Algorithm Selection for Discrete Black-Box Optimization

Raphaël Cosson,
supervised by Carola Doerr, LIP6, Sorbonne Université
and Thomas Weise, Hefei University, China

August 20, 2019

The General Context

Most of the time real-world optimization problems are difficult and can not be solved within a reasonable time limit. Effectively it is difficult to model or to express mathematically real-world problems. Though they need to be solved in a short amount of time with limited computational resources and insufficient expert knowledge.
A commonly used way to solve these problems is the use of Meta-Heuristic optimizers. They are iterative algorithms that try to find the best-possible approximate solution by evaluating the quality of candidates solutions and improving a current best one until a stopping criterion has been met.

It has been studied that there is typically no single algorithm more efficient than the others, instead they complement each other. Algorithms that performs well on some problems may performs poorly on others. In the same way, heuristic algorithms are often dependent on parameters influencing their effectiveness. These differences lead to the Algorithm Selection Problem and the Algorithm Configuration Problem. Different tools have been designed for Automated Algorithm Selection [2], [11],[12] in continuous optimization or for specific discrete problems such as Traveling Salesman Problem (TSP) [14] or propositional satisfiability (SAT) [25] and prove to exceed the expected results.

The research problem

Meta-heuristics algorithms are adapted to many discrete optimization problems in black-box settings, in particular for real-world problems difficult to model or to express mathematically. However, their performance varies from one problem to another.

In recent approaches, Algorithm Selection and Configuration for discrete optimization are mainly specific to a problem such as TSP or SAT, while different tools already exist for continuous optimization. The main difference between the continuous domain and the discrete domain is that there are already several ways for measuring structure of continuous optimization problems.

My internship focuses on the Algorithm Selection Problem for discrete optimization problems. The goal is to build a performance prediction model to learn which algorithm is the best for a given problem instance. Our approach aims to remain as general as possible by creating a performance prediction model covering a major part of discrete optimization functions.

Proposed contribution

Performance prediction models are based on three main elements: a set of benchmark problem instances, a set of algorithms and their performances over the benchmark problems, methods of numerically analysing structures of optimization problem, also called features and a learning strategy for the machine learning.

We use the W-model, recently proposed by Weise and Wu, to simulate functions with variation of hardness. The W-Model is a tunable benchmark problem with a number of scalable features for discrete optimization, which we used to collect performance data of nine classic algorithms among meta-heuristics algorithms.

Subsequently, we built two performance prediction models. The first is a reference model which knows the parameters of the W-Model. It can be seen as a white-box model with exact features. The second is a

real black-box model that uses features to learn and predict performances of algorithms.Then we perform cross validation tests and compare the respective performance of the two models.

Arguments supporting its validity

The W-Model is suitable as a benchmark problem for black-box discrete optimization. Indeed parameters influence simulated objective functions behavior by setting different known characteristics of optimization problems which modulate the complexity of the functions.

Algorithm Selection for discrete optimization has been empirically investigated. In our experiments we compared predictions with real features and with "exact" features. It has been demonstrated empirically that if features are cheap, performance prediction models reduces average cost for solving discrete optimization problems in black-box settings. Performance prediction models have already proven their effectiveness for continuous optimization problems and it turns out that they are also effective solutions for the discrete domain.

Both performance prediction models designed have empirically shown good results. Experiments have shown that using the performance model to choose the right algorithm for a given problem instance rather than always using the algorithm which had the best average can save an average of 10% of time on problem instances. Some features, although initially created for the continuous domain, also work for the discrete domain.

Summary and future work

The performance prediction model is an advance for discrete optimization. Experiments have shown that optimization problems in black-box settings can be solved more efficiently by using an algorithm selection tool. Instead of using an algorithm globally efficient for all objective functions, determining for each problem a specific algorithm can improve the computation time to solve a problem.

Many questions remain and many improvements are possible. In the short term, investigating the choice of feature sets seems to be a promising and necessary job. Allowing for a better understanding of which functions is well-covered by the W-Model but also by improving the precision of predictions and reducing the computation time of the estimation, defining a set of intuitive, precise and efficient features is a key objective to validate the performance prediction model as a state of the art optimizer.

In the long term, it is relevant to ca carry out cross validation tests on problems which are outside of the benchmark problems. This step would also help to understand which problems are poorly represented by the W-Model and to complete it. Due to our generic approach, the performance prediction model must determine for each problem in a black box a specific algorithm. Analyzing the completeness of the W-Model allows to have a better overview of the uncovered functions to be able to incorporate them.

1 Background

In theoretical contexts as well as for many industrial contexts, evolutionary algorithms (EAs) have been strongly studied to solve optimization problems. Problems whose analytical models are too complicated or that can not be formulated by a mathematical expression, these problems are often seen as black-box. Evolutionary algorithms (EAs) are a category of algorithms used to solved such problems. EAs evolve a population of candidate solution and they are well-performing for many optimization problems. They are at the origin of algorithms inspired by biology. By analogy, EAs are based on "genetic" variations of candidate solutions (mutation and crossover) and a natural selection operators.

However, performances of evolutionary algorithms contrast the difficulty to select the right algorithm and the ideal parameters for the problem to be solved. Although many evolutionary algorithms have been validated empirically, it is known that there is no perfect algorithm: No Free Lunch Theorem [24]. By this result, the first difficulty to solve a problem instance effectively is to determine which algorithm is best suited for the problem (Algorithm Selection problem) and what is the best configuration for the algorithm (Algorithm Configuration Problem).

Focusing in Algorithm Selection Problem, we consider two configurations of an algorithm as two distinct algorithms. Choosing the best algorithm can be seen as an optimization process. Given a problem instance, a set of algorithm and performance measure of each algorithm, the approach aims at finding the "best" algorithm to solve the problem.

1.1 Discrete Black Box Optimization

In this study, we consider problems instances as a black-box whose characteristics are not known. Each problem is an objective function $f(x) : x \in \{0,1\}^n \longrightarrow \mathbb{N}$ and algorithms aims to find x^* such that $f(x^*)$ is maximized.

Algorithms can use the black-box function to know the fitness of a candidate solution however they do not have any knowledge about the analytical form of the objective function.

It is common in practice to have problems whose structure can not be analysed or represented by a mathematical expression. For example, when a program uses an expensive library and must optimize parameters and minimizing the use of the library. An other real-life example: to test the safety of a car it is mandatory to perform crash tests. However these tests are very expensive for the designer. The objective of the manufacturer is to improve the resistance of the vehicle by minimizing the number of tests.

In case of black-box settings, algorithm performances are evaluated according to the number of function evaluations (FE). Objective function queries are counted as expensive operations. Operations to generate the candidate solutions (mutation, crossover) are considered cheap and the overall cost of optimization is reduced to the number of function evaluations.

1.2 Meta-heuristic Algorithms And Evolutionary Algorithms

Meta-heuristics are algorithms to solve optimization problems. These are generally iterative stochastic algorithms inspired by natural physics systems such as simulated annealing or by biology for Genetic Algorithm.

Meta-heuristics strategies evolve from exploration: a process that aims to gather information about objective function, to intensification also known as exploitation in which information is used to determine "best" areas of research.

Genetic Algorithms Evolutionary Algorithms are a meta-heuristics familly of bio-inspired algorithms [17]. These algorithms put in practice basic characteristics of the evolution (chromosome crossover, mutation, natural selection) to eliminate the least adapted individuals of the population. We can nevertheless distinguish the objective. The (real) evolution aims to improve average solutions, while algorithms aim to find the best fitted solution.

As in the rest of the document, candidate solutions are always represented by bit strings. Let $X \in \{0,1\}^n$ a candidate solution, also called *individuals*, such that $X = (x_1, x_2, ..., x_n)$.

We can define recurrent notions of genetic algorithms. A *population* P_k is a subset of feasible solutions $P_k \subset \{0,1\}^n$. These are the candidate solutions considered by the algorithm in the k^{th} iteration. A *gene* is

a bit or a subset of bits of a solution. The modification of a gene on an individual can have a drastic effect on its fitness.

Crossover operations generate new candidate solutions by combining solutions from the current population. A common crossover method is to exchange bits of two individuals with a predefined probability. The result of a crossover is called *offspring*, a solution inheriting genes from two "parent" solutions.

Mutation is an operator that changes the values of some genes (bits) of a solution. It preserves the diversity of a population.

Selection operators sort the newly composed solutions, selecting those that will be the new population P_{k+1}. Classical selections are often elitist (keep the bests) or integrate new individuals (randomly chosen) to renew the population.

The iterative process of genetics algorithms begins with initializing a population, usually at random, and evaluating their fitness. In a second step, a first selection determines the individuals used for the crossover. Offspring then mutate and their fitness is evaluated. If a stopping criterion has been met (satisfactory solution, budget exceeded, ...) the algorithm ends. Otherwise the algorithm iterates with the new generation as an individual population.

Although the overall process is similar for all genetic algorithms, they differ in strategies (type of crossover, selection techniques, stopping criteria, re-sampling, ...) and by parameters of such strategies (population size, probability of mutation, ...). These variants have a considerable impact on their efficiency and "best" choices rely on the objective functions.

1.3 Algorithm Selection

In a simple form, the problem is defined as : for an input algorithm set and a set of problems, we want to find which algorithm is the most efficient to solve each problem. As we have mentioned, algorithms performances show drastic changes from an objective function to an other.

Let Alg a set of algorithm and A_1, A_2,..., A_k its elements. We define a problem P_F by its features values $F = (f_1, f_2, ..., f_m)$. Features are numerical information about the landscape of the objective function and they are computed from a sample of fitness evaluation.

