introduction

The analysis of black-box complexities in evolutionary computation aims in several complementary ways at supporting the development of superior evolutionary algorithms. Comparing the runtime of currently used randomized search heuristics (RSH) with that of an optimal black-box algorithm allows a fair evaluation of the quality of today's heuristics. 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 runtime of RSH. Finally, research in black-box complexity proved to be a source of inspiration for developing new algorithmic ideas leading to the design of better search heuristics.

In this work, we analyze the unbiased black-box complexities of jump functions, which are observed as difficult for evolutionary approaches because of their large plateaus of constant (and low) fitness. Our results show surprisingly that even extreme

This paper is based on results published in the conference versions Doerr et al. (2011b; 2014a).

jump functions revealing 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. Perhaps the 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 nonstandard 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 of Droste et al. (2006), which does not restrict in any way the sampling or selection procedure of the algorithm; the ranking-based model of Teytaud and Gelly (2006) and Doerr and Winzen (2014a), in which the algorithms are required to base their selection only on relative and not on absolute fitness values; the memory-restricted model of Droste et al. (2006) and Doerr and Winzen (2012), in which the algorithm can store only a limited number of search points and their corresponding fitness values; and the unbiased model of Lehre and Witt (2012), which in addition to allowing the classification of algorithms according to the arity of the variation operators, 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 runtime of RSH is influenced by certain algorithmic choices, such as the population size, the use of crossover, the selection rules, and so on.

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 shown by Doerr et al. (2015). In that work, a new genetic algorithm is presented that optimizes the OneMax function in runtime $o(n \log n)$, thus showing that the $o(n \log n)$ bound for the binary unbiased black-box complexity of OneMax found by Doerr et al. (2011a) is not as unnatural as it might at first seem.

Here 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 (2012) for bit string representations and was later generalized to domains different from bit strings by Rowe and Vose (2011) and Doerr et al. (2013). 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

Table 1: Unbiased black-box complexities of Jump_ℓ for different regimes of ℓ . The $\Omega(n \log n)$ lower bound for arity $k = 1$ follows from Lehre and Witt (2012, Theorem 6). All other results are original to the present paper. The binary and ternary upper bounds for long jump functions follow from the corresponding ones for extreme jump functions.

Arity	Short Jump $\ell = O(n^{1/2-\varepsilon})$	Long Jump $\ell = (1/2 - \varepsilon)n$	Extreme Jump $\ell = n/2 - 1$
$k = 1$	$\Theta(n \log n)$	$O(n^2)$ Thm. 14	$O(n^{9/2})$ Thm. 20
$k = 2$	$O(n)$	$O(n \log n)$ Thm. 4	$O(n \log n)$ Thm. 18
$3 \leq k \leq \log n$	$O(n/k)$	$O(n/k)$ Cor. 7	$\Theta(n)$ Thm. 16, Lem. 15

evaluated solutions via operators that are unbiased (i.e., treat positions and bit contents equally). Section 3 gives 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, not fully agreeing, definitions used by Droste et al. (2002) and Lehre and Witt (2010). See Section 2.

Jump functions are well-known test functions for randomized search heuristics. Droste et al. (2002) 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 runtime 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 $\Theta(2^n)$ runtime becomes superior). For all ℓ , Jansen and Wegener (2002) 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 time¹ of $O(n \log^3 n)$ for constant ℓ and an optimization time of $O(n^{2c+1} \log n)$ for $\ell = \lceil c \log n \rceil$, c a constant.

1.3 Results

We analyze the unbiased black-box complexity of Jump functions for a broad range of jump sizes ℓ . We distinguish between *short*, *long*, and *extreme* jump functions for $\ell = O(n^{1/2-\varepsilon})$, $\ell = (1/2 - \varepsilon)n$, and $\ell = n/2 - 1$, respectively. Our findings are summarized in Table 1.

¹This is stated as $O(n \log^2 n \log \log n)$ in Jansen and Wegener (2002), but that is clearly an oversight.

Contrasting the runtime results for classic evolutionary approaches on Jump, we show that for jump functions with *small* jump sizes $\ell = O(n^{1/2-\varepsilon})$, 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 $\ell = 0$). As an intermediate result we prove (Lemma 3) that a black-box algorithm having access to a jump function with $\ell = O(n^{1/2-\varepsilon})$ can retrieve (with high probability) the true OneMax value of a search point using only a constant number of queries. This implies that we get the same runtime bounds for short jump functions as are known for OneMax. For $k = 1$ this is $\Theta(n \log n)$ (Lehre and Witt, 2012); for $k = 2$ it is $O(n)$ (Doerr et al., 2011a); and for $3 \leq k \leq \log n$ it is $O(n/k)$ (Doerr and Winzen, 2014b).

A result like Lemma 3 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, 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.

Our results indicate that even without the fitness function revealing useful information for search points close to the optimum, efficient optimization is still possible in the framework of unbiased black-box algorithms (provided there is enough knowledge about the location of the optimum).

The results regarding short jump can be found in Section 4, the results on long jump in Section 5, and the results on extreme jump in Section 6. Note that the bound on the binary unbiased black-box complexity of long jump follows from the same bound on extreme jump. The lower bounds partly follow from a more general result of independent interest (Theorem 2), which implies that for all $0 \leq \ell_1 \leq \ell_2$, for all k , the k -ary unbiased black-box complexity of Jump_{ℓ_1} is less than or equal to that of Jump_{ℓ_2} .

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 nonzero 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 Srinivasan (1997) and is of independent interest.

2 Jump Functions

As mentioned, several definitions for jump functions exist. We use here the version that is inspired by the idea of blanking out the full ℓ -neighborhood of the optimum and of its complement (the latter is needed because otherwise one could optimize the function by simply searching for the unique search point x' with Jump_ℓ value zero and complementing it). To be precise, for all $\ell < n/2$, Jump_ℓ is the function that assigns to each $x \in \{0, 1\}^n$ fitness

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

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

The jump function f_ℓ analyzed by Droste et al. (2002, def. 24) assigns to x fitness value

$$f_\ell(x) := \begin{cases} \ell + n, & \text{if } |x|_1 = n, \\ \ell + |x|_1, & \text{if } |x|_1 \leq n - \ell, \\ n - |x|_1, & \text{otherwise.} \end{cases}$$

Not only can this function be optimized by searching for the complement of the optimum (as described), but it also provides more information for those x with $n - \ell < |x|_1 < n$. While for classic runtime analysis of randomized search heuristics this does not pose any problems, these properties are not desirable for black-box complexity studies. Lehre and Witt (2010) therefore designed a different jump function g_ℓ , assigning to each x fitness value

$$g_\ell(x) := \begin{cases} n, & \text{if } |x|_1 = n, \\ |x|_1, & \text{if } \ell < |x|_1 \leq n - \ell, \\ 0, & \text{otherwise.} \end{cases}$$

Our version is mostly similar to the latter, with the only difference being the function values for bit strings x with $|x|_1 = n - \ell$. In our version the sizes of the blanked out areas around the optimum and its complement are equal, while for g_ℓ that area is larger around the complement than around the optimum.

Jansen (2015) introduced yet another version of the jump function, inspired by the idea that the spirit of jump functions is to “[locate] an unknown target string that is hidden in some distance to points a search heuristic can find easily.” Jansen’s definition also has black-box complexity analysis in mind. For some search point x^* with $|x^*|_1 > n - \ell$, his jump function h_{ℓ, x^*} assigns to bit string x the fitness value

$$h_{\ell, x^*}(x) := \begin{cases} n + 1, & \text{if } x = x^*, \\ n - |x|_1, & \text{if } n - \ell < |x|_1 \leq n, \\ \ell + |x|_1, & \text{otherwise.} \end{cases}$$

Since these functions do not reveal information about the optimum other than its ℓ -neighborhood, the (unrestricted) black-box complexity of the class $\{h_{\ell, x^*} \mid |x^*|_1 > n - \ell\}$ is $(\sum_{i=0}^{\ell-1} \binom{n}{i} + 1)/2$ (Jansen, 2015, Theorem 4). For constant ℓ this expression is $\Theta(n^{\ell-1})$, very different from the results on the unrestricted black-box complexity of Jump_ℓ in Buzdalov et al. (2015) or from our results.

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

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

3 The Unbiased Black-Box Model

The unbiased black-box model introduced in Lehre and Witt (2012) 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 et al. (2006). 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 give a brief definition of the unbiased black-box model, pointing to Lehre and Witt (2012) and Doerr and Winzen (2014b) for a more detailed introduction.

For all non-negative integers k , a k -ary unbiased distribution $(D(\cdot \mid y^{(1)}, \dots, y^{(k)}))_{y^{(1)}, \dots, y^{(k)} \in \{0, 1\}^n}$ 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] $\forall x \in \{0, 1\}^n \forall \sigma \in S_n :$

$$D(x \mid y^{(1)}, \dots, y^{(k)}) = D(\sigma(x) \mid \sigma(y^{(1)}), \dots, \sigma(y^{(k)})),$$

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 $UBB_k(\mathcal{F})$, 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 $*\text{-ary unbiased black-box complexity}$ 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 Doerr et al. (2014b) that helps us shorten 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.

