BENCHMARKING DISCRETE OPTIMIZATION HEURISTICS WITH IOHPROFILER

Carola Doerr
Sorbonne Université, CNRS, LIP6
Paris, France
Carola.Doerr@mpi-inf.mpg.de

Furong Ye
Leiden University
Leiden, The Netherlands
f.ye@liacs.leidenuniv.nl

Naama Horesh
Migal - The Galilee Research Institute
Upper Galilee, Israel
naamah@migal.org.il

Hao Wang
Leiden University
Leiden, The Netherlands
h.wang@liacs.leidenuniv.nl

Ofer M. Shir
Computer Science Department, Tel-Hai College
The Galilee Research Institute
Upper Galilee, Israel
ofersh@telhai.ac.il

Thomas Bäck
Leiden University
Leiden, The Netherlands
t.h.w.baeck@liacs.leidenuniv.nl

December 20, 2019

ABSTRACT

Automated benchmarking environments aim to support researchers in understanding how different algorithms perform on different types of optimization problems. Such comparisons provide insights into the strengths and weaknesses of different approaches, which can be leveraged into designing new algorithms and into the automation of algorithm selection and configuration. With the ultimate goal to create a meaningful benchmark set for iterative optimization heuristics, we have recently released IOHprofiler, a software built to create detailed performance comparisons between iterative optimization heuristics.

With this present work we demonstrate that IOHprofiler provides a suitable environment for automated benchmarking. We compile and assess a selection of 23 discrete optimization problems that subscribe to different types of fitness landscapes. For each selected problem we compare performances of twelve different heuristics, which are as of now available as baseline algorithms in IOHprofiler.

We also provide a new module for IOHprofiler which extents the fixed-target and fixed-budget results for the individual problems by ECDF results, which allows one to derive aggregated performance statistics for groups of problems.

Keywords combinatorial optimization · black-box optimization · randomized search heuristics · benchmarking · evolutionary computation

1 Introduction

Benchmarking optimization solvers aims at supporting practitioners in choosing the best algorithmic technique and its optimal configuration for a given problem. It is achieved through a systematic empirical assessment and comparison amongst competing techniques on a set of carefully selected optimization problems. Benchmarking may also benefit theoreticians by enhancing mathematically-derived ideas into techniques being broadly applicable in practical optimization. It also constitutes a catalyst in formulating new research questions. More re-

cently, carefully chosen benchmark problems covering various of the multifaceted characteristics of real-world optimization challenges are also needed as training sets for the automation of algorithm configuration and selection [Kerschke et al.(2018)Kerschke, Hoos, Neumann, and Trautmann].

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 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 [Wasik et al.(2016)Wasik, Antczak, Badura, Laskowski, and Sternal] 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 [Weise(2016)], have not yet received significant attention from the scientific community. In December 2018, Facebook announced its own benchmarking environment for black-box optimization [Rapin and Teytaud(2018)]. While their focus is mostly on noisy continuous optimization, the platform also comprises a few discrete problems.

Interestingly, the situation significantly differs in continuous optimization, where the BBOB workshop series [Hansen et al.(2010)Hansen, Auger, Ros, Finck, and Pošík] and its software framework COCO [Hansen et al.(2016)Hansen, Auger, Mersmann, Tušar, and Brockhoff] constitutes a well-established and widely recognized platform for benchmarking derivative-free black-box optimization heuristics. The COCO framework is under constant development. Apart from a noisy test bed, it has in recent years been extended multi-objective [Tusar et al.(2016)Tusar, Brockhoff, Hansen, and Auger] and mixed-integer [Tusar et al.(2019)Tusar, Brockhoff, and Hansen] problems. While COCO has been designed to analyze iterative optimization heuristics (IOHs) on different types of problems, its designers have chosen to pre-select these problems that the user can test his/her algorithms on. Benchmarking new problems with COCO requires substantial knowledge of its software design, and is therefore quite time-consuming.

For the design of IOHPROFILER [Doerr et al.(2018)Doerr, Wang, Ye, van Rijn, and Bäck], we have chosen a different way. Our goal is to make the software as flexible as possible, so that the user can easily test his/her algorithms on the problems and with respect to performance criteria of his/her choice, see Section 2 for a brief discussion. The original framework, however, only provided the experimental setup, but did not fix any benchmark problems nor reference algorithms.

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 twelve 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. All performance data is available in our data repository, and can be straightforwardly assessed through the web-based version of IOHANALYZER at http://iohprofiler.liacs.nl/. 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, which complements the previously available statistics in IOHPROFILER for the assessment of individual benchmark problems.

This report is an extension of [Doerr et al.(2019a)Doerr, Ye, Horesh, Wang, Shir, and Bäck], which has been presented at the 2019 ACM GECCO workshop on Black Box Discrete Optimization Benchmarking (BB-DOB).

2 The IOHprofiler Environment

IOHPROFILER is a new benchmarking environment for detailed, highly modular performance-analysis of iterative optimization heuristics. Given algorithms and test problems implemented in C++, Python, or R, IOHPROFILER outputs a statistical evaluation of the algorithms' performances in the form of the distribution of the fixed-target running time and the fixed-budget function values. In addition, IOHPROFILER also permits tracking the evolution of the algorithms' parameters, which is an attractive feature for the analysis, comparison, and design of (self-)adaptive algorithms. The user selects

which information (on top of performance data) is tracked per each algorithm and controls the granularity of the generated records. A documentation of IOHPROFILER is available at [Doerr et al.(2018)Doerr, Wang, Ye, van Rijn, and Bäck] and at https://iohprofiler.github.io/IOHanalyzer(Post-Processing)/GraphicUserInterface/.[1]

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. Data repositories for algorithms, benchmark problems, and performance data are currently under construction. For the time being, the algorithms used for this work as well as the here-described benchmark problems are available in the GitHub repository of IOHPROFILER, which is available at [Doerr et al.(2019b)Doerr, Wang, Ye, van Rijn, and Bäck]. Performance data can be assessed through a web-based interface at http://iohprofiler.liacs.nl/.

IOHEXPERIMENTER is designed to handle discrete optimization problems and to facilitate a user-defined selection of benchmark problems. The first version of IOHEXPERIMENTER was built upon the COCO framework [Hansen et al.(2016)Hansen, Auger, Mersmann, Tušar, and Brockhoff], but the software has now been restructured to allow for a more flexible selection of benchmark problems – adding new functions in COCO requires low-level reprogramming of the tool, whereas functions can be added more easily in IOHPROFILER.

IOHANALYZER 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 output formats of IOHEXPERIMENTER and of COCO. Extensions to other data formats are under investigation. IOHANALYZER is designed for a highly interactive evaluation, enabling the user to define their required precision and ranges for the displayed data. The design principle behind IOHANALYZER is a multi-dimensional view on "performance", which in our regard comprises at least three objectives: quality, time, and robustness. Unlike other existing benchmark software, IOHANALYZER offers the user to chose which projections of this multi-dimensional data to investigate. In addition to the web-based application at http://iohprofiler.liacs.nl/ and the version on GitHub, IOHANALYZER is also available as CRAN package at https://cran.r-project.org/web/packages/IOHanalyzer/index.html.

As mentioned in the introduction, prior to this work IOHPROFILER only provided the experimental setup for discrete optimization benchmarking, but did not provide a selection of built-in benchmark problems, nor algorithms – a gap that we start to fill with our present work. Our objectives are two-fold. On the one hand, we want to make a step towards identifying functions that are particularly suitable for discriminating the performance of different IOHs. On the other hand, we also want to demonstrate that IOHPROFILER can handle large benchmarking projects; for the present work we have performed a total number of 12 144 experiments: eleven runs of each of the twelve baseline algorithms on a total number of 23 functions in 4 dimensions each. To test the influence of the different instances, an additional 12 144 runs have been performed.

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 [Doerr et al.(2019b)Doerr, Wang, Ye, van Rijn, and Bäck].

Following the discussion in [Shir et al.(2018)Shir, Doerr, and Bäck], we restrict our attention to *pseudo-Boolean functions;* i.e., all the suggested benchmark problems are expressed as functions $f : \{0,1\}^n \to \mathbb{R}$. We also pay particular attention to the *scalability* of the problems, with the idea that good benchmark problems should allow to assess performances across different dimensions.

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 the usage of "dummy variables" that do not contribute to the function values, or due to other reductions of the search space dimensionality (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 (see 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.

[1] The arXiv paper is currently under revision and will be replaced by an updated version soon. Note that in particular the structure of IOHEXPERIMENTER has evolved significantly – this module is not any more based on the COCO framework.

We formulate all problems as *maximization* problems. For most, but not for all problems the value of an optimal solution is known. Where these maximum function values are not known or are not achieved by any of the algorithms within the allocated computational budget, we evaluate performances with respect to the best solution found by any of the algorithms in any of the runs.

Notation A search point $x \in \{0,1\}^n$ is written as $(x_1, \ldots, x_n)$. 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 are 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 the selection of the benchmark problems used for our experiments.

Ambition Our ultimate goal is to construct a benchmarking suite that covers a wide range of the problem characteristics found in real-world combinatorial optimization problems. A second ambition 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 work. To put it differently, we present here a first step towards a meaningful collection of discrete optimization benchmarks, but do not claim that that this selection is "final". In addition, 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 of how to distinguish among various heuristics, or if they can be replaced by other problems showing similar effects. Indeed, as we will argue in Section 5, some of the 23 assessed functions do not seem to contribute much to a better discrimination between the tested algorithms, and are therefore evaluated as being obsolete. Note though that this is evaluation crucially depends on the collection of algorithms, so that these functions may have their merit in the assessment of other IOHs. As we shall discuss in Section 6, we are confident that further advances in the research on exploratory landscape analysis [Mersmann et al.(2011)Mersmann, Bischl, Trautmann, Preuss, Weihs, and Rudolph] could help identify additional problems to be included in the benchmark suite.

Problem Properties As mentioned, we are mostly interested in problems that are arbitrarily scalable with respect to the dimension n. However, as will be the case with the N-queens problem, 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 for each algorithm. In particular, we do *not* address with this work settings that feature *expensive* evaluations. We believe that those should be treated separately, as they typically require different solvers than problems allowing for high level of adaption between the evaluations.

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 $f : \{0,1\}^n \to \mathbb{R}$, 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 $\sigma : [n] \to [n]$. 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 $\{f(\sigma(\cdot \oplus z)) \mid z \in \{0,1\}^n, \sigma \text{ permutation of } [n]\}$, all of which have fitness landscapes that are pairwise isomorphic. For further discussions of these *unbiasedness* transformations, the reader is referred to [Lehre and Witt(2012), Doerr et al.(2018)Doerr, Wang, Ye, van Rijn, and Bäck].

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. To allow future comparisons 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 is identified as 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. More precisely, 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 offset are chosen as for instances 2-6. For each function and each dimension the values of z, σ, a, and b are fixed per each instance, but different functions of the same dimensions may have different z and σ transformations.

For the reader's convenience we recall that IOHEXPERIMENTER uses as pseudo-random number generator the linear congruential generator (LCG), using Schrage's method, the user can find and replace it by other generators in the file `IOHprofiler_random.hpp`.

3.3 Overview of Selected Benchmark Problems

We summarize here our selected benchmark problems, on which we will elaborate in the subsequent subsections.

- **F1 and F4-F10:** ONEMAX and W-model extensions; details in Sections 3.4 and 3.7
- **F2 and F11-F17:** LEADINGONES; see Sections 3.5 and 3.7
- **F3:** HARMONIC; see Section 3.6
- **F18:** LABS: Low Autocorrelation Binary Sequences; see Section 3.8
- **F19-21:** Ising Models; see Section 3.9
- **F22:** MIVS: Maximum Independent Vertex Set; see Section 3.10
- **F23:** NQP: N-Queens; see Section 3.11

3.4 F1: OneMax

The ONEMAX function is the best-studied benchmark problem in the context of discrete evolutionary computation (EC), often referred to as the "drosophila of EC". It asks to optimize the function

$$\text{OM} : \{0,1\} \to [0..n], x \mapsto \sum_{i=1}^{n} x_i.$$

The problem has a very smooth and non-deceptive fitness landscape. Due to the well-known coupon collector effect (see, for example, [Dubhashi and Panconesi(2009)] for a detailed explanation of this 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 Section 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 twelve 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 [Garnier et al.(1999)Garnier, Kallel, and Schoenauer]. The $(1 + (\lambda, \lambda))$ GA from [Doerr et al.(2015)Doerr, Doerr, and Ebel] is the only EA known to optimize ONEMAX in $o(n \log n)$ time. The self-adjusting version used in our experiments (see Section 4.1.9) requires only a linear expected number of function evaluations to locate the optimum [Doerr and Doerr(2018)]. The best expected running time that any iterative optimization algorithm can achieve is $\Omega(n/\log n)$ [Erdős and Rényi(1963)]. However, it is also known that unary unbiased algorithms cannot achieve average running times better than $\Omega(n \log n)$ [Lehre and Witt(2012), Doerr et al.(2016)Doerr, Doerr, and Yang]. A more detailed survey of theoretical running-time results for ONEMAX can be found in [Doerr et al.(2018)Doerr, Ye, van Rijn, Wang, and Bäck], and a survey of lower bounds (in terms of black-box complexity results) can be found in [Doerr(2018)]. That ONEMAX is interesting beyond the study of theoretical aspects of evolutionary computation has been argued in [Thierens(2009)]. We believe that ONEMAX plays a similar role as the sphere function in continuous domains, and should be added to each benchmark set: it is not very time-consuming to evaluate, and can provide a first basic stress test for new algorithm designs.

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

$$\mathrm{LO} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\} = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j,$$

which counts the number of initial ones.

Most EAs require quadratic running time to solve LEADINGONES, see again [Doerr et al.(2018)Doerr, Ye, van Rijn, Wang, and Bäck] or the full version of [Doerr(2018)] for a summary of theoretical results. It is known that all elitist (1+1)-type algorithms [Doerr and Lengler(2018)] and all unary unbiased [Lehre and Witt(2012)] 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 [Afshani et al.(2019)Afshani, Agrawal, Doerr, Doerr, Larsen, and Mehlhorn, Doerr and Winzen(2012), Doerr et al.(2011)Doerr, Johannsen, Kötzing, Lehre, Wagner, and Winzen]. It is also known that the best-possible expected running time of any iterative optimization heuristic is $\Theta(n \log \log n)$ [Afshani et al.(2019)Afshani, Agrawal, Doerr, Doerr, Larsen, and Mehlhorn]. Similar to ONEMAX, we argue that LEADINGONES should form a default benchmark problem: it is fast to evaluate and can point at fundamental issues of algorithmic designs, see also the discussions in Section 5.

3.6 F3: A Linear Function with Harmonic Weights

Two extreme linear functions are ONEMAX with its constant weights and binary value $\mathrm{BV}(x) = \sum_{i=1}^{n} 2^{n-i} x_i$ with its exponentially decreasing weights. An intermediate linear function is

$$f : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_i i x_i$$

with harmonic weights, which was suggested to be considered in [Shir et al.(2018)Shir, Doerr, and Bäck]. We add this linear function to our assessment as F3.

Several results mentioned in the paragraph about ONEMAX apply more generally to linear functions, and hence to the special case F3. For example, it is well known that the $(1+1)$ EA and RLS need $\Theta(n \log n)$ function evaluations, on average, to optimize any linear function [Droste et al.(2002)Droste, Jansen, and Wegener]. No unary unbiased black-box algorithm can achieve a better expected running time [Lehre and Witt(2012)]. It is also known that for the $(1+1)$ EA the best static mutation rate for optimizing any linear function is $1/n$ [Witt(2013)]. The (unrestricted) black-box complexity of linear functions, however, is not known. However, we easily see that the mentioned greedy hill climber gHC described in Section 3.4 optimizes F3 in at most $n+1$ queries.

