Benchmarking discrete optimization heuristics with IOHprofiler

Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, Ofer Shir, Thomas Back

Benchmarking Discrete Optimization Heuristics with IOHprofiler

Carola Doerr
Sorbonne Université, CNRS
Paris, France

Furong Ye
LIACS
Leiden, The Netherlands

Naama Horesh
Migal Institute
Upper Galilee, Israel

Hao Wang
LIACS
Leiden, The Netherlands

Ofer M. Shir
Tel-Hai College and Migal Institute
Upper Galilee, Israel

Thomas Bäck
LIACS
Leiden, The Netherlands

ABSTRACT

Automated benchmarking environments aim to support researchers in understanding how different algorithms perform on different types of optimization problems. Such comparisons carry the potential to provide insights into the strengths and weaknesses of different approaches, which can be leveraged into designing new algorithms. Carefully selected benchmark problems are also needed as training sets in the context of algorithm selection and configuration. With the ultimate goal to create a meaningful benchmark set for iterative optimization heuristics, we compile and assess in this work a selection of discrete optimization problems that subscribe to different types of fitness landscapes. All problems have been implemented and tested within IOHprofiler, our recently released software built to assess iterative heuristics solving combinatorial optimization problems. For each selected problem we compare performances of eleven different heuristics. Apart from fixed-target and fixed-budget results for the individual problems, we also derive ECDF results for groups of problems. To obtain these, we have implemented an add-on for IOHprofiler which allows aggregation of performance data across different benchmark functions.

CCS CONCEPTS

• **Human-centered computing** → **Scientific visualization**; • **Theory of computation** → **Theory of randomized search heuristics**; • **Software and its engineering** → *Software libraries and repositories*;

1 INTRODUCTION

Benchmarking optimization solvers aims at supporting practitioners in choosing the best algorithmic technique and its optimal configuration for the problem at hand. It is achieved through a systematic empirical assessment and comparison amongst competing techniques on a set of carefully selected optimization problems. At the same time, benchmarking may benefit theoreticians by enhancing mathematically-derived ideas into techniques being broadly applicable in practical optimization. Moreover, empirical performance comparisons constitute a catalyst in formulating new research questions. Finally, carefully chosen benchmark problems covering various of the multifaceted characteristics of real-world optimization challenges serve as training sets for the automation of algorithm configuration and selection.

The fact that there already exists a broad range of available benchmarking environments demonstrates that the performance assessment of optimization solvers over a set of representative test-problems serves several complementary purposes. The nature of the Application Programming Interface, or the identity of the benchmark problems, define together the implementation, and are usually rooted in the sought target(s). In the context of discrete optimization, several attempts to construct widely accepted benchmarking environments have been undertaken, but these (1) are typically restricted to certain problem classes (often classical $\mathcal{NP}$-hard problems such as SAT, TSP, etc.), (2) strongly focus on constructive heuristics, which are assumed to have access to the instance data (in contrast to black-box optimization heuristics, which implicitly learn about the problem instance only through the evaluation of potential solutions), or (3) aim to bundle efforts on solving specific real-world problem instances, without the attempt to generate a set of scalable or otherwise generalizable optimization problems. Benchmark competitions and crowd-sourcing platforms such as [41] fall into this latter category.

The few attempts to create a sound benchmarking platform for discrete black-box optimization heuristics, e.g., Weise's optimization benchmarking platform [42], have not yet received significant attention from the scientific community. In December 2018 Facebook announced its own benchmarking environment for black-box optimization [37]. While their focus is mostly in noisy continuous optimization, the platform also comprises a few discrete problems.

Interestingly, the situation significantly differs in continuous optimization, where the BBOB workshop series [28] constitutes a well-established and widely recognized platform for benchmarking derivative-free black-box optimization heuristics. Capitalizing on the COCO software [27] – which is the primary tool on which the BBOB workshops rely – we recently released a discrete optimization benchmarking environment, IOHprofiler [21] (see Section 2 for a brief discussion). Prior to this work, however, IOHprofiler only provided the experimental setup, which enables a detailed performance analysis. It did not fix any benchmark problems nor reference algorithms (yet proposing default test-functions). IOHprofiler was used in [23] for the comparison of different $(1 + \lambda)$ EA variants on two elementary problems, OneMax and LeadingOnes. We are not aware of extensions to more complex problems.

With this work, we contribute to the development of IOHprofiler by compiling and evaluating a set of 23 functions for a possible inclusion to a reference set of benchmark problems. We also contribute a set of eleven different heuristics that can serve as a first baseline for the performance evaluation of user-defined heuristics. All problems and algorithms have been implemented and integrated in the environment of IOHprofiler, so that they are easily accessible for future comparative studies. An important by-product of our contribution is the identification of additional statistics, which should be included within the IOHprofiler environment. In this respect we contribute a new module for IOHprofiler, which can aggregate performance data across different benchmark problems. More precisely, our extension allows to compute ECDF curves for sets of benchmark problems, thereby extending the IOHprofiler statistics for individual functions.

While the focus of this present work is on the assessment of possible benchmark problems (Section 3) and reference algorithms (Section 4), we also prescribe a suitable experimental setup and briefly discuss the obtained performance results (Section 5).

2 THE IOHPROFILER ENVIRONMENT

IOHprofiler is a novel environment for analyzing behavior of iterative optimization heuristics when run on selected problems, and for drawing comparisons of performances when multiple heuristics are run. Given algorithms and test problems implemented in **C** or in python, it outputs a statistical evaluation of the algorithms' performance, formed as the distribution of the fixed-target running time and the fixed-budget function values. In addition, IOHprofiler also permits tracking the evolution of algorithms' parameters, an attractive feature for analyses, comparisons, and the design of (self-)adaptive algorithms.

IOHprofiler consists of two components: **IOHexperimenter**, a module for processing the actual experiments and generating the performance data, and **IOHanalyzer**, a post-processing module for compiling detailed statistical evaluations.

IOHexperimenter is built on the COCO software [27], which has been adjusted to handle discrete optimization problems, and to facilitate a user-defined selection of benchmark problems (unlike COCO, whose 24 functions form a fixed selection and adding new functions requires low-level reprogramming of the tool).

IOHanalyzer, in contrast, has been independently developed from scratch. This module can be utilized as a stand-alone tool for the running-time analysis of any algorithm on arbitrary benchmark problems. It supports various input file formats, among others the IOHexperimenter's and the COCO platform's output formats. IOHanalyzer is designed for an interactive evaluation, enabling the user to define their required precision and ranges for the displayed data.

Source-codes of both modules of IOHprofiler are available at [22], while its documentation is provided in [21]. A web-based application of IOHanalyzer is available at http://iohprofiler.liacs.nl/.

As mentioned in the introduction, IOHprofiler in its current state constitutes a standardized experimental setup for discrete optimization benchmarking, but does not provide yet a selection of built-in benchmark problems, nor algorithms - a gap that we aim to reduce with this present work.

3 SUGGESTED BENCHMARK FUNCTIONS

We evaluate a selection of 23 functions and assess their suitability for inclusion in a standardized discrete optimization benchmarking platform. All problems have been implemented and integrated within the IOHprofiler framework, and are available at [22].

