Limited Memory, Limited Arity Unbiased Black-Box Complexity: First Insights

Nina Bulanova
ITMO University
Saint Petersburg, Russia
ninasbulanova@gmail.com

Maxim Buzdalov
ITMO University
Saint Petersburg, Russia
mbuzdalov@gmail.com

ABSTRACT

Limited arity unbiased black-box complexity was proven to be a successful tool for understanding the working principles of randomized search heuristics and delivered insights to develop new efficient algorithms. While good upper bounds for simple problems were found long time ago, there are still no matching lower bounds.

On a road towards closing this gap, we introduce the notion of limited-memory, limited arity unbiased black-box complexity. We show that some efficient binary unbiased algorithms (almost) satisfy the memory-2 requirement, and present an algorithm to compute, for a given problem size, the exact lower bound on the runtime of any memory-m k-ary algorithm on any unimodal function.

CCS CONCEPTS

• **Theory of computation → Theory of randomized search heuristics**.

KEYWORDS

Unbiased complexity, memory-resticted, unimodal functions.

ACM Reference Format:
Nina Bulanova and Maxim Buzdalov. 2019. Limited Memory, Limited Arity Unbiased Black-Box Complexity: First Insights. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3319619.3326903

1 INTRODUCTION

In the current state of theory of randomized search heuristics there are two major building blocks that augment each other: runtime analysis and black-box complexity theory. The former evaluates the performance of particular randomized search heuristics on particular problems or problem classes, the latter strives to find how difficult it is to solve a problem (or any problem from the given class) by the best suitable randomized search heuristic (and why). The gaps between the complexities of various problems and the runtimes of existing algorithms on these problems are an important source of difficult questions and new inspiring results.

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326903

The notion of unbiased black-box complexity was introduced in [8] for pseudo-Boolean problems: as evolutionary algorithms and other randomized search heuristics are designed as general-purpose solvers, they shall not prefer one instance of a problem over another one. The definition of an unbiased black-box algorithm guarantees invariance under transformations preserving Hamming distances.

One of possible refinements of the unbiased black-box search model is the use of unbiased operators with restricted arity. The original paper [8] studied mostly *unary* unbiased black-box complexity, e.g. the class of algorithms allowing only unbiased operators taking one individual and producing another one, which can also be seen as *mutation-only* algorithms. This model appeared to be quite restrictive, e.g. the unary unbiased black-box complexity of OneMax was proven to be $\Theta(n \log n)$ [3, 8].

This inspired a number of works on higher-arity unbiased algorithms [2, 4, 7], since many crossovers are binary unbiased operators. In particular, in [7] it is proven that the k-ary unbiased black-box complexity of OneMax is $O(n/k)$ for $k = O(\log n)$. However, no matching lower bounds, other than the unbiased complexity $\Omega(n/\log n)$, are known. We aim at closing this gap by investigating the stricter *limited memory, limited arity* unbiased black-box complexity, for which we propose an algorithm to compute sharp lower bounds for all unimodal functions.

The rest of the paper is structured as follows. Section 2 introduces the necessary notation and definitions. Section 3 defines our new memory-m k-ary unbiased black-box complexity and shows that the existing algorithms that solve OneMax in linear time are (almost) memory-2 binary unbiased algorithms. In Section 4 we argue on switching to the BinVal function to continue the analysis. Section 5 describes our algorithm to compute memory-restricted fixed-arity unbiased complexities. Section 6 concludes the paper.

2 PRELIMINARIES

We denote as $[1..n]$ the set of integer numbers $\{1, 2, \ldots, n\}$. The class of functions OneMax is defined on bit strings of length n as

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

The class of functions called "binary value", or BinVal, was introduced in [6] and is defined on bit strings of length n as

$$\textsc{BinVal}_{z,\pi}(x) = \sum_{i=1}^{n} 2^{i-1} \cdot [z_i = x_{\pi(i)}],$$

where $z \in \{0;1\}^n$ is the hidden bit string representing the unknown optimum, $\pi : [1..n] \to [1..n]$ is a hidden permutation of indices from the range $[1..n]$ that defines which weights are given to which indices, and the *Iverson bracket* $[.]$ is the notation for a function that

converts the logic truth to 1 and the logic false to 0. This function has a single global maximum at $x = z$ with a value of $2^n - 1$.