Running algorithm A_i on the problem P_F has a cost $c(A_i, P_F)$ (number of function evaluation) that we want to minimize. As all algorithms are stochastic, we want to determine the best algorithm $A^*_{P_F}$ which minimize $\tau(c(A^*_{P_F}, P_F))$ where τ correspond to some statistics, e.g. its mean or its median. This cost can only be empirically approximated by running several times all algorithms on the problem. Then Algorithm Selection is :

$$A^*_{P_F} = \min_{A_i \in Alg} \tau(c(A_i, P_F)) \tag{1}$$

1.4 Fitness landscape Analysis

The notion of fitness landscape analysis is an analogy to real landscape to give an intuitive understanding of how meta heuristic algorithms work and why some fitness functions are harder than other. To analyse the structure of a problem, the objective function is not sufficient to understand everything. Landscape analysis uses the concept of neighborhood solutions to associate candidate solutions with each other. Different tools and techniques, called features, exist to obtain an approximation of the fitness landscape.

Exploratory Landscape Analysis (ELA) is used for combinatorial and continuous optimization, and in particular single-objective optimization problems. The main motivation for landscape analysis is to gain a better understanding of algorithms performances on a set of problems. The search space of a problem is imaged in the form of a map with valleys and peaks respectively representing the solutions with low and high fitness. The metaphor evokes the path run through by an algorithm in the search space to solve a problem. However the analogy can be misleading in discrete optimization problem due to the dimensionality gap.

Formally, given the set of all feasible solution S, in the case of bit strings optimisation problems $S = \{0,1\}^n$, we define an objective function $f : S \longrightarrow \mathbb{N}$. To define the fitness landscape of a function, a distance function D is used to connect solutions and define their neighborhood V. The most common distance relation used for discrete optimization is the Hamming Distance D_h, used in following parts of

this work. Therefore, we can express the neighborhood $V \subset \{0,1\}^n$ of a solution $X \in \{0,1\}^n$ as the set of $V(X) = \{D_h(X, X') = 1 \quad \forall X' \in \{0,1\}^n\}$.

To have a better understanding of the structure of fitness landscapes, it is essential to apprehend the concept of "walk". By analogy with real-world landscape, an algorithm runs continuously across the landscape to explore the field and observes the fitness on its path. There are several strategies. The most common are Random Walks which consists in choosing a neighbour randomly, Adaptive Walks in which we select the most fitted neighbour, neutral walks in which we select neighbour with same fitness value.

1.5 W-Model

The main goal of the Black-Box Discrete Optimization Benchmarking (BB-DOB) workshop is to establish a list of example problems to study algorithms in black-box settings for discrete optimization. The W-Model has been proposed by Weise [23] as an optimization problem with tunable hardness. It takes known features which increase hardness for meta-heuristics to form a problem with tunable layers that introduce these features such as neutrality, ruggedness and epistasis.

1.5.1 Tunable Features

Neutrality In the biology evolution theory, Kimura [15] introduced the neutrality as areas in the landscape with equal fitness. For an important characteristic in neutral areas is that if an algorithm has a restricted landscape exploration protocol, it can be blocked in the neutral zones. In order to cross a neutral zone, the algorithm must accept equal fitness solutions. The difficulty of neutral problems is that in black-box settings, the algorithms have no information to orientate themselves between two solutions of equal fitness.

In the W-Model, the neutrality layer with parameter μ is a transformation (Figure 1) $u_\mu(x)$ that divides by μ a bit string x. The i^{th} bit of $u_\mu(x)$ is defined as 0 if and only if every bits from x at position between $i \times \mu$ and $(i+1) \times \mu - 1$ contains a strict majority of 0. Otherwise, the i^{th} bit of $u_\mu(x)$ is 1. If the length of x is not a multiple of μ, the remaining bits are ignored.

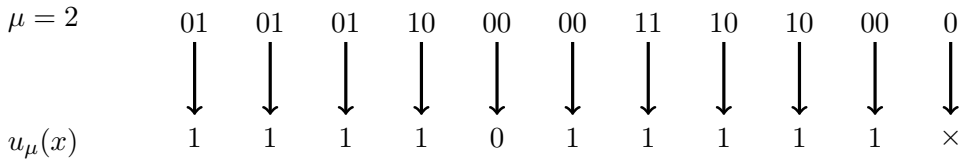

Figure 1: Introduction of Neutrality

Epistasis Epistasis is also a concept derived from biology [4]. In biology, an interaction between genes is said to be epistatic if the impact of a gene on the fitness depends on the allelic state of other genes. A gene can either inhibits or enhances the effect of other gene. This notion can be directly transposed to optimization as two (or more) variables whose contributions to the fitness rely on each other.

A fitness function f is not epistatic if and only if $f(x) = \sum_{i \in I} f_i(x_i)$, for some single-variable functions f_i. In other words, if a fitness function is not epistatic, a basic algorithm that find the best solution in $O(n)$ function evaluations consists of evaluating independently each variables to know if they inhibit or enhance the fitness. Davidor showed that it is useful to analyse epistasis as it severely influences the hardness of a fitness landscape [5].

The second layer of the W-Model aims to set a scalable transformation (Figure 2) of a fitness function into an epistatic one. Weise introduce a bijective function e_ν which translate a bit string x of length to a bit string of same length $e_\nu(x)$.

$$e_\nu(x) = \begin{cases} e_\nu(x)[i] = \bigotimes_{\substack{\forall j \in \mathbb{N}_0 : 0 \leq j < \nu, \\ j \neq (i-1) \bmod \nu}} x[j] & \forall x : 0 \leq x < 2^{\nu-1} \\ e_\nu(x - 2^{\nu-1}) & \text{otherwise} \end{cases} \tag{2}$$

The specificity of the equation 2 is that if two bit strings x_1 and x_2 differ from one bit, e.g. their hamming distance is one. So their respective transformation varies from $\nu - 1$ bits as in Equation 3. The transformation has two cases. If the most significant bit (the leftmost bit) is not set then the i^{th} bit in $e_\nu(x)$ equals the exclusive-or of all bits except one in x. If the most significant bit is set, then $e_\nu(x)$ to the negate e_ν transformation of x with the most significant bit cleared. If a bit string x is not a multiple of ν, then the length(x) mod ν last bits are transformed with the function $e_{\text{length}(x) \bmod \nu}$ instead of e_ν.

$$D_h(x_1, x_2) = 1 \implies D_h(e_\nu(x_1), e_\nu(x_2)) \geq \nu - 1 \qquad \forall\, x_1, x_2 \in \{0,1\}^\nu \tag{3}$$

This transformation ensures that each bit influences at least $\nu-1$ bits and therefore create a scalable epistasis layer.

One important note is that epistasis is not just introduced by this layer. Indeed, neutrality and ruggedness

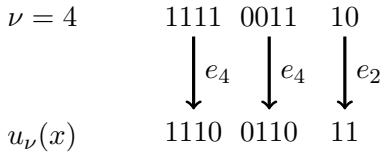

Figure 2: Introduction of Epistasis

are two layers of the W-Model that implicitly add epistasis but it seems difficult to study their effects separately. Also, these layers do not make it possible to constitute a scalable epistasis.

Ruggedness The ruggedness of a fitness landscape is an analogy to real landscape. The metaphor interprets the fitness landscape as a landscape with mountains and valleys. The peaks of the mountains represent solutions with high fitness and valleys with low fitness. Algorithms must chart their way to high fitness solutions. In the discrete domain, the distance between two solutions used is generally the Hamming distance. An easy problem will tend to have smooth plains in which a step from one solution to another lead to small changes in the objective value, while in a rugged fitness landscape small changes in the solution can value can cause large change in its objective values. Some landscape are also called deceptive. They have a smooth pathway of solutions that increase slightly the objective value but increase the distance from the optimal solution.

Ruggedness is implicitly created by the epistasis layer of the W-Model since a change of 1 bit in a solution x lead to a change of $\nu - 1$ bits in $e_\nu(x)$. However ruggedness is often generated by the objective function. The W-Model proposed a tunable ruggedness layer as a permutation r of objective values. In the initial problem [23], objective values are between 0 and n. The fitness landscape can be represented as in Figure 3. Exchanging two values of the fitness will insert ruggedness in the landscape. However, a permutation for the ruggedness must be bijective and preserves the optimal solution.

Let $\Delta(r_\gamma)$ a measure of ruggedness for a permutation r with parameter γ. The permutation proposed in the W-Model is a scalable permutation such that $\Delta(r_\gamma) = n + \gamma$ where n is the length of the bit string. This permutation can create rugged and deceptive landscape (Figure 3).

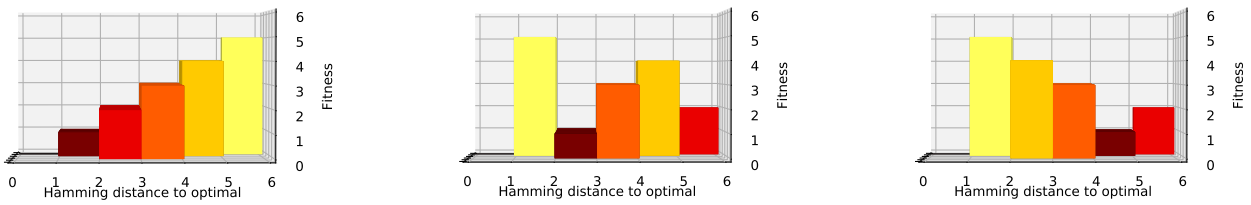

Figure 3: normal ($\gamma = 0$), rugged ($\gamma = 6$) and deceptive ($\gamma = 10$) landscape for minimization of OneMax at dimension 5

2 Project goal : Algorithm Selection for the W-Model

2.1 Adaptation of the W-Model to a new baseline

The data collection made in this work was done on a slightly different model. First, the baseline of the model problem is not to minimize the Hamming distance with the bit string 010101...01, but instead is the OneMax problem. Formally this problem can be described as finding a bit string X of length n, $X = (x_1, x_2, ..., x_n)$, $x_i \in \{0, 1\}$ which maximizes the following equation :

$$f(X) = \sum_{i=1}^{n} x_i \tag{4}$$

In this part, necessary modifications of the original model due to the new baseline as well as a new layer are presented.