3.7 F4-F17: The W-model

In [Weise and Wu(2018)] 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 [Weise and Wu(2018)] as well as the existing empirical evaluations [Weise(2018)] 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 [Kauffman and Levin(1987)] 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, see Section 3.7.3.. The assessment of combined transformations clearly forms a promising line for future work.

3.7.1 The Basic Transformations

The W-model comprises four basic transformations, and each of these transformations is parametrized, hence offering a huge set of different problems already. We provide a brief overview of the W-model transformations that are relevant for our work. A more detailed description can be found in the original work [Weise and Wu(2018)]. For some of the descriptions below we deviate from the exposition in [Weise and Wu(2018)], because in contrast to

there, we consider *maximization* as objective, not minimization. Note also that we write $x = (x_1, \ldots, x_n)$, whereas in [Weise and Wu(2018)] the strings are denoted as $(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$. Note also that the reduction of dummy variables is our own extension of the W-model, not originally proposed in [Weise and Wu(2018)].

1. **Reduction of dummy variables** $W(m, *, *, *)$**:** a reduction mapping each string $(x_1, \ldots, x_n)$ to a substring $(x_{i_1}, \ldots, x_{i_m})$ for randomly chosen, pairwise different $i_1, \ldots, i_m \in [n]$. This modification models a situation in which some decision variables do not have any or have only negligible impact on the fitness values. Thus, effectively, the strings $(x_1, \ldots, x_n)$ that the algorithm operates upon are reduced to substrings $(x_{i_1}, \ldots, x_{i_m})$ with $1 \leq i_1 < i_2 < \ldots < i_m \leq n$.

   We note that such scenarios have been analyzed theoretically, and different ways to deal with this *unknown solution length* have been proposed. Efficient EAs can obtain almost the same performance (in asymptotic terms) than EAs "knowing" the problem dimension [Einarsson et al.(2018)Einarsson, Lengler, Gauy, Meier, Mujika, Steger, and Weissenberger, Doerr et al.(2017)Doerr, Doerr, and Kötzing].

   Dummy variables are also among the characteristics of the benchmark functions contained in Facebook's nevergrad platform [Rapin and Teytaud(2018)], which might be seen as evidence for practical relevance.

   Example: With $n = 10$, $m = 5$, $i_1 = 1$, $i_2 = 2$, $i_3 = 4$, $i_4 = 7$, $i_5 = 10$, the bit string (1010101010) is reduced to (10010).

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 $(x_{(i-1)\mu+1}, x_{(i-1)\mu+2}, \ldots, x_{i\mu})$.[2] When $n/\mu \notin \mathbb{N}$, the last $n - \mu\lfloor n/\mu \rfloor$ remaining bits of x not fitting into any of the blocks are simply deleted; that is, we have $m = \lfloor n/\mu \rfloor$ and the entries x_i with $i > \mu\lfloor n/\mu \rfloor$ do not have any influence on y (and, thus, no influence on the function value).

   Example: With $n = 10$ and $\mu = 3$ the bit string (1110101110) is reduced to (101).

3. **Epistasis** $W(*, *, \nu, *)$**:** The idea 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 $e_\nu : \{0,1\}^\nu \to \{0,1\}^\nu$, 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 $\nu - 1$. Section 2.2 in [Weise and Wu(2018)] provides a construction for such permutations. For illustration purposes, we repeat below the map for $\nu = 4$, which is the parameter used in our experiments. This example can also be found, along with the general construction, in [Weise and Wu(2018)].

$$
\begin{array}{llll}
e_4(0000) = 0000 & e_4(0001) = 1101 & e_4(0010) = 1011 & e_4(0011) = 0110 \\
e_4(0100) = 0111 & e_4(0101) = 1010 & e_4(0110) = 1100 & e_4(0111) = 0001 \\
e_4(1000) = 1111 & e_4(1001) = 0010 & e_4(1010) = 0100 & e_4(1011) = 1001 \\
e_4(1100) = 1000 & e_4(1101) = 0101 & e_4(1110) = 0011 & e_4(1111) = 1110
\end{array}
$$

   When $n/\nu \notin \mathbb{N}$, the last bits of x are treated by $e_{n-\nu\lfloor n/\nu \rfloor}$; that is, the substring $(x_{\nu\lfloor n/\nu \rfloor+1}, x_{\nu\lfloor n/\nu \rfloor+2}, \ldots, x_n)$ is mapped to a new string of the same length via the function $e_{n-\nu\lfloor n/\nu \rfloor}$.

   Example: With $n = 10$, $\nu = 4$, and the permutation e_4 provided above, the bit string (1111011101) is mapped to (1110000110), because $e_4(1111) = 1110$ and $e_4(0111) = 0001$ and $e_2(01) = 10$.

4. **Fitness perturbation** $W(*, *, *, r)$**:** With these transformations 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 $r : \{f(x) \mid x \in \{0,1\}^n\} =: V \to V$ 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 $r \circ f$.

   To ease the analysis, it is required in [Weise and Wu(2018)] that the optimum $v_{\max} = \max\{f(x) \mid x \in \{0,1\}^n\}$ does not change, i.e., r must satisfy that $r(v_{\max}\}) = v_{\max}$ and $r(i) < v_{\max}$ for all $i < v_{\max}$. It is furthermore required in [Weise and Wu(2018)] that the ruggedness functions r are permutations (i.e., one-to-one maps). Both requirements are certainly not necessary, in the sense that additional interesting problems can be obtained by violating these constrains. We note in particular that in order to study *plateaus* of

[2] Note that with this formulation there is a bias towards ones in case of a tie. We follow here the suggestion made in [Weise and Wu(2018)], but we note that this bias may have a somewhat complex impact on the fitness landscape. For our first benchmark set, we therefore suggest to use this transformation with odd values for μ only.

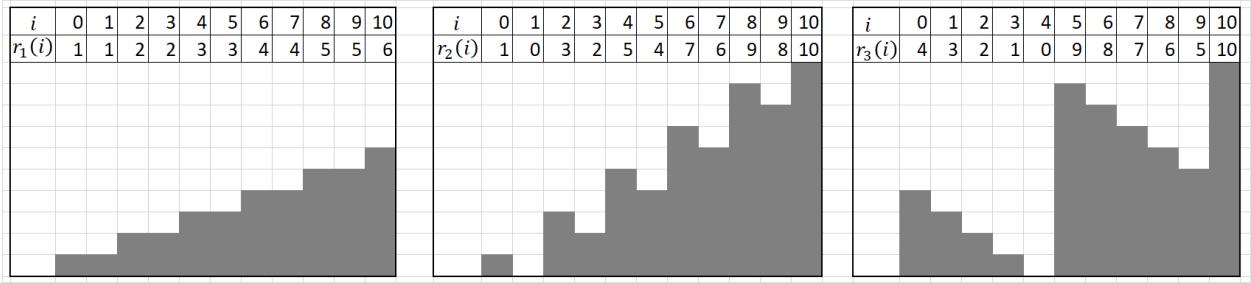

Figure 1: The ruggedness functions r_1, r_2, and r_3.

equal function values, one might want to choose functions that map several function values to the same value. We will include one such example in our testbed, see Section 3.7.3.

It should be noted that all functions of unitation (i.e., functions for which the function value depends only on the ONEMAX value of the search point, such as TRAP or JUMP) can be obtained from a superposition of the fitness perturbation onto the ONEMAX problem.

Example: The well-known, highly deceptive TRAP function can be obtained by superposing the permutation $r : [0..n] \to [0..n]$ with $r(i) = n - 1 - i$ for all $1 \leq i \leq n$ and $r(n) = n$.

3.7.2 Combining the Basic W-model Transformations

We note that any of the four W-model transformations can be applied independently of each other. The first three modification can, in addition, be applied in an arbitrary order, with each order resulting in a different benchmark problem. In line with the presentation in [Weise and Wu(2018)], we consider in our implementation only those perturbations that follow the order given above. Each set of W-model transformations can be identified by a string $(\{i_1, \ldots, i_m\}, \mu, \nu, r)$ with $m \leq n$, $1 \leq i_1 < \ldots < i_m \leq n$, $\mu \in [n]$, $\nu \in [n]$, and $r : V \to V$, all to be interpreted as in the descriptions given in Section 3.7.1 above. Setting $\{i_1, \ldots, i_m\} = [n]$, $\mu = 1$, $\nu = 1$, and/or r as the identity function on V corresponds to not using the first, second, third, and/or forth transformation, respectively.

As mentioned, the W-model can in principle be superposed on any benchmark problem. The only complication is that the search space on which the algorithm operates and the search space on which the benchmark problem is applied are not the same when $m < n$ or $\mu > 1$. More precisely, while the algorithm operates on $\{0,1\}^n$, the base problem has to be a function $f : \{0,1\}^s \to \mathbb{R}$ with $s = \lfloor m/\mu \rfloor$. We call s the *effective dimension* of the problem. When f is a scalable function defined for any problem dimension s—this is the case for most of our benchmark functions—we just reduce to the s-dimensional variant of the problem. When f is a problem that is only defined for a fixed dimension n, the algorithms should operate on the search space $\{0,1\}^\ell$ with $\ell \geq \mu s$ and $\ell - \mu s$ depending on the reduction that one wishes to achieve by the first transformation, the removal of dummy variables.

3.7.3 Selected W-Model Transformations

In contrast to existing works cited in [Weise and Wu(2018), Weise(2018)], we do not only study superpositions of W-model transformations to the ONEMAX problems (functions F4-F10), but we also consider LEADINGONES as a base problem (F11-17). This allows us to study the effects of the transformations on a well-understood separable and a well-understood non-separable problem. As mentioned, we only study individual transformations, and not yet combinations thereof.

We consider the reduction of $[n]$ to subsets of size $n/2$ and $0.9n$, i.e., only half and 90% of the bits, respectively, contribute to the overall fitness. We consider neutrality transformations of size $\mu = 3$, and we consider the epistasis perturbation of size 4. Finally, we consider the following ruggedness functions, where we denote by s the size of the effective dimension (see Section 3.7.2 for a discussion) and recall that both the s-dimensional ONEMAX and LEADINGONES functions take values in $[0..s]$. These functions are illustrated for $s = 10$ in Figure 1.

- $r_1 : [0..s] \to [0..\lceil s/2 \rceil + 1]$ 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.
- $r_2 : [0..s] \to [0..s]$ 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.
- $r_3 : [0..s] \to [-5..s]$ 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 $k \in [0..s - 5\lfloor s/5 \rfloor - 1]$.

We see that function r_1 keeps the order of the function values, but introduces small plateaus of the same function value. In contrast to r_1, function r_2 is a permutation of the possible function values. It divides the set of possible non-optimal function values $[0..s-1]$ into blocks of size two (starting at $s-1$ and going in the inverse direction) and interchanges the two values in each block. When s is odd, the value 0 forms its own block with $r_1(0) = 0$. Similarly, r_3 divides the set of possible function values in blocks of size 5 (starting at $s-1$ and going in inverse direction), and reverses the order of function values in each block.

Summarizing all these different setups, the functions F4-F17 are defined as follows:

F4: $\textsc{OneMax} + W([n/2], 1, 1, \mathrm{id})$	F11: $\textsc{LeadingOnes} + W([n/2], 1, 1, \mathrm{id})$
F5: $\textsc{OneMax} + W([0.9n], 1, 1, \mathrm{id})$	F12: $\textsc{LeadingOnes} + W(0.9n, 1, 1, \mathrm{id})$
F6: $\textsc{OneMax} + W([n], \mu = 3, 1, \mathrm{id})$	F13: $\textsc{LeadingOnes} + W([n], \mu = 3, 1, \mathrm{id})$
F7: $\textsc{OneMax} + W([n], 1, \nu = 4, \mathrm{id})$	F14: $\textsc{LeadingOnes} + W([n], 1, \nu = 4, \mathrm{id})$
F8: $\textsc{OneMax} + W([n], 1, 1, r_1)$	F15: $\textsc{LeadingOnes} + W([n], 1, 1, r_1)$
F9: $\textsc{OneMax} + W([n], 1, 1, r_2)$	F16: $\textsc{LeadingOnes} + W([n], 1, 1, r_2)$
F10: $\textsc{OneMax} + W([n], 1, 1, r_3)$	F17: $\textsc{LeadingOnes} + W([n], 1, 1, r_3)$

3.7.4 W-model vs. Unbiasedness Transformations and Fitness Scaling

To avoid confusion, we clarify the sequence of the transformations of the W-model and the unbiasedness and fitness value transformations discussed in Section 3.2. Both the re-ordering of the string by the permutation σ and the XOR with a fixed string $z \in \{0,1\}^n$ are executed *before* the transformations of the W-model are applied, while the multiplicative and additive scaling of the function values is applied to the result *after* the fitness perturbation of the W-model.

Example: Assume that the instance is generated from a base problem $f : \{0,1\}^n \to \mathbb{R}$, that the unbiasedness transformations are defined by a permutation $\sigma : [n] \to [n]$ and the string $z \in \{0,1\}^n$, the fitness scaling by a multiplicative scalar $b > 0$ and an additive term $a \in \mathbb{R}$. Assume further that the W-model transformations are defined by the vector $(i_1, \ldots, i_m, \mu, \nu, r)$. For each queried search point $x \in \{0,1\}^n$, the algorithm receives the function value $af(W(\sigma(x) \oplus z)) + b$, where $\sigma(x) = (x_{\sigma(1)}, \ldots, x_{\sigma(n)})$ and $W : \{0,1\}^n \to \mathbb{R}$ denotes the function that maps each string to the fitness value defined via the W-transformations $(i_1, \ldots, i_m, \mu, \nu, r)$.

3.8 F18: Low Autocorrelation Binary Sequences

Obtaining binary sequences possessing a high merit factor, also known as the Low-Autocorrelation Binary Sequence (LABS) problem, constitutes a grand combinatorial challenge with practical applications in radar engineering and measurements [Shapiro et al.(1968)Shapiro, Pettengill, Ash, Stone, Smith, Ingalls, and Brockelman, Pasha et al.(2000)Pasha, Moharir, and Rao]. It also carries several open questions concerning its mathematical nature. Given a sequence of length n, $S = (s_1, \ldots, s_n)$ with $s_i \in \{-1, +1\}$, the merit factor is proportional to the reciprocal of the sequence's autocorrelation. The LABS optimization problem is defined as searching over the sequence space to yield the maximum merit factor: $\frac{n^2}{2E(S)}$ with $E(S) = \sum_{k=1}^{n-1} \left(\sum_{i=1}^{n-k} s_i \cdot s_{i+k} \right)^2$. This hard, non-linear problem has been studied over several decades (see, e.g., [Militzer et al.(1998)Militzer, Zamparelli, and Beule, Packebusch and Mertens(2016)]), where the only way to obtain exact solutions remains exhaustive search. As a pseudo-Boolean function over $\{0,1\}^n$, it can be rewritten as follows:

$$F_{\text{LABS}}(\vec{x}) = \frac{n^2}{2 \sum_{k=1}^{n-1} \left(\sum_{i=1}^{n-k} x'_i \cdot x'_{i+k} \right)^2} \quad \text{where } x'_i = 2x_i - 1. \tag{1}$$

3.9 F19-F21: The Ising Model

The Ising Spin Glass model [Barahona(1982)] arose in solid-state physics and statistical mechanics, aiming to describe simple interactions within many-particle systems. The classical Ising model considers a set of spins placed on a regular lattice, where each edge $\langle i, j \rangle$ is associated with an interaction strength $J_{i,j}$. In essence, a problem-instance is defined upon setting up the coupling matrix $\{J_{i,j}\}$. Each spin directs *up* or *down*, associated with a value ± 1, and a set of n spin glasses is said to form a configuration, denoted as $S = (s_1, \ldots, s_n) \in \{-1, +1\}^n$. The configuration's energy function is described by the system's Hamiltonian, as a quadratic function of those n spin variables: $-\sum_{i<j} J_{i,j} s_i s_j - \sum_{i=1}^{n} h_i s_i$, where h_i is an external magnetic field. The optimization problem of interest is the study of the minimal energy configurations, which are termed *ground states*, on a final lattice. This is clearly a challenging combinatorial optimization problem,

which is known to be NP-hard, and to hold connections with all other NP problems [Lucas(2014)]. The Ising model has been investigated in light of EAs' operation, yielding some theoretical results on certain graph instances (see, e.g., [Briest et al.(2004)Briest, Brockhoff, Degener, Englert, Gunia, Heering, Jansen, Leifhelm, Plociennik, Röglin et al., Fischer and Wegener(2005), Sudholt(2005)]).

