Benchmarking discrete optimization heuristics: from building a sound experimental environment to algorithm configuration

Ye, F.

Chapter 2

Preliminaries

This chapter briefly introduces three topics of this thesis, covering *Optimization*, *Evolutionary Algorithms*, and *Algorithm Configuration*. Detailed descriptions of performance measures and benchmark problems used in the empirical study are also provided.

2.1 Optimization

Optimization aims at finding the *best* solution for a given problem. Optimization problems arise in many disciplines, e.g., biology [75], engineering [33], logistics [9, 66], physics [113], etc. Meanwhile, benchmarks have been built to unify ideas and methods for different domains such as numerical analysis [79], software engineering [81], traveling salesperson problems [132], etc.

Optimization problems consist of three elements, i.e., objective functions, decision variables, and constraints.

The **objective function** is a mapping that assesses the quality of a candidate solution. The assessment returns a value (i.e., single-objective optimization) or a set of values (i.e., multi-objective optimization). We only consider the single-objective optimization that is subject to maximization in this thesis. In short, we aim at maximizing a function:

$$f : S \to \mathbb{R}, x \mapsto f(x), \tag{2.1}$$

where we refer to S as the search space and its element $x \in S$ as a search point (or solution candidate). The decision variables of x can have different types: real

(i.e., continuous), integer (e.g., ordinal), and nominal (i.e., categorical). Continuous optimization considers only real-valued variables, discrete optimization considers integer and nominal variables, and mixed-integer problems consist of multiple types of variables.

Constraints restrict the values of variables that can be taken for solutions, which can be either hard constraints or soft constraints. Solutions must satisfy the requirements of hard constraints, and penalties will be assigned if soft constraints are violated [28]. Also, constraints can be distinguished by *equality constraints* $h(x) = 0$ and *inequality constraints* $g(x) \leq 0$.

2.1.1 Pseudo-Boolean Optimization

In this thesis, we study a subset of discrete optimization, whose variables consist of binary variables, namely *pseudo-Boolean optimization* [19]:

$$f : \{0,1\}^n \to \mathbb{R}, x \mapsto f(x). \tag{2.2}$$

A variety of problems are related to pseudo-Boolean optimization, including spin glass [11], maximum satisfiability [70], fault location [114], clustering [135], project selection [130], etc. Many techniques have been applied to solve these problems specifically. In this thesis, we study the performance of EAs and other IOHs on pseudo-Boolean optimization problems.

2.2 Evolutionary Algorithms

Though exact algorithms, such as dynamic programming, have been developed for solving optimization problems, these techniques usually require additional effort for specific problems and can not solve hard problems. However, EAs have achieved success in approximating solutions to problems in many fields [6, 9, 33, 64, 66, 70].

EAs were originally inspired by biological evolution. The general procedure of EAs is producing *offspring* using variation operators after initializing a *parent* population of solution candidates (i.e., individuals), then updating the population by selecting from offspring (and parent) solution candidates. This procedure is iterated until the termination criterion is reached.

Different design of operators will result in variations of EAs such as GAs, evolution strategies (ESs), etc. For instance, mutation and crossover are two common operators of EAs. Mutation allows exploiting promising search areas using small mutation rates,

and crossover creates offspring by recombining information of two or more parents. GAs use both mutation and crossover as variation operators. However, the EAs for pseudo-Boolean optimization usually concentrate on mutation only. Also, there are different strategies to form new populations. For example, a plus-strategy will select solution candidates from both parent and offspring populations, and a comma-strategy will consider only the offspring population.

Apart from the combinations of different operators, we have a variety of algorithms because most operators are parameterized, raising the question of how to configure them properly for the given optimization task. For example, EAs require proper settings of the population size, the mutation rate, and the selective pressure; GAs' performance can be affected by the values of the population size and the crossover probability. Note that we consider that the EAs for pseudo-Boolean optimization in this thesis are mutation-only algorithms.

2.3 Parameter Tuning Techniques

It is well known that the choice of parameters and operators influences the performance of EAs significantly [2, 93]. There are two classes of approaches to determine algorithm settings, namely static parameter setting and dynamic parameter control [42].