Random Dummy Variables Layer We modified the W-Model by adding a transformation at the beginning. It simulates the bits without influences on the fitness of the candidate solutions. This layer is sometimes found in natural problems in which one or more variables have no influence on fitness. For example, a problem that contains an obvious symmetry can be reduced to a problem of smaller dimension by an arbitrary choice for this symmetry.

More formally, the layer Random Dummy Variables, with parameter k reduce a bit string $(x_1, x_2, ..., x_n)$ to a substring $(x_1, x_2, ..., x_{n-k})$ by randomly selecting the retained bits.

Ruggedness for Maximization problems The Ruggedness layer was created for minimization problems, while our work is about maximization problems. However one of the important characteristics of the permutation is that it preserved the optimal solution. In order to adapt the layer to maximization problems and to retain all the properties we used the symmetrical permutation to the one proposed in the original W-Model.

Layer parameters A weakness of the W-Model is that comparing a fitness functions for two dimensions might be irrelevant. For example if a function has for ruggedness the parameter $\gamma = 150$, for dimension 20, such function should have a deceptive landscape. however for dimension 40 only a bit of ruggedness is introduced.

The number of possible configurations has a polynomial growth $O(n^5)$ according to the dimension n. In order to compare functions of the W-Model with same parameters but different dimensions, we set all parameters of the W-Model relative to the dimension. If a problem has parameter k for a layer of the W-Model whose minimal configuration is $c_{min}(n)$ and maximal configuration $c_{max}(n)$ for dimension n, then the real parameter of the layer is $c_{min}(n) + \text{Integer}(k * (c_{max}(n) - c_{min}(n)))/100)$. This modification makes it possible to compare configurations on different dimensions by transforming the parameters so that they retain a sense between different dimensions.

2.2 Algorithms studied

For our experiments, the performances of nine algorithms were evaluated. All algorithms start with randomly initialized solutions. For reasons of space, this section presents only a short presentation of the algorithms.

Algorithms were mainly selected because they are among the most classics and most studied metaheuristics.

- One Hill Climber : a randomized Local Search algorithm. Starting from a random solution, the algorithm flips one uniformly chosen bit in each iteration and keep the best so far solution.
- Two Hill Climber : Similar to One Hill Climber, the only difference is that the algorithm flips two bits at each iteration.
- $(1 + 1)$EA : (1+1) evolutionary algorithm with static mutation rate $p = 1/n$. This algorithm mutate every bit in its solution with probability p to create the offspring.

- $(1+10)$EA : the classical (1+10)EA with static mutation rate $p = 1/n$ and 10 offsprings.
- $(1+2)$EA : (1+2) evolutionary algorithm, in which two offspring are generated with a self-adjusting mutation rate.
- $(1+10)$EA : variant of the (1+10)EA with self-adjusting offspring size.
- $(1+(\lambda,\lambda))GA$: a self adjusting crossover based EA with five hyper parameters $\alpha = 1/2$, $\beta = 2$, $\gamma = 1/2$, $A = \frac{3}{2}^{1/4}$, $B = 2/3$, analysed in [6].
- $(1+1)$FGA : the Fast Genetic Algorithm is similar than $(1+\lambda)$EA but chooses the mutation length according to a power-law distribution.
- Random Search : solutions are chosen uniformly at random.

2.3 Implementation

The implementation of the W-Model has been integrated in IOHprofiler [7], a tool for analysing and comparing iterative optimization heuristic. IOHprofiler is composed of two components : IOH-experimenter is used to perform experiments over an algorithm and IOH-analyser is used for detailed evaluation statistics. This tool is built on the COCO software [9] adjusted for discrete optimization problems.

Originally IOHexperimenter contains a series of suggested benchmark functions called PBO suite. With the implementation of the W-Model, we added a second suite of benchmark functions in the IOHprofiler framework. This new suite is a modular collection of problems for generating W-Model problems from on a list of configurations. To ensure that the implementation works properly, a test code makes it possible to compare the results of the implemented W-Model (language C) with that of origin (Java) proposed by Weise.

2.4 Experimental Settings

We have performed a comprehensive set of experiments for all algorithm introduced in section 2.2. Our experiments cover 5 dimensions $n \in \{16, 20, 32, 40, 50\}$ for 2646 fitness functions, with Dummy Random Variables from 0 to $0.2 \times n$, neutrality 0 to $0.2 \times n$, epistasis 0 to $0.25 \times n$. As ruggedness presents different characteristic based on its parameter, we computed experiment of 20 values of ruggedness splits among ruggedness set of feasible parameters.

In order to avoid configurations which reduce to much the dimensions (layers Random Dummy Variables and Neutrality), we set a rule to skip a configurations if $(100 - \text{Dummy})/\text{Neutrality} \leq 5$. Such configurations significantly reduce the fitness range of values. They often have the effect of simplifying problems and improving the re-sampling strategies in algorithms.

All configurations have been performed for all previous algorithms with 20 runs per couple (Algorithm, W-Model parameters) and we set a maximal budget of $1000 \times n$ function evaluations.

2.5 Algorithms performances

For all algorithm and all fitness function f, the data obtained make it possible to determine the mean ERT (number of function evaluation) so that algorithms find a solution of $target \times f(\text{best})$, where target is an approximation ratio of the optimal solution. We are particularly interested in solutions approximating at least 0.9 the optimum. Analyzes of the results on the layers of the W-Model are in agreement with those obtained by Weise [23].

To begin the analysis of the computed data, it is interesting to observe and compare results of Hill Climber. These algorithms perform a local search and they vary by the size of their steps (number of bit flipped at each iteration). They have the particularity of getting stuck in local optima. Although in practice these algorithms are not very efficient on all configurations of the W-Model, the empirical data confirm the expected results. Not having the ability to perform mutations of more than 1 bit, One Hill Climber fails to solve problems as soon as they are neutral, epistatic or rugged. However Two Hill Climber manages to find optimal solutions when a short amount of neutrality, epistasis or ruggedness is introduced.

In the same way, each algorithm has configurations for which its performance sets it apart from the rest. Especially $(1+,(\lambda,\lambda))$GA (called $(1+(lda_l da))$ in Figure 4) has a surprising aspect. The algorithm proves to be the most efficient for a large number of configurations when the objective is to find exactly the optimal solution. In contrast if the objective is to find a good approximation of the optimal solution, $(1+,(\lambda,\lambda))$GA is significantly slower than other algorithms. This observation is justified by the ability (observed empirically) of the algorithm to find the optimum for deceptive problems or when the problems are strongly epistatics.

Generally (1+1)EA, (1+2)EA, $(1+,(\lambda,\lambda))$GA, and (1+10)EA (both version) are algorithms that struggle in solving neutral objective functions. As for neutrality, epistasis makes optimization problems complex. This effect is observed for algorithms with few offspring such as (1+1)EA and (1+2)EA but not for those with a larger number of offspring like $(1+10)EA_s$ and $(1+,(\lambda,\lambda))$GA. The problem hardness also increase with ruggedness and deceptiveness until most algorithm doesn't not succeed in finding the optimal solution.

Two algorithms stand out of the set. Random Search showed unexpected performance. In fact, this basic algorithm is not influenced by ruggedness, deceptiveness or epistasis. In addition, the results have been shown to peak efficiency for neutral configurations.

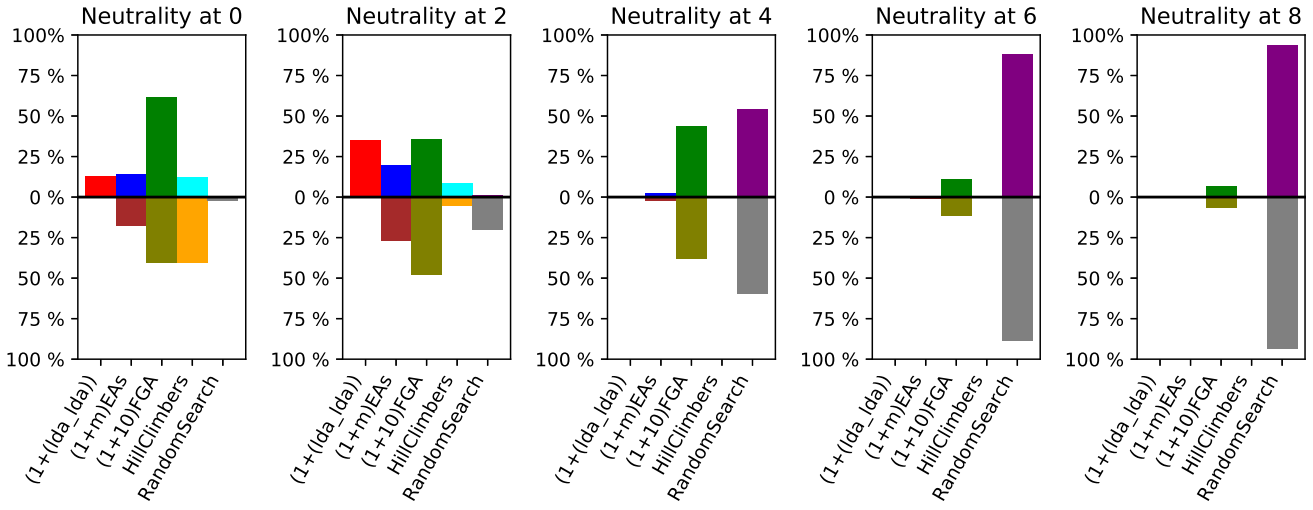

Figure 4: Percentage of the number of configurations in which algorithms are the most efficient at different level of neutrality for dimension 40 for target $1 \times Opt$ above the abscissa and target $0.90 \times Opt$ below the abscissa.