We have selected and integrated three Ising model instances in IOHPROFILER, assuming zero external magnetic fields, and applying *periodic boundary conditions* (PBC). In order to formally define the Ising objective functions, we adopt a strict graph perspective, where $G = (V, E)$ is undirected and $V = [n]$. We apply an affine transformation $\{-1, +1\}^n \rightsquigarrow \{0, 1\}^n$, where the n spins become binary decision variables (this could be interpreted, e.g., as a coloring problem; see [Sudholt(2005)]). A generalized, compact form for the quadratic objective function is now obtained:

$$F_{\text{Ising}}(\vec{x}) = \sum_{\{u,v\} \in E} \left[x_u x_v - (1 - x_u)(1 - x_v) \right], \tag{2}$$

thus leaving the instance definition within G.

In what follows, we specify their underlying graphs, whose edges are equally weighted as unity, to obtain their objective functions using (2).

3.9.1 F19: The Ring (1D)

This basic Ising model is defined over a one-dimensional lattice. The objective function follows (2) using the following graph:

$$\begin{aligned} & G_{\text{Is1D}}: \\ e_{ij} = 1 \quad & \Leftrightarrow \quad j = i + 1 \; \forall i \in \{1, \ldots, n-1\} \\ & \vee \quad j = n, i = 1 \end{aligned} \tag{3}$$

3.9.2 F20: The Torus (2D)

This instance is defined on a two-dimensional lattice of size N, using altogether $n = N^2$ vertices, denoted as (i, j), $0 \leq i, j \leq N - 1$ [Briest et al.(2004)Briest, Brockhoff, Degener, Englert, Gunia, Heering, Jansen, Leifhelm, Plociennik, Röglin et al.]. Since PBC are applied, a *regular graph with each vertex having exactly four neighbors* is obtained. The objective function follows (2) using the following graph:

$$\begin{aligned} & G_{\text{Is2D}}: \\ e_{(i,j)(k,\ell)} = 1 \quad & \Leftrightarrow \quad [k = (i+1) \bmod N \; \wedge \; \ell = j \; \forall i, j \in \{0, \ldots, N-1\}] \\ & \vee \quad [k = (i-1) \bmod N \; \wedge \; \ell = j \; \forall i, j \in \{0, \ldots, N-1\}] \\ & \vee \quad [\ell = (j+1) \bmod N \; \wedge \; k = i \; \forall j, i \in \{0, \ldots, N-1\}] \\ & \vee \quad [\ell = (j-1) \bmod N \; \wedge \; k = i \; \forall j, i \in \{0, \ldots, N-1\}] \end{aligned}$$

3.9.3 F21: Triangular (Isometric 2D Grid)

This instance is also defined on a two-dimensional lattice, yet constructed on an isometric grid (also known as triangular grid), *whose unit vectors form an angle of* $\frac{2\pi}{3}$ [Mellor(2011)]. The vertices are placed on integer-valued two-dimensional $n = N^2$ vertices, denoted as (i, j), $0 \leq i, j \leq N - 1$, yielding altogether a regular graph whose vertices have exactly six neighbors each (due to PBC):

$$\begin{aligned} & G_{\text{IsTR}}: \\ e_{(i,j)(k,\ell)} = 1 \quad & \Leftrightarrow \quad [k = (i+1) \bmod N \; \wedge \; \ell = j \; \forall i, j \in \{0, \ldots, N-1\}] \\ & \vee \quad [k = (i-1) \bmod N \; \wedge \; \ell = j \; \forall i, j \in \{0, \ldots, N-1\}] \\ & \vee \quad [\ell = (j+1) \bmod N \; \wedge \; k = i \; \forall j, i \in \{0, \ldots, N-1\}] \\ & \vee \quad [\ell = (j-1) \bmod N \; \wedge \; k = i \; \forall j, i \in \{0, \ldots, N-1\}] \\ & \vee \quad [\ell = (j+1) \bmod N \; \wedge \; k = (i+1) \bmod N \; \forall j, i \in \{0, \ldots, N-1\}] \\ & \vee \quad [\ell = (j-1) \bmod N \; \wedge \; k = (i-1) \bmod N \; \forall j, i \in \{0, \ldots, N-1\}] \end{aligned}$$

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 direct neighbors. A maximum independent vertex set (MIVS) (which generally is not equivalent to a *maximal* independent vertex set)

is defined as an independent subset $V' \subset [n]$ having the largest possible size. Using the standard binary encoding $V' = \{i \in [n] \mid x_i = 1\}$, MIVS can be formulated as the maximization of the function

$$F_{\text{MIVS}}(x) = \sum_i x_i - n \cdot \sum_{i,j} x_i x_j e_{i,j}, \tag{4}$$

where $e_{i,j} = 1$ if $\{i, j\} \in E$ and $e_{i,j} = 0$ otherwise.

In particular, following [Bäck and Khuri(1994)], we consider a specific, scalable problem instance, defining its Boolean graph as follows:

$$\begin{aligned} e_{ij} = 1 \quad &\Leftrightarrow \quad j = i + 1 \; \forall i \in \{1, \ldots, n-1\} - \{n/2\} \\ &\vee \quad j = i + n/2 + 1 \; \forall i \in \{1, \ldots, n/2 - 1\} \\ &\vee \quad j = i + n/2 - 1 \; \forall i \in \{2, \ldots, n/2\}. \end{aligned} \tag{5}$$

The resulting graph has a simple, standard structure as shown in Figure 2 for $n = 10$. The global optimizer has an objective function value of $|V'| = n/2 + 1$ for this standard graph. Notably, $n \geq 4$ and n is required to be even; given an odd n, we identify the n-dimensional problem with the $n - 1$-dimensional instance.

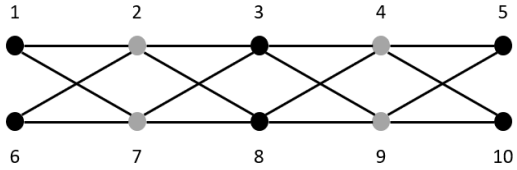

Figure 2: A scalable maximum independent set problem, with $n = 10$ vertices and the optimal solution of size 6 marked by the black vertices.

3.11 F23: N-Queens Problem

The N-queens problem (NQP) [Bell and Stevens(2009)] 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.[3] Figure 3 provides an illustration for the 8-queens problem. Notably, the *NQP is actually an instance of the MIVS problem* – when considering a graph on which all possible queen-attacks are defined as edges. NQP formally constitutes a *Constraints Satisfaction Problem*, but is posed here as a maximization problem using a binary representation:

$$\begin{aligned} &\text{maximize } \sum_{i,j} x_{ij} \\ &\text{subject to:} \\ &\sum_i x_{ij} \leq 1 \; \forall j \in \{1 \ldots, N\} \\ &\sum_j x_{ij} \leq 1 \; \forall i \in \{1 \ldots, N\} \\ &\sum_{j-i=k} x_{ij} \leq 1 \; \forall k \in \{-N+2, -N+3, \ldots, N-3, N-2\} \\ &\sum_{i+j=\ell} x_{ij} \leq 1 \; \forall \ell \in \{3, 4, \ldots, 2N-3, 2N-1\} \\ &x_{ij} \in \{0, 1\} \; \forall i, j \in \{1, \ldots, N\} \end{aligned}$$

This formulation utilizes $n = N^2$ binary decision variables x_{ij}, which are associated with the chessboard's coordinates, having an origin $(1, 1)$ at the top-left corner. Setting a binary to 1 implies a single queen assignment in that cell. This formulation promotes placement of as many queens as possible by means of the objective function, followed by four sets of constraints eliminating queens' *mutual threats*: the first two sets ensure a single queen on each row and each column, whereas the following two sets ensure a single queen at the increasing-diagonals (using the dummy indexing k) and decreasing-diagonals (using the dummy indexing ℓ). It should be noted that a *permutation formulation* also exists

[3]The NQP is traced back to the 1848 Bezzel article entitled "Proposal of the Eight Queens Problem"; for a comprehensive list of references we refer the reader to a documentation by W. Kosters at
http://liacs.leidenuniv.nl/~kosterswa/nqueens/nqueens_feb2009.pdf.

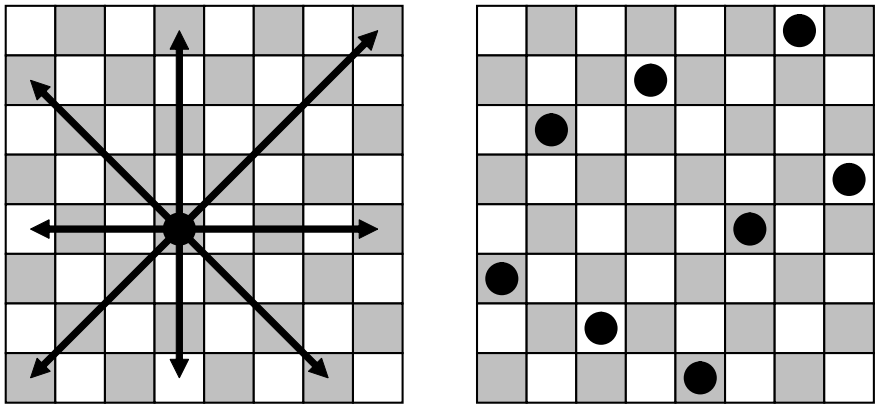

Figure 3: The 8-queens problem: [Left] all possible fields a queen can move to from position D4; [Right] a feasible solution.

for this problem, and is sometimes attractive for RSHs. Due to chessboard symmetries, NQP possesses multiplicity of optimal solutions. Its attractiveness, however, lies in its hardness. In terms of a black-box objective function, we formulate NQP as the maximization of the following function:

$$F_{\text{NQP}}(\vec{x}) = \sum_{i=1}^{N}\sum_{j=1}^{N} x_{ij} - N \cdot \left(\sum_{i=1}^{N} \max\left\{0, -1 + \sum_{j=1}^{N} x_{ij}\right\} + \sum_{j=1}^{N} \max\left\{0, -1 + \sum_{i=1}^{N} x_{ij}\right\} + \sum_{k=-N+2}^{N-2} \max\left\{0, -1 + \sum_{\substack{j-i=k \\ i,j\in\{1,2,\ldots,N\}}} x_{ij}\right\} + \sum_{\ell=3}^{2N-1} \max\left\{0, -1 + \sum_{\substack{j+i=\ell \\ i,j\in\{1,2,\ldots,N\}}} x_{ij}\right\} \right) \tag{6}$$

4 Algorithms

We evaluate a total number of twelve different algorithms on the problems described above. We have chosen algorithms that may serve for future references, since they all have some known strengths and weaknesses that will become apparent in the following discussions. Our selection therefore shows a clear bias towards algorithms for which theoretical analyses are available.

Note that most algorithms are parametrized, and we use here in this work only standard parametrizations (e.g., we use $1/n$ as mutation rates, etc.). Analyzing the effects of different parameter values as was done, for example in [Rodionova et al.(2019)Rodionova, Antonov, Buzdalova, and Doerr, Dang and Doerr(2019)], would be very interesting, but is beyond the scope of this present work.

We also note that, except for the so-called vGA, our implementations (deliberately) deviate slightly from the textbook descriptions referenced below. Following the suggestions made in [Carvalho Pinto and Doerr(2017)], we enforce that offspring created by mutation are different from their parent and resample without further evaluation if needed. Likewise, we do not evaluate recombination offspring that are identical to one of their immediate parents. Since we use this convention throughout, we omit the subscript $_{>0}$ used in [Carvalho Pinto and Doerr(2017), Doerr et al.(2018)Doerr, Ye, van Rijn, Wang, and Bäck].

All algorithms start with uniformly chosen initial solution candidates.

We list here the twelve implemented algorithms, and provide further details and pseudo-codes thereafter:

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. That is, RLS and gHC differ only in the choice of the bit which is flipped. While RLS is unbiased in the sense of Section 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 $\mathrm{Bin}_{>0}(n,p)$. That is, each bit is flipped independently with probability p and if none of the bits has been flipped this *standard bit mutation* is repeated until we obtain an offspring that differs from its parent in at least one bit.

4. **fGA:** The "fast GA" proposed in [Doerr et al.(2017a)Doerr, Le, Makhmara, and Nguyen] 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 as least as good.
6. $(1+10)$ $\mathbf{EA}_{r/2,2r}$**:** The two-rate EA with self-adjusting mutation rates suggested and analyzed in [Doerr et al.(2017b)Doerr, Gießen, Witt, and Yang].
7. $(1+10)$ $\mathbf{EA}_{\mathrm{norm.}}$**:** 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 [Ye et al.(2019)Ye, Doerr, and Bäck].
8. $(1+10)$ $\mathbf{EA}_{\mathrm{var.}}$**:** The $(1+10)$ $\mathrm{EA}_{\mathrm{norm.}}$ with an adaptive choice of the variance in the normal distribution from which the mutation strengths are sampled. Also from [Ye et al.(2019)Ye, Doerr, and Bäck].
9. $(1+10)$ $\mathbf{EA}_{\text{log-n.}}$ The (1+10) EA with log-normal self-adaptation of the mutation rate proposed in [Bäck and Schütz(1996)].
10. $(1+(\lambda,\lambda))$ **GA:** A binary (i.e., crossover-based) EA originally suggested in [Doerr et al.(2015)Doerr, Doerr, and Ebel]. We use the variant with self-adjusting λ analyzed in [Doerr and Doerr(2018)].
11. **vGA:** A $(30,30)$ "vanilla" GA (following the so-called traditional GA, as described, for example, in [Goldberg(1989), Bäck(1996)]).
12. **UMDA:** A univariate marginal distribution algorithm from the family of estimation of distribution algorithms (EDAs). UMDA was originally proposed in [Mühlenbein(1997)].

4.1 Detailed Description of the Algorithms

A detailed description of the algorithms follows. An operator frequently used in these descriptions is the $\mathrm{flip}_\ell(\cdot)$ mutation operator, which flips the entries of ℓ pairwise different, uniformly at random chosen bit positions, see Algorithm 1.

Algorithm 1: flip_ℓ chooses ℓ different positions and flips the entries in these positions.

Input: $x \in \{0,1\}^n$, $\ell \in \mathbb{N}$;
Select ℓ pairwise different positions $i_1, \dots, i_\ell \in [n]$ uniformly at random;
$y \leftarrow x$;
for $j = 1, ..., \ell$ **do** $y_{i_j} \leftarrow 1 - x_{i_j}$;

It is well known that standard bit mutation, which flips each bit in a length-n bit string with some probability p, can be equivalently defined as the operator calling flip_ℓ for a *mutation strength* chosen from the binomial distribution $\mathrm{Bin}(n,p)$. Since we want to avoid useless evaluations of offspring that are identical to their parents, we frequently make use of the conditional binomial distribution $\mathrm{Bin}_{>0}(n,p)$, which assigns probability $\mathrm{Bin}(n,p)(k)/(1-(1-p)^n)$ to each positive integer $k \in [n]$, and probability zero to all other values. Sampling from $\mathrm{Bin}_{>0}(n,p)$ is identical to sampling from $\mathrm{Bin}(n,p)$ until a positive value is returned ("resampling strategy").