A *k-ary variation operator* $\mathcal{X}$ produces a search point y from the given k search points $x_1, \ldots, x_k$ with probability $P_{\mathcal{X}}(y \mid x_1, \ldots, x_k)$. The operator $\mathcal{X}$ is *unbiased* [8] if the following relations hold for all search points $x_1, \ldots, x_k, y, z$ and all permutations π over $[1..n]$:

$$P_{\mathcal{X}}(y \mid x_1, \ldots, x_k) = P_{\mathcal{X}}(y \oplus z \mid x_1 \oplus z, \ldots, x_k \oplus z),$$
$$P_{\mathcal{X}}(y \mid x_1, \ldots, x_k) = P_{\mathcal{X}}(\pi(y) \mid \pi(x_1), \ldots, \pi(x_k)),$$

where $a \oplus b$ is the bitwise exclusive-or operation applied to two bit strings a and b of the same length, and $\pi(a)$ is an application of permutation π to a bit string a. We use pairs of terms *unary* and *1-ary*, as well as *binary* and *2-ary*, interchangeably throughout the paper and BBC abbreviation for black-box complexity.

3 MEMORY-m k-ARY UNBIASED BBC

In this section, we define the notion of the complexity which we are going to investigate in this paper. This is an intersection of the k-ary unbiased BBC [4, 8] and the limited memory BBC [5].

The motivation for this particular complexity is that the unbiased binary algorithm from [4] that solves OneMax in expected time $2n$ requires a population size of two, and the $(1 + (\lambda, \lambda))$ GA [2], as we are going to show later, can be easily transformed into such form. We will also show that this transformation leaves it unbiased and does not increase its expected running time on OneMax.

Definition 3.1. A black-box search algorithm is called a *memory-m k-ary unbiased algorithm* if it is limited to the following:

- before each iteration starts, the algorithm stores a population P consisting of at most m individuals, where the order of the element matters;
- based only on the fitness values of the individuals from P, the algorithm chooses a number $0 \le k' \le k$ and:
  - a sequence P' consisting of k' individuals chosen from P;
  - a k'-ary unbiased operator O;
- the algorithm applies the operator O to the chosen individuals P', produces a new individual g and evaluates it;
- based only on the fitness values of the individuals from $P \cup \{g\}$, the algorithm constructs a new population P_{new} of size at most m consisting of individuals from $P \cup \{g\}$;
- then the algorithm replaces P with P_{new} and continues.

The $(1 + 1)$ EA is a memory-1 unary unbiased algorithm, and the binary algorithm from [4] is a memory-2 binary unbiased algorithm.

Now we show that the $(1+(\lambda, \lambda))$ GA is almost a memory-2 binary unbiased algorithm as well. More precisely, we present an algorithm with a structure closely following the original $(1+(\lambda, \lambda))$ GA, where λ is chosen in a fitness-dependent way, which becomes memory-2 while remaining binary unbiased, and its expected running time on OneMax does not exceed the one of the original algorithm.

This algorithm is outlined as Algorithm 1. Note that an iteration of this algorithm is not the same as an iteration of the $(1+(\lambda, \lambda))$ GA, as Algorithm 1 is not allowed to store intermediate individuals. Instead, it finds out which phase (mutation or crossover) within an iteration of the original algorithm is to be simulated by checking whether the fitness values of the two individuals in the population are equal. When they are not equal, the algorithm maintains the invariant that the first individual corresponds to the parent individual, and the second individual corresponds to the best offspring with at least one new good bit found, so it is time to perform crossover. Otherwise, the algorithm is within the simulated mutation phase, so it mutates the parent and creates new offspring much like the original $(1 + (\lambda, \lambda))$ GA until an offspring with some new good bits, that are different from the ones in the parent, is found.

Algorithm 1 The memory-2 version of the $(1 + (\lambda, \lambda))$ GA

```
p0 ← UniformRandom({0,1}^n); evaluate p0
p1 ← p0
while true do                          ▷ The parent is p0 and f(p0) ≥ f(p1)
    λ ← sqrt(n/(n − f(p0)))
    if f(p0) = f(p1) then
        ℓ ~ B(n, λ/n)                                   ▷ Mutation phase
        p2 ← FlipBits(p0, ℓ); evaluate p2
        if f(p2) ≥ f(p0) then
            p0 ← p2, p1 ← p2                            ▷ Mutation too lucky
        else if f(p2) > f(p0) − ℓ then
            p1 ← p2                  ▷ New good bits → initiate crossover
        end if                       ▷ No new good bits → continue mutation
    else                                                ▷ Crossover phase
        p2 ← Crossover(p0, p1, 1/λ); evaluate p2
        if f(p2) > f(p0) then
            p0 ← p2, p1 ← p2                            ▷ Successful crossover
        end if                       ▷ Unsuccessful crossover, continue
    end if
end while
```