Following the suggestion in [38], we restrict our attention to *pseudo-Boolean functions*, i.e., all the suggested benchmark problems can be expressed as functions $f : \{0,1\}^n \to \mathbb{R}$.

Conventions. Throughout this work the variable n denotes the dimension of the problem that the algorithm operates upon. We assume that n is known to the algorithm; this is a natural assumption, since every algorithm needs to know the decision space that it is requested to search. Note though, that the *effective dimension* of a problem can be smaller than n, e.g., due to usage of "dummy variables" that do not contribute to the function values, or due to other reductions of the search space (see Section 3.7 for examples). In practice, we thus only require that n is an upper bound for the effective number of decision variables.

For *constrained problems*, such as the N-Queens problem, which we discuss in Section 3.11, we follow common practice in the evolutionary computation community and use penalty terms to discount infeasible solutions by the number and magnitude of constraint violations.

Notation. By $[k]$ we abbreviate the set $\{1, 2, \ldots, k\}$ and by $[0..k]$ the set $[k] \cup \{0\}$. All logarithms are to the base 10 and denoted by log. An exception is the natural logarithm, which we denote by ln. Finally, we denote by id the identity function, regardless of the domain.

3.1 Rationale Behind The Selection

We briefly discuss the ambition of our work, and the requirements that drove our selection and that should be imposed upon future selection of problems for benchmarking purposes.

Ambition. Our ultimate goal is to construct a benchmarking suite that covers wide ranges of the multifaceted problem characteristics found in real-world combinatorial optimization challenges.A second ambition, which is not necessarily perfectly identical to this goal, lies in building a suitable training set for automated algorithm design, configuration, and selection.

A core assumption of our work is that there is no room for a static, "ultimate" set of benchmark problems. We rather anticipate that a suitable training set should be extendable, to allow users to adjust the selection to their specific needs, but also to reflect advances of the field and to correct misconceptions. We particularly foresee a need for augmenting our set of functions, in order to cover landscape characteristics that are not currently present in the problems assessed in this present work. At the same time, we do not rule out the possibility of removing some of the functions considered below, for example if they do not contribute to our understanding how to distinguish amongst various heuristics, or if they can be replaced by other problems showing similar effects. Indeed, as we will argue in Section 5.4, some of the 23 assessed functions do not seem to contribute to a better discrimination between different algorithms, and are therefore evaluated as being obsolete. As we shall discuss in Section 6, we are confident that further advances in the research on exploratory landscape analysis [33] will help us to identify additional problems to include in the benchmark suite.

Problem Properties. We are mostly interested in problems that are arbitrarily scalable with respect to the dimension n. However, as the N-queens problem suggested below shows, we do not require that there exists a problem instance for *each and every n*, but we are willing to accept modest interpretations of scalability.

For the purposes of our work we demand that evaluating any search point is realizable in reasonable time. As a rule of thumb, we are mostly interested in experimental setups that allow one cycle of evaluating all problems within 24 hours per each algorithm. Put differently, we do *not* address with this work settings that feature *expensive* evaluations. We believe that those should be treated separately, as they typically require a different type of solvers.

3.2 Problems vs. Instances

While we are interested in covering different types of fitness landscapes, we care much less about their actual embedding, and mainly seek to understand algorithms that are invariant under the problem representation. In the context of pseudo-Boolean optimization $f : \{0,1\}^n \rightarrow \mathbb{R}$, a well-recognized approach to request representation invariance is to demand that an algorithm shows the same or similar performance on any instance mapping each bit string $x \in \{0,1\}^n$ to the function value $f(\sigma(x \oplus z))$, where z is an arbitrary bit string of length n, $\oplus$ denotes the bit-wise XOR function, and $\sigma(y)$ is to be read as the string $(y_{\sigma(1)}, \ldots, y_{\sigma(n)})$ in which the entries are swapped according to the permutation $\sigma : [n] \rightarrow [n]$. IOHprofiler supports such analysis by allowing to use these transformations (individually or jointly) with randomly chosen z and σ. Using these transformations, we obtain from one particular problem f a whole set of instances $\{f(\sigma(\cdot \oplus z)) \mid z \in \{0,1\}^n, \sigma \text{ permutation of } [n]\}$, which all have fitness landscapes that are pairwise isomorphic. The works [21, 32] provide further discussions of these *unbiasedness* transformations.

Apart from unbiasedness, we also focus in this work on *ranking-based heuristics,* i.e., algorithms which only make use of *relative*, and not of *absolute* function values. For a comparison with non-ranking-based algorithms, we test all algorithms on instances that are shifted by a multiplicative and an additive offset. That is, instead of receiving the values $f(\sigma(x \oplus z))$ only the transformed values $af(\sigma(x \oplus z)) + b$ are made available to the algorithms. We use here again the built-in functionalities of IOHprofiler to obtain these transformations.

In the following subsections we describe only the basic instance of each problem, which always forms instance 1 in IOHprofiler. We then test all algorithms on instances 1-6 and 51-55, which are obtained from this instance by the transformations described above. In theses instances the $\oplus$ and σ transformations are separated. Instances 2-6 are obtained from instance 1 by a '$\oplus z$' rotation with a randomly chosen $z \in \{0,1\}^n$, and random fitness offsets $a \in [1/5, 5]$, $b \in [-1000, 1000]$. For instances 51-55 there is no '$\oplus z$' rotation, but the strings are permuted by a randomly chosen σ and the ranges for the random fitness transformation are chosen as for instances 2-6. For each function and each dimension the values of z and σ are fixed per each instance, but different functions of the same dimensions may have different z and σ transformations.

3.3 Overview of Selected Benchmark Problems

The following list summarizes our 23 selected benchmark problems. For reasons of space, we will only provide short discussions in the subsequent sections.

- F1: OneMax, see Sec. 3.4
- F2: LeadingOnes, see Sec. 3.5
- F3: Harmonic, see Sec. 3.6
- F4: OneMax + $W([n/2], 1, 1, \text{id})$, see Sec. 3.7
- F5: OneMax + $W([0.9n], 1, 1, \text{id})$, see Sec. 3.7
- F6: OneMax + $W([n], \mu = 3, 1, \text{id})$, see Sec. 3.7
- F7: OneMax + $W([n], 1, \nu = 4, \text{id})$, see Sec. 3.7
- F8: OneMax + $W([n], 1, 1, r_1)$, see Sec. 3.7
- F9: OneMax + $W([n], 1, 1, r_2)$, see Sec. 3.7
- F10: OneMax + $W([n], 1, 1, r_3)$, see Sec. 3.7
- F11: LeadingOnes + $W([n/2], 1, 1, \text{id})$, see Sec. 3.7
- F12: LeadingOnes + $W(0.9n, 1, 1, \text{id})$, see Sec. 3.7
- F13: LeadingOnes + $W([n], \mu = 3, 1, \text{id})$, see Sec. 3.7
- F14: LeadingOnes + $W([n], 1, \nu = 4, \text{id})$, see Sec. 3.7
- F15: LeadingOnes + $W([n], 1, 1, r_1)$, see Sec. 3.7
- F16: LeadingOnes + $W([n], 1, 1, r_2)$, see Sec. 3.7
- F17: LeadingOnes + $W([n], 1, 1, r_3)$, see Sec. 3.7
- F18: LABS: Low Autocorrelation Binary Sequences, see Sec. 3.8
- F19: Ising-Ring, see Sec. 3.9
- F20: Ising-Torus, see Sec. 3.9
- F21: Ising-Triangular, see Sec. 3.9
- F22: MIVS: Maximum Independent Vertex Set, see Sec. 3.10
- F23: N-Queens, see Sec. 3.11