4.1.1 Greedy Hill Climber

The greedy hill climber (gHC, Algorithm 2) uses a deterministic mutation strength, and flips one bit in each iteration, going through the bit string from left to right, until being stuck in a local optimum, see Algorithm 2.

Algorithm 2: Greedy hill climber (gHC)

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
Optimization: for $t = 1, 2, 3, \dots$ **do**
  $x^* \leftarrow x$;
  Flip in x^* the entry in position $1 + (t \mod n)$ and evaluate $f(x^*)$;
  if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;

4.1.2 Randomized Local Search

RLS uses a deterministic mutation strength, and flips one randomly chosen bit in each iteration, see Algorithm 3.

Algorithm 3: Randomized local search (RLS)

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
Optimization: **for** $t = 1, 2, 3, \ldots$ **do**
  create $x^* \leftarrow \text{flip}_1(x)$, and evaluate $f(x^*)$;
  if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;

4.1.3 The EA with Static Mutation Rate

The $(1 + \lambda)$ EA is defined via the pseudo-code in Algorithm 4. We use $\lambda = 1$ and $\lambda = 10$ in our comparisons.

Algorithm 4: The $(1 + \lambda)$ EA with static mutation rates

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
Optimization: **for** $t = 1, 2, 3, \ldots$ **do**
  for $i = 1, \ldots, \lambda$ **do**
    Sample $\ell^{(i)} \sim \text{Bin}_{>0}(n, 1/n)$;
    create $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  $x^* \leftarrow \arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ (ties broken by selecting the first max $f(y^{(i)})$);
  if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;

4.1.4 Fast Genetic Algorithm

The *fast Genetic Algorithm* (fGA) chooses the mutation length ℓ according to a power-law distribution $D^\beta_{n/2}$, which assigns to each integer $k \in [n/2]$ a probability of $\Pr[D^\beta_{n/2} = k] = (C^\beta_{n/2})^{-1} k^{-\beta}$, where $C^\beta_{n/2} = \sum_{i=1}^{n/2} i^{-\beta}$. We use the (1+1) variant of this algorithm with $\beta = 1.5$.

Algorithm 5: Fast genetic algorithm (fGA) from [Doerr et al.(2017a)Doerr, Le, Makhmara, and Nguyen]

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
Optimization: **for** $t = 1, 2, 3, \ldots$ **do**
  for $i = 1, \ldots, \lambda$ **do**
    Sample $\ell^{(i)} \sim D^\beta_{n/2}$;
    create $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  $x^* \leftarrow \arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ (ties broken by favoring the largest index);
  if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;

4.1.5 The Two-Rate EA

The two-rate $(1 + \lambda)$ $\text{EA}_{r/2,2r}$ was introduced in [Doerr et al.(2017b)Doerr, Gießen, Witt, and Yang]. It uses two mutation rates in each iteration: half of offspring are generated with mutation rate $r/2n$, and the other $\lambda/2$ offspring

are sampled with mutation rate $2r/n$. The parameter r is updated after each iteration, from a biased random decision favoring the value from which the best of all λ offspring has been sampled.

Algorithm 6: The two-rate EA with adaptive mutation rates proposed in [Doerr et al.(2017b)Doerr, Gießen, Witt, and Yang]

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
Initialize $r \leftarrow r^{\text{init}}$; // we use $r^{\text{init}} = 2$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  for $i = 1, \ldots, \lambda/2$ **do**
    Sample $\ell^{(i)} \sim \text{Bin}_{>0}(n, r/(2n))$, create $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  for $i = \lambda/2 + 1, \ldots, \lambda$ **do**
    Sample $\ell^{(i)} \sim \text{Bin}_{>0}(n, 2r/n)$, create $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  $x^* \leftarrow \arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ (ties broken uniformly at random);
  if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;
  if x^* *has been created with mutation rate* $r/2$ **then** $s \leftarrow 3/4$ **else** $s \leftarrow 1/4$;
  Sample $q \in [0,1]$ uniformly at random;
  if $q \leq s$ **then** $r \leftarrow \max\{r/2, 2\}$ **else** $r \leftarrow \min\{2r, n/4\}$;

4.1.6 The EA with normalized standard bit mutation

The $(1+\lambda)$ $\text{EA}_{\text{norm.}}$, Algorithm 7, samples the mutation strength from a normal distribution with mean $r = pn$ and variance $pn(1-p) = r(1-r/n)$, which is identical to the variance of the binomial distribution used in standard bit mutation. The parameter r is updated after each iteration, in a similar fashion as in the 2-rate EA, Algorithm 6.

Algorithm 7: The $(1+\lambda)$ $\text{EA}_{\text{norm.}}$ with normalized standard bit mutation

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
Initialize $r \leftarrow r^{\text{init}}$; // we use $r^{\text{init}} = 2$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  for $i = 1, \ldots, \lambda$ **do**
    Sample $\ell^{(i)} \sim \min\{N_{>0}(r, r(1-r/n)), n\}$, create $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  $i \leftarrow \min\left\{j \mid f(y^{(j)}) = \max\{f(y^{(k)}) \mid k \in [\lambda]\}\right\}$;
  $r \leftarrow \ell^{(i)}$;
  if $f(y^{(i)}) \geq f(x)$ **then** $x \leftarrow y^{(i)}$;

4.1.7 The EA with normalized standard bit mutation and controlled variance

The $(1+\lambda)$ $\text{EA}_{\text{var.}}$, Algorithm 8 builds on the $(1+\lambda)$ $\text{EA}_{\text{norm.}}$ but uses, in addition to the adaptive choice of the mutation rate, an adaptive choice of the variance.

Algorithm 8: The $(1+\lambda)$ $\text{EA}_{\text{var.}}$ with normalized standard bit mutation and a self-adjusting choice of mean and variance

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
Initialize $r \leftarrow r^{\text{init}}$; // we use $r^{\text{init}} = 2$;
Initialize $c \leftarrow 0$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  for $i = 1, \ldots, \lambda$ **do**
    Sample $\ell^{(i)} \sim \min\{N_{>0}(r, F^c r(1-r/n)), n\}$, create $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  $i \leftarrow \min\left\{j \mid f(y^{(j)}) = \max\{f(y^{(k)}) \mid k \in [\lambda]\}\right\}$;
  if $r = \ell^{(i)}$ **then** $c \leftarrow c + 1$; **else** $c \leftarrow 0$;
  $r \leftarrow \ell^{(i)}$;
  if $f(y^{(i)}) \geq f(x)$ **then** $x \leftarrow y^{(i)}$;

4.1.8 The EA with log-Normal self-adaptation on mutation rate

The $(1+\lambda)$ $\text{EA}_{\text{log-n.}}$, Algorithm 9, uses a self-adaptive choice of the mutation rate.

Algorithm 9: The $(1+\lambda)$ $\text{EA}_{\text{log-n.}}$ with log-Normal self-adaptation of the mutation rate

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
$p = 0.2$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  for $i = 1, \ldots, \lambda$ **do**
    $p^{(i)} = \left(1 + \frac{1-p}{p} \cdot \exp(0.22 \cdot \mathcal{N}(0,1))\right)^{-1}$;
    Sample $\ell^{(i)} \sim \text{Bin}_{>0}(n, p^{(i)})$;
    create $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$, and evaluate $f(y^{(i)})$;
  $i \leftarrow \min\left\{j \mid f(y^{(j)}) = \max\{f(y^{(k)}) \mid k \in [\lambda]\}\right\}$;
  $p \leftarrow p^{(i)}$;
  $x^* \leftarrow \arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ (ties broken by favoring the smallest index);
  if $f(x^*) \geq f(x)$ **then** $x \leftarrow x^*$;

4.1.9 The Self-Adjusting $(1+(\lambda,\lambda))$ GA

The self-adjusting $(1+(\lambda,\lambda))$ GA, Algorithm 10, was introduced in [Doerr et al.(2015)Doerr, Doerr, and Ebel] and analyzed in [Doerr and Doerr(2018)]. The offspring population size λ is updated after each iteration, depending on whether or not an improving offspring could be generated. Since both the mutation rate and the crossover bias (see Algorithm 11 for the definition of the biased crossover operator cross) depend on λ, these two parameters also change during the run of the $(1+(\lambda,\lambda))$ GA. In our implementation we use update strength $F = 3/2$.

Algorithm 10: The self-adjusting $(1+(\lambda,\lambda))$ GA

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  Mutation phase:
    Sample $\ell \sim \text{Bin}_{>0}(n, \lambda/n)$;
    for $i = 1, \ldots, \lambda$ **do** create $y^{(i)} \leftarrow \text{flip}_\ell(x)$, and evaluate $f(y^{(i)})$;
    $x^* \leftarrow \arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ (ties broken by favoring the largest index);
  Crossover phase:
    for $i = 1, \ldots, \lambda$ **do** create $y^{(i)} \leftarrow \text{cross}_c(x, x^*)$, and evaluate $f(y^{(i)})$;
    $y^* \leftarrow \arg\max\{f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ (ties broken by favoring the largest index);
  Selection phase:
    if $f(y^*) > f(x)$ **then** $x \leftarrow y^*$; $\lambda \leftarrow \max\{\lambda/F, 1\}$;
    if $f(y^*) = f(x)$ **then** $x \leftarrow y^*$; $\lambda \leftarrow \min\{\lambda F^{1/4}, n\}$;
    if $f(y^*) < f(x)$ **then** $\lambda \leftarrow \min\{\lambda F^{1/4}, n\}$;

Algorithm 11: Crossover operation $\text{cross}_c(x, x^*)$ with crossover bias c

$y \leftarrow x$;
Sample $\ell \sim \text{Bin}_{>0}(n, c)$;
Select ℓ different positions $\{i_1, \ldots, i_l\} \in [n]$;
for $j = 1, 2, \ldots, \ell$ **do** $y_{i_j} \leftarrow x^*_{i_j}$;

4.1.10 The "Vanilla" GA

The vanilla GA (vGA, Algorithm 13) constitutes a textbook realization of the so-called Traditional GA [Goldberg(1989), Bäck(1996)]. The algorithm holds a parental population of size μ. It employs the Roulette-Wheel-Selection (RWS, that is, probabilistic fitness-proportionate selection which permits an individual to appear multiple times) as the sexual selection operator to form $\mu/2$ pairs of individuals that generate the offspring population. 1-point crossover

Algorithm 12: 1-Point crossover of two parents $x^{(1)}$ and $x^{(2)}$

Sample $\ell \in [n]$ uniformly at random;
for $i = 1, 2, \ldots, \ell$ **do** Set $y_i^{(1)} \leftarrow x_i^{(1)}$ and $y_i^{(2)} \leftarrow x_i^{(2)}$;
for $i = \ell + 1, \ldots, n$ **do** Set $y_i^{(1)} \leftarrow x_i^{(2)}$ and $y_i^{(2)} \leftarrow x_i^{(1)}$;

(Algorithm 12) is applied to every pair with a fixed probability of $p_c = 0.37$. A mutation operator is then applied to every individual, flipping every bit with a fixed probability of $p_m = 2/n$. This completes a single cycle.

Algorithm 13: The (μ, μ)-"Vanilla-GA" with mutation rate p_m and crossover probability p_c

Initialization:
  for $i = 1, \ldots, \mu$ **do** sample $x^{(i)} \in \{0,1\}^n$ uniformly at random and evaluate $f(x^{(i)})$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  Parent selection phase: Apply roulette-wheel selection to $\{x^{(1)}, \ldots, x^{\mu}\}$ to select μ parent individuals $y^{(1)}, \ldots, y^{(\mu)}$;
  Crossover phase:
    for $i = 1, \ldots, \mu/2$ **do** with probability p_c replace $y^{(i)}$ and $y^{(2i)}$ by the two offspring that result from a 1-point crossover of these two parents, for a randomly chosen crossover point $j \in [n]$;
  Mutation phase:
    for $i = 1, \ldots, \mu$ **do** Sample $\ell^{(i)} \sim \mathrm{Bin}(n, p_m)$, set $y^{(i)} \leftarrow \mathrm{flip}_{\ell^{(i)}}(y^{(i)})$, and evaluate $f(y^{(i)})$;
  Replacement:
    for $i = 1, \ldots, \mu$ **do** Replace $x^{(i)}$ by $y^{(i)}$;

4.1.11 The Univariate Marginal Distribution Algorithm

The univariate marginal distribution algorithm (UMDA, Algorithm 14) is one of the simplest representatives of the family of so-called estimation of distribution algorithms (EDAs). The algorithm maintains a population of size s (we use $s = 50$ in our experiments) and uses the best $s/2$ of these to estimate the marginal distribution of each decision variable, by simply counting the relative frequency of ones in the corresponding position. These frequencies are capped at $1/n$ and $1 - 1/n$, respectively. In the t-th iteration, a new population is created by sampling from these marginal distributions. Building upon previous work made in [Mühlenbein and Paaß(1996)], the UMDA was introduced in [Mühlenbein(1997)]. Theoretical results for this algorithm are summarized in [Krejca and Witt(2018)].

Algorithm 14: The Univariate Marginal Distribution Algorithm (UMDA), representing the family of EDAs

Initialization:
  for $i = 1, \ldots, s$ **do** sample $x^{(0,i)} \in \{0,1\}^n$ uniformly at random and evaluate $f(x^{(0,i)})$;
  Let $\vec{P_0}$ be the collection of the best $s/2$ of these search points, ties broken uniformly at random (u.a.r.);
Optimization:
  for $t = 1, 2, 3, \ldots$ **do**
    for $j = 1, \ldots, n$ **do**
      $p_j \leftarrow 2|\{x \in \vec{P_{t-1}} \mid x_j = 1\}|/s$;
      if $p_j < 1/n$ **then** $p_j = 1/n$;
      if $p_j > 1 - 1/n$ **then** $p_j = 1 - 1/n$;
    for $i = 1, \ldots, s$ **do** sample $x^{(t,i)} \in \{0,1\}^n$ by setting, independently for all $j \in [n]$, $x_j^{(t,i)} = 1$ with probability p_j and setting $x_j^{(t,i)} = 0$ otherwise. Evaluate $f(x^{(t,i)})$;
    Let $\vec{P_t}$ be the collection of the best $s/2$ of the points $x^{(t,1)}, x^{(t,2)}, ..., x^{(t,s)}$, ties broken u.a.r.;

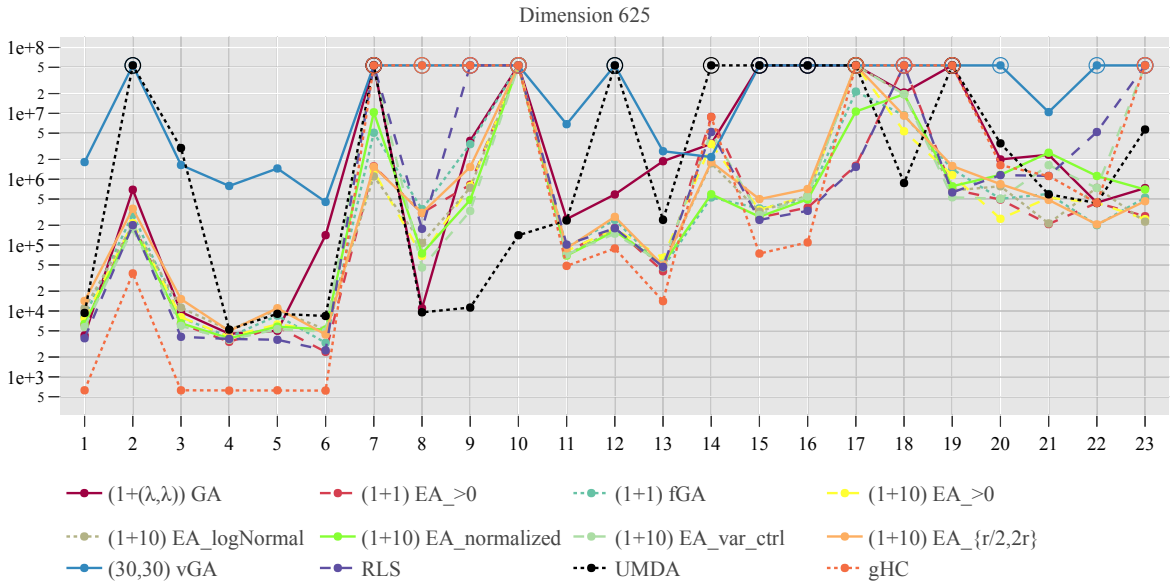

Figure 4: ERT values of the twelve baseline algorithms for the 625-dimensional test suite, with respect to the best solution quality found by any of the algorithms in any of the eleven runs. These target values can be found in Table 1.