Static parameter setting identifies parameter values and operator choices of the algorithms for a given problem. The settings are predefined for the optimization process. Usually, the settings are based on empirical studies or theoretical works. For the empirical study methods, the design of experiments (DOE) [26] methods can help us understand the relationship between input parameters and decide a promising algorithm setting. Nowadays, automatic tuning tools have also been applied to identify algorithms' parameters, and the obtained results have shown significant advantages against manual settings. In the theory research domain, researchers prove bounds for the running time of algorithms with respect to specific parameters values, such that we can find suitable parameter settings by minimizing these bounds [163].

Dynamic parameter control adjusts parameter values and even operator choices during the optimization process. The aim is to benefit from applying promising settings at different stages of the optimization process. Both empirical [2, 61, 92] and theoretical studies [42] have been conducted for dynamic algorithm control. Recent work has formulated dynamic algorithm configuration as a contextual Markov decision process, and the authors compared the reinforcement learning and the classic sequential model-based optimization for general algorithm configuration (SMAC) on

their test bed [17]. Also, a similar topic, dynamic algorithm selection, has attracted attention in recent research [153]. Theoreticians also investigate the optimal parameters for theory-oriented problems such as OneMax and proposed theory-inspired self-adaptation methods [40, 46].

In the following, we introduce *algorithm configuration (AC)*, also known as hyperparameter optimization, which belongs to the class of static parameter tuning approaches. AC is applied to explore promising operator combinations and parameter settings of the algorithms for a given problem. Most AC methods do not require preliminary knowledge of the algorithm and the problem. Consequently, we can not directly explain why (or if) the obtained configurations perform well. Benefiting from benchmarking, we can understand the behavior of both algorithms and the AC methods.

We provide here the definition of the algorithm configuration problem [60]:

Definition 2.1 (AC: Algorithm Configuration)**.** Given a set of problem instances P, a parametrized algorithm A with parameter space Θ, and a cost metric $c : \Theta \times P \to \mathbb{R}$ that is subject to minimization, the objective of the AC problem is to find a configuration $\theta^* \in \Theta$ such that the cost $c(\theta, P)$ is as small as possible.

The AC problem can be seen as a meta-optimization problem, asking to optimize performance of a specific solver on a given set of problem instances.

Many approaches, e.g., Bayesian optimization [155], local search [83], evolutionary algorithms [109], gradient-based optimization [16], etc., can be used for the AC problem. Among the best-known tools are paramILS [83], SMAC [82], and Irace [111]. These methods have been applied to boost the performance of algorithms in many domains such as TSP [111], software engineering [14], and machine learning [97].

2.4 Algorithm Performance Measures

We introduce here the performance measures used in this thesis to assess algorithms' behaviour, concerning three different objectives, namely *fixed-target performance*, *fixed-budget performance*, and *anytime performance*. The measures can also be used as the cost metric in Definition 2.1.

2.4.1 Fixed-target Performance

For the fixed-target results, we consider the cost needed by each algorithm to find a solution that is at least as good as a certain target. In this thesis, we consider the

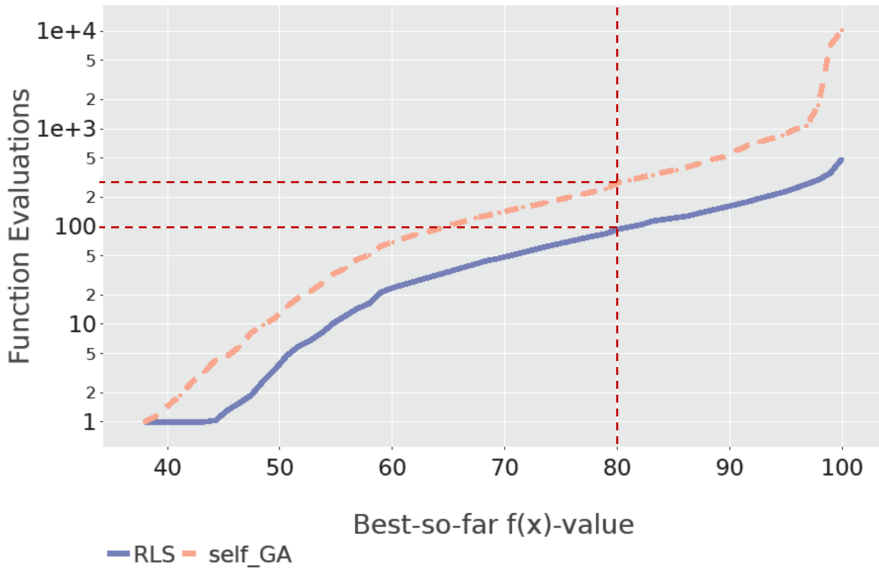

Figure 2.1: ERT values of the two algorithms RLS and self_GA for a maximization problem in a fixed-target perspective.

time cost measured by the expected running time (ERT) value, where *time* indicates the number of function evaluations.