3.4 F1: OneMax (Hamming Distance)

The OneMax function is the best-studied benchmark problem in the context of evolutionary computation (EC), often coined the "drosophila of EC". It asks to optimize the function $\text{OM} : \{0,1\} \rightarrow [0..n], x \mapsto \sum_{i=1}^{n} x_i$. The problem has a very smooth and non-deceptive fitness landscape. Due to the well-known coupon collector effect, it is relatively easy to make progress when the function values are small, and the probability to obtain an improving move decreases considerably with increasing function value.

With the '$\oplus z$' transformations introduced in Sec. 3.2, the OneMax problem becomes the problem of minimizing the Hamming distance to an unknown target string $z \in \{0,1\}^n$.

OneMax is easily solved in n steps by a greedy hill climber that flips exactly one bit in each iteration, e.g., the one recursively going through the bit string from left to right until no local improvement can be obtained. This algorithm is included in our set of 11 reference algorithms as gHC, see Section 4. Randomized local search (RLS) and several classic evolutionary algorithms require $\Theta(n \log n)$ function evaluations on OneMax, due to the mentioned coupon collector effect [25]. The self-adjusting $(1 + (\lambda, \lambda))$ GA from [11] is the only EA known to optimize OneMax in $o(n \log n)$ time, it requires only a linear expected number of function evaluations to locate the optimum. The best expected running time that any iterative optimization algorithm can achieve is $\Omega(n/\log n)$ [24]. Unary unbiased algorithms cannot achieve average running times better than $\Omega(n \log n)$ [13, 32]. A more detailed survey of theoretical running-time results for OneMax can be found in [23], and a survey of lower bounds (in terms of black-box complexity results) can be found in [19].

3.5 F2: LeadingOnes

Among the non-separable functions, the LeadingOnes function is certainly the one receiving most attention in the theory of EC community. The LeadingOnes problem asks to maximize the function LO : $\{0,1\}^n \rightarrow [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\} = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$, which counts the number of initial ones.

Most EAs require quadratic running time to solve LeadingOnes, see again [23] or the full version of [10] for a summary of theoretical results. It is known that all elitist (1+1)-type algorithms [20] and all unary unbiased [32] are restricted to an $\Omega(n^2)$ expected running time. However, some problem-tailored algorithms optimizing LeadingOnes in sub-quadratic expected optimization time have been designed [1, 15, 18]. It is also known that the best-possible expected running time of any iterative optimization heuristic is $\Theta(n \log \log n)$ [1].

3.6 F3: Harmonic Weights Linear Function

Two extreme linear functions are OneMax with its constant weights and binary value $\text{BV}(x) = \sum_{i=1}^{n} 2^{n-i} x_i$ with its exponentially decreasing weights. An intermediate linear function is $f : \{0,1\}^n \rightarrow \mathbb{R}, x \mapsto \sum_i i x_i$ with harmonic weights, which was suggested to be considered in [38]. We add this linear function as F3.

3.7 F4-F17: The W-model

In [44] a collection of different ways to "perturb" existing benchmark problems in order to obtain new functions of scalable difficulties and landscape features has been suggested, the so-called W-model. These W-model transformations can be combined arbitrarily, resulting in a huge set of possible benchmark problems. In addition, these transformations can, in principle, be superposed to any base problem, giving yet another degree of freedom. Note here that the original work [44] as well as the existing empirical evaluations [43] only consider OneMax as underlying problem, but there is no reason to restrict the model to this function. We expect that in the longer term, the W-model, similarly to the well-known NK-landscapes [29] may constitute an important building block for a scalable set of discrete benchmark problems. More research, however, is needed to understand how the different combinations influence the behavior of state-of-the-art heuristic solvers. In this work, we therefore restrict our attention to instances in which the different components of the W-model are used in an isolated way. The assessment of combined transformations clearly forms a promising line for future work.

Basic Transformations. The W-model comprises 4 basic transformations, each coming with different instances. We use $W(\cdot,\cdot,\cdot,\cdot)$ to denote the configuration chosen in our benchmark set. For reasons of space we can only provide a very short summary of the W-model, focusing on the transformations assessed in our work. We note that, following the suggestion in [44], the transformations are executed in the same order as induced by the following description.

(1) **Reduction of dummy variables** $W(k,*,*,*)$**:** a reduction mapping each string $(x_1,\ldots,x_n)$ to a substring $(x_{i_1},\ldots,x_{i_k})$ for randomly chosen, pairwise different $i_1,\ldots,i_k \in [n]$.

(2) **Neutrality** $W(*,\mu,*,*)$**:** The bit string $(x_1,\ldots,x_n)$ is reduced to a string $(y_1,\ldots,y_m)$ with $m := n/\mu$, where μ is a parameter of the transformation. For each $i \in [m]$ the value of y_i is the majority of the bit values in a size-μ substring of x. More precisely, $y_i = 1$ if and only if there are at least $\mu/2$ ones in the substring $(x_{(i-1)\mu+1}, x_{(i-1)\mu+2},\ldots,x_{i\mu})$. When $n/\mu \notin \mathbb{N}$, the last bits of x are simply copied to y. In our assessment, we regard only the case $\mu = 3$.

(3) **Epistasis** $W(*,*,\nu,*)$**:** The idea of epistasis is to introduce local perturbations to the bit strings. To this end, a string $x = (x_1,\ldots,x_n)$ is divided into subsequent blocks of size ν. Using a permutation $e_\nu : \{0,1\}^\nu \rightarrow \{0,1\}^\nu$, each substring $(x_{(i-1)\nu+1},\ldots,x_{i\nu})$ is mapped to another string $(y_{(i-1)\nu+1},\ldots,y_{i\nu}) = e_\nu((x_{(i-1)\nu+1},\ldots,x_{i\nu}))$. The permutation e_ν is chosen in a way that Hamming-1 neighbors $u,v \in \{0,1\}^\nu$ are mapped to strings of Hamming distance at least $\nu - 1$. In our evaluation, we use $\nu = 4$ only, and the construction given in [44, Section 2.2].