5 Experimental Results

As a demonstration of the results that can be obtained from the IOHPROFILER environment, we report here on some basic experiments that were run on it. All data is available for interactive evaluation with IOHANALYZER at http://iohprofiler.liacs.nl (the data sets can be loaded by selecting the data set 2019gecco-inst11-1run in the "Load Data from Repository" section on the "upload data" page. The data set with 11 runs on the first instance is available as data set 2019gecco-inst1-11run). In addition to analyzing the performance statistics of the baseline algorithms described above, the user can upload his/her own data sets for a performance comparison against the twelve algorithms, or for an extension of this work to more benchmark problems.

Following the primary scope of this work, which is a demonstration of the ability of IOHPROFILER to handle such large-scale experiments, our report features only the major empirical findings. A more detailed analysis by each algorithm is left for future work. We note, though, that the insights obtained through the here-described experiments have already inspired the development of new algorithmic ideas, see [Ye et al.(2019)Ye, Doerr, and Bäck, Rodionova et al.(2019)Rodionova, Antonov, Buzdalova, and Doerr, Horesh et al.(2019)Horesh, Bäck, and Shir].

5.1 Experimental Setup

Our experimental setup can be summarized as follows:

- 23 test-functions F1-F23, described in Section 3
- Each function is assessed over the four problem dimensions $n \in \{16, 64, 100, 625\}$
- Each algorithm is run on 11 different instances of each of these 92 (F, n) pairs, yielding a total number of 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 $n = 625$. More precisely, each algorithm performs one run on each of the instances $1 - 6$ and $51 - 55$ described in Section 3.2.

  As mentioned in Section 4, most of our algorithms are unbiased and comparison-based. For these algorithms all 11 instances look the same, i.e., performing one run each is equivalent to 11 independent runs on instance 1, which is the "pure" problem instance without fitness scaling nor any other transformation applied to it. However, in order to understand how the transformations impact the behavior of vGA and gHC, we also performed 11 independent runs of each algorithm on instance 1 of each (F, n) pair, yielding another 1,012 runs per each algorithm.

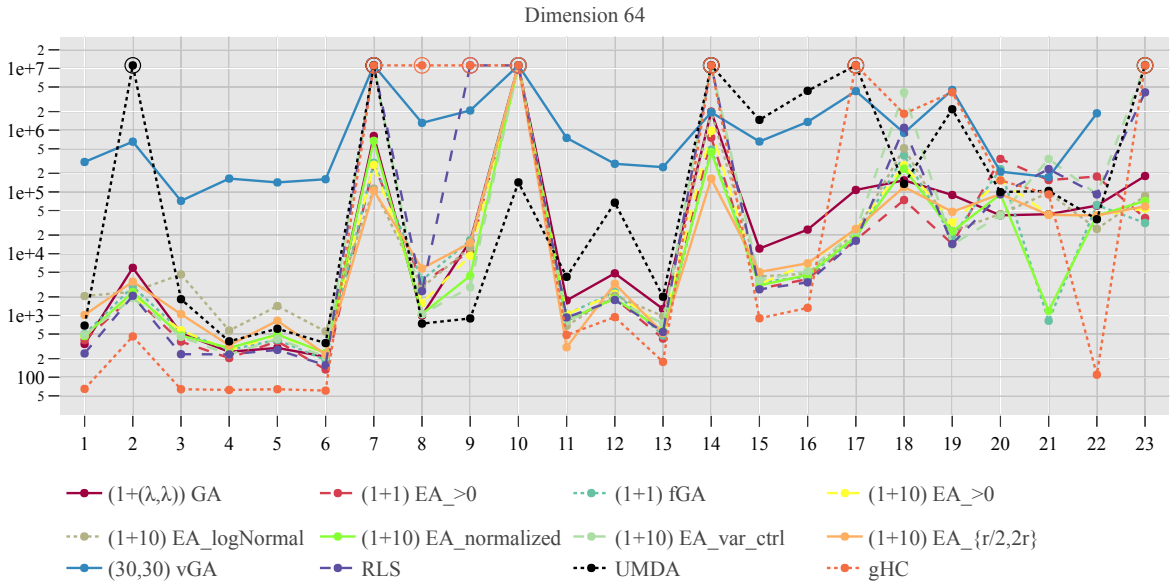

Figure 5: ERT values of the twelve baseline algorithms for the 64-dimensional test suite, with respect to the best solution quality found by any of the algorithms in any of the eleven runs. These target values can be found in Table 1.

- For each run we store the current and the best-so-far function value at each evaluation. This setup allows very detailed analyses, since we can zoom into each range of fixed budgets and/or fixed-targets of choice, and obtain our anytime performance statistics in terms of quantiles, averages, probabilities of success, ECDF curves, etc.

  For some of the algorithms we also store information about the self-adjusting parameters, for example the value of λ in the $(1+(\lambda,\lambda))$ GA and the mutation rates for the (1+10) $\text{EA}_{r/2,2r}$, the (1+10) $\text{EA}_{\text{var.}}$, and the (1+10) $\text{EA}_{\text{norm.}}$. From this data we can derive how the parameters evolve with respect to the time elapsed and with respect to the quality of the best-so-far solutions.

As discussed, we are interested in experimental setups that allow to evaluate one entire experiment (for one algorithm) within **24 CPU hours.** The majority of our tested algorithms completed the experiment in around 12 CPU hours on an Intel(R) Xeon(R) CPU E5-4667 server, and all twelve algorithms finished experimentation within the 24 hours time frame.

Concerning the number of repetitions, we note that with 11 runs we already get a good understanding of the key differences between the algorithms. 11 runs can be enough to get statistical significance, if the differences in performance are substantial. We refer the interested reader to the tutorial [Hansen(2018)], which argues that for a first experiment a small number of experiments can suffice. We also note that IOHANALYZER offers statistical tests, in the form of pairwise Kolmogorov-Smirnov tests. An extension to Bayesian inference statistics is in progress. We do not report on these statistical tests here, but refer the interested reader to our work [Calvo et al.(2019)Calvo, Shir, Ceberio, Doerr, Wang, Bäck, and Lozano], which discusses statistical significance of the results presented here in this work using the framework previously presented in [Calvo et al.(2018)Calvo, Ceberio, and Lozano]. We also note that other statistical tests could make a lot of sense to analyze our data, but this is beyond the scope of this work.

5.2 Performance Measures

We are mostly interested in fixed-target results, i.e., we consider the average "time" (=number of function evaluations) needed by each algorithm to find a solution that is at least as good as a certain threshold value. To be very explicit, Figure 6 is a example showing fixed-target curves, which plot the average number of function evaluations (y-axis) needed to find a solution x satisfying $f(x) \geq \phi$, where the target value ϕ is the value on the x-axis.

Given performance data of r independent runs of an algorithm A with a maximal budget of B function evaluations, the expected running time (ERT) value of A for a target value v is $\frac{r-s}{s}B + \text{AHT}$, 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.

	$F1$	$F2$	$F3$	$F4$	$F5$	$F6$	$F7$	$F8$	$F9$	$F10$	$F11$
$n = 64$	64	64	2080	32	57	21	64	33	64	63.2	32
$n = 625$	625	625	195 625	312	562	208	576.4	314	625	625	312

	$F12$	$F13$	$F14$	$F15$	$F16$	$F17$	$F18$	$F19$	$F20$	$F21$	$F22$	$F23$
$n = 64$	57	21	43.8	33	64	64	3.981492	128	230.4	384	28	8
$n = 625$	562	208	36.6	314	625	625	4.2655266	1 242	2 420	3 532.8	268.4	24

Table 1: Target values for which the ERT curves in Figures 4 and 5 are computed.

Another important concept in the analysis of IOHs are empirical cumulative distribution function (ECDF) curves, which allow to aggregate performance across different functions. For each of the d different problems $i = 1, \ldots, d$ a list of target values $\phi_{i,j}$, $j = 1, \ldots, m(i)$, is chosen. The ECDF value at budget t is defined as the fraction of (run,target)-pairs $(s, \phi_{i,j})$ which satisfy that in run s on problem i a solution has been identified that is at least as good as $\phi_{i,j}$. See [Hansen et al.(2016)Hansen, Auger, Brockhoff, Tusar, and Tušar] for more information and motivation.

5.3 Function-wise Raw Observations Across Dimensions

Figures 4 and 5 depict the ERT of the baseline algorithms on the 625-dimensional and the 64-dimensional functions, respectively, when considering the best function value found by any of the algorithms in any of the runs. These target values are summarized in Table 1.

We summarize a few basic observations for each function.

F1 This baseline ONEMAX problem is easily solved, having the gHC winning (it solves each n-dimensional ONEMAX instance in at most $n+1$ queries), the majority of the algorithms clustered with a practically-equivalent performance, the (1+10)-EA$_{r/2,2r}$ lagging behind, and the vGA outperformed by far. All algorithms locate the global optimum eventually. Figure 6 presents the average fixed-target performance of the algorithms on F1 at $n = 625$, in terms of ERT. Evidently, the vGA and the UMDA obtain a clear advantage in the beginning of the optimization process, although the vGA eventually uses the largest number of evaluations, by far, to locate the optimum. We also see here that, as expected, the performances of the unbiased algorithms (i.e., all algorithms except the vGA) are identical for the 11 runs on instance 1 and the 1 run on 11 different instances. For the vGA this is clearly not the case, the fixed-target performances of these two settings differ substantially.

F2 The LEADINGONES problem introduces more difficulty when compared to F1, with the ERT consistently shifting upward, but it is still easily solved. The gHC wins, the vGA loses, and the majority of the algorithms are again clustered, but now the $(1 + (\lambda, \lambda))$-GA lags behind. The UMDA fails to find the optimum within the given time budget, for all tested dimensions except for $n = 16$. An example of the evolution of the parameter λ in the $(1 + (\lambda, \lambda))$ GA is visualized in Figure 7. We observe – as expected – that larger function values are evidently correlated with larger population sizes (and, thus, larger mutation rates).

F3 The behavior on this problem, the linear function with harmonic weights, is similar to F1 for most algorithms. Exceptions are the vGA, for which it is slightly easier, and the UMDA, which shows worse performance on F3 than on F1.

F4 This problem, ONEMAX with 50% dummy variables, is the most easily-solved problem in the suite, with an even simpler performance classification – the gHC performs at the top, the vGA at the bottom, and the rest are tightly clustered. Given the consistent correlation with the F1 performance profiles, across all twelve algorithms, it seems debatable whether or not to keep this function in a benchmark suite, since it seems to offer only limited additional insights, which could be of a rather specialized interest, e.g., for theoretical investigations addressing precise running times of the algorithms.

F5 Solving this problem, ONEMAX with 10% dummy variables, exhibits equivalent behavior to F1. Similarly to F4, we suggest to ignore this setup for future benchmarking activities. Note, however, that the exclusion of F4 and F5 does *not* imply that the dummy variables do not play an interesting role – in an ongoing evaluation of the W-model, we are currently investigating their impact when combined with other W-model transformations.

F6 The neutrality ("majority vote") transformation apparently introduces difficulty to the $(1 + (\lambda, \lambda))$ GA, which exhibits deteriorated performance compared to F1. The vGA, despite a slightly better performance compared to F1, is the worst among the twelve algorithms. At the same time, the (1+10)-EA$_{\text{log-n.}}$ lags behind its competitors in the beginning, but it eventually shows a competitive result in the later optimization process, ending up with an overall fine ERT value. The gHC outperforms all other algorithms also on this function.

F7 The introduction of local permutations to ONEMAX, within the current problem, introduced difficulties to all the algorithms. The ability to locate the global optimum within the designated budget deteriorated for all of them,

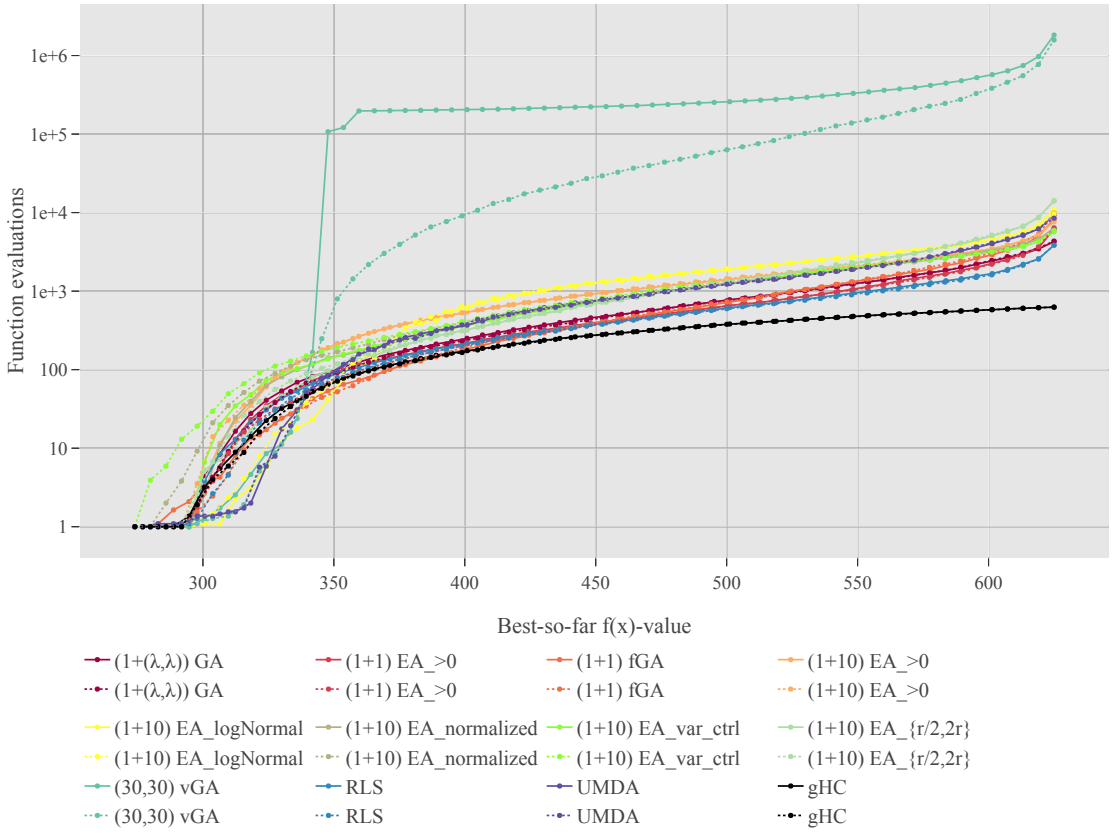

Figure 6: ERT values for F1 (ONEMAX) at dimension $n = 625$ in a fixed-target perspective. The dashed lines are the average running times of 11 independent runs on instance 1, while the solid lines are average running times for one run on each of the eleven different instances 1-6 and 51-55.