This algorithm is clearly unbiased, binary and memory-2. It differs from the $(1 + (\lambda, \lambda))$ GA in the following important points:

- Both the mutation phase and crossover phase employ synthesizing not exactly λ individuals, but as many individuals as it is necessary for a successful completion of the phase. The reason for this change is that a memory-m algorithm cannot store any counters in any way except for encoding them in the individuals.
- The condition to enter the mutation phase heavily depends on the properties of OneMax, so this modification can not be advised to serve as a replacement for the $(1 + (\lambda, \lambda))$ GA.
- The choice for λ is fitness-dependent, as the self-adjusting version would require to store the intermediate λ values somewhere, which is cumbersome for the above reasons.

The last two points, despite they turn the algorithm into a heavily specialized one, are still acceptable for the BBC research. The first change does not increase the running time on OneMax, as it just redistributes the risk for an iteration to have no improvement onto the individual sampling procedures.

4 FROM ONEMAX TO BINVAL

Unlike the unary unbiased BBC of OneMax, which was proven in [8] to be reachable by $(1 + 1)$-style algorithms so that the proof is able to use only the information about the best found individual, reasonably precise lower bounds on the unbiased BBC for higher arities seem to be impossible without considering also the inferior individuals, and their mutual relations. Indeed, the OneMax fitness does not reveal enough information to judge on the distribution

of offspring of two given individuals for the most of cases, even assuming a particular binary unbiased operator. More information can be achieved by taking into consideration the chain(s) of operators that connect these individuals, and the necessity of doing this complicates the proofs.

For this reason, it makes sense to analyse the BinVal function instead. The particular version of this function that we are going to use has been introduced and studied in [6], where the $\log_2 n + O(1)$ upper bound on its unbiased BBC was shown. For many complexity classes, the complexity of OneMax, as well as of any unimodal function, is bounded from below by the complexity of BinVal Indeed, the BinVal function serves as a bijection between the genotype and the fitness that preserves the Hamming distance. This means that any algorithm, which is capable of optimizing a unimodal function $\mathcal{F}$, can be applied to BinVal by applying $\mathcal{F}$ to the fitness returned by BinVal written in the binary notation, assuming the complexity class permits such operations.

On the other hand, the BinVal function ensures that all the information about the individuals, which is possible to access by unbiased means, is available to the algorithm through the fitness function, so our analysis does not have to cope with the problems imposed by a fitness function which can hide some information. In total, this motivates the research on the BBC of the BinVal function, including the one in this paper.

5 ALGORITHM FOR COMPUTING MEMORY-m k-ARY UNBIASED BBC OF BINVAL

In this section we describe the algorithm for computing, for any given n, the expected number of iterations required for an optimal memory-m k-ary unbiased algorithm to find the optimum of the BinVal function. Our procedure actually constructs such an algorithm, and its optimality is ensured by the construction procedure, so the resulting value is exact.

For the sake of brevity, we illustrate the concepts on the memory-2 binary example, giving some remarks for the general case. The algorithm can be easily generalized to arbitrary m and k, although its complexity quickly increases as these parameters grow.

First we note that, given a pair of individuals, a binary unbiased operator can make separate decisions for the coinciding bits and the differing bits. In each of the group, some of the bits are guessed right while others are guessed wrong. While choosing the operator (and the pair of individuals to work with, in general), the algorithm actually knows how the bits are distributed by looking at the BinVal fitness. As a result, we can describe the *state* of a memory-2 algorithm as a quadruple of non-negative integer numbers a_{00}, a_{01}, a_{10}, a_{11}, such that $a_{00} + a_{01} + a_{10} + a_{11} = n$, where:

- a_{00}: the number of bits guessed wrong in both strings;
- a_{11}: the number of bits guessed right in both strings;
- a_{01}: wrong in the first string, right in the second one;
- a_{10}: right in the first string, wrong in the second one.

Similarly, the state of a memory-m algorithm is represented by 2^m numbers with the similar semantics. There are $\Theta(n^3)$ states of a memory-2 algorithm and $\Theta(n^{2^m-1})$ states of a memory-m algorithm. Various symmetries can be used to reduce the number of the actually stored states, although the order will be the same. One can prove by induction that the decisions of an optimal unbiased memory-m algorithm for BinVal are a function of this state exclusively.