(4) **Fitness perturbation** $W(*,*,*,r)$**.** With this transformation we can determine the *ruggedness* and *deceptiveness* of a function. Unlike the previous transformations, this perturbation operates on the function values, not on the bit strings. To this end, a *ruggedness* function $r : \{f(x) \mid x \in \{0,1\}^n\} =: V \rightarrow V$ is chosen. The new function value of a string x is then set to $r(f(x))$, so that effectively the problem to be solved by the algorithm becomes $r \circ f$. We use the following three ruggedness functions.
   - $r_1 : [0..s] \rightarrow [0..\lceil s/2 \rceil + 1]$ with $r_1(s) = \lceil s/2 \rceil + 1$ and $r_1(i) = \lfloor i/2 \rfloor + 1$ for $i < s$ and even s, and $r_1(i) = \lceil i/2 \rceil + 1$ for $i < s$ and odd s. This function maintains the order of the search points (i.e., for all x and y with $f(x) \geq f(y)$ it holds that $r_1(f(x)) \geq r_1(f(y))$), but introduces small fitness plateaus.
   - $r_2 : [0..s] \rightarrow [0..s]$ with $r_2(s) = s$, $r_2(i) = i + 1$ for $i \equiv s \pmod 2$ and $i < s$, and $r_2(i) = \max\{i-1, 0\}$ otherwise. This function introduces moderate ruggedness at each fitness level.
   - $r_3 : [0..s] \rightarrow [0..s]$ with $r_3(s) = s$ and $r_3(s - 5j + k) = s - 5j + (4 - k)$ for all $j \in [s/5]$ and $k \in [0..4]$ and $r_3(k) = s - (5\lfloor s/5 \rfloor - 1) - k$ for $k \in [0..s - 5\lfloor s/5 \rfloor - 1]$. With this function the problems become quite deceptive, since the distance between two local optima implies a difference of 5 in the function values.

We study superpositions of individual W-model transformations to the OneMax (F1) and the LeadingOnes (F2) problem, so as to study their effects on a well-understood separable and a well-understood non-separable problem.

3.8 F18: Low Autocorrelation Binary Sequences

The Low Autocorrelation Binary Sequences (LABS) problem poses a non-linear objective function over a binary sequence space, with the goal to maximize the reciprocal of the sequence's autocorrelation:

[LABS:] $\frac{n^2}{2E(S)}$ with $E(S) = \sum_{k=1}^{n-1} \left(\sum_{i=1}^{n-k} s_i \cdot s_{i+k} \right)^2$, where the sequence is of length n, $S := (s_1, \ldots, s_n)$ with $s_i = \pm 1$. To obtain a pseudo-Boolean problem, we use the straightforward interpretation $s_i = 2x_i - 1$ for all $i \in [n]$. The LABS problem has been studied over several decades (see, e.g., [34, 35]), but exact solutions are known only for problem dimensions $n \leq 66$ [35].

3.9 F19-F21: The Ising Model

The classical Ising model [5] considers a set of spins placed on a regular lattice $G = ([n], E)$, where each edge $(i, j) \in E$ is associated with an interaction strength J_{ij}. Given a configuration of n spins, $S := (s_1, \ldots, s_n)$, this problem poses a quadratic function, representing the system's energy and depending on its structure J_{ij}. Assuming zero external magnetic fields and using $s_i = 2x_i - 1$ we obtain the following pseudo-Boolean maximization problem:

[ISING:] $\sum_{\{i,j\} \in E} \left[x_i x_j - (1 - x_i)(1 - x_j) \right]$

In our benchmark set we consider three instances: the one-dimensional ring (F19), the two-dimensional torus (F20), and the two-dimensional triangular (F21).

3.10 F22: Maximum Independent Vertex Set

Given a graph $G = ([n], E)$, an independent vertex set is a subset of vertices where no two vertices are linked by an edge. A maximum independent vertex set (MIVS) is defined as an independent subset $V' \subset [n]$ having largest possible size.

[MIVS:] $\sum_i x_i$ s.t. $\sum_{i<j} x_i x_j e_{ij} = 0$.

Following [3], we use as F22 the instance of concatenated modules of X shape with horizontal edges between any two neighboring vertices on the top and the bottom, respectively. We use a straightforward penalty approach mentioned above to discount infeasible solutions.

3.11 F23: N-Queens Problem

The N-queens problem (NQP) [7] is defined as the task to place N queens on an $N \times N$ chessboard in such a way that they cannot attack each other. Using binary representation with $n := N^2$ variables x_{ij}, NQP subscribes to the following objective function:

[NQP:]

$$\sum_{i=1}^{N} \sum_{j=1}^{N} x_{ij} - N \cdot \left(\sum_{i=1}^{N} \max\left\{0, -1 + \sum_{j=1}^{N} x_{ij}\right\} + \sum_{j=1}^{N} \max\left\{0, -1 + \sum_{i=1}^{N} x_{ij}\right\} \right.$$

$$\left. + \sum_{k=-N+2}^{N-2} \max\left\{0, -1 + \sum_{\substack{j-i=k \\ i,j \in \{1,2,\ldots,N\}}} x_{ij}\right\} + \sum_{\ell=3}^{2N-1} \max\left\{0, -1 + \sum_{\substack{j+i=\ell \\ i,j \in \{1,2,\ldots,N\}}} x_{ij}\right\} \right)$$

4 ALGORITHMS

For our comparison, we have implemented the following 11 algorithms, which can serve as a baseline for future tests. For reasons of space, we can again only give the main references here.

We note that, except for the vGA, our implementations (deliberately) deviate slightly from the text-book descriptions referenced below. Following the suggestions made in [36] (and numerous other works), we enforce that offspring created by mutation are different from their parent (by resampling if needed), and, secondly, we do not evaluate recombination offspring that are identical to one of their immediate parents. With this convention, we omit the subscript $_{>0}$ previously used in [23, 36].

All algorithms start with uniformly chosen initial solution candidates.

With these conventions, our algorithms are as follows:

(1) **gHC:** A (1+1) greedy hill climber, which goes through the string from left to right, flipping exactly one bit per each iteration, and accepting the offspring if it is at least as good as its parent.

(2) **RLS:** Randomized Local Search, the elitist (1+1) strategy flipping one uniformly chosen bit in each iteration. I.e., RLS and gHC differ only in the choice of the bit which is flipped. While RLS is unbiased in the sense of Sec. 3.2, gHC is not permutation-invariant and thus biased.

(3) $(1+1)$ **EA:** The (1+1) EA with static mutation rate $p = 1/n$. This algorithm differs from RLS in that the number of uniformly chosen, pairwise different bits to be flipped is sampled from the conditional binomial distribution $\text{Bin}_{>0}(n, p)$.

(4) **fGA:** The "fast GA" proposed in [17] with $\beta = 1.5$. Its *mutation strength* (i.e., the number of bits flipped in each iteration) follows a power-law distribution with exponent β. This results in a more frequent use of large mutation-strength, while maintaining the property that small mutation strengths are still sampled with reasonably large probability.

(5) $(1+10)$ **EA:** The (1+10) EA with static $p = 1/n$, which differs from the (1+1) EA only in that 10 offspring are sampled (independently) per each iteration. Only the best one of these (ties broken at random) replaces the parent, and only if it is at least as good.

(6) $(1+10)$ $\textbf{EA}_{r/2,2r}$**:** The two-rate EA with self-adjusting mutation rates suggested and analyzed in [14].