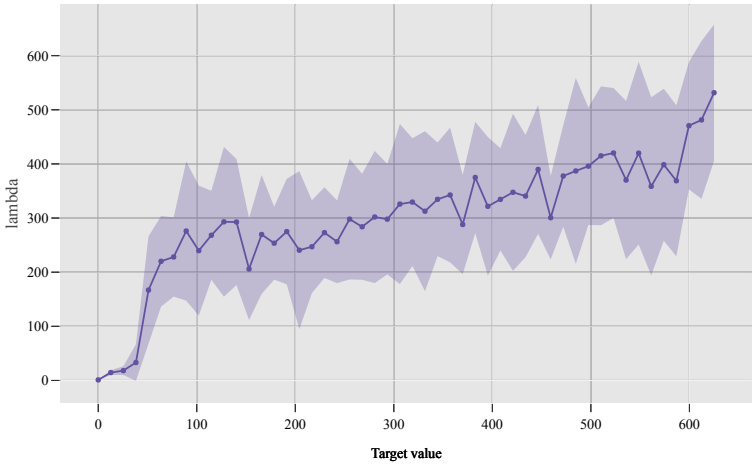

Figure 7: Evolution of the population size λ of the $(1 + (\lambda, \lambda))$ GA on the LEADINGONES problem F2 at dimension n = 625, correlated to the best-so-far objective function values (horizontal axis). The line shows the average value of λ for iterations starting with a best-so-far solution of the value indicated by the x-axis. The shade represents the standard deviation.

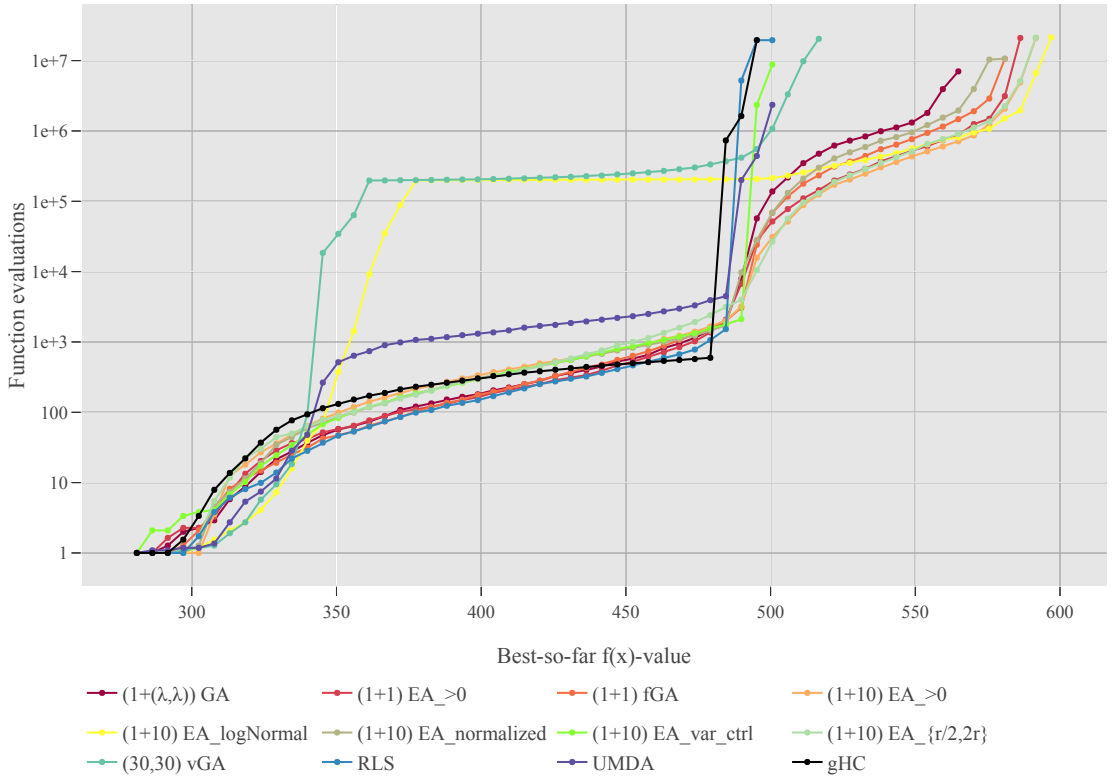

Figure 8: ERT values for F7 at dimension $n = 625$ in a fixed-target perspective.

except for the (1+10)-EA$_{r/2,2r}$ on "low-dimensional" scales ($n \in \{16, 64, 100\}$). Figure 8 depicts the ERT values of the algorithms on F7 at $n = 625$, where it is evident that they all failed to locate the global optimum. Note that this figure encompasses results for both instantiations (a single instance or 11 instances). The twelve algorithms' performances are clustered in two groups that are associated with two fitness regions (and likely two basins of attraction) - the first around an objective function value of 500 (including the UMDA, with the gHC being the fastest to approach it and get stuck), and the other below 600. It seems that the latter cluster could use additional budget to further improve the results.

F8 Being ONEMAX with the small fitness plateaus induced by the ruggedness function r_1, the UMDA performs best on this problem, with the $(1 + (\lambda, \lambda))$ GA following very closely. It seems to introduce medium difficulty to all the algorithms, except for the gHC, whose performance is dramatically hampered and becomes worse than the vGA. Interestingly, the ERT values are distributed sparsely compared to other ONEMAX variants.

F9 The UMDA also performs best for this problem, with the (1+10) EA$_{\text{var.}}$ being the runner-up. Generally, the behavior on this problem, ONEMAX with small fitness perturbations, is close to F8, but with certain differences. It is evidently harder, as the algorithms experience larger ERT values. Importantly, unlike F8, the RLS always fails on F9 (since it gets stuck in local optima), and "joins" the gHC and vGA at the bottom of the performance table. The $(1 + (\lambda, \lambda))$ GA also shows worse performance on F9 than on F8.

F10 This problem, ONEMAX with fitness perturbations of block size five, presents a dramatic difficulty to all the algorithms, including the UMDA, which, however, clearly outperforms all other algorithms. It is evidently the hardest ONEMAX variant for all the tested algorithms, among the eight variants studied in this work. For $n = 625$ the UMDA finds the optimum after an average of 141 243 evaluations, while none of the other algorithms finds a solution better than 575.

F11 The gHC performs strongly on this problem, namely the LEADINGONES with 50% dummy variables, consistently with its winning behavior on F2. The vGA performs poorly, and the UMDA is also at the bottom of the table. Notably, the problem should become easier compared to F2, since the effective number of variables is reduced. The RLS, however, which generally performs well on LEADINGONES, only ranks third from the bottom on ERT values when solving this problem.

F12 The behavior of the algorithms on this problem, LEADINGONES with 10% dummy variables, is very similar to F11, with excellent performance of the gHC. However, one major difference is the dramatic deterioration of UMDA and vGA, which fail to find the optimum with given time budget for $n = 625$ (see, Figure 4). UMDA performs better than vGA for $n = 64$, but still obtains clear disadvantage comparing to other algorithms (see Figure 5).

F13 The introduction of neutrality on LEADINGONES makes this problem easier in practice (that is, by observing ERT decrease compared to F2). The gHC wins, while the vGA, $(1+(\lambda, \lambda))$ GA as well as the UMDA lag behind the other methods. The poor performance of these three algorithms is consistent with their performance on LEADINGONES.

F14 Being LEADINGONES with epistasis, this problem introduces high difficulty. For high dimensions, $n \in \{64, 100, 625\}$, none of the algorithms was capable of locating an optimal solution within the allocated budget. The vGA tops the ERT values on this problem, followed by the (1+1)-fGA, the (1+10)-EA$_{r/2,2r}$, and the (1+10)-EA$_{\text{norm.}}$. On the other hand, three algorithms, namely the gHC, the UMDA, and the RLS, seem to get trapped with low objective function values.

F15 The introduction of fitness perturbations to LEADINGONES makes this problem difficult. The UMDA exhibits the worst performance among the competing methods. The remainder of the algorithms, except for the vGA and the $(1 + (\lambda, \lambda))$ GA, are still able to hit the optimum of this problem, but with significantly larger ERT values. The gHC performs best, and the first runner-up is the RLS.

F16 The obtained ranks of algorithms, with respect to the ERT values, are similar to those of F15, but generally exhibit higher ERT values. Notably, the UMDA is still the worst performer.

F17 As expected, the rugged LEADINGONES function is the second-hardest among the LEADINGONES variants, following F14. Only the RLS and the (1+1)-EA are able to hit the optimum in dimension 625, while the gHC has a diminished performance on this problem. This can be explained by the fact that the gHC has a very high probability of getting stuck in a local traps, while the RLS is capable of performing random walks about local optima, until eventually escaping them (e.g., by flipping the right bit when all the consecutive four bits are also identical to the target string). This is of course a rare event, and the ERT values are therefore significantly worse than all other LEADINGONES variants, except F14. As on the previous two functions, the UMDA performs poorly, with similar ERT values as the gHC.

Comparing to F10, the effect of the fitness permutation r_3 on LEADINGONES is not as significant as on ONEMAX, which can be explained by the ability of most of the algorithms to perform random walks about local traps, through which the four first bits of the tail are eventually set correctly, at which point flipping the significant bit (i.e., the bit in position $\text{LO}(x) + 1$) results in a LEADINGONES fitness increase of at least five, and consequently a fitness increase of at least one for the problem $r_3 \circ$ LEADINGONES. This candidate solution is thus accepted by all of our algorithms, and the next phase of optimizing the following consecutive five bits begins.

F18 The LABS problem is the most complex problem in our assessment. For the higher dimensions, $n \in \{64, 100, 625\}$, none of the algorithms obtained the maximally attainable values, or got fitness values close to those of the best-known sequences (see, e.g., [Packebusch and Mertens(2016)]). Additionally, a couple of algorithms (e.g., the gHC and the RLS) did not succeed to escape low-quality "local traps" on most dimensions. Surprisingly, the vGA was superior to the other algorithms at $n = 16$ but, as expected, over the higher dimensions presented weaker performance. Notably, the UMDA outperforms the other methods at $n = 625$.

F19 The simplest problem among the Ising instances. Most of the algorithms exhibited similar performance, except for the vGA, the UMDA and the gHC, which obtained weak results. The latter preformed worst among all algorithms, and obtained the lowest objective function values across all the dimensions for the given time budget. As a demonstration of the performance statistics that IOHPROFILER provides, average fixed-target and fixed-budget running times are provided in Figure 9. This figure illustrates that ERT values tell only one side of the story: the performance of UMDA is comparable to that of the other algorithms for all targets up to around 170; only then it starts to perform considerably worse.

F20 In contrast to its poor performance on the 1D-Ising (F19), the gHC outperformed the other algorithms on the 2D-Ising for target values up to around 2,300 ($d = 625$), after which its performance becomes worse than most of the other algorithms, except for the vGA, which is consistently the worst except for a few initial target values. For the 16-dimensional F20 problem, gHC performs best for all recorded target values. For $d = 625$, however, the (1+10) EA achieves the best ERT value for the target recorded in Table 1, followed by the (1+1) EA, the (1+10) EA$_{\text{var.}}$, and the fGA.

F21 As expected, the most complex among the Ising model instances. The observed performances resemble the observations on F20, whereas the vGA was among the slowest and the gHC was among the fastest. In addition, the RLS preformed poorly and obtained worse ERT values compared to the other algorithms (apart from the vGA, which performs even worse). The UMDA improved its ranking compared to F19, exhibiting performance close to the vGA. For $d = 625$, the best ERT is obtained by the (1+10) EA$_{\text{log-n.}}$.

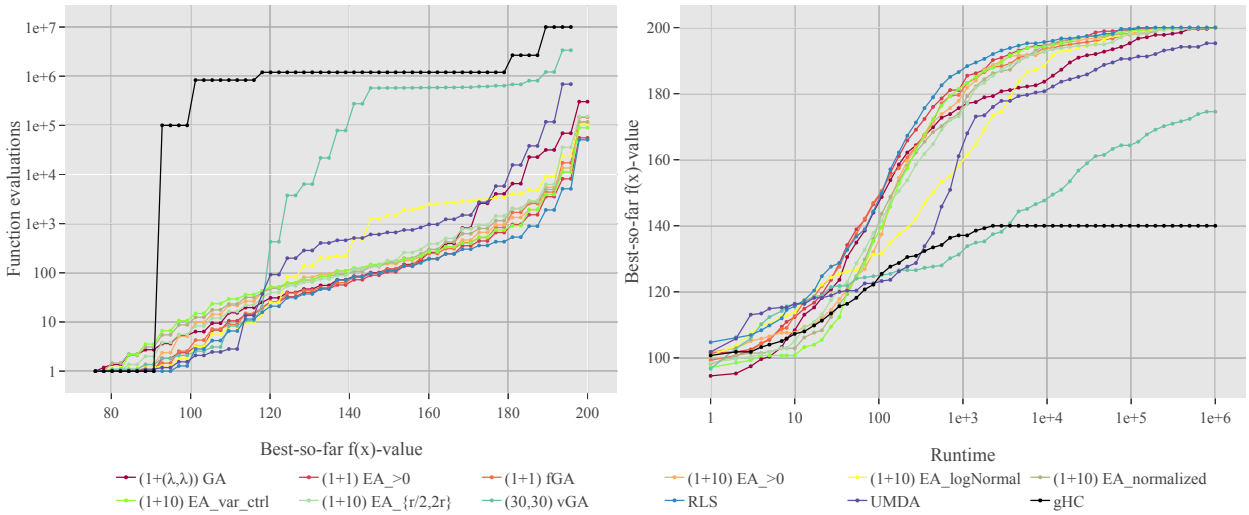

Figure 9: Demonstration of the basic performance plots for F19 at dimension $n = 100$: [LEFT] best obtained values as a function of evaluations calls ("fixed-target perspective"), versus [RIGHT] evaluations calls as a function of best obtained values ("fixed-budget perspective"). For F19, these patterns of relative behavior are observed across all dimensions.

F22 None of the algorithms succeeded in locating the global optimum across all dimensions of this problem. This is explained by the existence of a local optimum with a strong basin of attraction. The gHC and the vGA exhibited inferior performance compared to the other algorithms.

F23 Some algorithms failed to locate the global optimum of the N-Queens problem in high dimensions, yet the vGA, the gHC and the UMDA constantly possessed the worst ERT values. Fine performance was observed for the (1+10)-EA$_{>0}$ and the (1+10) EA$_{\text{log-n.}}$.

5.4 Grouping of Functions and Algorithms

In this section we are aiming to recognize patterns and identify classes within (i) the set of all functions, and (ii) the set of all algorithms. Our analyses are based on human experts' observations, alongside automated clustering of the mean ERT vectors using *K-means* [Hastie et al.(2013)Hastie, Tibshirani, and Friedman].

Functions' Empirical Grouping It is evident that problems F1-F6, F11-F13 and F15-F16 are treated relatively easily by the majority of the algorithms, with those functions based on LEADINGONES (i.e., F2, F11-13, F15, F16) being more challenging within this group. On the other extreme, F7, F9-F10, F14, F18-F19 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. F8, 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 performs very poorly 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 performs very well on F1-F6, while having difficulties on F7-F10. Clearly, RLS also gets trapped by the deceptive functions, while at the same time it shows fine performance on most of the non-deceptive problems. The UMDA's performance stands out, and also appears as an independent cluster in the automated K-Means analysis. Evidently, it performs well on the ONEMAX-based problems, but fails to optimize the LEADINGONES function F2 and its derivatives F11-F17, with the exception of F11 and F13 – a behavior that might be interesting to analyze further in future work. 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. In terms of ERT values, the (1+10)-EA$_{\text{log-n.}}$ could also be grouped into this class of seven algorithms, but it behaves quite differently during the optimization process, often showing an opposite trend of convergence speed at the early stages of the optimization procedure.