THEOREM 2: For all sets of pseudo-Boolean functions C , all $k \in \mathbb{N}$, and all $f : \mathbb{R} \rightarrow \mathbb{R}$ such that for all $g \in C$ with $\{x \mid f(g(x)) \text{ optimal}\} = \{x \mid g(x) \text{ optimal}\}$ we have $UBB_k(C) \leq UBB_k(f(C))$.

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. \square

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 by Droste et al. (2006) and Erdős and Rényi (1963). A similar line of arguments proves the lower bound for extreme jump functions (see Section 6).

The lower bound for the unary unbiased black-box complexity of Jump follows immediately from the $\Omega(n \log n)$ bound proven by Lehre and Witt (2012, Theorem 6) for all pseudo-Boolean functions with unique global optimum.

4 Short Jump Functions

The key idea for obtaining the bounds on short jump functions, i.e., jump functions with jump size $\ell \in O(n^{1/2-\varepsilon})$, is the following lemma. It shows that one can compute, with high probability, the OneMax value of any search point x with few black-box calls to Jump_ℓ . With this, we can orient ourselves on the large plateau surrounding the optimum and thus revert to the problem of optimizing OneMax.

We collect these computations in a subroutine, to be called by black-box algorithms.

LEMMA 3: For all constants $\varepsilon, c > 0$ 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 , $s(x) = \text{OneMax}(x)$ with probability $1 - O(n^{-c})$.

PROOF: We assume n to be large enough so that $\ell \leq n/4$. We use a unary unbiased variation operator flip_ℓ , which samples uniformly an ℓ -neighbor (a bit string that differs in exactly ℓ positions) of the argument. Next we give the subroutine s , which uses Jump_ℓ to approximate OneMax as desired (see Algorithm 2). Intuitively, the subroutine samples $t = \lceil c/(2\varepsilon) \rceil \in O(1)$ bit strings in the ℓ -neighborhood of x ; if $|x|_1 \geq n - \ell$, then it is likely that at least once only 1s of x have been flipped, leading to a Jump_ℓ value of $|x|_1 - \ell$; as no sample will have a lower Jump_ℓ value, adding ℓ to the minimum non-zero fitness of one of the sampled bit strings gives the desired output. The case of x with $|x|_1 \leq \ell$ is analogous.

Algorithm 2 Simulation of OneMax using the jump function

```

1 subroutine  $s(x)$  is
2   if  $\text{Jump}_\ell(x) \neq 0$  then output  $\text{Jump}_\ell(x)$ ;
3    $M \leftarrow \{\text{JUMP}_\ell(\text{flip}_\ell(x)) \mid i \in [\lceil c/(2\varepsilon) \rceil]\}$ ;
4   if  $\max(M) < n/2$  then  $m \leftarrow \max(M) - \ell$ ;
5   else  $m \leftarrow \min(M \setminus \{0\}) + \ell$ ;
6   output  $m$ ;
```

Clearly, the subroutine is correct with certainty on all x with $\ell < |x|_1 < n - \ell$. The other two cases are nearly symmetric, so we only analyze x with $|x|_1 \geq n - \ell$. Clearly, the return value of the subroutine is correct if and only if at least one of the t samples flips only 1s in x (note that $\max(M) > n/2$ holds due to $\ell \leq n/4$). We denote the probability of this event with p . We start by bounding the probability that a single sample flips only 1s. We choose which k bits to flip iteratively so that after i iterations, there are at least $n - \ell - i$ bit positions with a 1 out of $n - i$ unchosen bit positions left to choose. This gives the bound of

$$\begin{aligned} & \left(\frac{n - \ell}{n}\right) \cdot \left(\frac{n - \ell - 1}{n - 1}\right) \cdots \left(\frac{n - \ell - (\ell - 1)}{n - (\ell - 1)}\right) \\ &= \prod_{i=0}^{\ell-1} \left(1 - \frac{\ell}{n - i}\right) \geq \left(1 - \frac{\ell}{n - \ell}\right)^\ell \geq \left(1 - \frac{\ell^2}{n - \ell}\right), \end{aligned}$$

using Bernoulli's inequality. Let c' be such that $\ell \leq c'n^{1/2-\varepsilon}$. We have

$$p \geq 1 - \left(\frac{\ell^2}{n - \ell}\right)^t \geq 1 - \left(\frac{2\ell^2}{n}\right)^t \geq 1 - (2c'n^{-2\varepsilon})^t \geq 1 - (2c')^t n^{-c}.$$

□

With Lemma 3 at hand, the results stated in Table 1 follow easily from the respective OneMax bounds proven by Doerr et al. (2011a), Doerr and Winzen (2012), and Droste et al. (2006).

THEOREM 4: For $\varepsilon > 0$ and $\ell \in O(n^{1/2-\varepsilon})$, the unbiased black-box complexity of Jump_ℓ is $O(n \log n)$ for unary variation operators, and it is $O(n/k)$ for k -ary variation operators with $2 \leq k \leq \log n$.

PROOF: First note that the black-box complexities claimed for Jump_ℓ are shown for OneMax in Droste et al. (2006) for $k = 1$, in Doerr et al. (2011a) for $k = 2$, and in Doerr and Winzen (2014b) for $3 \leq k \leq \log n$.

We use Lemma 3 with $c = 4$ and run the unbiased black-box algorithms of the appropriate arity for OneMax; all sampled bit strings are evaluated using the subroutine s . Thus, this algorithm samples as if working on OneMax and finds the bit string with all 1s after the desired number of iterations. Note that for up to $n \log n$ uses of s , we expect no more than $n \log(n) \cdot O(n^{-4}) = O(n^{-2})$ incorrect evaluations of s . Therefore, there is a small chance of failing, and the claim follows from Remark 1. □

5 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.

5.1 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 in Section 4, we can revert to optimizing OneMax, and the missing fitness values will not show in any partial problems.

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

LEMMA 5: *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 $\text{OneMax}(z)$ 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.*

PROOF: Let x and y with Hamming distance a be given as detailed in the statement of the theorem. Note that $[x, y]$ is isomorphic to $\{0, 1\}^a$. Without loss of generality, assume that x and y differ on the first a bits, and let \tilde{x} be the last $n - a$ bits of x (which equal the last $n - a$ bits of y). Thus, $[x, y] = \{z\tilde{x} \mid z \in \{0, 1\}^a\}$.

We employ A optimizing $\{0, 1\}^a$. Sampling a uniformly random point $z\tilde{x}$ in $[x, y]$ is clearly unbiased in x and y . However, the resulting OneMax value is not the value that A requires, unless \tilde{x} is the all-0s string. In order to correct for this, we need to know the number of 1s $|\tilde{x}|_1$ in \tilde{x} . This we can compute from $|x|_1, |y|_1$ and a as follows. Let z_x and z_y be such that $x = z_x\tilde{x}$ and $y = z_y\tilde{x}$. We have

$$\frac{|x|_1 + |y|_1 - a}{2} = \frac{|z_x|_1 + |z_y|_1 + 2|\tilde{x}|_1 - a}{2} = \frac{|z_x|_1 + a - |z_x|_1 + 2|\tilde{x}|_1 - a}{2} = |\tilde{x}|_1.$$

Thus, for any bit string $z\tilde{x}$ sampled by `simulateOnSubcube` we can pass the OneMax value of z to A . In iteration t , when A uses a k -ary unbiased operator that samples according to the distribution $D(\cdot \mid x^{(i_1)}, \dots, x^{(i_k)})$, `simulateOnSubcube` uses the $(k + 2)$ -ary unbiased operator that samples according to the distribution $D'(\cdot \mid x^{(i_1)}\tilde{x}, \dots, x^{(i_k)}\tilde{x}, x, y)$ such that

$$\forall u \in \{0, 1\}^n : D'(u \mid x^{(i_1)}\tilde{x}, \dots, x^{(i_k)}\tilde{x}, x, y) = \begin{cases} D(z \mid x^{(i_1)}, \dots, x^{(i_k)}), & \text{if } u = z\tilde{x}, \\ 0, & \text{otherwise.} \end{cases}$$

For any $v^{(1)}, \dots, v^{(k+2)}$ such that $D'(\cdot \mid v^{(1)}, \dots, v^{(k+2)})$ is not defined by this equation, we let this distribution be the uniform distribution over $\{0, 1\}^n$. From the additional conditioning on x and y we see that D' is indeed unbiased. Note that D' samples only points from $[x, y]$. As described, `simulateOnSubcube` can now use the OneMax value

Algorithm 3 k -ary unbiased black-box algorithm for Jump_ℓ

```

1 repeat
2   |  $x \leftarrow \text{uniform}()$ ;
3 until  $f(x) = \lceil n/2 \rceil$ ;
4  $z^{(0)} \leftarrow x$ ;
5 for  $i = 1$  to  $n/a$  do
6   |  $z^{(i)} \leftarrow \text{flipWhereEqual}_a(z^{(i-1)}, x)$ ;
7   |  $y^{(i)} \leftarrow \text{selectBits}(x, z^{(i-1)}, z^{(i)})$ ;
8 for  $i = 1$  to  $n/a$  do
9   |  $u^{(i)} \leftarrow \text{simulateOnSubcube}(x, y^{(i)}, a)$ ;
10  $b \leftarrow x$ ;
11 for  $i = 1$  to  $n/a$  do
12  |  $b \leftarrow \text{copySecondIntoFirstWhereDifferentFromThird}(b, u^{(i)}, x)$ ;

```

of the resulting $z\bar{x}$ to compute the OneMax value of z and pass that on to A as the answer to the query. This shows that the simulation is successful as desired. \square

THEOREM 6: Let $\ell \leq (1/2 - \varepsilon)n$. For all $k \geq 3$, the k -ary unbiased black-box complexity of Jump_ℓ is $O(\text{UBB}_{k-2}(\text{OneMax}))$.