Definition 2.2 (ERT: Expected Running Time)**.** Given a target ϕ for a problem P, the ERT of an algorithm A hitting ϕ is

$$\mathrm{ERT}(A, P, \phi) = \frac{\sum_{i=1}^{r} \min\{t_i(A, P, \phi), B\}}{\sum_{i=1}^{r} \mathbb{1}\{t_i(A, P, \phi) < \infty\}} , \tag{2.3}$$

where r is the number of independent runs of A, B is the given budget (i.e., the maximal number of function evaluations), $t_i(A, P, \phi) \in \mathbb{N} \cup \{\infty\}$ is the running time (for finite values, the running time is the number of function evaluations that the i-th run of A on the problem P uses to hit the target ϕ, and $t_i(A, P, \phi) = \infty$ is used if none of the solutions is better than ϕ), and $\mathbb{1}(\mathcal{E})$ is the indicator function returning 1 if event $\mathcal{E}$ happens and 0, otherwise. $t_i(A, P, \phi) < \infty$ indicates that the algorithm hits the target within the given budget B in the i-th run. If the algorithm hits the target ϕ in all r runs, the ERT is equal to the average hitting time (AHT).

Definition 2.3 (AHT: Average Hitting Time)**.** Given a target ϕ for a problem P, the

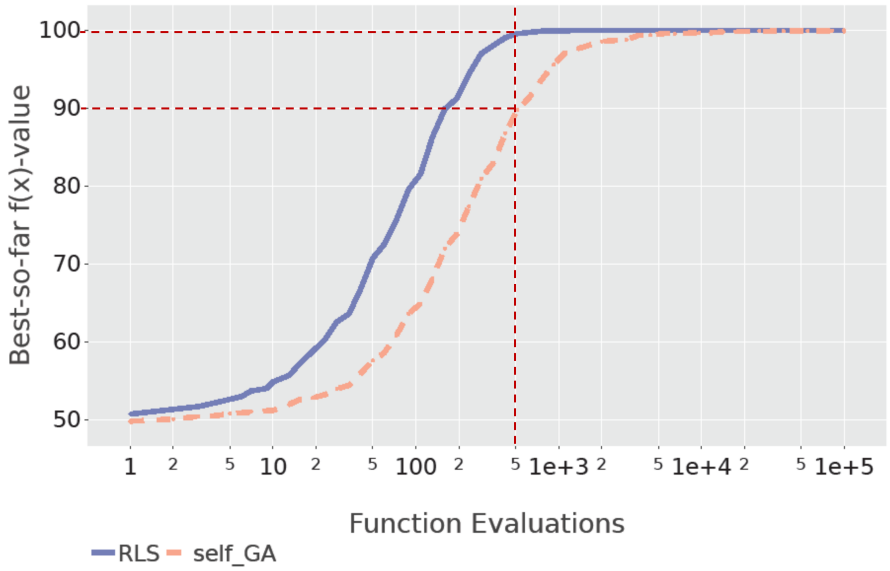

Figure 2.2: The mean of best-found fitness values of the two algorithms RLS and self_GA for a maximization problem in a fixed-budget perspective.

AHT of an algorithm A hitting ϕ is

$$\mathrm{AHT}(A, P, \phi) = \frac{\sum_{i=1}^{r} t_i(A, P, \phi)}{r}\,, \tag{2.4}$$

where r is the number of independent runs of A, $t_i(A, P, \phi)$ is the running time (i.e., the number of function evaluations) that the i-th run of A uses to hit the target ϕ of P.