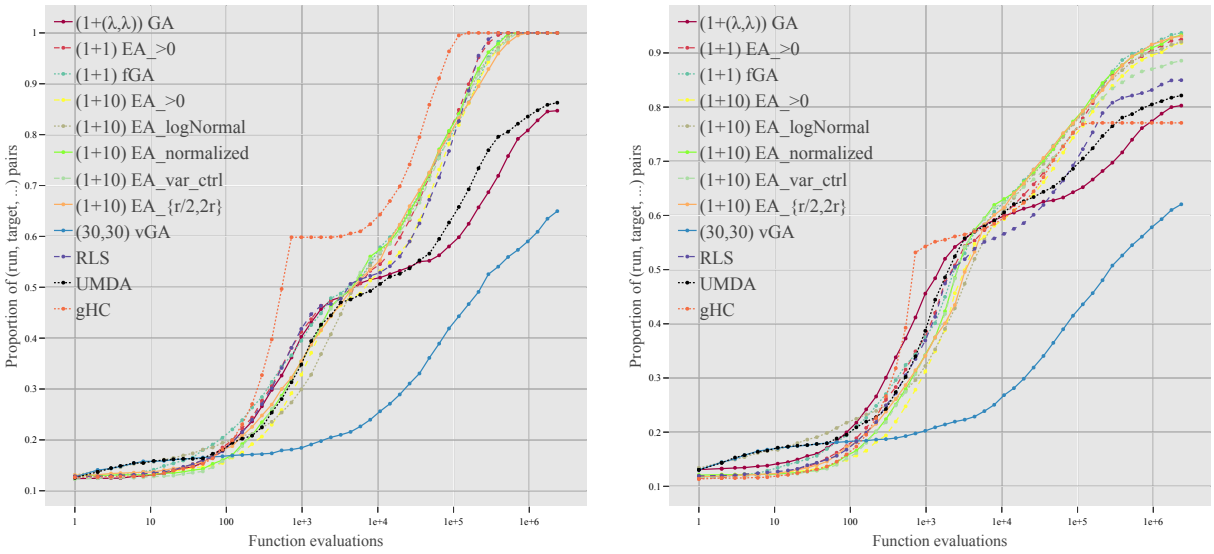

Figure 10: ECDF curve for the class of "easily-solved" functions in dimension $n = 625$: F1-F6, F11-F13, and F15-F16 [LEFT] and of all 23 functions [RIGHT], with respect to equally spaced target values.

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 ($n \in \{16,\ 64\}$), with (1+1)-fGA and (1+10)-EA$_{\text{norm.}}$ being together the first runner-up. The (1+10)-EA also exhibits high ranking across all dimensions. (1+10)-EA$_{\text{norm.}}$ leads the grouped hitting numbers on $n = 100$, whereas the (1+1)-EA leads the hitting numbers on the "high-dimensional" functions at $n = 625$, with (1+10)-EA being the runner-up. Across all dimensions, UMDA, gHC and vGA are with the lowest rankings.

Visual Analytics As a demonstration of the performance statistics offered by IOHPROFILER, we provide snapshots of visual analytics that supported our examination. Figure 9 depicts basic performance plots for F19 at dimension $n = 100$, in so-called *fixed-target* and *fixed-budget* perspectives. For clarity of the plots we only show the ERT values and the average function values achieved per each budget, respectively. Standard deviations as well as the $2, 5, 10, 15, 50, 75, 90, 95, 98\%$ quantiles are available on http://iohprofiler.liacs.nl/.

In Figure 10 we provide two plots obtained from our new module which computes ECDF curves for user-specified target values. The plot on the left depicts an ECDF curve for the "easily-solved" functions identified above (i.e., F1-F6, F11-F13, and F15-F16) in dimension $n = 625$. The one on the right shows the ECDF curves across all 23 benchmark functions. For both figures we have chosen ten equally spaced target values per each function, with the largest value being again the best function value identified by any of the algorithms in any of the runs. Since the number of "easy" problems dominates our overall assessment the curves on the right are to a large extent dominated by the performances depicted on the left. This indicates once again the need for a thorough revision of our benchmark selection.

5.5 Unbiasedness

Following our experimental planning to test the hypothesized "biasedness" effect for the vGA, we compared its averaged performance on instance 1 versus on all the other instances (1-6 and 51-55) altogether. Figure 11 depicts a comparison of attained objective function values, by means of box-plots, on F1 and on F2 for $n = 64$. Performance deterioration is indeed evident on the permuted instances; that is, instances 51-55, for which the base functions are composed with a σ-transformation of the bit strings, as described in Section 3.2. The box-plots in Figure 11 show very clearly that the vGA treats the plain F1 and F2 much better, in terms of attained target values, than their transformed variants. The plots are for $n = 64$ and after exhausting the full budget of $100n^2$ function evaluations.

5.6 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

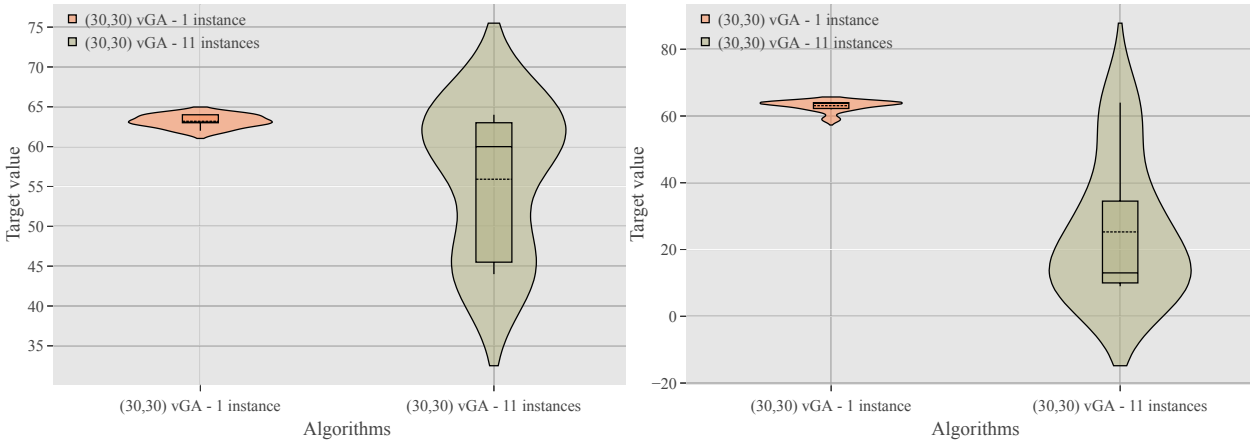

Figure 11: Statistical box-plots for vGA's attained function values on instance 1 alone versus on the eleven different instances 1-6 and 51-55, after exploiting the entire budget (namely, 409 600 function evaluations): F1 [LEFT] and F2 [RIGHT]. Both plots are for $n = 64$.

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. However, we have also observed that some of the components of the W-model, in particular the introduction of dummy variables, might be less interesting as "stand-alone"-effects. A revised selection of the W-model functions is currently under investigation.

Finally, we note that the hardness of all functions consistently increase with their problem dimension – exhibiting a desirable property that we mentioned in Section 3.1.

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 are continuously strengthening its statistical repertoire by introducing new performance measures any by devising better procedures. 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. To shed better light on the tradeoff with respect to time, quality, and robustness, we are also considering to add an automated computation of the empirical attainment function, as available from [Fonseca et al.(2011)Fonseca, Guerreiro, López-Ibáñez, and Paquete]. We are furthermore looking into the option of adding a variant of the so-called performance profiles [Moré and Wild(2009)] for comparison across multiple dimensions. In terms of statistical tests, two-sample nonparametric tests (e.g., Mann–Whitney U or Kolmogorov–Smirnov test) can be applied pairwise among algorithms with corrections (e.g., Holm-Bonferroni correction). The Kolmogorov-Smirnov tests are available in IOHANALYZER, while extension, in particular in terms of nonparametric tests that are designed for two or more samples, e.g., Kruskal–Wallis or Friedman test, are left for future work. For the large-scale multiple testing scenario (thousands of pairwise tests are performed, for instance), we are considering to add Bayesian inference, as suggested, e.g., in [Benavoli et al.(2017)Benavoli, Corani, Demsar, and Zaffalon]. Such an approach has recently been proposed for comparing performance of IOHs [Calvo et al.(2018)Calvo, Ceberio, and Lozano] and has been applied to the data set of this paper in [Calvo et al.(2019)Calvo, Shir, Ceberio, Doerr, Wang, Bäck, and Lozano].

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 [Kerschke et al.(2018)Kerschke, Hoos, Neumann, and Trautmann], which use landscape features and performance data to build a model that predicts how the tested algorithms will perform on a previously *unseen* 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 optimiza-

tion [Belkhir et al.(2017)Belkhir, Dréo, Savéant, and Schoenauer]. 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 has been defined and can be computed with the flacco package [Kerschke and Trautmann(2016)], 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 [Thomson et al.(2018a)Thomson, Vérel, Ochoa, Veerapen, and McMenemy, Thomson et al.(2018b)Thomson, Vérel, Ochoa, Veerapen, and Cairns] 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 develop a more systematic approach towards the identification of 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 be dynamic, with the possibility to add new problems, so that users can focus their experimentation on problems relevant to their work. IOHPROFILER is built 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 research line that we are currently addressing in a parallel work stream.

Integration of Algorithm Design Software: IOHs are to a large extent modular algorithms, whose components can be exchanged and executed in various different ways. This has let the community to develop software which enables an easier algorithmic design. Examples for such software are ParadisEO [Cahon et al.(2004)Cahon, Melab, and Talbi] for single-objective and multi-objective optimization and jMetal [Durillo and Nebro(2011)] for multi-objective algorithms. Building or integrating such software could allow much more comprehensive algorithm benchmarking, and could eventually automate the detection of promising algorithmic variants. In the continuous domains, the modular CMA-ES framework [van Rijn et al.(2016)van Rijn, Wang, van Leeuwen, and Bäck] can be seen as a proof of concept for this idea: it was shown there that some of the automatically generated CMA-ES variants could improve upon existing human-designed algorithms. In a similar direction, we are also planning to ease parallelization of IOHEXPERIMENTER, so that parametric algorithms can be batch-tested for various configurations.

Acknowledgments

We are very grateful to our colleagues Arina Buzdalova and Maxim Buzdalov (both at ITMO University, St. Petersburg, Russia), Johann Dreo (Thales Research), Michal Horovitz and Mordo Shalom (both at Tel-Hai College, Israel), Dirk Sudholt (Sheffield, UK), and Thomas Weise (Hefei University, China) for valuable discussions around different aspects of benchmarking IOHs. We have very much appreciated the detailed comments of the anonymous reviewers of this paper, which have helped us improve the presentation of our contribution. In particular, the suggestion to add an EDA to the set of baseline algorithms was made by one of the reviewers.

Our work was supported by the Chinese scholarship council (CSC No. 201706310143), a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH, in a joint call with the Gaspard Monge Program for optimization, operations research, and their interactions with data sciences, by Paris Ile-de-France Region, and by COST Action CA15140 "Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO)".

References

[Kerschke et al.(2018)Kerschke, Hoos, Neumann, and Trautmann] P. Kerschke, H. H. Hoos, F. Neumann, H. Trautmann, Automated algorithm selection: Survey and perspectives, CoRR abs/1811.11597 (2018). URL: http://arxiv.org/abs/1811.11597. arXiv:1811.11597.

[Wasik et al.(2016)Wasik, Antczak, Badura, Laskowski, and Sternal] S. Wasik, M. Antczak, J. Badura, A. Laskowski, T. Sternal, 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, 2016, pp. 433–436. URL: https://doi.org/10.1145/2818052.2869098. doi:10.1145/2818052.2869098.

[Weise(2016)] T. Weise, Optimization benchmarking, 2016. Available at http://optimizationbenchmarking.github.io/.

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

[Hansen et al.(2010)Hansen, Auger, Ros, Finck, and Pošík] N. Hansen, A. Auger, R. Ros, S. Finck, P. Pošík, 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, 2010, pp. 1689–1696. URL: http://doi.acm.org/10.1145/1830761.1830790. doi:10.1145/1830761.1830790.

[Hansen et al.(2016)Hansen, Auger, Mersmann, Tušar, and Brockhoff] N. Hansen, A. Auger, O. Mersmann, T. Tušar, D. Brockhoff, COCO: A platform for comparing continuous optimizers in a black-box setting, CoRR abs/1603.08785 (2016). URL: http://arxiv.org/abs/1603.08785. arXiv:1603.08785.

[Tusar et al.(2016)Tusar, Brockhoff, Hansen, and Auger] T. Tusar, D. Brockhoff, N. Hansen, A. Auger, COCO: the bi-objective black box optimization benchmarking (bbob-biobj) test suite, CoRR abs/1604.00359 (2016). URL: http://arxiv.org/abs/1604.00359.

[Tusar et al.(2019)Tusar, Brockhoff, and Hansen] T. Tusar, D. Brockhoff, N. Hansen, Mixed-integer benchmark problems for single- and bi-objective optimization, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'19), ACM, 2019, pp. 718–726. URL: https://doi.org/10.1145/3321707.3321868. doi:10.1145/3321707.3321868.

[Doerr et al.(2018)Doerr, Wang, Ye, van Rijn, and Bäck] C. Doerr, H. Wang, F. Ye, S. van Rijn, T. Bäck, IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics, arXiv e-prints:1810.05281 (2018). arXiv:1810.05281, available at https://arxiv.org/abs/1810.05281.

[Doerr et al.(2019a)Doerr, Ye, Horesh, Wang, Shir, and Bäck] C. Doerr, F. Ye, N. Horesh, H. Wang, O. M. Shir, T. Bäck, Benchmarking discrete optimization heuristics with IOHprofiler, in: Companion Material of Proc. of Genetic and Evolutionary Computation Conference (GECCO'19), ACM, 2019a, pp. 1798–1806. URL: https://doi.org/10.1145/3319619.3326810. doi:10.1145/3319619.3326810.

[Doerr et al.(2019b)Doerr, Wang, Ye, van Rijn, and Bäck] C. Doerr, H. Wang, F. Ye, S. van Rijn, T. Bäck, Github page of IOHprofiler data sets, 2019b. Available at https://github.com/IOHprofiler.

[Shir et al.(2018)Shir, Doerr, and Bäck] O. M. Shir, C. Doerr, T. Bäck, 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, 2018, pp. 1753–1760. URL: https://doi.org/10.1145/3205651.3208251. doi:10.1145/3205651.3208251.

[Mersmann et al.(2011)Mersmann, Bischl, Trautmann, Preuss, Weihs, and Rudolph] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, G. Rudolph, Exploratory landscape analysis, in: Proc. of Genetic and Evolutionary Computation (GECCO'11), ACM, 2011, pp. 829–836. URL: http://doi.acm.org/10.1145/2001576.2001690. doi:10.1145/2001576.2001690.

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

[Dubhashi and Panconesi(2009)] D. P. Dubhashi, A. Panconesi, Concentration of Measure for the Analysis of Randomised Algorithms, Cambridge University Press, 2009.

[Garnier et al.(1999)Garnier, Kallel, and Schoenauer] J. Garnier, L. Kallel, M. Schoenauer, Rigorous hitting times for binary mutations, Evolutionary Computation 7 (1999) 173–203.