PROOF: We optimize Jump_ℓ blockwise, where each block is optimized by itself and the correct OneMax value is available as long as only bits within the block are modified. Then the different optimized blocks are merged to obtain the optimum.

Let $a := \lfloor n/2 \rfloor - \ell$, and assume for the moment that a divides n . Algorithm 3 gives a formal description of the intuitive idea. This algorithm uses the following unbiased operators:

- $\text{uniform}()$. The operator uniform is a 0-ary operator that samples a bit string uniformly at random.
- $\text{flipWhereEqual}_k(x, y)$. For two search points x and y and an integer k , the operator flipWhereEqual_k generates a search point by randomly flipping k bits in x among those bits where x and y agree. If x and y agree in fewer than k bits, then all bits where x and y agree are flipped.
- $\text{selectBits}(x, y, z)$. For three search points x, y , and z , the operator selectBits returns a bit string identical to the first argument, except where the second and third differ; there the bits of x are flipped. Note that this operator is deterministic.
- $\text{copySecondIntoFirstWhereDifferentFromThird}(x, y, z)$. For three search points x, y , and z , the operator $\text{copySecondIntoFirstWhereDifferentFromThird}$ copies x , except where the second differs from the third; there it copies y . This is also a deterministic operator. \square

Furthermore, we use the subroutine simulateOnSubcube from Lemma 5 with a fixed time budget that guarantees constant success at each call, returning the best bit

string found (note that if a does not divide n , the last call to `simulateOnSubcube` has to be with respect to a different Hamming distance).

5.1.1 Expected Number of Queries

A uniformly sampled bit string has exactly $n/2$ 1s with probability $\Theta(1/\sqrt{n})$, which shows that the first line takes an expected number of $\Theta(\sqrt{n})$ queries. Since $a \in \Theta(n)$, all loops have a constant number of iterations. The body of the second loop takes as long as a single optimization of `OneMax` with arity k , which is in $\Omega(n/\log n)$, so that initial sampling in line 1 makes no difference in the asymptotic runtime. Thus, the total number of queries is $O(\text{UBB}_{k-2}(\text{OneMax}))$.

5.1.2 Correctness

The algorithm first generates a reference string x with $f(x) = \lceil n/2 \rceil$. The first loop generates bit strings $y^{(i)}$, which have a Hamming distance of a to the reference string x ; in this way the different $y^{(i)}$ partition the bit positions into $\lceil n/a \rceil$ sets of at most a positions each. The next loop optimizes (copies of) x on each of the selected sets of a bits independently as if optimizing `OneMax`. For the bit strings encountered during this optimization we always observe the correct `OneMax` value, as their Hamming distance to x is at most a , and x has exactly $\lceil n/2 \rceil$ 1s. The last loop copies the optimized bit positions into b by copying the bits in which $u^{(i)}$ and x differ (those are the incorrect ones). This selects the correct bits in each segment with constant probability according to Lemma 5. As all segments have a constant independent failure probability, we get a constant overall failure probability (since a is constant), and Remark 1 concludes the proof. The proof trivially carries over to the case of n not divisible by a .

Thus, we immediately get the following corollary, using the known run time bounds for `OneMax` from Doerr and Winzen (2014b).

COROLLARY 7: *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 improve in Section 6 to $O(n)$. For all higher arities, the theorem presented in this section gives the best known bound.

5.2 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 nonzero fitness. When ℓ is small, that is, when 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 `OneMax`(x). This was the main idea for dealing with short jump functions (see Section 4).

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 leads to a unary unbiased black-box complexity of $O(n^2)$ for all jump functions Jump_ℓ with $\ell \leq n/2 - \varepsilon n$.

5.2.1 Proof Outline and Methods

Since we aim at an asymptotic statement, let us assume that n is sufficiently large and even. Also, since we do 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 Jump_ℓ 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 $n/2 - \varepsilon n/2 + \varepsilon \text{OneMax}(x)$. If ε is constant, then by Chernoff bounds $O(n \log n)$ samples u are enough to ensure that the average observed fitness $n/2 - \varepsilon n/2 + \varepsilon v$ satisfies $v = \text{OneMax}(x)$ 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 1s—if $\text{OneMax}(x) = n - a$, then we observe that only $O(a \log n)$ samples suffice to estimate the 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 to 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 to argue that Chernoff bounds also hold for these. This was stated explicitly by Doerr (2011) 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 Panconesi and Srinivasan (1997).

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

5.2.2 Estimating the Distance to the Optimum

We start with some preliminary considerations that might be helpful for similar problems as well. Let $x \in \{0, 1\}^n$. Let $a := a(x) := n - \text{OneMax}(x) = H(x, \mathbf{1}_n)$ be its Hamming distance from the all-1s string. Fix some enumeration $1, \dots, a$ of the zero-bits of x . Let u be an offspring of x obtained from flipping exactly $n/2 - \varepsilon n/2$ bits. For $i \in [a]$, define a $\{-1, +1\}$ -valued random variable X_i by $X_i = 1$ if and only if the i th zero of x is flipped in u . We first argue that

$$\text{OneMax}(u) = n/2 + \varepsilon n/2 + \sum_{i \in [a]} X_i. \quad (1)$$

Suppose first x has exactly $n/2 + \varepsilon n/2$ many 1-bits, so the number of bits flipped equals the number of 0-bits. Any bit that was 0 in x increases the OneMax value by 1 when flipped, while *not* flipping such a bit means that instead some 1-bit is flipped, leading to a decrease in the OneMax value; this gives (1). If x has k more 1-bits than $n/2 + \varepsilon n/2$, then certainly k 1-bits have to flip, and for the 0-bits the same argument as in the previous

case holds, leading to (1). If x has k fewer 1-bits than $n/2 + \varepsilon n/2$, then certainly k 0-bits cannot flip; counting them with a value of -1 in (1) needs to be offset by adding k , leading again to (1).

By construction, $\Pr(X_i = 1) = (1 - \varepsilon)/2$ and $\Pr(X_i = -1) = (1 + \varepsilon)/2$. Consequently, $E(\text{OneMax}(u)) = n/2 + \varepsilon n/2 - \varepsilon a$, which is in $[n/2 - \varepsilon n/2, n/2 + \varepsilon n/2]$ for all a .

Let $X = \sum_{i \in [a]} X_i$. Note that the X_i are not independent. However, they are negatively correlated and for this reason still satisfy the usual Chernoff bounds. This was made precise by Doerr (2011, Theorems 1.16, 1.17) but only giving multiplicative Chernoff bounds (Doerr, 2011, Theorem 1.9). Since for our purposes an additive Chernoff bound (sometimes called Hoeffding bound) is more suitable, we look at the paper by Panconesi and Srinivasan (1997). There, Theorem 3.2 applied with correlations parameter $\lambda = 1$ and the \hat{X}_i simply being independent copies of the X_i together with Equation (2) give the first part of the following lemma. By setting the random variables $Y_i := 1 - X_i$, the second claim of the lemma follows from the first.

LEMMA 8: Let X_1, \dots, X_n be binary random variables. Let $X = \sum_{i \in [n]} X_i$.

- (1) Assume that for all $S \subseteq [n]$, we have $\Pr(\forall i \in S : X_i = 1) \leq \prod_{i \in S} \Pr(X_i = 1)$. Then $\Pr(X \geq E(X) + d) \leq \exp(-2d^2/n)$.
- (2) Assume that for all $S \subseteq [n]$, we have $\Pr(\forall i \in S : X_i = 0) \leq \prod_{i \in S} \Pr(X_i = 0)$. Then $\Pr(X \leq E(X) - d) \leq \exp(-2d^2/n)$.

Note that our $\{-1, +1\}$ -valued X_i are derived from a hypergeometric distribution (which leads to random variables fulfilling the assumptions of both parts of the lemma) via a simple affine transformation. Consequently, the following corollary directly implied by the lemma applies to these X_i .

COROLLARY 9: Let X_1, \dots, X_n be $\{-1, +1\}$ -valued random variables. Assume that for all $S \subseteq [n]$ and both $b \in \{-1, +1\}$, we have $\Pr(\forall i \in S : X_i = b) \leq \prod_{i \in S} \Pr(X_i = b)$. Let $X = \sum_{i \in [n]} X_i$. Then $\Pr(|X - E(X)| \geq d) \leq 2 \exp(-d^2/(2n))$.