(7) $(1+10)$ $\textbf{EA}_{\text{norm.}}$**:** a variant of the $(1+10)$ EA sampling the mutation strength from a normal distribution $N(pn, pn(1-p))$ with a self-adjusting choice of p [46].

(8) $(1+10)$ $\textbf{EA}_{\text{var.}}$**:** The $(1+10)$ $\text{EA}_{\text{norm.}}$ with an adaptive choice of the variance in the normal distribution from which the mutation strengths are sampled. Also from [46].

(9) $(1+10)$ $\textbf{EA}_{\text{log-n.}}$ The (1+10) EA with log-normal self-adaptation of the mutation rate proposed in [4].

(10) $(1+(\lambda,\lambda))$ **GA:** A binary (i.e., crossover-based) EA originally suggested in [12]. We use the variant with self-adjusting λ analyzed in [11].

(11) **vGA:** A (30, 30) "vanilla" GA (following the so-called traditional GA, as described, for example, in [2, 26]).

5 EXPERIMENTAL RESULTS

As a demonstration of the proposed benchmarking environment, we report here on basic experiments that were run on it. Notably, due to space limitations and following the primary scope of this work, the experimental report features only the major empirical findings.

5.1 Experimental Setup

We summarize our experimental setup:

- 23 test-functions

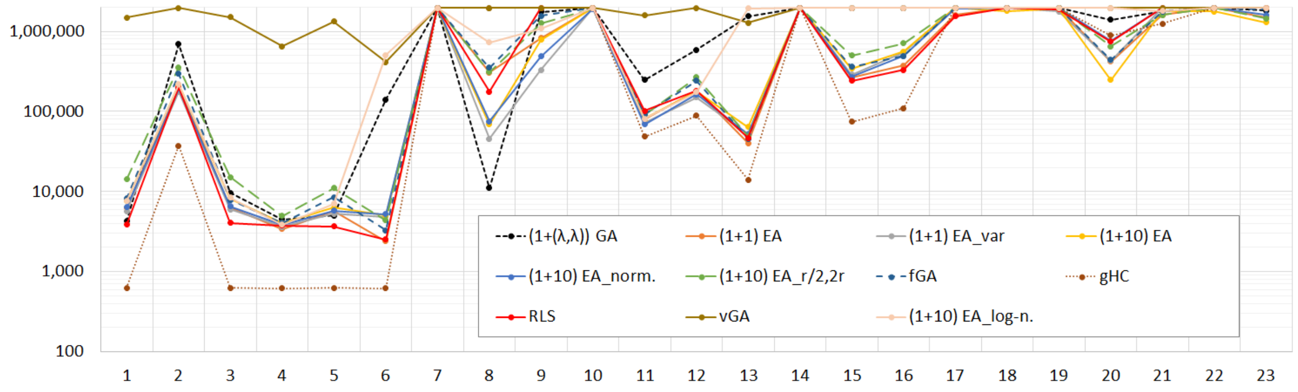

Figure 1: ERT values of all 11 algorithms for the 625-dimensional test suite, with respect to the best solution quality found by any of the algorithms (see Section 5.2 for the ERT value derivation).

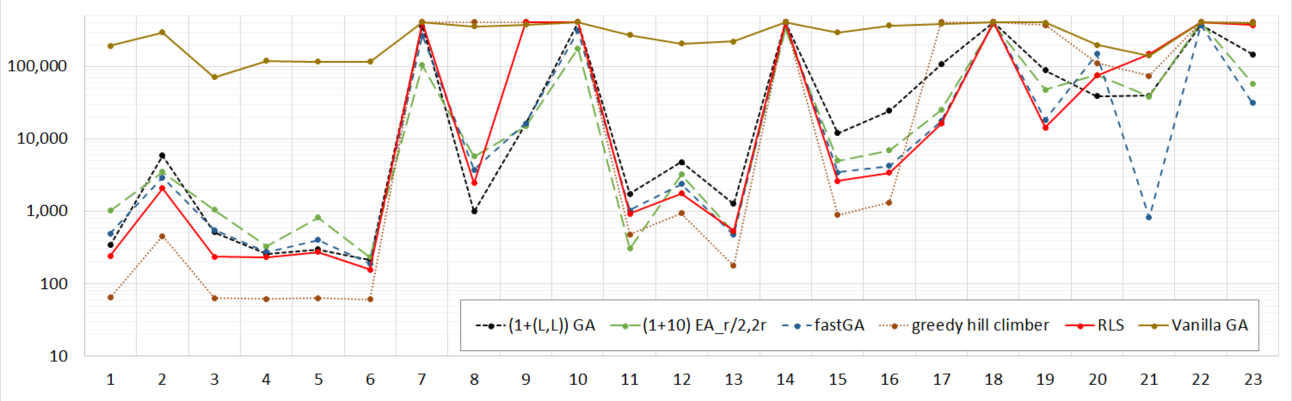

Figure 2: ERT values of selected algorithms for the 64-dimensional test suite, with respect to the best solution quality found by any of the algorithms (see Section 5.2 for the ERT value derivation).

- Each test-function is assessed over four problem dimensions, $n \in \{16, 64, 100, 625\}$, yielding altogether 92 (F, n) pairs.
- Each algorithm is run on 11 different instances (instances $1-6$ and $51-55$, see Sec. 3.2) of each of these 92 pairs, yielding 1,012 different runs per each algorithm.
- Each run is granted a budget of $100n^2$ function evaluations for dimensions $n \in \{16, 64, 100\}$ and a budget of $5n^2$ function evaluations for $n = 625$.

As mentioned, we are interested in experimental setups that allow to evaluate one entire experiment (for one algorithm) within 24 CPU hours. All 11 tested algorithms finished experimentation within this time frame.

We note that for each set of results (instance 1 only and instances 1-6+51-55) we collect very detailed performance data, so as to compare not only ERT and AHT values, but also the anytime behavior in terms of fixed-target runtimes, fixed-target results, and EDCF curves (see Section 5.2 for a recap of these performance measures). For the adaptive algorithms we also track the evolution of the most relevant algorithmic parameters, e.g., the value of λ in the $(1 + (\lambda, \lambda))$ GA, and the mutation rates for the (1+10) EA$_{r/2,2r}$, (1+10) EA$_{\text{var.}}$, (1+10) EA$_{\text{norm.}}$.

All data is available for interactive evaluation with IOHanalyzer at http://iohprofiler.liacs.nl/. The reader can upload her/his own runtime data, to compare performances against our 11 baseline algorithms. In the remainder of this section we show some selected statistics created with this tool. For reasons of space, we can only highlight a few of the findings, a detailed report will be made available on arXiv in due time.

5.2 Performance Measures

Given performance data of r independent runs of an algorithm A with a maximal budget of B function evaluations, the ERT value of A for a target value v is $(n - s)B + \text{AHT}$, where $s \leq r$ is the number of *successful* runs in which a solution of fitness at least v has been found, and AHT is the average first hitting time of these successful runs. We mostly concentrate on ERT values in our analysis. A definition of ECDF curves can be found, for example, in [21, Section 4.1].