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

[Doerr and Doerr(2018)] B. Doerr, C. Doerr, Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm, Algorithmica 80 (2018) 1658–1709.

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

[Doerr et al.(2016)Doerr, Doerr, and Yang] B. Doerr, C. Doerr, J. Yang, Optimal parameter choices via precise black-box analysis, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'16), ACM, 2016, pp. 1123–1130.

[Doerr et al.(2018)Doerr, Ye, van Rijn, Wang, and Bäck] C. Doerr, F. Ye, S. van Rijn, H. Wang, T. Bäck, 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, 2018, pp. 951–958. URL: https://doi.org/10.1145/3205455.3205621. doi:10.1145/3205455.3205621.

[Doerr(2018)] C. Doerr, Complexity theory for discrete black-box optimization heuristics, CoRR abs/1801.02037 (2018). arXiv:1801.02037, available at http://arxiv.org/abs/1801.02037.

[Thierens(2009)] D. Thierens, On benchmark properties for adaptive operator selection, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'09), ACM, 2009, pp. 2217–2218. URL: https://doi.org/10.1145/1570256.1570306. doi:10.1145/1570256.1570306.

[Doerr(2018)] B. Doerr, Better runtime guarantees via stochastic domination, in: Proc. of Evolutionary Computation in Combinatorial Optimization (EvoCOP'18), volume 10782 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 1–17. URL: https://doi.org/10.1007/978-3-319-77449-7_1. doi:10.1007/978-3-319-77449-7_1, full version available at http://arxiv.org/abs/1801.04487.

[Doerr and Lengler(2018)] C. Doerr, J. Lengler, The (1+1) elitist black-box complexity of LeadingOnes, Algorithmica 80 (2018) 1579–1603. URL: https://doi.org/10.1007/s00453-017-0304-6.

[Afshani et al.(2019)Afshani, Agrawal, Doerr, Doerr, Larsen, and Mehlhorn] P. Afshani, M. Agrawal, B. Doerr, C. Doerr, K. G. Larsen, K. Mehlhorn, The query complexity of a permutation-based variant of mastermind, Discrete Applied Mathematics (2019). doi:10.1016/j.dam.2019.01.007, in press.

[Doerr and Winzen(2012)] B. Doerr, C. Winzen, Black-box complexity: Breaking the $O(n \log n)$ barrier of LeadingOnes, in: Artificial Evolution (EA'11), Revised Selected Papers, volume 7401 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 205–216.

[Doerr et al.(2011)Doerr, Johannsen, Kötzing, Lehre, Wagner, and Winzen] B. Doerr, D. Johannsen, T. Kötzing, P. K. Lehre, M. Wagner, C. Winzen, Faster black-box algorithms through higher arity operators, in: Proc. of Foundations of Genetic Algorithms (FOGA'11), ACM, 2011, pp. 163–172.

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

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

[Weise and Wu(2018)] T. Weise, Z. Wu, 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, 2018, pp. 1769–1776. doi:10.1145/3205651.3208240.

[Weise(2018)] T. Weise, The W-Model, a tunable black-box discrete optimization benchmarking (BB-DOB) problem, implemented for the BB-DOB@GECCO workshop, 2018. Data is available at https://github.com/thomasWeise/BBDOB_W_Model.

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

[Einarsson et al.(2018)Einarsson, Lengler, Gauy, Meier, Mujika, Steger, and Weissenberger] H. Einarsson, J. Lengler, M. M. Gauy, F. Meier, A. Mujika, A. Steger, F. Weissenberger, The linear hidden subset problem for the (1 + 1) EA with scheduled and adaptive mutation rates, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'18), ACM, 2018, pp. 1491–1498. URL: https://doi.org/10.1145/3205455.3205519. doi:10.1145/3205455.3205519.

[Doerr et al.(2017)Doerr, Doerr, and Kötzing] B. Doerr, C. Doerr, T. Kötzing, Unknown solution length problems with no asymptotically optimal run time, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'17), ACM, 2017, pp. 1367–1374. URL: http://doi.acm.org/10.1145/3071178.3071233.

[Shapiro et al.(1968)Shapiro, Pettengill, Ash, Stone, Smith, Ingalls, and Brockelman] I. I. Shapiro, G. H. Pettengill, M. E. Ash, M. L. Stone, W. B. Smith, R. P. Ingalls, R. A. Brockelman, Fourth test of general relativity: Preliminary results, Phys. Rev. Lett. 20 (1968) 1265–1269. doi:10.1103/PhysRevLett.20.1265.

[Pasha et al.(2000)Pasha, Moharir, and Rao] I. A. Pasha, P. S. Moharir, N. S. Rao, Bi-alphabetic pulse compression radar signal design, Sadhana 25 (2000) 481–488. doi:10.1007/BF02703629.

[Militzer et al.(1998)Militzer, Zamparelli, and Beule] B. Militzer, M. Zamparelli, D. Beule, Evolutionary search for low autocorrelated binary sequences, IEEE Transactions on Evolutionary Computation 2 (1998) 34–39. doi:10.1109/4235.728212.

[Packebusch and Mertens(2016)] T. Packebusch, S. Mertens, Low autocorrelation binary sequences, Journal of Physics A: Mathematical and Theoretical 49 (2016) 165001.

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

[Lucas(2014)] A. Lucas, Ising formulations of many NP problems, Frontiers in Physics 2 (2014) 5. doi:10.3389/fphy.2014.00005. arXiv:1302.5843.

[Briest et al.(2004)Briest, Brockhoff, Degener, Englert, Gunia, Heering, Jansen, Leifhelm, Plociennik, Röglin et al.] P. Briest, D. Brockhoff, B. Degener, M. Englert, C. Gunia, O. Heering, T. Jansen, M. Leifhelm, K. Plociennik, H. Röglin, et al., The Ising model: Simple evolutionary algorithms as adaptation schemes, in: Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference, Birmingham, UK, September 18-22, 2004, Proceedings, 2004, pp. 31–40. URL: https://doi.org/10.1007/978-3-540-30217-9_4. doi:10.1007/978-3-540-30217-9_4.

[Fischer and Wegener(2005)] S. Fischer, I. Wegener, The one-dimensional Ising model: Mutation versus recombination, Theoretical Computer Science 344 (2005) 208–225.

[Sudholt(2005)] D. Sudholt, Crossover is provably essential for the Ising model on trees, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'05), 2005, pp. 1161–1167.

[Mellor(2011)] V. Mellor, Numerical simulations of the Ising model on the union jack lattice, arXiv 1101.5015 (2011). Available at https://arxiv.org/abs/1101.5015.

[Bäck and Khuri(1994)] T. Bäck, S. Khuri, An evolutionary heuristic for the maximum independent set problem, in: Proc. 1st IEEE Conference on Evolutionary Computation, IEEE, 1994, pp. 531–535. doi:10.1109/ICEC.1994.350004.

[Bell and Stevens(2009)] J. Bell, B. Stevens, A survey of known results and research areas for N-queens, Discrete Math. 309 (2009) 1–31. URL: http://dx.doi.org/10.1016/j.disc.2007.12.043. doi:10.1016/j.disc.2007.12.043.

[Rodionova et al.(2019)Rodionova, Antonov, Buzdalova, and Doerr] A. Rodionova, K. Antonov, A. Buzdalova, C. Doerr, Offspring population size matters when comparing evolutionary algorithms with self-adjusting mutation rates, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'19), ACM, 2019, pp. 855–863. URL: https://doi.org/10.1145/3321707.3321827. doi:10.1145/3321707.3321827, full version available online at https://arxiv.org/abs/1904.08032.

[Dang and Doerr(2019)] N. Dang, C. Doerr, Hyper-parameter tuning for the (1 + (λ, λ)) GA, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'19), ACM, 2019, pp. 889–897. URL: https://doi.org/10.1145/3321707.3321725. doi:10.1145/3321707.3321725.

[Carvalho Pinto and Doerr(2017)] E. Carvalho Pinto, C. Doerr, Discussion of a more practice-aware runtime analysis for evolutionary algorithms, in: Proc. of Artificial Evolution (EA'17), 2017, pp. 298–305. URL: https://ea2017.inria.fr//EA2017_Proceedings_web_ISBN_978-2-9539267-7-4.pdf, full version available at http://arxiv.org/abs/1812.00493.

[Doerr et al.(2017a)Doerr, Le, Makhmara, and Nguyen] B. Doerr, H. P. Le, R. Makhmara, T. D. Nguyen, Fast genetic algorithms, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'17), ACM, 2017a, pp. 777–784. URL: http://doi.acm.org/10.1145/3071178.3071301. doi:10.1145/3071178.3071301.

[Doerr et al.(2017b)Doerr, Gießen, Witt, and Yang] B. Doerr, C. Gießen, C. Witt, J. Yang, The $(1 + \lambda)$ evolutionary algorithm with self-adjusting mutation rate, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'17), ACM, 2017b, pp. 1351–1358.

[Ye et al.(2019)Ye, Doerr, and Bäck] F. Ye, C. Doerr, T. Bäck, Interpolating Local and Global Search by Controlling the Variance of Standard Bit Mutation, in: Proc. of Congress on Evolutionary Computation (CEC'19), IEEE, 2019, pp. 2292–2299. Also available at http://arxiv.org/abs/1901.05573.

[Bäck and Schütz(1996)] T. Bäck, M. Schütz, Intelligent mutation rate control in canonical genetic algorithms, in: International Symposium on Foundations of Intelligent Systems (ISMIS'96), volume 1079 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 158–167.

[Goldberg(1989)] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, Reading, MA, 1989.

[Bäck(1996)] T. Bäck, Evolutionary Algorithms in Theory and Practice, Oxford University Press, New York, NY, USA, 1996.

[Mühlenbein(1997)] H. Mühlenbein, The equation for response to selection and its use for prediction, Evolutionary Computation 5 (1997) 303–346. URL: https://doi.org/10.1162/evco.1997.5.3.303. doi:10.1162/evco.1997.5.3.303.

[Mühlenbein and Paaß(1996)] H. Mühlenbein, G. Paaß, From recombination of genes to the estimation of distributions i. binary parameters, in: H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.), Proc. of Parallel Problem Solving from Nature (PPSN'96), Springer, 1996, pp. 178–187.

[Krejca and Witt(2018)] M. S. Krejca, C. Witt, Theory of estimation-of-distribution algorithms, CoRR abs/1806.05392 (2018). URL: http://arxiv.org/abs/1806.05392.

[Horesh et al.(2019)Horesh, Bäck, and Shir] N. Horesh, T. Bäck, O. M. Shir, Predict or screen your expensive assay: Doe vs. surrogates in experimental combinatorial optimization, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'19), ACM, 2019, pp. 274–284. URL: https://doi.org/10.1145/3321707.3321801. doi:10.1145/3321707.3321801.

[Hansen(2018)] N. Hansen, A practical guide to experimentation, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'18), Companion material, ACM, 2018, pp. 432–447.

[Calvo et al.(2019)Calvo, Shir, Ceberio, Doerr, Wang, Bäck, and Lozano] B. Calvo, O. M. Shir, J. Ceberio, C. Doerr, H. Wang, T. Bäck, J. A. Lozano, Bayesian performance analysis for black-box optimization benchmarking, in: Proc. of the Genetic and Evolutionary Computation Conference (GECCO'19, Companion Material), ACM, 2019, pp. 1789–1797. URL: https://doi.org/10.1145/3319619.3326888. doi:10.1145/3319619.3326888.

[Calvo et al.(2018)Calvo, Ceberio, and Lozano] B. Calvo, J. Ceberio, J. A. Lozano, Bayesian inference for algorithm ranking analysis, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'18), companion material, ACM, 2018, pp. 324–325. URL: https://doi.org/10.1145/3205651.3205658. doi:10.1145/3205651.3205658.

[Hansen et al.(2016)Hansen, Auger, Brockhoff, Tusar, and Tušar] N. Hansen, A. Auger, D. Brockhoff, D. Tusar, T. Tušar, COCO: performance assessment, CoRR abs/1605.03560 (2016). URL: http://arxiv.org/abs/1605.03560.

[Hastie et al.(2013)Hastie, Tibshirani, and Friedman] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer Series in Statistics, Springer New York, 2013.

[Fonseca et al.(2011)Fonseca, Guerreiro, López-Ibáñez, and Paquete] C. M. Fonseca, A. P. Guerreiro, M. López-Ibáñez, L. Paquete, On the computation of the empirical attainment function, in: Proc. of Evolutionary Multi-Criterion Optimization (EMO'11), volume 6576 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 106–120. URL: https://doi.org/10.1007/978-3-642-19893-9_8. doi:10.1007/978-3-642-19893-9_8.

[Moré and Wild(2009)] J. J. Moré, S. M. Wild, Benchmarking derivative-free optimization algorithms, SIAM Journal on Optimization 20 (2009) 172–191.

[Benavoli et al.(2017)Benavoli, Corani, Demsar, and Zaffalon] A. Benavoli, G. Corani, J. Demsar, M. Zaffalon, Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis, Journal of Machine Learning Research 18 (2017) 77:1–77:36. URL: http://jmlr.org/papers/v18/16-305.html.

[Belkhir et al.(2017)Belkhir, Dréo, Savéant, and Schoenauer] N. Belkhir, J. Dréo, P. Savéant, M. Schoenauer, Per instance algorithm configuration of CMA-ES with limited budget, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'17), ACM, 2017, pp. 681–688. URL: https://doi.org/10.1145/3071178.3071343. doi:10.1145/3071178.3071343.

[Kerschke and Trautmann(2016)] P. Kerschke, H. Trautmann, 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, 2016, pp. 5262–5269. URL: https://doi.org/10.1109/CEC.2016.7748359. doi:10.1109/CEC.2016.7748359.

[Thomson et al.(2018a)Thomson, Vérel, Ochoa, Veerapen, and McMenemy] S. L. Thomson, S. Vérel, G. Ochoa, N. Veerapen, P. McMenemy, On the fractal nature of local optima networks, in: Proc. of Evolutionary Computation in Combinatorial Optimization (EvoCOP'18), volume 10782 of *Lecture Notes in Computer Science*, Springer, 2018a, pp. 18–33. URL: https://doi.org/10.1007/978-3-319-77449-7_2. doi:10.1007/978-3-319-77449-7_2.

[Thomson et al.(2018b)Thomson, Vérel, Ochoa, Veerapen, and Cairns] S. L. Thomson, S. Vérel, G. Ochoa, N. Veerapen, D. Cairns, Multifractality and dimensional determinism in local optima networks, in: Proc. of Genetic and Evolutionary Computation Conference (GECCO'18), ACM, 2018b, pp. 371–378. URL: https://doi.org/10.1145/3205455.3205472. doi:10.1145/3205455.3205472.

[Cahon et al.(2004)Cahon, Melab, and Talbi] S. Cahon, N. Melab, E. Talbi, ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics, J. Heuristics 10 (2004) 357–380. URL: https://doi.org/10.1023/B:HEUR.0000026900.92269.ec. doi:10.1023/B:HEUR.0000026900.92269.ec.

[Durillo and Nebro(2011)] J. J. Durillo, A. J. Nebro, jMetal: A Java framework for multi-objective optimization, Advances in Engineering Software 42 (2011) 760–771. URL: http://www.sciencedirect.com/science/article/pii/S0965997811001219. doi:DOI:10.1016/j.advengsoft.2011.05.014.

[van Rijn et al.(2016)van Rijn, Wang, van Leeuwen, and Bäck] S. van Rijn, H. Wang, M. van Leeuwen, T. Bäck, Evolving the structure of evolution strategies, in: 2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016, IEEE, 2016, pp. 1–8. URL: https://doi.org/10.1109/SSCI.2016.7850138. doi:10.1109/SSCI.2016.7850138.