The decision of the algorithm, based on the current state, consists, as per Definition 3.1, of choosing the individuals, choosing the operator, and (based on the fitness of the operator's result) choosing the new sequence of m individuals which define the next state. There is a finite number of choices of the individuals and a finite number of choices for the new sequence. On the contrast, there are infinitely many unbiased binary operators. However, each unbiased operator amounts to choosing (in the binary case) the number of coinciding bits to flip, and the number of differing bits to flip. Due to the linearity of expectation the operator that would minimize the expected running time will choose these numbers deterministically, and there are finitely many such operators, as well as finitely many possible outcomes of each operator, where each outcome is associated with the probability to happen. This means that the total number of possible decisions in each state is finite, so all they can be iterated over and considered.

Our **proposed algorithm** associates with each state S the value $E(S)$, which will eventually store the expected number of steps needed to reach the optimum from S. It is conceptually similar to the Dijkstra algorithm [1, Section 24.3], where the equivalent of $E(S)$ would be the distance to the origin. Initially, all $E(S) \leftarrow \infty$ except for those states which correspond to having an optimum, for which $E(S) \leftarrow 0$. Unless all $E(S)$ are finite, we estimate for all states S where $E(S) = \infty$ their values $E'(S)$ by iterating over all decisions conjoined with the probabilities related to application of operators, and estimating their results by taking the current values of E into account. The state S_{best}, for which $E'(S_{\text{best}})$ is the smallest, undergoes promotion: $E(S_{\text{best}}) \leftarrow E'(S_{\text{best}})$, and the process repeats. The correctness of this algorithm follows from

Theorem 5.1. *Assume* $S^+ = \{s \mid E(s) < \infty\}, S^- = \{s \mid E(s) = \infty\}$, *as well as* $e = \max\{E(s) \mid s \in S^+\}$. *The following invariant holds:*

- $\min\{E'(s) \mid s \in S^-\} \geq e$;
- *if* $s_{best} = \arg\min\{E'(s) \mid s \in S^-\}$, *no* $s \in S^+$ *will need to change its decision after executing* $E(s_{best}) \leftarrow E'(s_{best})$.

Proof. We prove this theorem by induction.

Base: After the initialization, each state s_{opt} containing the optimum will have $E(s_{\text{opt}}) = 0$, and each other state s will have $E(s) = \infty$. As for each other state the expected number of steps is strictly positive, the statement holds.

Step: The first part of the statement is proven by contradiction. Assume $s_w = \arg\max\{E(s) \mid s \in S^+\}$ is the worst already estimated state, it holds that $\min\{E'(s) \mid s \in S^-\} < E(s_w)$ and s_{best} is the state where the minimum is reached. Note that the optimal decision for s_{best} shall not ever choose to transfer to the state s_w, as retaining in s_{best} instead will result in a better estimation. This means that the algorithm should have promoted s_{best} strictly before promoting s_w, which contradicts its definition.

The second part follows from noticing that, since the first part holds, any decision $s \in S^+$ that would transfer the algorithm to the state s_{best} can be replaced with the decision to retain in s, since $E(s) \geq E(s_{\text{best}}) = E'(s_{\text{best}})$. □

We ran our algorithm for the binary operators ($k = 2$) and for two memory sizes ($m = 2$ and $m = 3$) for as many problem sizes n