From this, we observe that $\Pr(\text{OneMax}(u) \notin [n/2 - \varepsilon n, n/2 + \varepsilon n]) \leq \Pr(|\text{OneMax}(u) - E(\text{OneMax}(u))| > \varepsilon n/2) = \Pr(|X - E(X)| > \varepsilon n/2) \leq 2 \exp(-\varepsilon^2 n/8)$. In particular,

$$\Pr(f(u) = 0) \leq 2 \exp(-\varepsilon^2 n/8). \tag{2}$$

Independent copies of sets of negatively correlated random variables again are negatively correlated. Let Y be the sum of T independent copies of X . Then the corollary again yields

$$\Pr(Y \notin [-(a + 1/2)\varepsilon T, -(a - 1/2)\varepsilon T]) = \Pr(|Y - E(Y)| > \varepsilon T/2) \leq 2 \exp(-\varepsilon^2 T/(8a)).$$

Similarly, $\Pr(Y \notin [(3/2)E(X), E(Y)/2]) \leq 2 \exp(-|E(Y)|^2/(8aT)) = 2 \exp(-a\varepsilon^2 T/8)$. We summarize these findings in the following lemma.

LEMMA 10: Let $x \in \{0, 1\}^n$. Let u_1, \dots, u_T be obtained independently from x , each by flipping exactly $n/2 - \varepsilon n/2$ random bits. Let $s = \sum_{i \in [T]} \text{OneMax}(u_i)$ and $\hat{a} := -(s - T(n/2 - \varepsilon n/2))/(T\varepsilon)$. The probability that $\lfloor \hat{a} + 1/2 \rfloor$ does not equal $a := a(x)$ is at most $2 \exp(-\varepsilon^2 T/(8a))$. The probability that \hat{a} is not in $[a/2, (3/2)a]$ is at most $2 \exp(-\varepsilon^2 T a/8)$.

PROOF: By construction, $y = s - T(n/2 - \varepsilon n/2)$ has the same distribution as Y above. The probability that $y/(T\varepsilon)$ deviates from its expectation $-a$, or equivalently, that

Algorithm 4 `estimate`(x, T). Estimation of the Hamming distance of x to the optimum via roughly T samples to a jump function (see Corollary 11)

```

1  $s \leftarrow 0$ ;
2 for  $i = 1$  to  $T$  do
3    $u \leftarrow \text{flip}_{n/2 - \varepsilon n/2}(x)$ ;
4    $f_u \leftarrow f(u)$ ;
5   if  $f_u = 0$  then breakandgoto1;
6    $s \leftarrow s + (f_u - n/2 - \varepsilon n/2)$ ;
7 return  $\lfloor -s/(T\varepsilon) + 1/2 \rfloor$ ;
```

$-y/(T\varepsilon)$ deviates from its expectation a , by at least an additive term of $1/2$, is at most $2 \exp(-\varepsilon^2 T/(8a))$. \square

Building on the previous analysis, we now easily derive an estimator for a OneMax value not revealed by a jump function. It overcomes the possible problem of sampling an offspring with fitness zero by restarting the procedure using the command `breakandgoto1`.

We analyze the function `estimate` for the case that $q := T/(1 - 2T \exp(-\varepsilon^2 n/8))$ is positive, which is the only situation in which we use this function in the following.

COROLLARY 11: *The function `estimate` given in Algorithm 4 takes as inputs a search point x and an integer T ; if $q := T/(1 - 2T \exp(-\varepsilon^2 n/8))$ is positive, then the algorithm terminates using an expected number of at most q fitness evaluations, returning an integer \hat{a} .*

Assume that $\varepsilon n \geq 5\sqrt{n \log n}$ and $T \in O(n)$. Then the expected number of fitness evaluations is $q = T + O(1/n^2)$. Let $a = a(x) = H(x, \mathbf{1}_n)$ denote the unknown Hamming distance of x to the optimum. The probabilities for the events $\lfloor \hat{a} + 1/2 \rfloor \neq a$ and $\hat{a} \notin [a/2, (3/2)a]$ are at most $2 \exp(-\varepsilon^2 T/(8a)) + O(1/n^3)$ and $2 \exp(-\varepsilon^2 T a/8) + O(1/n^3)$, respectively. If $T \geq 24a \ln(6n/a)/\varepsilon^2$, these probabilities become $2/(6n/a)^3 + O(1/n^3)$ and $O(1/n^3)$.

PROOF: A run of `estimate` in which the `breakandgoto1` statement is not executed uses exactly T fitness evaluations. The probability that one execution of the for-loop leads to the execution of the `breakandgoto1` statement is at most $2T \exp(-\varepsilon^2 n/8)$ by a simple union bound argument and (2). If this number is less than 1, then an expected total number of $(1 - 2T \exp(-\varepsilon^2 n/8))^{-1}$ times the for-loop is started, given an expected total number of at most $T(1 - 2T \exp(-\varepsilon^2 n/8))^{-1}$ fitness evaluations.

When $\varepsilon \geq 5\sqrt{n \log n}$ and $T \in O(n)$, the probability for a restart is $2T \exp(-\varepsilon^2 n/8) \in O(1/n^3)$, the expected number of fitness evaluations becomes $T + O(1/n^2)$. Consequently, conditioning on none of the u_i in Lemma 10 having a OneMax value outside $[n/2 - \varepsilon n, n/2 + \varepsilon n]$ changes the probabilities computed there by at most an additive $O(1/n^3)$ term. \square

The main argument of how such an estimator for the number of 0s in a bit string can be used to derive a good black-box algorithm is reused in a later section in Theorem 20. Thus, we make the following definition.

DEFINITION 12: *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 that 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*

Algorithm 5 Unary optimization of a jump function f using a p -estimator g (see Definition 12)

```

1 repeat
2   |  $x \leftarrow \text{uniform}()$ ;
3 until  $f(x) \neq 0$ ;
4  $\alpha \leftarrow f(x)$ ;
5 while  $f(x) \neq n$  do
6   |  $x' \leftarrow \text{flip}_1(x)$ ;
7   | if  $f(x') = n$  then
8     |  $x \leftarrow x'$ ;
9   | else
10    |  $\alpha' \leftarrow g(x', \alpha)$ ;
11    |  $\alpha'' \leftarrow g(x, \alpha)$ ;
12    | if  $\alpha' > \alpha''$  then  $(x, \alpha) \leftarrow (x', \alpha')$ ;

```

g a p -estimator using f if, for all bit strings x , $a := n - \text{OneMax}(x)$, and for all $\alpha \in [a/2, 3a/2]$ we have

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

LEMMA 13: 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 $O(p(n)n^2)$.

PROOF: Let us consider a run of Algorithm 5.

We first assume that all calls of the p estimator g return a value that lies in $[1/2, 3/2]$ times the a -value of the first argument.

During a run of the algorithm, $a(x)$ performs a biased random walk on the state space $[0..n]$. The walk ends when the state 0 is reached. From the definition of a p -estimator we derive the following bounds on the transition probabilities. From state a , with probability at least $(a/n)(1 - 2(a/(16n))) \geq (a/n)(7/8)$, we move to state $a - 1$. This is the probability that a 1-bit flip reduces the Hamming distance to the optimum times a lower bound on the probability that we correctly identify both the resulting a -value and the a -value of our current solution x . With probability at most $((n - a)/n)(a/(16n) + (a + 1)/(16n))$, we move from state a to state $a + 1$. If we neither move to $a - 1$ or $a + 1$, we stay in a . Observe that when conditioning on not staying in a , then with probability at least $3/4$ we go to $a - 1$ and with probability at most $1/4$, we go to $a + 1$ (these are coarse estimates).

Our first goal is to show that the expected number of different visits to each particular state a is constant. To this aim, we may regard a speedy version of the random walk ignoring transitions from a to itself. In other words, we may condition on actually moving in each step to a different state. For any state $i \in [0..n]$, let e_i denote the expected number of visits to a starting the walk from i . To be more precise, we count the number of times we leave state a in the walk starting at i . We easily observe the following. For $i > a$, we have $e_i = e_a$, simply because we know that the walk at some time will reach a (because 0 is the only absorbing state). Hence we can split the walk started in i into

two parts, one from the start until the first visit to a (this contributes zero to e_i) and the other from the first visit to a until reaching the one absorbing state (this contributes e_a to e_i). For $i < a$, we have $e_i = (1 - q_i)e_a$, where q_i denotes the probability that a walk started in i visits 0 prior to a . This implies that $e_{a-1} \leq e_{a+1}$. Consequently, for the state a itself, we may use the pessimistic estimates on the transition probabilities and derive $e_a \leq 1 + (1/4)e_{a+1} + (3/4)e_{a-1} = 1 + (1/4 + (3/4)(1 - q_{a-1}))e_a$. To prove that $e_a \in O(1)$, it thus suffices to show that q_{a-1} is bounded from below by an absolute constant greater than zero. This follows easily from the gambler’s ruin theorem (see, e.g., Jansen, 2013, Theorem A.4). Consider a game in which the first player starts with $r = a - 1$ dollars, the second with $s = 1$ dollar. In each round of the game, the first player wins with probability $P = 1/4$, the second with probability $Q = 3/4$. The winner of the round receives one dollar from the other player. The game ends when one player has no money left. In this game, the probability that the second player runs out of money before the first, is exactly

$$\frac{(Q/P)^r - 1}{(Q/P)^{r+s} - 1},$$

which in our game is $(3^{a-1} - 1)/(3^a - 1)$. Using the pessimistic estimates of the transition probabilities, we hence see that $1 - q_{a-1} \leq (3^{a-1} - 1)/(3^a - 1) \leq 1/3$ as desired.