Figure 2.1 is an example showing fixed-target curves, which plots the ERT values (y-axis) that the two algorithms RLS and self_GA need to find a solution satisfying $f(x) \geq \phi$, where the target value ϕ is the value on x-axis. We see that the ERT of the RLS for the target value 80 is around 100, while the ERT of the self_GA is around 300.

2.4.2 Fixed-budget Performance

For the fixed-budget results, we consider the quality of solutions found by each algorithm with a given budget. Figure 2.2 is an example showing fixed-budget curves, which plots average of the best-found fitness values (y-axis) after using specific budget

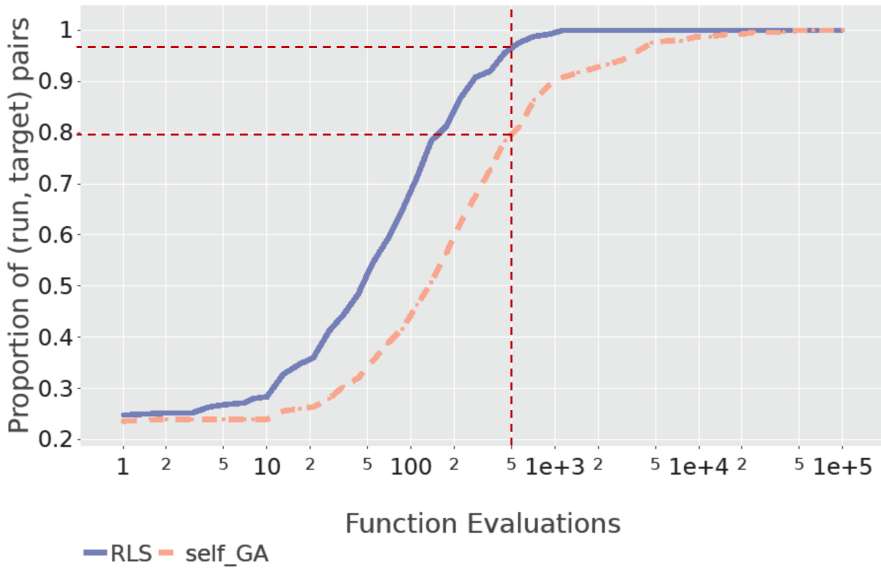

Figure 2.3: ECDF values of the two algorithms RLS and self_GA for different budgets.

B, where the budget value B is the value on x-axis. We see that, after using 500 function evaluations, the mean of best-found fitness value of the RLS is 100, while the mean of best-found fitness value of the self_GA is 90.

2.4.3 Anytime Performance

Another important concept in the analysis of IOHs are empirical cumulative distribution function (ECDF) curves, which allow to aggregate performance across different targets. The definition of ECDF is given below. Figure 2.3 plots ECDF values (y-axis) after using specific budget B, where the budget value B is the value on x-axis. We see that, around 96% runs of the (run,target) pairs hit the corresponding target within the given budget of 500 function evaluations for the RLS, while this number is 80% for the self_GA. Note that ECDF can also be applied for evaluating algorithm performance across different functions.

Definition 2.4 (ECDF: empirical cumulative distribution function of the running time)**.** Given a set of targets $\Phi = \{\phi_i \in \mathbb{R} \mid i \in \{1, 2, \ldots, m\}\}$ for a real-valued problem P and a set of budgets $T = \{t_j \in \mathbb{N} \mid j \in \{1, 2, \ldots, B\}\}$ for an algorithm A, the ECDF value of A at budget t_j is the fraction of (run, target)-pairs (r, ϕ_i) that

satisfy that the run r of the algorithm A finds a solution has fitness at least as good as ϕ_i within the budget t_j.

In this thesis, we evaluate anytime performance of algorithms using the area under the ECDF curve (AUC). The domain of AUC values is $[0, 1]$. The definition is given below. Note that the given definition is for a discretized version of AUC, of which values can be affected by the given targets Φ and budgets T.