5.3 ERT Analysis

We first regard the ERT values per each (function, dimension) pair, accounting for the best value found by any of the 11 algorithms in any of the 11 independent runs. Figure 1 shows these ERT values for all 11 algorithms for problem dimension $n = 625$. Performance data of selected algorithms for $n = 64$ is displayed in Figure 2.

Our analysis considers two complementary perspectives, aiming to recognize patterns and identify classes within (i) the set of all functions, and (ii) the set of all algorithms.

Functions' Empirical Grouping. It is evident that the low-dimensional F1-F6, F8, F11-F13 and F15-F16 are easily treated by the majority of the algorithms, with those functions based on LeadingOnes (i.e., F2, F11-13, F15, F16) and the OneMax with plateaus (F8) being more challenging within this group. On the other extreme, F7, F10, F14, F18 and F22 evidently constitute a class of hard problems, on which all algorithms consistently exhibit difficulties (except for $n = 16$); the LABS function (F18) seems the most difficult among them. F9, F17, and the instances of the Ising model (F19-F21), as well as the NQP (F23), constitute a class of moderate level of difficulty.

Algorithms' Observed Trends. The gHC and the vGA usually exhibited extreme performance with respect to the other algorithms. The vGA consistently suffers from poor performance over all functions, while the gHC either leads the performance on certain functions or undergoes deteriorated performance on other. The gHC's behavior is to be expected, since it is correlated with the existence of local traps (by construction) – for instance, it consistently excels on F1-F6, while having difficulties on F7-F10. Otherwise, we observe one primary class of algorithms exhibiting equivalent performance over all problems in all dimensions: The 7 algorithms $(1+(\lambda,\lambda))$-GA, (1+1)-EA, (1+10)-EA$_{\text{var.}}$, (1+10)-EA, (1+10)-EA$_{\text{norm.}}$, (1+10)-EA$_{r/2,2r}$, and (1+1)-fGA behave consistently, typically exhibiting fine performance.

Ranking. We also examined the overall number of runs per test-function in which an algorithm successfully located the best recorded value – the so-called *hitting number*. We then grouped those hitting numbers by dimension, and ranked the algorithms per each dimension. The (1+10)-EA$_{r/2,2r}$ consistently leads the grouped hitting numbers on the "low-dimensional" functions ($n \in \{16,\ 64,\ 100\}$), with (1+1)-fGA being the first runner-up (on $n = 64$ in close tie with (1+10)-EA$_{\text{norm.}}$). The (1+10)-EA also exhibits high ranking across all dimensions. On the other hand, the (1+1)-EA leads the grouped hitting numbers on the "high-dimensional" functions at $n = 625$, with (1+10)-EA being the runner-up. Across all dimensions, gHC, vGA and (1+10)-EA$_{\text{log-n.}}$ are with the lowest rankings.

Visual Analytics. As a demonstration of the proposed tool, we provide snapshots of visual analytics that supported our examination. Figure 3 depicts basic performance plots for F19 at dimension $n = 100$, in so-called *fixed-target* and *fixed-budget* perspectives. Furthermore, Figure 4 depicts an ECDF curve for the "easily-solved" functions in dimension $n = 625$: F1-F6, F11-F13, and F15-F16, with 10 equally spaced target values per each function.

5.4 Aftermath

The reported experiments reveal interesting findings on the test-functions and the algorithms' behavior when solving them – among which we highlight a few. By construction, the current test-suite proposed functions with various degrees of difficulty. Certain functions are inherently difficult (e.g., F18), and some synthetically (e.g., F7 and F14, regenerated by "epistasis"). Our empirical observations on such synthetically regenerated hard problems corroborated the effectiveness of the W-model in such a benchmarking environment. Moreover, on a different note, the hardness of all functions consistently increase with their problem dimension – exhibiting a desirable property that we mentioned in Sec. 3.1.

We also observe that functions F3, F4, F5, F11, F12 do not contribute much additional information to discriminate between the eleven reference algorithms, as the rankings of these algorithms is almost identical as for OneMax (F3-5) and LeadingOnes (F11, F12), respectively. In general, our analysis shows that further research is required to determine configurations of the W-model which maximize the information gain. The empirical results for F3 suggest that linear functions may not be suitable candidates for a benchmarking suite. While the performance analysis of EAs on these functions has inspired significant advances in our theoretical understanding of randomized search heuristics [16, 45], they may be less meaningful for a performance comparison targeting to understand performance across broad sets of combinatorial optimization problems. This example underlines the importance of the statement already made in Section 3.1: **the most suitable set of benchmark problems strongly depends on the question that one aims to answer**.

6 OUTLOOK

Among the many possible directions for future work, we consider the following ones particularly interesting.

Additional Performance Measures. While IOHprofiler already provides a very detailed assessment of algorithms' performance data, we suggest to extend its statistics by additional measures. We have implemented for this report and for future use in IOHprofiler the possibility to generate ECDF curves, for a user-defined set of functions and target values, thereby following the interactive performance evaluation paradigm which distinguished IOHprofiler from other existing benchmarking platforms (where the targets or budgets are typically set fixed). Going forward, we suggest to include modules that allow performance comparisons across different dimensions. Statistical tests such as the Wilcoxon-Mann-Whitney-type tests might be useful, but one should also evaluate the possibility to automate the generation of Bayesian statistics as suggested, for example, in [8]. Such an approach has recently been suggested for the EA context in [9].

Feature-Based Analyses. Another interesting extension of IOHanalyzer would be the design of modules that allow us to couple the performance evaluation with an analysis of the fitness landscape of the considered problems. Such *feature-based analyses* are at the heart of algorithm selection techniques [30], which use landscape features and performance data to build a model that predicts how the tested algorithms will perform on a previously not tested problem. Similar approaches can be found in per-instance-algorithm configuration (PIAC) approaches, which have recently shown very promising performance in the context of continuous black-box optimization [6]. A key step towards such a feature-based performance analysis are the selection and the efficient computation of meaningful features: while in continuous optimization a large set of features