We have just shown that the expected number of times the algorithm has to leave a state “ $a(x) = a$ ” is constant. Since the probability of leaving this state in one iteration of the while-loop is at least $a/(2n)$, and by the definition of an estimator one iteration takes an expected number of $O(p(n)a \log(2 + n/a))$ fitness evaluations, we see that the expected total number of fitness evaluations spent in state “ $f(x) = a$ ” is $O(p(n)n \log(2 + n/a))$, with all constants hidden in the O -notation being absolute constants independent of a and n . Consequently, the expected total number of fitness evaluations in one run of the algorithm is at most $\sum_{a \in [n]} O(p(n)n \log(2 + n/a)) = O(p(n)n^2)$.

So far we assumed that all $g(y, T)$ calls return a value that is in $[a(y)/2, (3/2)a(y)]$, and conditional on this, proved an expected optimization time of $O(p(n)n^2)$. By the definition of an estimator, the probability that we receive a value outside this interval is $O(1/p(n)n^3)$. Consequently, the probability that this happens within the first $O(p(n)n^2)$ fitness evaluations is at most $O(1/n)$. A simple Markov bound shows that Algorithm 5 after $O(p(n)n^2)$ fitness evaluations (assuming the implicit constant high enough), with probability at least $1/2$, has found the optimum. Hence Remark 1 proves the claim. \square

THEOREM 14: *Let $\ell \leq (1/2 - \varepsilon)n$. The unary unbiased black-box complexity of Jump_ℓ is $O(n^2)$.*

PROOF: By Corollary 11, $g(x, \alpha) := \text{estimate}(x, \lceil 24(2\alpha + 1) \ln(6n/(2\alpha + 1))/\varepsilon^2 \rceil)$ is a p -estimator for $\text{Jump}_{n/2 - \varepsilon n}$ with $p(n)$ a sufficiently large constant. To see this, note that $a \mapsto a \ln(6n/a)$ is increasing for all $a \in [2n]$. Consequently, Lemma 13 shows the claim. \square

6 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 $n/2$ 1s. Surprisingly, despite some additional difficulties, we still find polynomial time black-box algorithms.

Throughout this section, let n be even (we comment on the case of odd n in Section 6.5). We call a jump function Jump_ℓ an *extreme jump function* if $\ell = n/2 - 1$.

Consequently, this function is zero except for the optimum (where it has the value n) and for bit strings having $n/2$ 1s (where it has the value $n/2$).

The information-theoretic argument of Droste et al. (2006) 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 15: *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 Droste et al. (2006) 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 (see end of Section 3). \square

6.1 Upper Bounds on Extreme Jump Functions

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

For an extreme jump function, we cannot distinguish between having OneMax values of $n/2 + k$ and $n/2 - k$ until we have encountered the optimum or its inverse. More precisely, let $x^{(1)}, x^{(2)}, \dots$ be a finite sequence of search points not containing the all-1s and all-0s strings. Define $y^{(i)} = x^{(i)} \oplus \mathbf{1}_n$ to be the inverse of $x^{(i)}$ for all i . 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 1s. However, if we know such a search point, we are done anyway.

Despite these difficulties, we develop a linear time ternary unbiased black-box algorithm. Then we show that restricting ourselves to binary variation operators at most increases the black-box complexity to $O(n \log n)$. For unary operators, the good news 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 $d(x) := |\text{OneMax}(x) - n/2|$ a *symmetricized version* of OneMax taking into account this difficulty. Also, let us define the *sign* $\text{sgn}(x)$ of x to be -1 if $\text{OneMax}(x) < n/2$, $\text{sgn}(x) := 0$, if $\text{OneMax}(x) = n/2$, and $\text{sgn}(x) = +1$ if $\text{OneMax}(x) > n/2$. In other words, $\text{sgn}(x)$ is the sign of $\text{OneMax}(x) - n/2$.

6.2 Ternary Unbiased Optimization of Extreme Jump Functions

When ternary operators are allowed, we easily obtain an unbiased black-box complexity of $O(n)$, which is best possible by Lemma 15. The reason for this fast optimization progress is that 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 1. If the new fitness is zero, then the new OneMax value is $n/2 + 2$, and the original bit was set to zero. We thus can learn all bit values and flip those bits that do not have the desired value.

One difficulty to overcome is that we 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 this strategy in a sufficiently symmetric way, we end

Algorithm 6 Ternary unbiased black-box algorithm for an extreme jump function f

```

1 repeat
2   |  $x \leftarrow \text{uniform}()$ ;
3 until  $f(x) \neq 0$ ;
4  $z^{(0)} \leftarrow x$ ;
5 for  $i = 1$  to  $n$  do
6   |  $z^{(i)} \leftarrow \text{flipWhereEqual}_1(z^{(i-1)}, x)$ ;
7   |  $y^{(i)} \leftarrow \text{selectBits}(x, z^{(i-1)}, z^{(i)})$ ;
8  $b \leftarrow y^{(1)}$ ;
9 for  $i = 2$  to  $n$  do
10  |  $a^{(i)} \leftarrow \text{selectBits}(x, y^{(1)}, y^{(i)})$ ;
11  | if  $f(a^{(i)}) = 0$  then
12  |   |  $b \leftarrow \text{selectBits}(b, x, y^{(i)})$ ;
13 Sample complement}(b);

```

up with a search point having OneMax value either n or 0 and in the latter case output its complement.

THEOREM 16: For $k \geq 3$, the k -ary unbiased black-box complexity of extreme jump functions is $\Theta(n)$.

PROOF: We show that Algorithm 6 optimizes any extreme jump function with $O(n)$ black-box queries using only operators of arity at most 3. This algorithm uses the three operators `uniform`, `selectBits` and `flipWhereEqual`, introduced in the proof of Theorem 6, as well as the following unbiased operator.

- `complement(x)`: Given a bit string x , `complement` flips all bits in x . This is a deterministic operator.

Note that a uniformly sampled bit string has exactly $n/2$ 1s with probability $\Theta(1/\sqrt{n})$. Consequently, the expected total number of queries is at most $4n + O(\sqrt{n}) = \Theta(n)$.

Let us now analyze the correctness of our algorithm. Let x be the initial bit string with fitness different from 0. This is either the optimal bit string, in which case nothing is left to be done, or a bit string with fitness $n/2$. In the latter case, consider the first for-loop. For each $i \leq n$, $z^{(i)}$ has a Hamming distance of i to x ; in fact, the sequence $(z^{(i)})_{0 \leq i \leq n}$ is a path in the hypercube flipping each bit exactly once. Thus, for each i , $y^{(i)}$ differs from x in exactly one position, and for each position there is exactly one i such that $y^{(i)}$ differs from x in that position. We can thus use the $y^{(i)}$ to address the individual bits, and we will call the bit where x and $y^{(i)}$ differ the i th bit.

We use $y^{(1)}$ now as a baseline and check which other bits in x contribute to the OneMax value of x in the same way (both are 0 or both are 1), as follows. The bit string $a^{(i)} = \text{selectBits}(x, y^{(1)}, y^{(i)})$ is obtained from x by flipping the first and i th bit. Thus, the fitness of $a^{(i)}$ is 0 if and only if the first and the i th bit contribute to the OneMax value of x in the same way; otherwise it is $n/2$.

Thus, b is the bit string with either all bits set to 0 or all bits set to 1. This means that we are either done after the last loop or after taking the complement of b . \square

6.3 Binary Unbiased Optimization of Extreme Jump Functions

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 implement the strategy used in the previous section, which relies on being able to copy particular bit values into the best-so-far solution.

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 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 $\text{OneMax}(x) > n/2$. 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 $\text{OneMax}(y) = n/2 - k$. 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 $\text{OneMax}(u) = n/2$, 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 $O(n \log n)$.

To make this idea precise, let us call a pair (x, y) of search points *opposing* if they have opposite signs, that is, if $\text{sgn}(x)\text{sgn}(y) = -1$. We call (x, y) an opposing k -pair for some integer k if x and y are opposing, $d(x) = d(y) = k$, and $H(x, y) = 2k$. Clearly, in this case, one of x and y has a OneMax value of $n/2 - k$, while the other has one of $n/2 + k$. To further ease the language, let us call bits having the value 1 *good*, those having the value 0 *bad*. Then the definition of an opposing k -pair implies that in one of x or y all the bit positions in which x and y differ are good, whereas in the other, they are all bad. The remaining positions contain the same number of good and bad bits.

There are some additional technicalities to overcome. For example, since we cannot decide the sign of a search point, it is nontrivial to generate a first opposing pair. Our solution is to generate different x and y in Hamming distance 1 from m and create offspring of x and y via a mixing crossover `mix` that inherits from x exactly one of the

Algorithm 7 subroutine `movefirstk`, $k \in [1, \dots, n/2 - 1]$, which takes $x, y \in \{0, 1\}^n$ as input and returns an $x' \in \{0, 1\}^n$. If (x, y) is an opposing k -pair, then the output x' with probability at least $1 - (2 \log n)/n^2$ satisfies $d(x') = k + 1$ and $H(x', y) = 2k + 1$.

```

1 status ← failure;
2 while status ≠ success do
3    $x' \leftarrow \text{flipWhereEqual}_1(x, y)$ ;
4   status ← success;
5    $i \leftarrow 0$ ;
6   while (status = success) and ( $i \leq 2 \log n$ ) do
7      $i \leftarrow i + 1$ ;
8      $u \leftarrow \text{flipWhereDifferent}_{k-1}(x', y)$ ;
9     if  $f(u) = n/2$  then status ← failure;
10 return  $x'$ ;

```

two bits in which x and y differ, and the other from y . If x and y are opposing, then this offspring with probability 1 has a fitness of $n/2$. Otherwise, it has a fitness of $n/2$ only with probability 1/2. Consequently, a polylogarithmic number of such tests with sufficiently high probability distinguishes the two cases.