For the original problem, the rule that assigns a default value of 1 when as many bits are set to 1 as bits set to 0, had no effect in average. As the problem was to find the solution with minimal Hamming distance with the bit string 0101...0101 of length n, half of the time, draw situations reduce the fitness by 1 and in the other case, it increase fitness by 1.

However this rule has an unwanted impact on the OneMax problem. As the problem seeks to maximize the number of 1 in the bit string, draw situations constantly increase the fitness. As a result algorithms with a re-sampling strategy, especially Random Search, have improved performance for small dimensions such as Figure 4.

The second algorithm that stands out is (1+10)FGA proposed by [8, 21] and considered later in this document as the Single Best Solver in this document. The empirical results show good performances for neutral, epistatic and rugged problems.

Overall, the three originally proposed layers in the W-Model strongly influence the problem hardness. On the other hand, the variable Random Dummy layer seems to have a minor or non-existent impact for the algorithms.

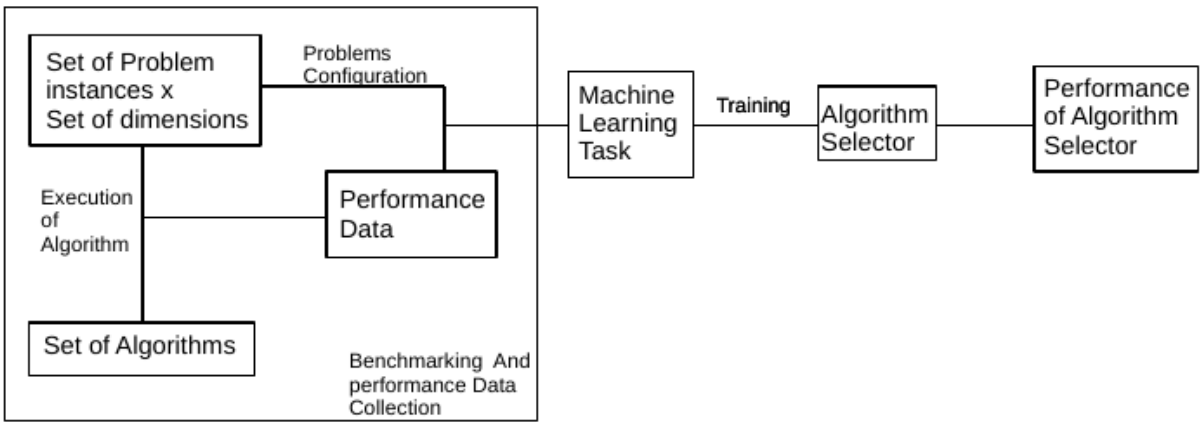

Figure 5: Schematic overview of a simplified learning model for the Algorithm Selection Problem

3 Using the W-Model parameters as feature

This section is an empirical study of the Algorithm Selection for the W-Model with a simplified performance prediction model. In section 3.1, we introduce how the model works, its steps and encountered problems. In the following section 3.2 we discuss about the experimental settings. Finally, the results are presented and discussed in section 3.4. The novelty brought by this internship is that the similar works relate to the continuous domain, sometimes the continuous domain and some discrete variables or are specific to NP-hard problems. Therefore, the first task is to confirm that the benchmark problems are apt to lead to a performance prediction system. This section focuses on the effectiveness of predictions by considering the parameters of the W-Model as exact and free features. Another interesting aspect of this model is that it serves as a reference for comparison to the final model.

3.1 Performance Prediction Model with Knowledge of Model Parameter

3.1.1 Algorithm Selection

Let a set of algorithm Alg and A_1, A_2,..., A_k its elements. Instead of using features, we define a W-Model P_θ problem by its parameters $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$.

As said in section 1.3, the objective is to find the best algorithm $A^*_{P_\theta}$ which minimize $\tau(c(A^*_{P_\theta}, P_\theta))$. The Algorithm Selection Problem can be rewritten as :

$$A^*_{P_\theta} = \min_{A_i \in Alg} \tau(c(A_i, P_\theta)) \tag{5}$$

3.1.2 Performance Prediction Model

To build a prediction model, we must already have computed the cost $\tau(c(A^*_{P_\theta}, P_\theta))$ of all algorithms for all problem instances. One of the future goals would be to reduce the number of problems instances that are required by performing a clustering job on the W-Model as this first step of computation has a huge cost.

At first, we only predict the performances of the algorithms according to the parameters of the W-Model as in Figure 5. These parameters act as a description of the difficulty of the problems since each layer of the W-Model is a transformation that complicates the original problem (OneMax). The negative point of this method is that these parameters only exist for functions belonging to the W-Model and are not known in the black-box settings. In section 4, we will discuss how to compute an approximation of the hardness for all objective functions. This new approximation will then be used as problem description in the second performance prediction model.

3.2 Experimental Settings

3.2.1 Learning Method

Different studies have focused on which learning method is most effective for the Algorithm Selection problem [1],[16]. The main techniques fall into two categories: the classifications, the objective is to predict the best algorithm, it is not necessarily necessary to predict the performances of all the algorithms and the regressions which aim at predicting the performances of all the algorithms for a given problem. More than the exact values of the predicted performances, it is the rank of the algorithms that is important.

Another question is what to predict. Different models of prediction have been studied. Some predict directly the rank of the algorithms, other the difference of ERT for each pair of algorithms.

In this project, we used Random Forest Regression to directly predict the performance of algorithms. Some tests were done with Random Forest Classification to predict the algorithm with the best performances but because of the first results, we preferred to use Random Forest Regression. In the continuity of the project, it would be interesting to make a comparison study between the different methods.

The model has been programmed in Python3 using the Scikit-Learn library for the regression with defaults hyper-parameters.

3.2.2 Training and Cross Validation

As with every machine learning project, one of the crucial tasks is to establish a set to train the model as well as to define a validation procedure for the model. One of the difficulties encountered is that it is difficult to create the training set. The computed functions of the W-Models have for 5 different possible values for the layer Dummy random Variables, 5 values for neutrality, 6 values for the epistasis and 20 for the ruggedness. As some reduce the size of the problem too much, we have obtained 2646 functions over 5 dimensions which give 13230 configurations $(function, dimensions)$ in our data collection. This point highlights one of the future works to be done. Performing a clustering task according to the performances of the algorithms on the functions of the W-Model will establish a selection of interesting functions, not according to their parameters of the W-Model but according to the empirical results previously obtained.

To avoid this problem, we choose uniformly at random configurations that are part of the training set. As this method sets aside a large number of configurations, those that have not been selected are assigned in a test set. We studied results for different training set size.

Machine learning project usually use a leave-one-out procedure as model validation technique. However we construct our training test such that we dispose plenty of left-out functions. In order to avoid any bias, every left out functions of the training set is used for testing the model. We made the model predict performances for each and assessed the results by comparing the best predicted algorithm with the best from the data collection.

3.3 Measure of Quality

3.3.1 Quantity of Successful Tests

Each configuration of the W-Model is used, either in the training set or in the testing set. The first possible measure to confirm the good functioning of the performance prediction model is simply to evaluate the ratio of successful tests to the number of tests performed. However this raises an important question: is the test only successful when the performance prediction model predicts the best algorithm according to the data for the tested configuration. It is common for a configuration to have several algorithms whose performances are of the same order of magnitude. Another possibility is to define a successful test when the model predicts an algorithm "almost" as effective as the best one.

Let C_{best} the ERT of the best algorithm and $C_{predicted}$ the ERT of the best predicted algorithm (ERT from the data collection, not the actual value predicted by the model) for a configuration. The condition of success can be written as if $\alpha \times C_{best} + \beta \geq C_{predicted}$ then the prediction is successful.

The nuance is that if such a model commits many errors but of small scales then it will show better results than a model that makes few errors but whose unsuccessful tests predict an algorithm considerably longer than the best. Two arguments can be put forward in favor of this definition: algorithms are stochastic and the data is empirical therefore not perfect. Hence, it is difficult to consider an algorithm as more efficient

than a second one when the difference in number of function evaluations is small. It also makes it possible to evaluate the number of "serious" errors of the model by counting only the predictions in which the algorithm predicted is clearly less effective than the best one.

3.3.2 Per Configuration Satisfaction

We used a second way of measuring the Satisfaction of a unique test. Let $ERT^{i,n}_{Best}$ the average ERT of the best algorithm for a function i at dimension n and $ERT^{i,n}_{Pred}$ the average ERT of the predicted algorithm over the same function. We define the following Per Configuration Satisfaction (for a single test):

$$S_{i,n} = \frac{ERT^{i,n}_{Pred} - ERT^{i,n}_{Best}}{ERT^{i,n}_{Best}} \tag{6}$$

Given a problem instance, if the model predicts the best algorithm then $S_{i,n} = 0$. The bigger $S_{i,n}$ is, the more important is the error. This measurement makes it possible to determine the configurations difficult to predict whatever which functions is in the training set.

3.3.3 Single and Virtual best Solver

A third notion used to determine the quality of a performance prediction model can be seen in [16, 13]. Let C be the set of all configurations (W-Model parameters $\times$ dimension) and ERT^{c}_{Alg} the average time for an algorithm to solve the configuration $c \in C$.

$$V_{Alg} = \sum^{c \in C} ERT^{c}_{Alg} \tag{7}$$

We define SBS the Single Best Solver, the best algorithm in average for all configuration.

$$SBS = \frac{\min_{\forall Alg}(V_{Alg})}{|C|} \tag{8}$$

$$\tag{9}$$

VBS the Virtual Best Solver, The time used by a perfect prediction model to solve all configuration.

$$VBS = \frac{\sum^{c \in C} \min_{\forall Alg}(ERT^{c}_{Alg})}{|C|} \tag{10}$$

Rs the Real solver, from the performance prediction model.

$$Rs = \frac{\sum^{c \in C} ERT^{c}_{BestPredicted}}{|C|} \tag{11}$$

We defined the Merit M of a performance prediction model.

$$M = \frac{Rs - VBS}{SBS - VBS} \tag{12}$$

Merit can be used as an indicator of learning performance. In the ideal case, when the model does not make prediction errors, the merit is 0. When the merit is between 0 and 1, the model is more efficient than the Single Best Solver.. On the other hand, for any merit value $M > 1$, the model makes too many errors and is less efficient than the Single Best Solver. This measurement directly gives information as to whether it is preferable to use the best algorithm in general or whether to use the performance prediction model to determine the best suited algorithm.