Definition 2.5 (AUC: area under the ECDF curve)**.** Given a set of targets $\Phi = \{\phi_i \in \mathbb{R} \mid i \in \{1, 2, \ldots, m\}\}$ and a set of budgets $T = \{t_j \in \{1, 2, \ldots, B\} \mid j \in \{1, 2, \ldots, z\}\}$, the AUC $\in [0, 1]$ (normalized over B) of algorithm A on problem P is the area under the ECDF curve of the running time over multiple targets. For maximization, it reads

$$\mathrm{AUC}(A, P, \Phi, T) = \frac{\sum_{h=1}^{r} \sum_{i=1}^{m} \sum_{j=1}^{z} \mathbb{1}\{\phi_h(A, P, t_j) \geq \phi_i\}}{rmz},$$

where r is the number of independent runs of A and $\phi_h(A, P, t)$ denotes the value of the best solution that A evaluated within its first t evaluations of the run h.

2.5 Benchmark Problems

In this thesis, we focus on *pseudo-Boolean optimization problems*, 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 the benchmark problems should allow to assess performances across different dimensions. All problems have been implemented and integrated within the IOHprofiler software.

Conventions Throughout this thesis, 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 2.5.6 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 2.5.10), 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.

We formulate all problems as *maximization* problems.

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.

2.5.1 Problems vs. Instances

We define a *problem* in this thesis as a collection of *functions* sharing some common properties. For example, the NK landscape problem refers to problems with different gene interactions [94]. 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, a well-recognized approach to request representation invariance 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 $\sigma : [n] \to [n]$. 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. Further discussions of these *unbiasedness* transformations can be found in [54, 106].

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 the 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.

2.5.2 Overview of Selected Benchmark Problems

We summarize here the benchmark problems that we repeatedly use in this thesis to compare algorithms' performance, which are from the suite of IOHprofiler for

pseudo-Boolean optimization (PBO). The PBO suite originally consisted of twenty three problem [55], and two problems were added afterwards in [170].

- **F1 and F4-F10:** OneMax and its W-model extensions; details in Sections 2.5.3 and 2.5.6
- **F2 and F11-F17:** LeadingOnes and its W-model extensions; details in Sections 2.5.4 and 2.5.6
- **F3:** Harmonic; see Section 2.5.5
- **F18:** LABS: Low Autocorrelation Binary Sequences; see Section 2.5.7
- **F19-21:** Ising Models; see Section 2.5.8
- **F22:** MIVS: Maximum Independent Vertex Set; see Section 2.5.9
- **F23:** NQP: N-Queens; see Section 2.5.10
- **F24:** CT: Concatenated Trap; see Section 2.5.11
- **F25:** NKL: Random NK landscapes; see Section 2.5.12

2.5.3 F1: OneMax

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

$$\text{F1} : \text{OM} : \{0,1\}^n \to [0..n], x \mapsto \sum_{i=1}^{n} x_i.$$

The problem has a very smooth and non-deceptive fitness landscape. Due to the well-known coupon collector effect (see, for example, [57] 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 2.5.1, the OneMax problem becomes the problem of minimizing the Hamming distance to an unknown target string $z \in \{0,1\}^n$.

That OneMax is interesting beyond the study of theoretical aspects of evolutionary computation has been argued in [147]. 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.

2.5.4 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

$$\mathrm{F2} : \mathrm{LO} : \{0,1\}^n \to [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\} = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j, \quad (2.5)$$

which counts the number of initial ones.

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 4.1.

2.5.5 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

$$\mathrm{F3} : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_i i x_i$$

with harmonic weights, which was suggested to be considered in [139].

2.5.6 F4-F17: The W-model

In [160], 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 problems, giving yet another degree of freedom. Note here that the original work [160] and the existing empirical evaluations [159] 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 [94] 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 thesis, we therefore restrict our attention to instances in which the different components of the W-model are used in an isolated way, see Section 2.5.6. The assessment of combined transformations clearly forms a promising line for future work.

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 the study in this thesis. A more detailed description can be found in the original work [160]. For some of the descriptions below we deviate from the exposition in [160], 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 [160] 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 [160].

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 [44, 62].

   Dummy variables are also among the characteristics of the benchmark functions contained in the Nevergrad platform [131], 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** $W(*, u, *, *)$**:** The bit string $(x_1, \ldots, x_n)$ is reduced to a string $(y_1, \ldots, y_m)$ with $m = n/u$, where u 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-u substring of x. More precisely, $y_i = 1$ if and only if there are at least $u/2$ ones in the substring $(x_{(i-1)u+1}, x_{(i-1)u+2}, \ldots, x_{iu})$.[1] When $n/u \notin \mathbb{N}$, the last $n - u\lfloor n/u \rfloor$ remaining bits of x not fitting into any of the blocks are simply deleted; that is, we have $m = \lfloor n/u \rfloor$ and the entries x_i with $i > u\lfloor n/u \rfloor$ do not have any influence on y (and, thus, no influence on the function value).

