Landscape-Aware Selection of Metaheuristics for the Optimization of Radar Networks

Quentin Renau

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAX002

Thèse de doctorat

Landscape-Aware Selection of Metaheuristics for the Optimization of Radar Networks

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École Polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 24/01/2021, par

QUENTIN RENAU

Composition du Jury :

Marc Schoenauer Directeur de Recherches, INRIA Saclay - Île-de-France (TAU)	Président du jury
Bilel Derbel Maître de Conférences, Université de Lille (CRIStAL)	Rapporteur
Günter Rudolph Professeur, Technical University of Dortmund (LS XI)	Rapporteur
Rémy Chevrier Chef de projets R&D, SNCF Innovation & Recherche	Examinateur
Claudia D'Ambrosio Directrice de Recherches, CNRS et École Polytechnique (LIX)	Examinatrice
Benjamin Doerr Professeur, École Polytechnique (LIX)	Directeur de thèse
Carola Doerr Chargée de recherche, CNRS et Sorbonne Université (LIP6)	Co-directrice de thèse
Johann Dreo Ingénieur-Chercheur, Institut Pasteur	Invité
Yann Semet Ingénieur-Chercheur, Thales Research & Technology	Invité

Contents

Acknowledgments

I am thankful to my supervisors Benjamin Doerr, Carola Doerr and Johann Dreo for these three years. Thank you for your guidance, your advice and for cheering me up in hard times. I had a lot of fun working with you during my PhD. I would also like to thank Yann Semet for doing his best entering the supervision team after two years.

I would like to thank Bilel Derbel and Günter Rudolph for agreeing to be reviewers of my manuscript. Also, I would like to thank Claudia D'Ambrosio, Rémy Chevrier and Marc Schoenauer for being part of my jury.

I would like to address special thanks to Alain Peres and Stéphane Brisson. Thank you Alain for all that you have done on the application part of this thesis which could not have existed without you. Thank you for still coming in our meetings and still showing a great interest even if we did not talk about radars. Thank you Stéphane for all that you have done for me so that I could work in a less difficult environment. 80% of data presented in this thesis would not have existed without your help.

I am grateful to Claire Laudy, Nicolas Museux, Cédric Buron, Yann Briheche and, Simon Fossier for their help with my work, our coffee breaks and discussions. I am grateful for all gaming sessions organized by Bruno Marcon or Martin Krejca which really helped me escaping my work.

Thanks to all my fellow PhD students: Anja, Roman, Douae, Erwann, Paul, Adam, Océane, Mara and, François. I started this journey at the same time as most of you and it is been a real pleasure to do it with you.

Thanks to all members of the labs in Thales, LIX and LIP6 that I may have forgot. I really enjoyed working with all of you.

Finally, I would also like to thank my family who was always supportive whatever I was doing during my studies.

Synthèse

Contexte

Des réseaux de radars sont positionnés en vue de former des barrières de détection dont l'efficacité doit être optimisée. Cette efficacité peut se visualiser en mesurant la couverture radar ou la probabilité de détection d'une cible. L'optimisation des réseaux de radars doit prendre en compte des contraintes systémiques et environnementales telles que des zones à défendre ou des zones d'exclusion.

Un réseau est composé de plusieurs radars. Ces radars diffèrent par leur type et possèdent plusieurs paramètres à configurer. Optimiser un réseau de radars, c'est donc choisir une localisation et des paramètres internes du radar. En plus de ces paramètres, il est possible d'avoir pour certains radars des contraintes supplémentaires telles que des zones interdites dans le domaine ou une obligation d'intervisibilité entre radars. L'optimisation de la configuration de réseaux de radars est donc un problème complexe, non-convexe et non-différentiable ce qui rend très difficile sa description analytique.

La résolution de ce problème s'appuie sur *l'optimisation boîte noire*. Le problème n'est pas explicitement donné, mais les valeurs de la fonction objectif f sont accessibles en utilisant un simulateur dont l'entrée est une solution x représentant une configuration du réseau.

Dans la littérature et dans un contexte industriel, le développement et l'emploi des métaheuristiques ont été grandement étudiés pour résoudre des problèmes d'optimisation boîte noire continus. Les métaheuristiques sont des méthodes stochastiques et itératives ne nécessitant pas d'informations *a priori* sur le problème à résoudre. Ces méthodes n'ont pas de garanties d'optimalité, toutefois elles se sont révélées performantes pour différents types de problèmes, de la résolution de problèmes d'embouteillages [BSF+21] au design d'ARN [MS21].