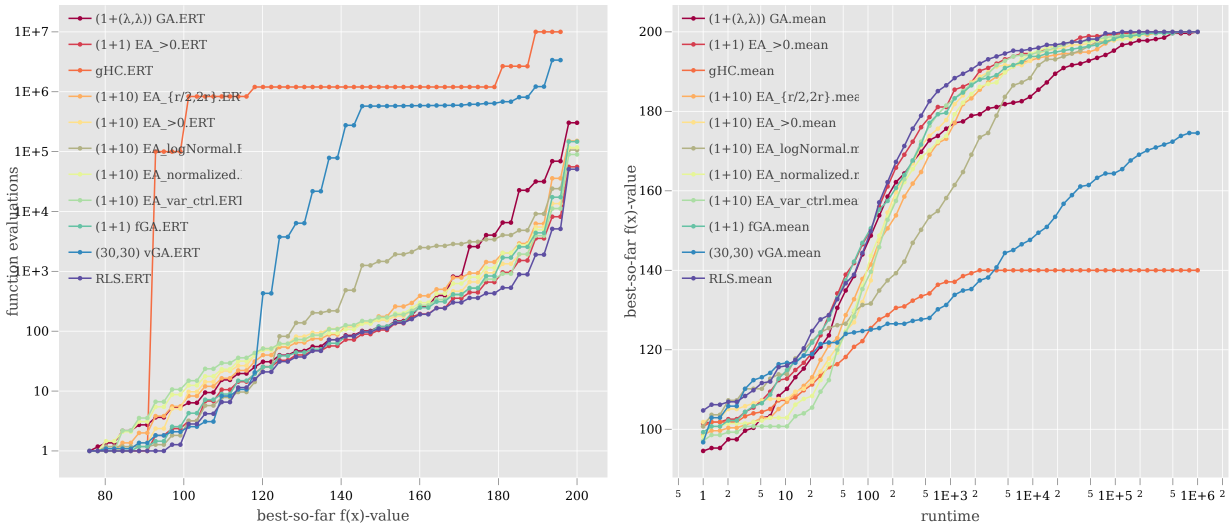

Figure 3: Demonstration of the basic performance plots for F19 at dimension $n = 100$: [LEFT] best obtained values as a function of evaluations calls (“fixed-target perspective”), versus [RIGHT] evaluations calls as a function of best obtained values (”fixed-budget perspective”). For F19, these patterns of relative behavior are observed across all dimensions.

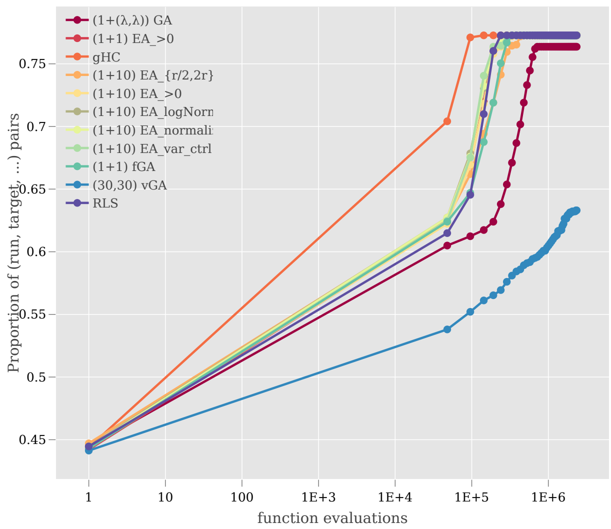

Figure 4: An ECDF curve for the class of “easily-solved” functions in dimension $n = 625$: F1-F6, F11-F13, and F15-F16.

has been defined and can be computed with the flacco package [31], the research community currently lacks a meaningful analog for discrete optimization problems. We note though, that several advances in this direction have been made, including the above-introduced features covered by the W-model (size of the effective dimension, neutrality, epistasis, ruggedness) and the local optima networks (see [39, 40] and references mentioned therein). We suggest to start a first prototype using these existing features, while at the same time intensifying research efforts to find additional landscape features that can be used to characterize pseudo-Boolean optimization problems.

Critical Assessment of Benchmark Problems. With such a feature-based approach, we also hope to more systematically identify problem characteristics that are not well represented in the selection of problems described above. We emphasize once again the fact that we see the here-suggested set of benchmark problems as a *first step* towards a sound benchmarking suite, not as a static “ultimate” selection. Quite the contrary, another important direction of our future research concerns the identification, critical selection, and implementation of additional benchmark problems. We believe that a good benchmark environment should offer the possibility to add new problems, so that users can focus their experimentation on problems relevant to their work. IOHprofiler is build with this functionality in mind, making it a particularly suitable testbed for our studies.

Combinations of W-model Transformations. As discussed in Section 3.7, the transformations of the W-model can be combined with each other. To analyze the individual effects of each transformation, and in order to keep the size of the experimental setup reasonable, we have not considered such combinations in this work. A critical consideration of adding such combinations, and of extending the base transformations (e.g., with respect to the fitness transformation, but also the size of the neutrality transformation, etc.) forms another direction that we will address in future work.

ACKNOWLEDGMENTS

We are very grateful to our colleagues Arina Buzdalova, Maxim Buzdalov, Michal Horovitz, Mordo Shalom, Dirk Sudholt, and Thomas Weise for valuable discussions on the topic of benchmarking IOHs.

This work was supported by the Chinese scholarship council (CSC No. 201706310143), by ANR-11-LABX-0056-LMH, by the Paris Ile-de-France Region, and by COST Action CA15140.

REFERENCES

[1] Peyman Afshani, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, and Kurt Mehlhorn. 2019. The query complexity of a permutation-based variant of Mastermind. *Discrete Applied Mathematics* (2019). https://doi.org/10.1016/j.dam.2019.01.007 In press.

[2] Thomas Bäck. 1996. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, NY, USA.

[3] Thomas Bäck and Sami Khuri. 1994. An evolutionary heuristic for the maximum independent set problem. In *Proc. 1st IEEE Conference on Evolutionary Computation*. IEEE, 531–535. https://doi.org/10.1109/ICEC.1994.350004

[4] Thomas Bäck and Martin Schütz. 1996. Intelligent Mutation Rate Control in Canonical Genetic Algorithms. In *International Symposium on Foundations of Intelligent Systems (ISMIS'96) (Lecture Notes in Computer Science)*, Vol. 1079. Springer, 158–167.

[5] F. Barahona. 1982. On the computational complexity of Ising spin glass models. *Journal of Physics A Mathematical General* 15 (Oct. 1982), 3241–3253. https://doi.org/10.1088/0305-4470/15/10/028

[6] Nacim Belkhir, Johann Dréo, Pierre Savéant, and Marc Schoenauer. 2017. Per instance algorithm configuration of CMA-ES with limited budget. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*. ACM, 681–688. https://doi.org/10.1145/3071178.3071343

[7] Jordan Bell and Brett Stevens. 2009. A Survey of Known Results and Research Areas for N-queens. *Discrete Math.* 309, 1 (Jan. 2009), 1–31. https://doi.org/10.1016/j.disc.2007.12.043

[8] Alessio Benavoli, Giorgio Corani, Janez Demsar, and Marco Zaffalon. 2017. Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis. *Journal of Machine Learning Research* 18 (2017), 77:1–77:36. http://jmlr.org/papers/v18/16-305.html

[9] Borja Calvo, Josu Ceberio, and José Antonio Lozano. 2018. Bayesian inference for algorithm ranking analysis. In *Proc. of Genetic and Evolutionary Computation Conference, companion material.* ACM, 324–325. https://doi.org/10.1145/3205651.3205658

[10] Benjamin Doerr. 2018. Better Runtime Guarantees via Stochastic Domination. In *Proc. of Evolutionary Computation in Combinatorial Optimization (EvoCOP'18) (Lecture Notes in Computer Science)*, Vol. 10782. Springer, 1–17. https://doi.org/10.1007/978-3-319-77449-7_1 Full version available at http://arxiv.org/abs/1801.04487.

[11] Benjamin Doerr and Carola Doerr. 2018. Optimal Static and Self-Adjusting Parameter Choices for the $(1+(\lambda,\lambda))$ Genetic Algorithm. *Algorithmica* 80 (2018), 1658–1709.