Example: With $n = 10$ and $u = 3$ 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 [160] 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 [160].

$e_4(0000) = 0000$	$e_4(0001) = 1101$	$e_4(0010) = 1011$	$e_4(0011) = 0110$
$e_4(0100) = 0111$	$e_4(0101) = 1010$	$e_4(0110) = 1100$	$e_4(0111) = 0001$
$e_4(1000) = 1111$	$e_4(1001) = 0010$	$e_4(1010) = 0100$	$e_4(1011) = 1001$
$e_4(1100) = 1000$	$e_4(1101) = 0101$	$e_4(1110) = 0011$	$e_4(1111) = 1110$

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 de-

[1] Note that with this formulation there is a bias towards ones in case of a tie. We follow here the suggestion made in [160], 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 u only.

termine 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 [160] 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 [160] 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 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 2.5.6.

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 $r(n) = n$.

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 modifications can, in addition, be applied in an arbitrary order, with each order resulting in a different benchmark problem. In line with the presentation in [160], 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 $(\{i_1, \ldots, i_m\}, u, \nu, r)$ with $m \leq n$, $1 \leq i_1 < \ldots < i_m \leq n$, $u \in [n]$, $\nu \in [n]$, and $r : V \to V$, all to be interpreted as in the descriptions given in Section 2.5.6 above. Setting $\{i_1, \ldots, i_m\} = [n]$, $u = 1$, $\nu = 1$, 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 $u > 1$. 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 $s = \lfloor m/u \rfloor$. 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 $\ell \geq us$ and $\ell - us$ depending on the reduction that one wishes to achieve by the first transformation, the removal of dummy variables.

Selected W-Model Transformations

In contrast to existing works cited in [159, 160], 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 $u = 3$, and we consider the epistasis perturbation of size $\nu = 4$. Finally, we consider the following ruggedness functions, where we denote by s the size of the effective dimension (see Section 2.5.6 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 2.4.

- $r_1 : [0..s] \to [0..\lceil s/2 \rceil + 1]$ 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)

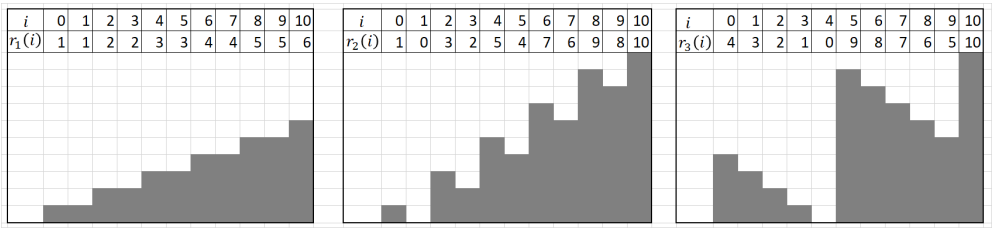

Figure 2.4: The ruggedness functions r_1, r_2, and r_3.

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: $\textsc{OneMax} + W(\lfloor n/2 \rfloor, 1, 1, \mathrm{id})$
F5: $\textsc{OneMax} + W(\lfloor 0.9n \rfloor, 1, 1, \mathrm{id})$
F6: $\textsc{OneMax} + W(n, u = 3, 1, \mathrm{id})$
F7: $\textsc{OneMax} + W(n, 1, \nu = 4, \mathrm{id})$
F8: $\textsc{OneMax} + W(n, 1, 1, r_1)$
F9: $\textsc{OneMax} + W(n, 1, 1, r_2)$
F10: $\textsc{OneMax} + W(n, 1, 1, r_3)$

F11: $\textsc{LeadingOnes} + W(\lfloor n/2 \rfloor, 1, 1, \mathrm{id})$
F12: $\textsc{LeadingOnes} + W(\lfloor 0.9n \rfloor, 1, 1, \mathrm{id})$
F13: $\textsc{LeadingOnes} + W(n, u = 3, 1, \mathrm{id})$
F14: $\textsc{LeadingOnes} + W(n, 1, \nu = 4, \mathrm{id})$
F15: $\textsc{LeadingOnes} + W(n, 1, 1, r_1)$
F16: $\textsc{LeadingOnes} + W(n, 1, 1, r_2)$
F17: $\textsc{LeadingOnes} + W(n, 1, 1, r_3)$

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 2.5.1. 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 $(i_1, \ldots, i_m, u, \nu, r)$. 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 $(i_1, \ldots, i_m, u, \nu, r)$.