Before giving the precise algorithms, let us define the operators used. We use the operator `flipWhereEqualk`, introduced in the proof of Theorem 6, as well as the following operators.

- `flipWhereDifferentk`(x, y). For two search points x and y and an integer k , the operator `flipWhereDifferentk` generates a search point by randomly flipping k bits in x among those bits where x and y disagree. If x and y disagree in fewer than k bits, a random bit string is returned.
- `mix`(x, y). If x and y disagree in exactly two bits, then a bit string is returned that inherits exactly one of these bits from x and one from y and that is equal to both x and y in all other bit positions. If x and y do not disagree in exactly two bits, a random bit string is returned.

We are now in a position to formally state the algorithms. We start with the key routine `movefirstk` (Algorithm 7), which, from an opposing k -pair (x, y) computes a Hamming neighbor x' of x with $d(x') = d(x) + 1$. Applying this function to both (x, y) and (y, x) , we obtain an opposing $(k + 1)$ -pair in the main algorithm (Algorithm 8).

LEMMA 17: *Let $k \in [1..n/2 - 1]$. The function `movefirstk` is binary unbiased. Assume that it is called with an opposing k -pair (x, y) . Let X be a geometrically distributed random variable with success probability 1/2 and denote by T the random variable counting the number of fitness evaluations in one run of `movefirstk`. Then the following holds.*

- (1) T is stochastically dominated by $(1 + 2 \log n)X$, also if we condition on the output satisfying $d(x') = k + 1$.
- (2) With probability at least $1 - (2 \log n)/n^2$, the output $x' \in \{0, 1\}^n$ satisfies $d(x') = k + 1$, $H(x, x') = 1$, and $H(x', y) = 2k + 1$.

PROOF: Since the operators `flipWhereEqual` and `flipWhereDifferent` are 2-ary and unbiased, `moveFirstk` is a 2-ary unbiased algorithm. Also note that any x' generated in line 3 necessarily has $H(x, x') = 1$ and $H(x', y) = 2k + 1$, the latter because $H(x, y) = 2k$ and x' is obtained from x by flipping a bit in which x and y agree. Hence the main challenge is to distinguish between the two cases that $d(x') = k + 1$ and $d(x') = k - 1$.

We first argue that the inner while-loop terminates surely with `status = success` if $d(x') = k + 1$, and with probability $1 - 1/n^2$ terminates with `status = failure` if $d(x') = k - 1$. Assume first that $d(x') = k + 1$. Since x' and y then are opposing and $d(y) = k$, the $2k + 1$ bits x' and y differ in are all good bits in x' (and thus bad bits in y) or vice versa. Consequently, flipping any $k - 1$ of them in x' surely reduces its d -value to 2, leading to $f(u) = 0$. Hence “`status = success`” is never changed to “`status = failure`.” Assume now that $d(x') = k - 1$. Then x' and y differ in $2k$ good bits and one bad bit, or in $2k$ bad bits and one good bit. Therefore, with probability $(k + 2)/(2k + 1) \geq 1/2$, all $k - 1$ bits flipped in the creation of u are of the same type. In this case, the `OneMax` values of u and x' differ by $k - 1$, implying $f(u) = n/2$. Thus, a single iteration of the while-loop sets the status variable to failure with probability at least $1/2$, as desired when $d(x') = k - 1$. The probability that this does not happen in one of the up to $2 \log n$ iterations is at most $1/n^2$.

We now analyze the `flipWhereEqual` statement in line 3. If (x, y) is an opposing k -pair, then exactly half of the bits in which x and y agree are good and the other half are bad. In either case, flipping one of the agreeing bits in x has a chance of exactly $1/2$ of increasing the d -value (and the same chance of $1/2$ of decreasing it). By what we proved about the inner while-loop, we see that the random variable describing the number of iterations of the outer while-loop is stochastically dominated by a geometric random variable with success probability $1/2$ (it would be equal to such a geometric random variable if the inner while-loop would not with small probability accept a failure as success). Since each execution of the inner while-loop leads to at most $2 \log n$ fitness evaluations, this proves the first part of (1). If we condition on the output satisfying $d(x') = k + 1$, then indeed the inner while-loop does not misclassify an x' . Consequently, the number of iterations in the outer while-loop has distribution X , and again T is dominated by $(1 + 2 \log n)X$.

For the failure probability estimate, we use the following blunt estimate. Since the `flipWhereEqual` statement with probability $1/2$ produces an x' that we view as success (and that will become the output of the function finally), with probability at least $1 - 1/n^2$ there will be at most $2 \log n$ iterations of the outer while-loop each generating a failure- x' . Each of them has a chance of at most $1/n^2$ of being misclassified as success. Consequently, the probability that `moveFirstk` returns a failure- x' is at most $1 - (2 \log n)/n^2$. □

THEOREM 18: *Algorithm 8 is a 2-ary unbiased black-box algorithm for extreme jump functions. It finds the optimum of an unknown extreme jump function with probability $1 - o(1)$ within $O(n \log n)$ fitness evaluations.*

PROOF: As in the previous section, line 1 of Algorithm 8 found a search point m having fitness $n/2$ after $O(n)$ fitness evaluations with probability $1 - \exp(-\Omega(\sqrt{n}))$.

Lines 2–11 are devoted to generating an opposing 1-pair (x, y) . Since m has exactly $n/2$ good and bad bits, x has a `OneMax` value of $n/2 - 1$ or $n/2 + 1$, each with probability

Algorithm 8 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

```

1 repeat
2   |  $m \leftarrow \text{uniform}()$ ;
3 until  $f(m) = n/2$ ;
4 status  $\leftarrow$  failure;
5 while status = failure do
6   |  $x \leftarrow \text{flipWhereEqual}_1(m, m)$ ;
7   |  $y \leftarrow \text{flipWhereEqual}_1(m, x)$ ;
8   | status  $\leftarrow$  success;
9   |  $i \leftarrow 0$ ;
10  | while (status  $\neq$  failure) and ( $i \leq \sqrt{n}$ ) do
11  |   |  $i \leftarrow i + 1$ ;
12  |   |  $u \leftarrow \text{mix}(x, y)$ ;
13  |   | if  $f(u) \neq n/2$  then status  $\leftarrow$  failure;
14 for  $k = 1$  to  $n/2 - 1$  do
15 |  $x' \leftarrow \text{movefirst}_k(x, y)$ ;
16 |  $y' \leftarrow \text{movefirst}_k(y, x)$ ;
17 |  $(x, y) \leftarrow (x', y')$ ;
18 if  $f(x) = n$  then
19 | return  $x$ ;
20 else
21 | return  $y$ ;
```

exactly $1/2$. Independent of this outcome, y in line 7 has a chance of $(n/2)/(n-1) > 1/2$ of having the opposite value, which means that (x, y) is an opposing 1-pair.

Observe that if x and y are opposing, then $\text{mix}(x, y)$ with probability 1 has a fitness of $n/2$, simply because both possible outcomes of $\text{mix}(x, y)$ have this fitness. If x and y are not opposing, then x and y have both one good bit more than m or both have one more bad bit. Consequently, the one outcome of $\text{mix}(x, y)$ that is different from m has a d -value of 2, visible from a fitness different from $n/2$. We conclude that the inner while-loop using at most \sqrt{n} fitness evaluations surely ends with **status** = success if (x, y) is an opposing 1-pair, and with probability $1 - 2^{-n}$ ends with **status** = failure if not. A coarse estimate thus shows that after $O(n)$ fitness evaluations spent in lines 2–11, which involves at least $\Omega(\sqrt{n})$ executions of the outer while-loop, with probability $1 - \exp(-\Omega(\sqrt{n}))$ we exit the outer while-loop with an opposing 1-pair (x, y) .

We now argue that if (x, y) is an opposing k -pair right before line 13 is executed, then with probability at least $1 - (4 \log n)/n^2$, the new (x, y) created in line 15 is an opposing $(k+1)$ -pair. This follows from twice applying Lemma 17 and noting that the condition $H(x, x') = 1$ in the statement of Lemma 17 implies that $H(x, y) = 2(k+1)$ after executing line 15. Consequently, a simple induction shows that with probability at least $1 - O(\log(n)/n)$, the pair (x, y) when leaving the for-loop is an opposing 1-pair, and thus exactly one of x and y is the optimum.