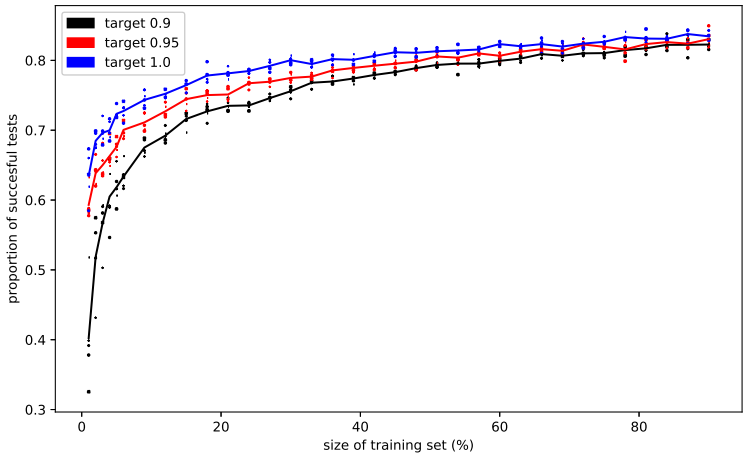

Figure 6: Number of successful tests according to the size of the training set for targets at 0.90,0.95,1.00 $\times Opt$

3.4 Results

Figure 6 shows the evolution of the number of successful tests according to the size of the training set. The smaller the training set is, the more difficult it is to predict other problems effectively. These data were obtained by running 5 times the learning model for each size of the learning set. This figure also shows that a size of 15% of problems (randomly selected) is enough to get 75% of correct predictions for target 1.00 and 0.95. Let $C_{predicted}$ be the cost of the predicted algorithm and C_{best} the cost of the best algorithm for a problem instance (cost in number of function evaluations). In this plot, a test is successful test if $1.05 \times C_{best} + 5 \geq C_{predicted}$. This size of 15% achieve a Merit of 0.30 in average for target at 1.0 and 0.95. Table 1 displays the mean satisfaction for all configuration. The mean have been computed over ten runs of the leaning performance prediction model with independent random training set. The first column corresponds to models in which we predict algorithms to optimize the cost to find $0.90 * Opt$ where Opt is the best solution for the configuration. In the second column we predict algorithms to minimize the cost for finding the optimal solution.

It shows that less than 5% of configurations (W-Model parameters,dimension) are difficult to predict independently of the training set. We have identified these configurations by selecting different orders. The second line of plots show the configurations ordered by increasing neutrality then increasing epistasis and as we can see, the model does not predict efficiently some configurations with low neutrality and low epistasis. In the same way deceptive ruggedness leads the model to make mistakes, especially when the target is the optimal solution. These difficult-to-predict configurations append to be configurations with none to low epistasis, none to low neutrality, and high ruggedness (deceptive landscape).

In the bottom-left plot, all dimensions contains configurations difficult to predict. Two explanations are possible. On one hand, having no better easily identifiable algorithm, the system does not learn correctly to predict for these configurations. The performances of the algorithms are not representative of their efficiency and they have not been calculated on a sufficiently large set In fact, when the target is to easy to find, some algorithms succeed in finding it at similar costs. In the other hand, by mixing some layers of the W-Model, complexity effects difficult to predict may appear. These effects may be poorly represented by the W-Model parameters.

When we increase the target, almost all the configurations that the model can not predict have dimensions 16 or 20. This observation argues that the behavior of the small functions and which reduce the dimensions by neutrality is different from their behavior for larger dimensions. After an investigation of these functions, it turned out that Random Search outperforms the other algorithms on the functions that reduce the dimension (neutrality and random dummy variable layers). Fortunately, its effectiveness decreases considerably when the dimension grows.

Although the validity tests were made only from problems belonging to the W-Model, this empirical study confirms that the W-Model is adapted to serve as a reference for the Algorithm Selection Problem.

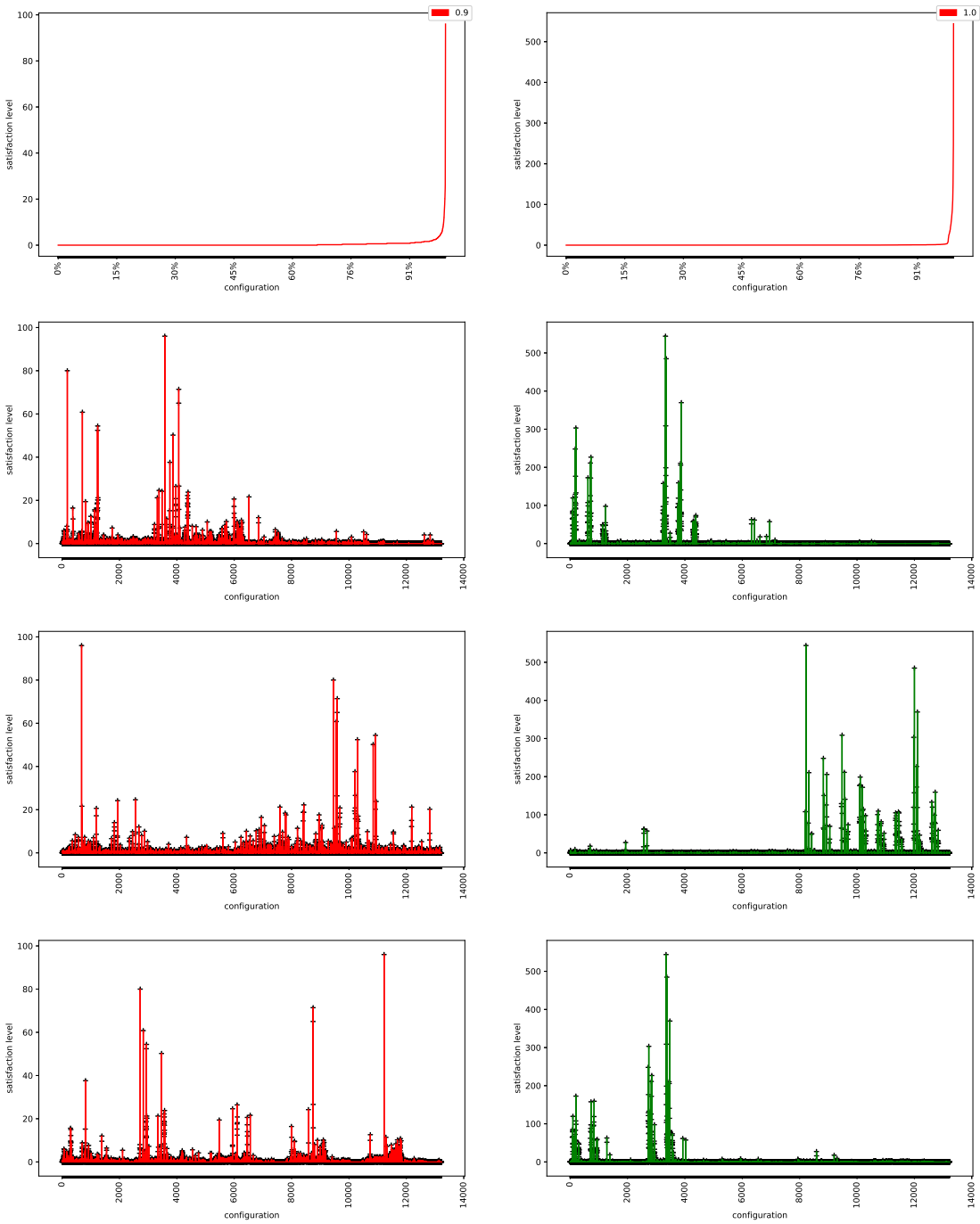

Table 1: Mean satisfaction for all configuration at target 0.90 (left column) and target 1.00 (right column). The first line plot configuration ordered by their satisfaction. In the second line, configurations are ordered by their neutrality parameter, the third line by their ruggedness and the last is ordered by dimension.

4 Algorithm Selection using Landscape Features

4.1 Motivation

This chapter is an empirical study of the Algorithm Selection for the W-Model. We previously introduced a first performance prediction model which use algorithms performances and W-Model parameters to determine the best suited algorithm for a function. However, these parameters are not known for a problem that is not part of or that can not be created by the W-Model.
The Algorithm Selection and its other variants (Algorithm Configuration, PIAC) is based on features that describe the problem properties [19, 20].