2.5.7 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 [138, 126]. 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., [116, 125]), 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:

$$F_{\text{LABS}}(\vec{x}) = \frac{n^2}{2 \sum_{k=1}^{n-1} \left(\sum_{i=1}^{n-k} x'_i \cdot x'_{i+k} \right)^2} \quad \text{where } x'_i = 2x_i - 1. \tag{2.6}$$

2.5.8 F19-F21: The Ising Model

The Ising Spin Glass model [11] 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 [113]. EAs have been investigated concerning the impact of their operators for the Ising model, yielding some theoretical results on certain graph instances (see, e.g., [22, 65, 141]).

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 [141]). A generalized, compact form for the quadratic objective function is now obtained:

$$F_{\text{Ising}}(\vec{x}) = \sum_{\{u,v\} \in E} \left[x_u x_v + (1 - x_u)(1 - x_v) \right], \tag{2.7}$$

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 (2.7).

F19: The Ring (1D)

This basic Ising model is defined over a one-dimensional lattice. The objective function follows (2.7) using the following graph:

$$\begin{aligned} & G_{\text{Is1D}}: \\ & e_{ij} = 1 \quad \Leftrightarrow \quad j = i + 1 \quad \forall i \in \{1, \ldots, n-1\} \\ & \qquad\qquad \vee \quad j = n, i = 1 \end{aligned} \tag{2.8}$$

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), $0 \leq i, j \leq N-1$ [22]. Since PBC are applied, a *regular graph with each vertex having exactly four neighbors* is obtained. The objective function follows (2.7) using the following graph:

$$\begin{aligned} & G_{\text{Is2D}}: \\ e_{(i,j)(k,\ell)} = 1 \quad \Leftrightarrow \quad & [k = (i+1) \bmod N \quad \wedge \quad \ell = j \quad \forall i, j \in \{0, \ldots, N-1\}] \\ \vee \quad & [k = (i-1) \bmod N \quad \wedge \quad \ell = j \quad \forall i, j \in \{0, \ldots, N-1\}] \\ \vee \quad & [\ell = (j+1) \bmod N \quad \wedge \quad k = i \quad \forall j, i \in \{0, \ldots, N-1\}] \\ \vee \quad & [\ell = (j-1) \bmod N \quad \wedge \quad k = i \quad \forall j, i \in \{0, \ldots, N-1\}] \end{aligned} \tag{2.9}$$

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* $\frac{2\pi}{3}$ [115]. 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):

$$
\begin{aligned}
& G_{\mathrm{IsTR}}: \\
e_{(i,j)(k,\ell)} = 1 \quad \Leftrightarrow \quad & [k = (i+1) \bmod N \ \wedge \ \ell = j \quad \forall i, j \in \{0, \ldots, N-1\}] \\
\vee \quad & [k = (i-1) \bmod N \ \wedge \ \ell = j \quad \forall i, j \in \{0, \ldots, N-1\}] \\
\vee \quad & [\ell = (j+1) \bmod N \ \wedge \ k = i \quad \forall j, i \in \{0, \ldots, N-1\}] \\
\vee \quad & [\ell = (j-1) \bmod N \ \wedge \ k = i \quad \forall j, i \in \{0, \ldots, N-1\}] \\
\vee \quad & [\ell = (j+1) \bmod N \ \wedge \ k = (i+1) \bmod N \ \forall j, i \in \{0, \ldots, N-1\}] \\
\vee \quad & [\ell = (j-1) \bmod N \ \wedge \ k = (i-1) \bmod N \ \forall j, i \in \{0, \ldots, N-1\}]
\end{aligned}
\tag{2.10}
$$