For the runtime statement, let us again assume that all opposing k -pairs are indeed created as discussed. We already argued that up to before line 12, with probability $1 - \exp(-\Omega(\sqrt{n}))$, we spent at most $O(n)$ fitness evaluations. We then perform $n - 2$ calls to movefirst_k functions, each leading to a number of fitness evaluations that are,

independent from what happened in the other calls, dominated by $(1 + 2 \log n)$ times a geometric random variable with success probability $1/2$. By Lemma 1.20 of Doerr (2011), we may assume in the following runtime estimate that the number of fitness evaluations in each such call is indeed $(1 + 2 \log n)$ times such an (independent) geometric random variable. By Theorem 1.14 of Doerr (2011), the probability that a sum of $\Theta(n)$ such geometric random variables deviates from its $\Theta(n)$ expectation by more than a factor of 2 is $\exp(-\Theta(n))$. Consequently, with high probability the total number of fitness evaluations is $O(n \log n)$. \square

Note that by using different constants in the `movefirstk` algorithm we can easily improve the success probability from the stated $1 - o(1)$ to any $1 - O(n^{-c})$, c a constant. We do not care that much about this because black-box complexity usually deals with expected number of fitness evaluations. To obtain now such a result, our claim in Theorem 18 is sufficient.

COROLLARY 19: *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.*

PROOF: By Theorem 18, a single run of this budgeted version of Algorithm 8 with probability $1 - o(1)$ finds the optimum. Thus, with Remark 1, we can conclude that we have an overall expected number of $O(n \log n)$ black-box queries. \square

6.4 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 nontrivial 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 $a := a(x) = \min\{|x|_1, |x|_0\}$ of x to optimum or its opposite. Here, and henceforth, let $|x|_0 := n - |x|_1$.

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 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 an expected total of $O(n^{9/2})$ fitness evaluations.

THEOREM 20: *Let n be even and $\ell = n/2 - 1$. Then the unary unbiased black-box complexity of Jump_ℓ is $O(n^{9/2})$.*

We start by analyzing the probability of receiving a bit string y with fitness $n/2$ when flipping a certain number λ of bits in a fixed bit string x . When λ has a parity different from the one of $\text{OneMax}(x)$, offspring y will never have a fitness of $n/2$. Consequently, we need to treat the cases of $\text{OneMax}(x)$ even or odd separately. Since n is even, the parities of $\text{OneMax}(x)$ and $a(x)$ are equal.

In the following, we frequently need the following estimate, due to Robbins (1955).

$$\forall n \geq 1 : \frac{4^n}{\sqrt{2\pi n}} \leq \binom{2n}{n} \leq \frac{4^n}{\sqrt{\pi n}}. \quad (3)$$

Let us first assume that $a := a(x)$ is *even*. Let y be obtained from x by flipping exactly $n/2$ random bits. Note that $\Pr(f(y) = n/2)$ depends on a , but not more specifically on x

or $\text{OneMax}(x)$. We may thus write $p_a := \Pr(f(y) = n/2)$. Note that $f(y) = n/2$ holds if and only if we flipped half the 1-bits and half the 0-bits in x . Hence,

$$p_a = \frac{\binom{n-a}{(n-a)/2} \binom{a}{a/2}}{\binom{n}{n/2}}, \tag{4}$$

which for $a \geq 1$ implies

$$p_a = \Theta\left(\frac{1}{\sqrt{a}}\right) \tag{5}$$

by Equation (3).

Let $a \geq 2$. We are interested in the difference $p_{a-2} - p_a$ in order to estimate the number of samples that are necessary to discriminate between the cases $a(x) = a$ and $a(x) = a - 2$. By Equation (4) we have

$$\begin{aligned} p_{a-2} - p_a &= \frac{\binom{n-a+2}{(n-a+2)/2} \binom{a-2}{(a-2)/2}}{\binom{n}{n/2}} - \frac{\binom{n-a}{(n-a)/2} \binom{a}{a/2}}{\binom{n}{n/2}} \\ &= \frac{\binom{n-a+2}{(n-a+2)/2} \binom{a}{a/2}}{\binom{n}{n/2}} \left[\frac{\binom{a-2}{(a-2)/2}}{\binom{a}{a/2}} - \frac{\binom{n-a-2}{(n-a-2)/2}}{\binom{n-a+2}{(n-a+2)/2}} \right] \\ &= \frac{\binom{n-a+2}{(n-a+2)/2} \binom{a}{a/2}}{\binom{n}{n/2}} \left[\frac{(a-2)!}{((a-2)/2)!^2} \frac{(a/2)!^2}{a!} - \frac{(n-a)!}{((n-a)/2)!^2} \frac{((n-a+2)/2)!^2}{(n-a+2)!} \right] \\ &= \frac{\binom{n-a+2}{(n-a+2)/2} \binom{a}{a/2}}{\binom{n}{n/2}} \left[\frac{(a/2)^2}{a(a-1)} - \frac{((n-a+2)/2)^2}{(n-a+2)(n-a+1)} \right] \\ &= \frac{\binom{n-a+2}{(n-a+2)/2} \binom{a}{a/2}}{\binom{n}{n/2}} \frac{1}{4} \left[\frac{a}{a-1} - \frac{n-a+2}{n-a+1} \right] \\ &= \frac{\binom{n-a+2}{(n-a+2)/2} \binom{a}{a/2}}{\binom{n}{n/2}} \frac{1}{4} \left[\frac{(n-a+1)a - (n-a+2)(a-1)}{(n-a+1)(a-1)} \right] \\ &= \frac{\binom{n-a+2}{(n-a+2)/2} \binom{a}{a/2}}{\binom{n}{n/2}} \frac{1}{4} \left[\frac{n-2a+2}{(n-a+1)(a-1)} \right]. \end{aligned} \tag{6}$$

LEMMA 21: For any constant k , there is a constant K such that the following holds. Let a be even. Let $y \in \{0, 1\}^n$ such that its distance from the optimum or its opposite $a(y) := \min\{|y|_1, |y|_0\}$ equals a . Consider the following random experiment. Let $N \geq N_a := Ka^{5/2}n^2/(n-2a)^{3/2}$. Exactly N times we independently sample an offspring of y by flipping exactly $n/2$ bits. Let Y denote the number of times we observed an offspring with fitness exactly $n/2$. Then, if $a \geq 2$,

$$\Pr(Y \geq N(p_a + p_{a-2})/2) \leq \exp(-k(n-2a)^{1/2}).$$

If $a \leq n/2 - 2$, then

$$\Pr(Y \leq N(p_a + p_{a+2})/2) \leq \exp(-k(n-2a)^{1/2}).$$

PROOF: To prove the first part, observe that by definition, Y is the sum of $N \geq N_a$ independent indicator random variables, each of which is 1 with probability p_a . Consequently, $Y \geq N(p_a + p_{a-2})/2$ is equivalent to

$$Y \geq E(Y) \left(1 + \frac{p_{a-2} - p_a}{2p_a}\right).$$

Writing $\delta = (p_{a-2} - p_a)/(2p_a)$, we compute from Equations (5) and (6) that $\delta \geq c(n - 2a)n^{-1}a^{-1}$ for some absolute constant c . By a standard Chernoff bound, $\Pr(Y \geq N(p_a + p_{a-2})/2) \leq \exp(-\delta^2 E(Y)/2) \in \exp(-\Theta(K)(n - 2a)^{1/2})$. By choosing K large enough, the first claim follows. The second claim can be proven with analogous arguments. \square

For $a := a(x)$ odd, things are not much different, so we only sketch some details. If $a(x)$ is odd, x has an odd number both of 1s and 0s, and these numbers are a and $n - a$ in some order. If $\lfloor |x|_1/2 \rfloor$ of the 1-bits and $\lfloor |x|_0/2 \rfloor$ of the 0-bits flip, we obtain a bit string with $|x|_1 - \lfloor |x|_1/2 \rfloor + \lfloor |x|_0/2 \rfloor = n/2$ 1s, that is, with fitness $n/2$. Also, there is no other way of obtaining a fitness of $n/2$ by flipping $n/2 - 1 = \lfloor |x|_0/2 \rfloor + \lfloor |x|_1/2 \rfloor$ bits. Consequently, the probability that flipping exactly $n/2 - 1$ bits in x gives a bit string y with fitness $n/2$ is, independent of the particular x ,

$$p_a := \frac{\binom{n-a}{(n-a-1)/2} \binom{a}{(a-1)/2}}{\binom{n}{n/2-1}} = \Theta\left(\frac{1}{\sqrt{a}}\right), \tag{7}$$

where the asymptotic statement follows, among others, from noting that $\binom{2k+1}{k} = (1/2)\binom{2(k+1)}{k+1}$ and $\binom{2k}{k-1} = \frac{k}{k+1}\binom{2k}{k}$ valid for all k and Equation (3).

Analogous to the case of even a , we compute for all odd $a \geq 3$ that

$$p_{a-2} - p_a = \frac{1}{4} \frac{\binom{n-a+2}{(n-a+1)/2} \binom{a}{(a-1)/2}}{\binom{n}{n/2-1}} \frac{n - 2a + 2}{a(n - a + 2)}.$$

From this, we derive the following analogue of Lemma 21.