These features aim to describe the fitness landscape of the problem. Functional features are already

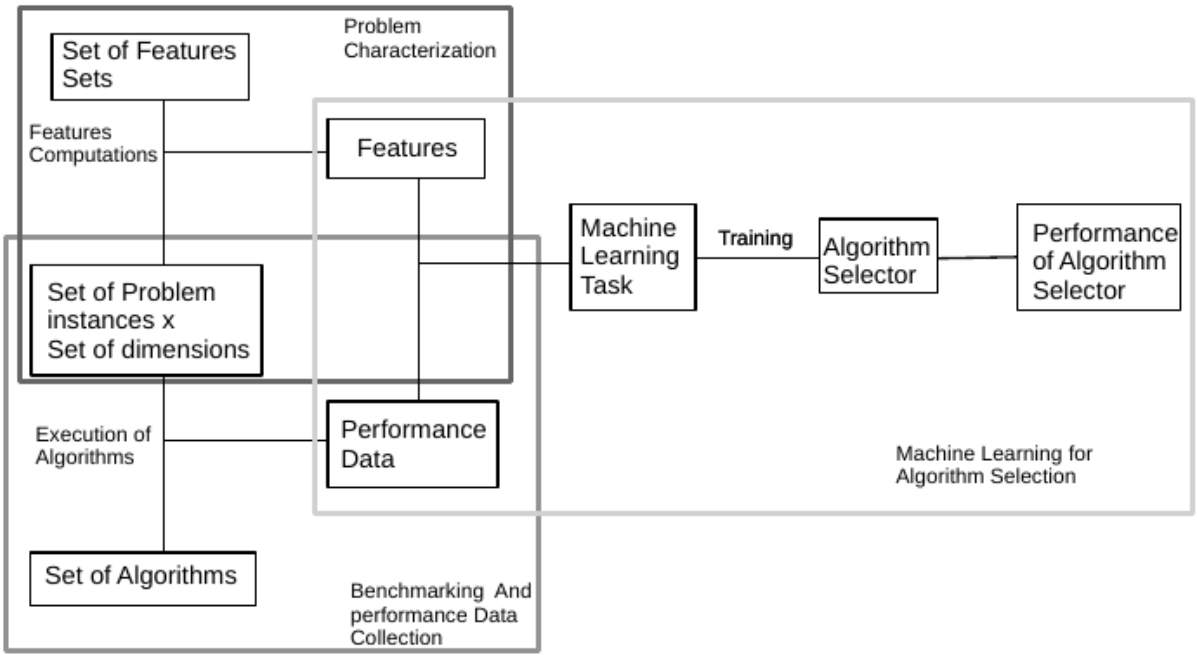

Figure 7: Schematic overview of the performance prediction model for the Algorithm Selection Problem

commonly used, especially for continuous optimization problems or for some specific NP-hard problems. The delay of the discrete domain with respect to the continuous domain is explained by the fact that there is already several ways to analyse the landscape for continuous functions. However, features are statistics values computed from a sample of functions and some of them are compatible with the discrete domain.

4.2 Performance Prediction Model

The main difference with the previous prediction model is the use of features instead of W-Model parameters. For the performance prediction model to be realistic, we can not continue to use the parameters of the W-Model. On the one hand, they only make sense for the functions belonging to the W-Model and the prediction system must work for all functions in the black box. On the other hand, the W-Model parameters describe the expected theoretical complexity of the function. But the model must predict for functions that are unknown to it and whose complexity is unknown.
A new preliminary step is necessary for this model. Features need to be computed for all functions. This is another step that proves to have a huge computation cost, as will be described in the next section where we will detail the features used, each feature has its own more or less long computing time. This issue introduces a problem of balance between the quality and the effectiveness of the features.
An overview of the performance prediction model can be seen Figure 7. Let Alg a set of all algorithms and A_1, A_2,..., A_k its elements. We define a problem P_F by its features $F = (f_1, f_2, ..., f_k)$ and the objective is to find the best algorithm $A^*_{P_F}$ such that $A^*_{P_F} = \min_{A_i \in Alg} \tau(c(A_i, P_F))$.

4.3 Landscape Analysis And Features Sets

As mentioned section 1.4, Exploratory Landscape Analysis (ELA) aims to specify problems hardness with numerical features. Features are often grouped by set of techniques (similar calculations) or objectives (eg evaluating the neutrality) however they are not necessarily understandable intuitively. There have been successful attempts of using features for Algorithm Selection. Different features have been proposed in the literature [19, 3, 18, 22] to characterize black-box problems, most of them for continuous optimization. They requires sets of samples for their estimation.

In this section, we briefly present the three features sets used in the work, namely three Distribution features, nine Meta-Model features and 15 Level-Set features. They are computed using the R package FLACCO made publicy available by Pascal Kerschke at https://github.com/flacco.

Level-Set Level-Set has been proposed by Mersmann [19]. This feature set divide samples into two parts and train different classification techniques such as LDA (Linear Discriminant Analysis), QDA (Quadratic Discriminant Analysis) and MDA (Mixture Discriminant Analysis) to compute 15 numerical features. Classifiers are trained to predict if a fitness value fall below or exceeds given thresholds. Features are a summary of statistic of the resulting cross validation mean misclassification errors for each classifier.

Meta-Model The Meta-Model feature set studies the property of an unknown problem using regression methods to approximate the objective function from a sample of candidate solutions. Computing meta-models (linear, quadratic and mixture regression models) for the initial samples of evaluated solutions, the coefficient of determination R^2 is used as measurement of the accuracy of the model correlation. This feature set approximates the problem structure with an analytical function, identifying relationship between the variables and reflecting the difficulty of the problem.

Y-distribution This feature set contains three features. It computes the kurtosis, skewness and number of peaks of the kernel-based estimation of the density of fitness sample. This set gives an overview of the "shape" of the search space. The kurtosis measures the sharpness of a probability distribution. This value makes it possible to identify the neutral functions. High kurtosis indicates that the fitness landscape contains neutral areas.

Skewness measures the asymmetry of a probability distribution. In addition to kurtosis, this feature improves the precision of the landscape estimation according to the samples.

Number of peaks determines the ruggedness of the landscape. Given the probability density of the fitness in the sample, it determines the number of peaks. This estimation observes the presence of plateau and gives a deeper insight into the landscape

4.3.1 Features Computation

According to the literature, a sample of size 500 $\times$ dimension is enough to have a correct estimation of the features for mixture of continuous and discrete optimization problems. Although for all features the computation time is dependent on the size of the sample, the complexity in time of their calculation varies. We had to compute features for more than 13000 functions. To obtain this data, we drastically reduced the sample size to 10 $\times$ dimension. This sample is sufficient to have a satisfactory result at first.

In our experiments, we considered features computation time as negligible. This choice is explained because it is above all a priority to show that performance prediction models work and that the W-Model can be used as benchmark problem for discrete optimization. Used features set were considered cheap in [1]. However this consideration is false in practice. The Y-distribution feature set is a really cheap, but Level-set cheapness can be discussed. The Meta-Model Set has a computation time increasing strongly with the sample size, and it gives in practice an unrealistic cost as a cheap features set.

4.4 Results

We conducted experiments similar to those of the previous model and the results have several points in common.

We can observes similarities between predictions from W-Model parameters and predictions from features. First, we compared the number of successful tests according to the size of the training set. In order to make correct comparison to be correct, as for the previous model, a test is considered successful if $1.05 \times C_{best} + 5 \geq C_{predicted}$.

Figure 8 concludes that 65% of tests can be successful if the size of the training set is greater than 20%. In the previous prediction model a size of 15% was enough to achieve 75% of successful test for all targets. This difference in performance confirms the veracity of the previous hypotheses. Either the features sets wrongly represent the functions of the W-Model. (At least not as well as the W-Model parameters). Either some samples are not representative of their objective function.

Despite the drop in performance, the model remains justified globally with only 10% of configurations difficult to predict, see Table 2. the remaining 90% of configurations are at the same level of satisfaction as the previous prediction model. We can therefore conclude that even if the features set used are not

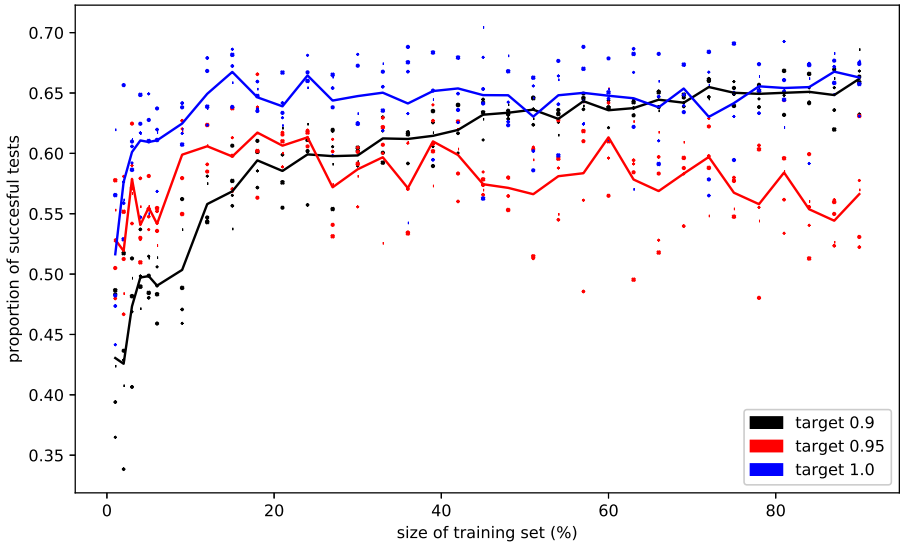

Figure 8: Number of successful tests according to the size of the training set for targets at 0.90,0.95,1.00 $\times Opt$

the most suitable for W-Model, they nevertheless allow to partially describe the configurations. Table 2 is the same as table 1. It shows the mean satisfaction for all configuration of the performance prediction model (features-based predictions) at target 0.90 (left column) and target 1.00 (right column). The mean satisfaction was computed from 10 runs (training + testing) with a training set of size 20%. The first line has configurations ordered by satisfaction, the second by neutrality, the third by ruggedness and the last by dimension. This table makes it possible to compare predictions from the parameters of the W-Model to those made from the features by comparing which problems are difficult to predict.

Globally, features partially succeed in replacing the parameter of the W-model. The presence of peaks in the graphs show that as for the first model, the configurations difficult to predict for target 1.00 have low neutrality, low epistasis, low dimension and high ruggedness, but also low ruggedness unlike the first model. This difference is explained as before, it is hard for the features to detect ruggedness since they are computed from a random sampling and not by walks on the fitness landscape.
For target 0.95, results are not clear. On one hand hard-to-predict configurations are similar to those of the previous model. On the other hand performances according to the size of the training set show that the system is less efficient than for the other targets.