Table 1: Memory-2 binary unbiased BBC of BinVal

n	BBC	Step
2	2.0	+2.0
3	2.875	+0.875
4	3.625	+0.75
5	4.84375	+1.21875
6	6.078125	+1.234375
7	7.6171875	+1.5390625
8	8.889322916666668	+1.2721354166666696
9	10.636456564198369	+1.747133647531701
10	12.371170960313359	+1.73471439611499
11	14.038303461036381	+1.6671325007230227
12	15.699654859327913	+1.6613513982915311
13	17.447830553785792	+1.7481756944578795
14	19.0861176120001	+1.6382870582143063
15	20.921320866238506	+1.835203254238408
16	22.647312350057515	+1.7259914838190085
17	24.470455239318323	+1.823142889260808
18	26.20716584775122	+1.7367106084328974
19	28.027618816736567	+1.8204529689853466
20	29.813320310706573	+1.7857014939700058
21	31.639064455561932	+1.82574414485536
22	33.42679155241885	+1.7877270968569157
23	35.25538202472911	+1.8285904723102604
24	37.06383510084354	+1.8084530761144322
25	38.87284223330957	+1.8090071324660286
26	40.71184044975941	+1.8389982164498377
27	42.53445596805067	+1.822615518291265
28	44.35589812642954	+1.821442158378865
29	46.22928990971105	+1.8733917832815123
30	48.03055269421843	+1.8012627845073794
31	49.8944168488537	+1.863864154635273
32	51.708736279416016	+1.814319430562314
33	53.58955127372054	+1.8808149943045223
34	55.39662101796029	+1.8070697442397545
35	57.256590612226496	+1.8599695942662038
36	59.10667818052142	+1.8500875682949243
37	60.95153719120732	+1.8448590106859015
38	62.78320787455084	+1.8316706833435177
39	64.6580888992715	+1.8748810247206649
40	66.49815076211127	+1.840061862839761
41	68.36507159911925	+1.8669208370079815
42	70.20843935621512	+1.8433677570958764
43	72.08905164631821	+1.8806122901030875
44	73.93061687323592	+1.8415652269177087
45	75.798488053211	+1.867871179975083
46	77.65072866957095	+1.8522406163599499
47	79.52642764368056	+1.8756989741096106
48	81.37204113567641	+1.8456134919958487
49	83.23935173105438	+1.8673105953779725
50	85.10708177253406	+1.8677300414796747

as it was affordable on our hardware. The results are presented in Tables 1 and 2. In both tables, a clear linear trend is observed, and one can see that the memory-3 complexity is strictly smaller than

Table 2: Memory-3 binary unbiased BBC of BinVal

n	BBC	Step
2	2.0	+2.0
3	2.875	+0.875
4	3.625	+0.75
5	4.84375	+1.21875
6	6.041666666666667	+1.197916666666667
7	7.572368476823275	+1.5307018101566081
8	8.853198722782997	+1.2808302459597218
9	10.49695541739944	+1.6437566946164424
10	12.040784731611982	+1.5438293142125428
11	13.606199696329744	+1.5654149647177622
12	15.115284915797506	+1.509085219467762
13	16.72988334807649	+1.614598432278985
14	18.311115741599444	+1.5812323935229529
15	19.959933643823767	+1.648817902224323
16	21.572013544291508	+1.6120799004677409
17	23.237874414429374	+1.665860870137866
18	24.900849518865293	+1.662975104435919
19	26.59131342415072	+1.6904639052854264

memory-2 starting at $n = 6$. This means that at least the memory-2 unbiased BBC of BinVal is strictly greater than the memory-3 complexity, so it is strictly greater than the general unbiased BBC of the same problem.

6 CONCLUSION

Our investigation of the lower bounds on the binary unbiased BBC of unimodal functions were limited in this paper by memory-2 and memory-3 complexities, which we investigated for BinVal.

Our computations hint that the memory-2 binary unbiased BBC of unimodal functions is likely linear, and the leading constant is very close to the algorithm from [4] with the expected running time of $2n$, which shows that it is nearly optimal in its class. We also disproved the hypothesis that the binary unbiased BBC of the BinVal function coincides with its memory-2 flavour, as memory-3 complexity already differs.

This research was supported by the Russian Scientific Foundation, agreement No. 17-71-20178.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms, 2nd Ed.* MIT Press, Cambridge, Massachusetts.
[2] Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science* 567 (2015), 87–104.
[3] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. Optimal Parameter Choices via Precise Black-Box Analysis. In *Proceedings of Genetic and Evolutionary Computation Conference*. 1123–1130.
[4] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Per Kristian Lehre, Markus Wagner, and Carola Winzen. 2011. Faster black-box algorithms through higher arity operators. In *Proceedings of Foundations of Genetic Algorithms*. 163–172.
[5] Benjamin Doerr and Carola Winzen. 2014. Playing Mastermind with Constant-Size Memory. *Theory of Computing Systems* 55, 4 (2014), 658–684.
[6] Benjamin Doerr and Carola Winzen. 2014. Ranking-Based Black-Box Complexity. *Algorithmica* 68, 3 (2014), 571–609.
[7] Benjamin Doerr and Carola Winzen. 2014. Reducing the arity in unbiased black-box complexity. *Theoretical Computer Science* 545 (2014), 108–121.
[8] Per Kristian Lehre and Carsten Witt. 2012. Black-box Search by Unbiased Variation. *Algorithmica* 64 (2012), 623–642.