LEMMA 22: *For any constant k , there is a constant K such that the following holds. Let a be odd. Let $y \in \{0, 1\}^n$ such that its distance from the optimum or its opposite $a(y) := \min\{|y|_1, |y|_0\}$ equals a . Consider the following random experiment. Let $N \geq N_a := K a^{5/2} n^2 / (n - 2a)^{3/2}$. Exactly N times we independently sample an offspring of y by flipping exactly $n/2 - 1$ bits. Let Y denote the number of times we observed an offspring with fitness exactly $n/2$. Then, if $a \geq 3$,*

$$\Pr(Y \geq N(p_a + p_{a-2})/2) \leq \exp(-k(n - 2a)^{1/2}).$$

If $a \leq n/2 - 2$, then

$$\Pr(Y \leq N(p_{a+2} + p_a)/2) \leq \exp(-k(n - 2a)^{1/2}).$$

These observations immediately yield an estimator for an invisible a -value of a bit string y . Since we only need it to distinguish between two possible values $a(y) = a - 1$ and $a(y) = a + 1$, we formulate the estimator and the following lemma tailored for this purpose. It is clear that stronger statements, relying less on a preknowledge of $a(y)$, could be derived as well.

LEMMA 23: *The procedure estimate, described in Algorithm 9, 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 arbitrarily close to 1.*

PROOF: The claim is trivial for $a = 1$ and $a = n/2 - 1$, hence let $a \in [2..n/2 - 2]$. Assume first that a is odd, consequently, $a(y)$ is even. If $a(y) = a + 1$, then by Lemma 21, with probability at least $1 - \exp(-\Theta((n - 2a)^{1/2}))$ we have $Y < N(p_{a+1} + p_{a-1})/2$, leading to the correct return value of $a + 1$. If $a(y) = a - 1$, the second part of the lemma yields that

Algorithm 9 The procedure $\text{estimate}(y, a)$, analyzed in Lemma 23

```

1 if  $a = n/2 - 1$  then
2   if  $f(y) = n/2$  then
3     return  $n/2$ ;
4   else
5     return  $n/2 - 2$ ;
6 if  $a = 1$  then
7   if  $f(y) = n$  or  $f(\text{flip}_n(y)) = n$  then
8     return  $n$ ;
9   else
10    return  $n - 2$ ;
11  $N \leftarrow \max\{N_{a-1}, N_{a+1}\}$ ;
12  $Y \leftarrow 0$ ;
13 for  $i = 1$  to  $N$  do
14   if  $a$  odd then
15      $z \leftarrow \text{flip}_{n/2}(x)$ ;
16   else
17      $z \leftarrow \text{flip}_{n/2-1}(x)$ ;
18   if  $f(z) = n/2$  then
19      $Y \leftarrow Y + 1$ ;
20 if  $Y < N(p_{a+1} + p_{a-1})/2$  then
21   return  $(a + 1)$ ;
22 else
23   return  $(a - 1)$ ;

```

we have $Y > N(p_{a+1} + p_{a-1})/2$ with probability at least $1 - \exp(-\Theta((n - 2a)^{1/2}))$, which again leads to the correct return value of $a - 1$. The case of even a is done analogously by invoking Lemma 22. \square

THEOREM 24: *The unary unbiased black-box complexity of extreme jump functions is $O(n^{9/2})$. This is witnessed by Algorithm 10, 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.

In the analysis of Algorithm 10, let us first assume that all $\text{estimate}(y, a)$ calls with $a(y) \in \{a - 1, a + 1\}$ return $a(y)$ correctly.

Assume that we start the while-loop in Algorithm 10 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.

Algorithm 10 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

```

1 repeat
2   |  $m \leftarrow \text{uniform}()$ ;
3 until  $f(m) = n/2$ ;
4  $x \leftarrow \text{flip}_1(m)$ ;
5 for  $a = n/2 - 1$  DownTo 1 do
6   | status  $\leftarrow$  failure;
7   | while status = failure do
8     |  $y \leftarrow \text{flip}_1(x)$ ;
9     | if estimate( $y, a$ ) =  $a - 1$  then
10    |   |  $x \leftarrow y$ ;
11    |   | status  $\leftarrow$  success;
12 Sample complement( $x$ );
    
```

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

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

This is true if we assume that none of the exceptional events (failures) of Lemma 23 occur. We now argue that in fact with constant probability, none of them occur. 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 23 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 $\sum_{a=1}^{n/2-1} (n/a) \exp(-k(n - 2a)^{1/2}) = \sum_{b=1}^{n/2-1} n(n/2 - b)^{-1} \exp(-k\sqrt{b}) = O(1)$, 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 no failure at all. \square

6.5 Extreme Jump Functions for Odd n

So far we restricted ourselves to extreme jump functions for even n . The reason is that we believe the case of odd n does not contain interesting new aspects but only adds notational difficulties. Since a reviewer explicitly asked for a discussion of the odd case, we briefly do this now.

It is not immediately clear what is a good definition for extreme jump functions when n is odd. Most likely, the most natural candidate is the function $f : \{0, 1\}^n \rightarrow \mathbb{R}$ defined by $f(x) = \text{OneMax}(x)$ when $\text{OneMax}(x) \in \{n, \lfloor n/2 \rfloor\}$ and $f(x) = 0$ for all other x .

However, the not-blanked-out fitness layer now is not invariant under inversion of the bit string (that is, the unbiased operator that flips all bits). So we do not have here the difficulty with the symmetry described in Section 6.1. Hence this definition of an extreme jump function does not serve our aim of investigating the most difficult jump setting.

Note that the alternative definition of taking $f : \{0, 1\}^n \rightarrow \mathbb{R}$ as $f(x) = \text{OneMax}(x)$ when $\text{OneMax}(x) \in \{n, \lceil n/2 \rceil, \lfloor n/2 \rfloor\}$ and $f(x) = 0$ for all other x does not reinstall the symmetry, simply because the fitness values break the symmetry.

These reasons limit our motivation to study extreme jump functions for odd n . At the same time, an inspection of the proofs of Section 6 reveals that all algorithms and arguments can easily be extended to cope with either definition of the extreme jump function for odd n .

7 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 us and where certain regimes change. This would also require studying lower bounds, a problem that is likely to require completely new analytical tools (recall that apart from information-theoretic lower bounds, even for simple functions such as OneMax the unbiased black-box complexities are not known for arities strictly greater than 1).

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 Jump_ℓ 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). Such precise analyses for runtimes of given algorithms have attracted interest; see Böttcher et al. (2010); Witt (2013); Sudholt (2013); Doerr and Doerr (2014); and Hwang et al. (2014).

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

Acknowledgments

Parts of this work were done during the Dagstuhl seminar 10361 “Theory of Evolutionary Algorithms.”

References

- Böttcher, S., Doerr, B., and Neumann, F. (2010). Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Parallel Problem Solving from Nature*, pp. 1–10. Lecture Notes in Computer Science, Vol. 6238.
- Buzdalov, M., Kever, M., and Doerr, B. (2015). Upper and lower bounds on unrestricted black-box complexity of $\text{jump}_{n,\ell}$. In *Evolutionary Computation in Combinatorial Optimization*, pp. 209–221. Lecture Notes in Computer Science, Vol. 9026.

- Doerr, B. (2011). Analyzing randomized search heuristics: Tools from probability theory. In A. Auger and B. Doerr (Eds.), *Theory of randomized search heuristics*, pp. 1–20. Singapore: World Scientific Publishing.
- Doerr, B., and Doerr, C. (2014). The impact of random initialization on the runtime of randomized search heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1375–1382.
- 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 the Genetic and Evolutionary Computation Conference (GECCO)*, 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., Johannsen, D., Kötzing, T., Lehre, P. K., Wagner, M., and Winzen, C. (2011a). Faster black-box algorithms through higher arity operators. In *Proceedings of the ACM Workshop on 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. (2011b). Too fast unbiased black-box algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 2043–2050.
- Doerr, B., and Winzen, C. (2012). Playing Mastermind with constant-size memory. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, pp. 441–452.
- Doerr, B., and Winzen, C. (2014a). Ranking-based black-box complexity. *Algorithmica*, 68:571–609.
- Doerr, B., and Winzen, C. (2014b). Reducing the arity in unbiased black-box complexity. *Theoretical Computer Science*, 545:108–121.
- 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*, 39:525–544.
- Erdős, P., and Rényi, A. (1963). On two problems of information theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézet Közleményei*, 8:229–243.
- Hwang, H., Panholzer, A., Rolin, N., Tsai, T., and Chen, W. (2014). Probabilistic analysis of the (1+1)-evolutionary algorithm. Retrieved from arXiv:1409.4955.
- Jansen, T. (2013). *Analyzing evolutionary algorithms: The computer science perspective*. Berlin: Springer.
- Jansen, T. (2015). On the black-box complexity of example functions: The real jump function. In *Proceedings of the ACM Workshop on 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*, 34:47–66.
- Lehre, P. K., and Witt, C. (2010). Black-box search by unbiased variation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 1441–1448.
- Lehre, P. K., and Witt, C. (2012). Black-box search by unbiased variation. *Algorithmica*, 64:623–642.

- Panconesi, A., and Srinivasan, A. (1997). Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. *SIAM Journal on Computing*, 26:350–368.
- Robbins, H. (1955). A remark on Stirling’s formula. *American Mathematical Monthly*, 62:26–29.
- Rowe, J., and Vose, M. (2011). Unbiased black box search algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 2035–2042.
- Sudholt, D. (2013). A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17:418–435.
- Teytaud, O., and Gelly, S. (2006). General lower bounds for evolutionary algorithms. In *Parallel Problem Solving from Nature*, pp. 21–31. Lecture Notes in Computer Science, Vol. 6238.
- Witt, C. (2013). Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability and Computing*, 22:294–318.