[12] Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science* 567 (2015), 87–104.

[13] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. Optimal Parameter Choices via Precise Black-Box Analysis. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'16)*. ACM, 1123–1130.

[14] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. 2017. The $(1+\lambda)$ Evolutionary Algorithm with Self-Adjusting Mutation Rate. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*. ACM, 1351–1358.

[15] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Per Kristian Lehre, Markus Wagner, and Carola Winzen. 2011. Faster black-box algorithms through higher arity operators. In *Proc. of Foundations of Genetic Algorithms (FOGA'11)*. ACM, 163–172.

[16] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. 2012. Multiplicative Drift Analysis. *Algorithmica* 64 (2012), 673–697.

[17] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. 2017. Fast genetic algorithms. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'17)*. ACM, 777–784. https://doi.org/10.1145/3071178.3071301

[18] Benjamin Doerr and Carola Winzen. 2012. Black-Box Complexity: Breaking the $O(n \log n)$ Barrier of LeadingOnes. In *Artificial Evolution (EA'11), Revised Selected Papers (Lecture Notes in Computer Science)*, Vol. 7401. Springer, 205–216.

[19] Carola Doerr. 2018. Complexity Theory for Discrete Black-Box Optimization Heuristics. *CoRR* abs/1801.02037 (2018). arXiv:1801.02037 http://arxiv.org/abs/1801.02037

[20] Carola Doerr and Johannes Lengler. 2018. The (1+1) Elitist Black-Box Complexity of LeadingOnes. *Algorithmica* 80 (2018), 1579–1603. https://doi.org/10.1007/s00453-017-0304-6

[21] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. 2018. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *arXiv e-prints:1810.05281* (Oct. 2018). arXiv:1810.05281 https://arxiv.org/abs/1810.05281

[22] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. 2019. Data Repository for IOHprofiler data sets. (2019). https://github.com/IOHprofiler

[23] Carola Doerr, Furong Ye, Sander van Rijn, Hao Wang, and Thomas Bäck. 2018. Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: profiling $(1+\lambda)$ EA variants on OneMax and LeadingOnes. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*. ACM, 951–958. https://doi.org/10.1145/3205455.3205621

[24] Paul Erdős and Alfréd Rényi. 1963. On Two problems of Information Theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézet Közleményei* 8 (1963), 229–243.

[25] Josselin Garnier, Leila Kallel, and Marc Schoenauer. 1999. Rigorous Hitting Times for Binary Mutations. *Evolutionary Computation* 7 (1999), 173–203.

[26] David Goldberg. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, Reading, MA.

[27] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. 2016. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *CoRR* abs/1603.08785 (2016). arXiv:1603.08785 http://arxiv.org/abs/1603.08785

[28] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. 2010. Comparing Results of 31 Algorithms from the Black-box Optimization Benchmarking BBOB-2009. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '10)*. ACM, New York, NY, USA, 1689–1696. https://doi.org/10.1145/1830761.1830790

[29] Stuart Kauffman and Simon Levin. 1987. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology* 128 (1987), 11–45.

[30] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2018. Automated Algorithm Selection: Survey and Perspectives. *CoRR* abs/1811.11597 (2018). arXiv:1811.11597 http://arxiv.org/abs/1811.11597

[31] Pascal Kerschke and Heike Trautmann. 2016. The R-Package FLACCO for exploratory landscape analysis with applications to multi-objective optimization problems. In *Proc. of Congress on Evolutionary Computation (CEC'16)*. IEEE, 5262–5269. https://doi.org/10.1109/CEC.2016.7748359

[32] Per Kristian Lehre and Carsten Witt. 2012. Black-Box Search by Unbiased Variation. *Algorithmica* 64 (2012), 623–642.

[33] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory Landscape Analysis. In *Proc. of Genetic and Evolutionary Computation (GECCO'11)*. ACM, 829–836. https://doi.org/10.1145/2001576.2001690

[34] Burkhard Militzer, Michele Zamparelli, and Dieter Beule. 1998. Evolutionary search for low autocorrelated binary sequences. *IEEE Transactions on Evolutionary Computation* 2, 1 (Apr 1998), 34–39. https://doi.org/10.1109/4235.728212

[35] Tom Packebusch and Stephan Mertens. 2016. Low autocorrelation binary sequences. *Journal of Physics A: Mathematical and Theoretical* 49, 16 (2016), 165001.

[36] Eduardo Carvalho Pinto and Carola Doerr. 2017. Discussion of a More Practice-Aware Runtime Analysis for Evolutionary Algorithms. In *Proc. of Artificial Evolution (EA'17)*. 298–305. https://ea2017.inria.fr//EA2017_Proceedings_web_ISBN_978-2-9539267-7-4.pdf

[37] J. Rapin and O. Teytaud. 2018. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad. (2018).

[38] Ofer M. Shir, Carola Doerr, and Thomas Bäck. 2018. Compiling a benchmarking test-suite for combinatorial black-box optimization: a position paper. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18), Companion.* ACM, 1753–1760. https://doi.org/10.1145/3205651.3208251

[39] Sarah L. Thomson, Sébastien Vérel, Gabriela Ochoa, Nadarajen Veerapen, and David Cairns. 2018. Multifractality and dimensional determinism in local optima networks. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18)*. ACM, 371–378. https://doi.org/10.1145/3205455.3205472

[40] Sarah L. Thomson, Sébastien Vérel, Gabriela Ochoa, Nadarajen Veerapen, and Paul McMenemy. 2018. On the Fractal Nature of Local Optima Networks. In *Proc. of Evolutionary Computation in Combinatorial Optimization (EvoCOP'18) (Lecture Notes in Computer Science)*, Vol. 10782. Springer, 18–33. https://doi.org/10.1007/978-3-319-77449-7_2

[41] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. 2016. Optil.io: Cloud Based Platform For Solving Optimization Problems Using Crowdsourcing Approach. In *Proc. of ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW'16), Companion Volume.* ACM, 433–436. https://doi.org/10.1145/2818052.2869098

[42] Thomas Weise. 2016. Optimization Benchmarking. (2016). http://optimizationbenchmarking.github.io/

[43] Thomas Weise. 2018. The W-Model , a tunable black-box discrete optimization benchmarking (bb-dob) problem, implemented for the bb-dob@gecco workshop. (2018).

[44] Thomas Weise and Zijun Wu. 2018. Difficult Features of Combinatorial Optimization Problems and the Tunable W-Model Benchmark Problem for Simulating them. In *Proc. of Genetic and Evolutionary Computation Conference (GECCO'18), Companion Material)*. ACM, 1769–1776. https://doi.org/10.1145/3205651.3208240

[45] Carsten Witt. 2013. Tight Bounds on the Optimization Time of a Randomized Search Heuristic on Linear Functions. *Combinatorics, Probability & Computing* 22 (2013), 294–318.

[46] Furong Ye, Carola Doerr, and Thomas Bäck. 2019. Interpolating Local and Global Search by Controlling the Variance of Standard Bit Mutation. *coRR* abs/1901.05573 (2019). http://arxiv.org/abs/1901.05573