Using the performance prediction model is nevertheless a better solution than using the best algorithm (single best solver). For a training set of size 20%, the average Merit of the performance prediction model (computed over 30 runs) is 0.36 for target 1.00 against 0.30 previously, which confirms the slight decline in results from the ideal to the real prediction model. Once again, the variance from one run to another greatly influences Merit. On the sample of 30 learning and testing the Merit is between 0.22 and 0.5.

The first remark is that the variance of the results between different training set is much more important for predictions from features than for prediction from W-Model parameters. This variance is reflected in particular in the Figure 2. Few possible explanations may be the origin.

Features were computed from random samples of points, therefore, adding a non-deterministic step to the prediction model. If the training set contains too many configurations (W-Model parameters $\times$ dimensions) whose features were computed from non representative samples, learning may require more configurations or worse be distorted by “bad” features.

Features sets are often divided into two categories. Efficiency features and those whose purpose is to help understand the nature of the problem. The sets used were chosen for their performance as well as their compatibility with the discrete optimization problems. Although they were also used in some works for continuous and discrete optimization. There is usually a bias between the dimensions used for problem of the continuous and discrete domain compared to the discrete domain. For example Belkhir in his thesis, [1], was interested in dimensions (2,3,5,10) while in our case, the results suggest that the smaller dimensions

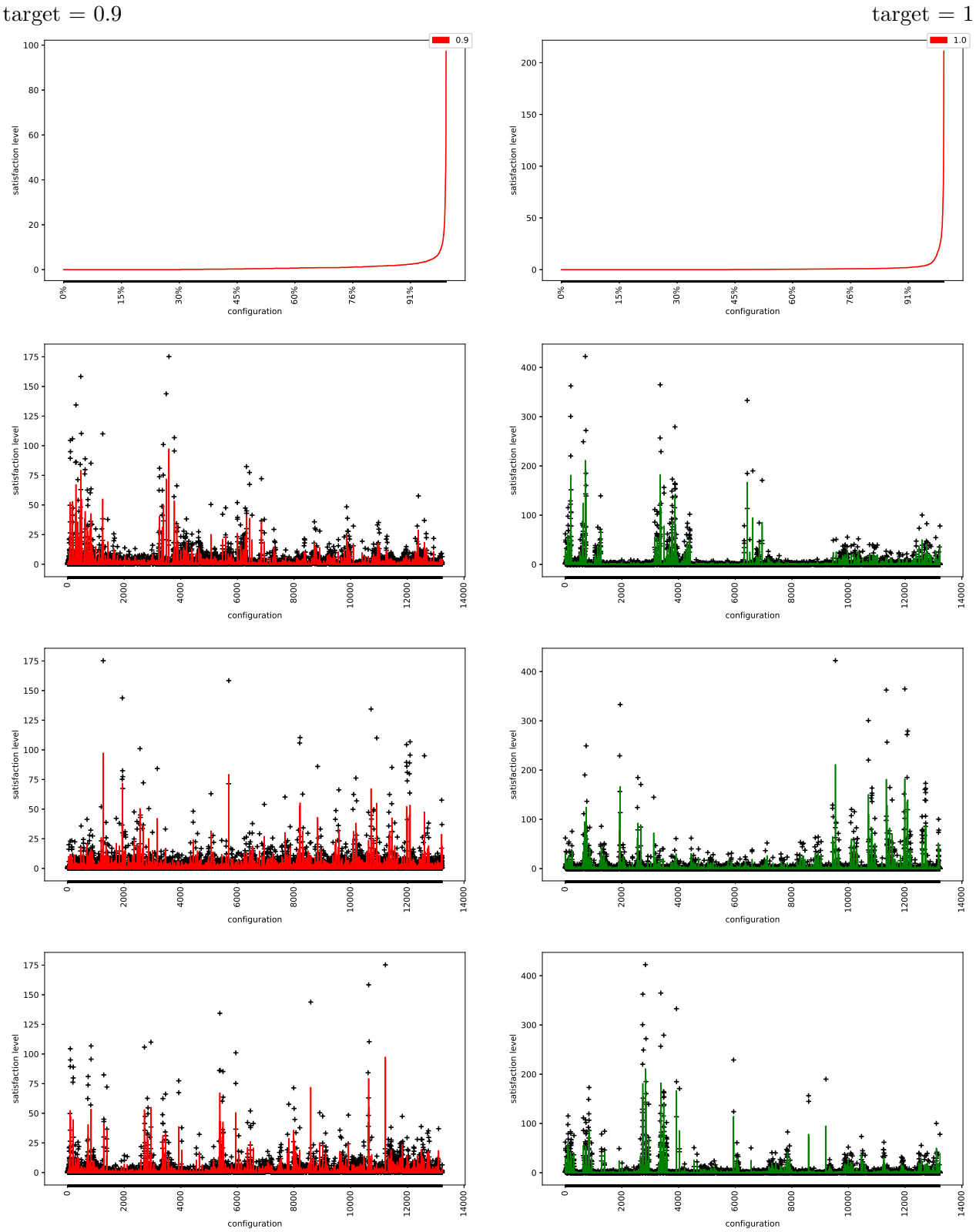

Table 2: Mean satisfaction for all configuration at target 0.90 (left column) and target 1.00 (right column). The first line plots configuration ordered by their satisfaction. In the second line, configurations are ordered by their neutrality parameter, the third line by their ruggedness and the last is ordered by dimension.

(16,20) complicates the prediction. A strongly probable hypothesis is that these features set do not constitute a good enough description of objective functions.

One last possible explanation is simply that the sample used to compute features is too small. As a result, the features are not completely representative of the configurations. Of course, the variance of its results

can come from these three remarks. However, we can deduce two important points to study later : the evolution of the performance according to the size of the sample of a prediction model and compare different features set in terms of their impacts on predictions, their costs as well as different sampling methods.

5 Conclusion

5.1 Summary of the Contributions

During this internship, we set up and validate a methodology for the Algorithm Selection in discrete black-box optimization.Our approach retains all genericity and can work for any choice of algorithms, features or learning methods.

The use of the W-Model as a benchmark problem made it possible to build a large data set containing performances of algorithms necessary for our approach. The performances obtained, computed according to a limited budget, show that algorithms efficiency vary according to the complexity of the objective functions and the strategies of the algorithm. Algorithms Performances which reflect the scalability complexity of objective functions, attest the W-Model as potential Black-box Discrete Optimization Benchmark.

Following the general experimental protocol, we validated the Algorithm Selection for discrete optimization. The performance prediction model showed very good results by selecting the most appropriate algorithm for each instance of problem, under conditions that the cost of the features is considered as cheap or free. These results are particularly promising when the dimension is large, when the empirical performances of the algorithms are clear which facilitates the selection of a more adapted algorithm.

The comparison of the results between the predictions with knowledge of the W-Model parameters and those with Landscape features attest the proper functioning of the approach and ascertain that some features initially conceived for the continuous domain work for the discrete domain.

We released a C implementation of the W-Model integrated to IOH-profiler as well as a Python3 implementation of the performance prediction model at `https://github.com/rf-csn/DBBO_AlgorithmSelection`. As additional contribution, we provide performance data required to use the prediction model.

5.2 Perspectives and Future Work

Several directions for future research are highlighted in this work. One of the most important objectives is to assess the performance prediction model as a state of the art optimizer, performing well for discrete optimization problems. To achieve this goal we must study the real gain of the model without considering features as free data.

Many research steps are needed to design a powerful optimizer. The first of these tasks is the development of a more precise and efficient set of features. We can distinguish three objectives of this stage: knowledge, precision and optimization. Finding intuitive and understandable features would be ideal to gain a better understanding of the complexity of objective functions and help determine which functions are well covered by the W-Model to complement it as needed. The accuracy of features is obviously another important goal. Although the features set used generally give a good estimate of the problem, some are still poorly estimated by the features. Of course the features must be computed in a negligible time and from small data samples to make a powerful optimizer. This stage is complex due to the conflict of interest between optimization and accuracy. However, we have a trail for a set of features dedicated to objective functions in black boxes for the discrete domain. ParadisEO [10] is a framework to design meta-heuristics. It that contains intuitive tools for analysing fitness landscapes devoted to discrete optimization.

To achieve our goal of evaluating the performance prediction model as a state of the art optimizer, expand our data collection by selecting the best performing algorithms in the literature and under different parameters would make predictions more interesting. Although our current collection already contains some popular algorithms, we have restricted their settings. Completing this data collection would improve the real gain of the performance prediction model by predicting algorithms specific to certain classes of objective functions but more efficient.

Performing a clustering work according to the efficiency of the algorithms on W-Model configurations would also improve the efficiency of the performance prediction model. Understanding which configurations are interesting (according to the data obtained and not according to the parameters of the W-Model) makes

it possible to strictly reduce the size of the performance data by selecting only objective functions necessary for learning.

Finally, a last crucial step is to perform cross validation tests with real-world problems therefore outside the W-Model. This step will give an insight into the actual gain provided by the optimizer.

Acknowledgements

I wish to thanks my supervisors Carola Doerr and Thomas Weise for their regular monitoring and their genuine help. This work was supported by the Paris Ile-de-France Region.

Appendices

References

[1] Nacim Belkhir. *Per Instance Algorithm Configuration for Continuous Black Box Optimization*. PhD thesis, 2017. 2017SACLS455.

[2] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Frechette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237, 06 2015.

[3] Manuel Clergue, Philippe Collard, Marco Tomassini, and Leonardo Vanneschi. Fitness distance correlation and problem difficulty for genetic programming. pages 724–732, 01 2002.

[4] Heather J. Cordell. Epistasis: what it means, what it doesn't mean, and statistical methods to detect it in humans. *Human Molecular Genetics*, 11(20):2463–2468, 10 2002.

[5] Yuval Davidor. Epistasis variance: A viewpoint on ga-hardness. 1:23 – 35, 1991.