2.5.9 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

$$
F_{\mathrm{MIVS}}(x) = \sum_i x_i - n \cdot \sum_{i,j} x_i x_j e_{i,j}, \tag{2.11}
$$

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

In particular, following [6], we consider a specific, scalable problem instance, defining its Boolean graph as follows:

$$
\begin{aligned}
e_{ij} = 1 \quad \Leftrightarrow \quad & j = i + 1 \quad \forall i \in \{1, \ldots, n-1\} - \{n/2\} \\
\vee \quad & j = i + n/2 + 1 \quad \forall i \in \{1, \ldots, n/2 - 1\} \\
\vee \quad & j = i + n/2 - 1 \quad \forall i \in \{2, \ldots, n/2\}.
\end{aligned}
\tag{2.12}
$$

The resulting graph has a simple, standard structure as shown in Figure 2.5 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.

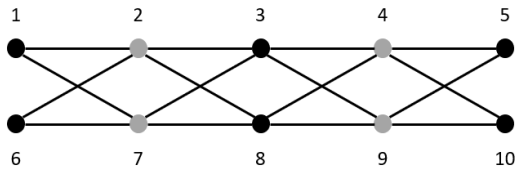

Figure 2.5: A scalable maximum independent set problem, with $n = 10$ vertices and the optimal solution of size 6 marked by the black vertices.

2.5.10 F23: N-Queens Problem

The N-queens problem (NQP) [15] 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.[2] Figure 2.6 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:

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

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

[2]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.

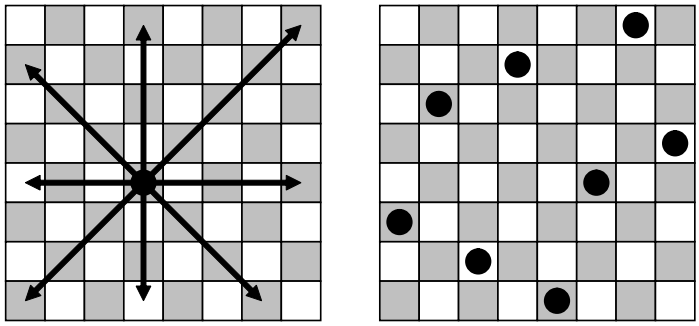

Figure 2.6: The 8-queens problem: [Left] all possible fields a queen can move to from position D4; [Right] a feasible solution.

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 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:

$$F_{\text{NQP}}(\vec{x}) = \sum_{i=1}^{N}\sum_{j=1}^{N} x_{ij} - N \cdot \left(\sum_{i=1}^{N} \max\left\{0, -1 + \sum_{j=1}^{N} x_{ij}\right\} + \sum_{j=1}^{N} \max\left\{0, -1 + \sum_{i=1}^{N} x_{ij}\right\} + \sum_{k=-N+2}^{N-2} \max\left\{0, -1 + \sum_{\substack{j-i=k \\ i,j\in\{1,2,\ldots,N\}}} x_{ij}\right\} + \sum_{\ell=3}^{2N-1} \max\left\{0, -1 + \sum_{\substack{j+i=\ell \\ i,j\in\{1,2,\ldots,N\}}} x_{ij}\right\} \right) \tag{2.13}$$

2.5.11 F24: Concatenated Trap

Concatenated Trap (CT) is defined by partitioning a bit-string into segments of length k and concatenating $m = n/k$ trap functions that takes each segment as input. The trap function is defined as follows: $f_k^{\text{trap}}(u) = 1$ if the number u of ones satisfies $u = k$ and $f_k^{\text{trap}}(u) = \frac{k-1-u}{k}$ otherwise. We use $k = 5$ in our experiments.

2.5.12 F25: Random NK Landscapes

Random NK landscapes (NKL). The function values are defined as the average of n sub-functions $F_i : [0..2^{k+1} - 1] \to \mathbb{R}, i \in [1..n]$, where each component F_i only takes as input a set of $k \in [0..n - 1]$ bits that are specified by a neighborhood matrix. In this paper, k is set to 1 and entries of the neighbourhood matrix are drawn u.a.r. in $[1..n]$. The function values of F_i's are sampled independently from a uniform distribution on $(0, 1)$.