[6] Benjamin Doerr and Carola Doerr. A tight runtime analysis of the (1+(λ, λ)) genetic algorithm on onemax. [6], pages 1423–1430.

[7] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *CoRR*, abs/1810.05281, 2018.

[8] Carola Doerr, Furong Ye, Sander van Rijn, Hao Wang, and Thomas Bäck. Towards a theory-guided benchmarking suite for discrete black-box optimization heuristics: profiling (1 + λ) EA variants on onemax and leadingones. pages 951–958, 2018.

[9] Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *CoRR*, abs/1603.08785, 2016.

[10] Jérémie Humeau, Arnaud Liefooghe, El-Ghazali Talbi, and Sébastien Verel. Paradiseo-mo: From fitness landscape analysis to efficient local search algorithms. Research Report RR-7871, INRIA, 2013.

[11] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, and Thomas StÃ$\frac{1}{4}$tzle. Paramils: An automatic algorithm configuration framework. *J. Artif. Intell. Res. (JAIR)*, 36:267–306, 01 2009.

[12] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *CoRR*, abs/1811.11597, 2018.

[13] Pascal Kerschke, Lars Kotthoff, Jakob Bossek, Holger Hoos, and Heike Trautmann. Leveraging tsp solver complementarity through machine learning. *Evolutionary Computation*, 26, 08 2017.

[14] Pascal Kerschke, Lars Kotthoff, Jakob Bossek, Holger H. Hoos, and Heike Trautmann. Leveraging tsp solver complementarity through machine learning. *Evolutionary Computation*, 26(4):597–620, 2018.

[15] Motoo Kimura. Evolutionary rate at the molecular level. *Nature*, 217:624–626, 1968.

[16] Lars Kotthoff, Pascal Kerschke, Holger Hoos, and Heike Trautmann. Improving the state of the art in inexact tsp solving using per-instance algorithm selection. pages 202–217, 01 2015.

[17] John R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, Jun 1994.

[18] K. M. Malan and A. P. Engelbrecht. Ruggedness, funnels and gradients in fitness landscapes and the effect on pso performance. pages 963–970, June 2013.

[19] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 829–836, New York, NY, USA, 2011. ACM.

[20] M. A. Munoz, M. Kirley, and S. K. Halgamuge. Exploratory landscape analysis of continuous space optimization problems using information content. *IEEE Transactions on Evolutionary Computation*, 19(1):74–87, Feb 2015.

[21] Eduardo Carvalho Pinto and Carola Doerr. Towards a more practice-aware runtime analysis of evolutionary algorithms. *CoRR*, abs/1812.00493, 2018.

[22] Colin R. Reeves and Christine Wright. Genetic algorithms and the design of experiments. 1996.

[23] Thomas Weise and Zijun Wu. Difficult features of combinatorial optimization problems and the tunable w-model benchmark problem for simulating them. pages 1769–1776, 2018.

[24] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[25] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *CoRR*, abs/1111.2249, 2011.

Algorithms Pseudo-Codes

This section contains pseudo-code of algorithms used in our work. Two operator filp_l and cross_c are used in some of them. The mutation operator filp_l is chooses l different positions of a bit string x and flips bits in these positions.

Algorithm: filp_l

Input: Sample $x \in \{0,1\}^n$, $l \in \mathbb{N}$;
Select l **different** positions $i_1, ..., i_l$ uniformly at random;
$y \longleftarrow x$;
for $j = 1, ..., l$ **do**
  $y_{i_j} \longleftarrow 1 - x_{i_j}$;

.1 Hill Climber

Hill Climbers algorithms flips k bits randomly chosen at each iteration until being stuck in local optimum.

Algorithm: Hill climber

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$
Optimization: *for* $t = 1, 2, 3, ...$ **do**
  $x^* \longleftarrow x$;
  flip k bits in x^*;
  if $f(x^*) \geq f(x)$ **then**
    $x \longleftarrow x^*$;
  end

.2 EA with static mutation rate

This pseudo-code correspond to the classical $(1 + \lambda)$EA with static mutation rate.

Algorithm: $(1 + \lambda)$EA with static parameters

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$
Optimization: *for* $t = 1, 2, 3, ...$ **do**
  for $i = 1, ..., \lambda$ **do**
    Sample $l^{(i)} \sim \text{Bin}_{>0}(n, 1/n)$;
    Create $u^{(i)} \longleftarrow \text{flip}_{l^{(i)}}(x)$, and evaluate $f(x^*)$;
  $x^* \longleftarrow \max\left\{f(y^{(1)}, ..., f(y^{(\lambda)}\right\}$;
  if $f(x^*) \geq f(x)$ **then**
    $x \longleftarrow x^*$;
  end

.3 EA with self-adjusting mutation rate

The $(1 + \lambda)$EA with normalized standard bit mutation samples the mutation strength from a normal distribution with mean $r = pn$ and variance $pn(1-p) = r(1-r/n)$. The parameter r is updated at each iteration. We used $r^{\text{init}} = 2$.

Algorithm: $(1 + \lambda)$EA with normalized standard bit mutation

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$;
$r \longleftarrow r^{\text{init}}$;
Optimization: *for* $t = 1, 2, 3, ...$ **do**
  for $i = 1, ..., \lambda$ **do**
    Sample $l^{(i)} \sim \min\{N_{>0}(r, r(1 - r/n)), n\}$;
    Create $u^{(i)} \longleftarrow \text{flip}_{l^{(i)}}(x)$, and evaluate $f(x^*)$;
  $x^* \longleftarrow \max\{f(y^{(1)}, ..., f(y^{(\lambda)}\}$;
  $r \longleftarrow H_d(x, x^*)$;
  if $f(x^*) \geq f(x)$ **then**
    $x \longleftarrow x^*$;
  end

.4 EA with self-adjusting offspring size

The $(1 + \lambda)$EA with self-adjusting offspring size double λ if all offspring have a fitness value smaller than the parent. Otherwise λ is divided by s.

Algorithm: $(1 + \lambda)$EA with self-adjusting offspring size

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$
Optimization: *for* $t = 1, 2, 3, ...$ **do**
  for $i = 1, ..., \lambda$ **do**
    Sample $l^{(i)} \sim \text{Bin}_{>0}(n, 1/n)$;
    Create $u^{(i)} \longleftarrow \text{flip}_{l^{(i)}}(x)$, and evaluate $f(x^*)$;
  $x^* \longleftarrow \max\{f(y^{(1)}, ..., f(y^{(\lambda)}\}$;
  $s \longleftarrow \sum_{i=1}^{\lambda} 1$ if $f(y^{(i)} \geq f(x)$ otherwise 0;
  if $s > 0$ **then**
    $\lambda \longleftarrow \lambda / s$;
  else
    $\lambda \longleftarrow \lambda * 2$;
  end
  if $f(x^*) \geq f(x)$ **then**
    $x \longleftarrow x^*$;
  end

.5 Fast Genetic Algorithm

The Fast Genetic Algorithm (FGA) chooses the mutation strength according to a power-law distribution $D^{\beta}_{n/2}$. We use the (1+1) variant of this algorithm with $\beta = 1.5$.

Algorithm: $(1 + \lambda)$GA

Initialization: Sample $x \in \{0,1\}^n$ uniformly at random and evaluate $f(x)$
Optimization: *for* $t = 1, 2, 3, ...$ **do**
  for $i = 1, ..., \lambda$ **do**
    Sample $l^{(i)} \sim D^{\beta}_{n/2}$;
    Create $u^{(i)} \longleftarrow \text{flip}_{l^{(i)}}(x)$, and evaluate $f(x^*)$;
  $x^* \longleftarrow \max\{f(y^{(1)}, ..., f(y^{(\lambda)}\}$;
  if $f(x^*) \geq f(x)$ **then**
    $x \longleftarrow x^*$;
  end

.6 Self-Adjusting $(1 + (\lambda, \lambda))$GA

The $(1+(\lambda, \lambda))$GA is a crossover based algorithm wich update the offspring population size at each iteration according to a strength parameter $F = 3/2$.

Algorithm: Self-Adjusting $(1 + (\lambda, \lambda))$GA

Initialization: Sample $x \in \{0, 1\}^n$ uniformly at random and evaluate $f(x)$

Optimization: $t = 1, 2, , 3...$ **do**

  Mutation phase: Sample $l^{(i)} \sim \text{Bin}_{>0}(n, \lambda/n)$;

  for $i = 1, ..., \lambda$ **do**

    Create $y^{(i)} \longleftarrow \text{flip}_{l^{(i)}}(x)$, and evaluate $f(x^*)$;

  $x^* \longleftarrow \max\left\{f(y^{(1)}, ..., f(y^{(\lambda)}\right\}$;

  Crossover phase: $i = 1, ..., \lambda$

  create $y^(i) \longleftarrow \text{cross}_c(x, x^*)$ and evaluate $f(y^{(i)})$;

  $y^* \longleftarrow \max\left\{f(y^{(1)}, ..., f(y^{(\lambda)}\right\}$; **Selection phase:**

  if $f(y^*) > f(x)$ **then**

    $x \longleftarrow y^*$;

    $\lambda \longleftarrow \max\{\lambda/F, 1\}$;

  end

  if $f(y^*) = f(x)$ **then**

    $x \longleftarrow y^*$;

    $\lambda \longleftarrow \min\left\{\lambda/F^{1/4}, 1\right\}$;

  end

  if $f(y^*) < f(x)$ **then**

    $\lambda \longleftarrow \min\left\{\lambda/F^{1/4}, 1\right\}$;

  end

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

$y \longleftarrow x$;

Sample $l \sim \text{Bin}_{>0}(n, c)$;

Select l different position $\{i_1, ..., i_l\}$;

for $j = 1, ..., l$ **do**

  $y_{i_j} \longleftarrow 1 - x^*_{i_j}$;