De nombreuses métaheuristiques ont été développées et montrent des performances complémentaires sur différents types de problèmes. Un exemple de cette complémentarité est notamment visible pour des algorithmes de recherche locale. Ces derniers sont souvent très efficaces et atteignent des solutions de bonne qualité sur des problèmes convexes, mais la recherche locale peut obtenir de faibles performances sur des problèmes complexes. De fait, choisir l'algorithme le plus adapté pour un problème donné est compliqué. Les algorithmes ayant des forces et des faiblesses complémentaires, choisir le bon algorithme afin de résoudre un problème d'optimisation est une tâche difficile et cruciale.

Résultats clés

L'objectif de cette thèse est d'appliquer une méthode de sélection automatique de métaheuristique guidée par le paysage de recherche à un problème de configuration de réseau de radars. Les travaux de cette thèse sont divisés en deux parties, une première partie concernant l'étude des mesures permettant de caractériser un problème d'optimisation boîte noire. La deuxième partie concerne l'application de la sélection automatique de métaheuristique guidée par le paysage de recherche au problème radar.

L'étude des mesures permettant de caractériser un problème d'optimisation boîte noire nous a menée à définir six propriétés et à étudier si les mesures satisfaisaient ces propriétés. Ces mesures sont obtenues à partir d'un échantillonnage et les six propriétés définies correspondent à :

Expressivité : capacité d'une mesure à différentier plusieurs problèmes d'optimisation ;

Stabilité : capacité d'une mesure à garder des valeurs similaires lors de différents échantillonnages indépendants de même taille ;

Sensibilité au bruit : capacité d'une mesure à garder des valeurs similaires lorsque que l'échantillon de départ est modifié par du bruit ;

Robustesse : capacité d'une mesure à garder des valeurs similaires lorsque la taille de l'échantillonnage est réduite ;

Sensibilité à la méthode d'échantillonnage : capacité d'une mesure à garder des valeurs similaires lorsque différentes méthodes d'échantillonnage sont utilisées ;

Sensibilité aux transformations : capacité d'une mesure à garder des valeurs similaires lorsque que des transformations sont appliquées à la fonction ou aux échantillons.

L'étude des mesures montre qu'aucune d'entre elles ne satisfait les six propriétés. Contrairement aux préconisations dans la littérature, nous avons remarqué que la méthode d'échantillonnage importait car une grande partie des mesures y est sensible. Nous avons trouvé d'importantes différences entre les distributions des mesures provenant de différentes méthodes d'échantillonnage. De plus, il semblerait qu'échantillonner avec la méthode de Sobol´ donne de meilleures performances en terme d'expressivité.

La résolution du problème de configuration de réseau de radars par des métaheuristiques a révélé que la différence de performance entre le meilleur algorithme sur les instances d'entraînement (SBS) et le sélecteur parfait (VBS) était faible. De fait, il est difficile d'obtenir des performances supérieures au SBS. L'application de la sélection automatique de métaheuristique guidée par le paysage de recherche a montré des performances similaires au SBS.

Les problèmes radars considérés dans cette thèse sont intentionnellement simplifiés. Les modèles très réalistes comportent plus de paramètres et des fonctionnement radar plus complexes. Sur ces modèles plus complexe, la complémentarité entre les différentes métaheuristiques pourrait être accrue et donc cette complémentarité impliquerait un plus grand écart VBS-SBS.

Part I

Introduction

Chapter 1

Introduction

Contents

1.1 Motivation

Radar networks are placed to form detection barriers with the goal to detect targets. The efficiency of these barriers, *i.e.*, the network coverage or the detection probability has to be optimized taking into account several systemic and environmental constraints such as areas to defend and exclusion areas in the domain.

A radar network is composed of several radars. The radars may differ in type, and are typically configurable, *e.g.*, one classical parameter is the angle between the horizontal plane and the antenna axis, which is called *tilt*. Optimizing a radar network therefore compromises the choice of the locations of the radars and their specific configuration. On top of that, several operational constraints can be defined for the network such as intervisibility between radars or forbidden locations in the domain. The number of parameters and these non-linear constraints cause the problem to be non-convex, non-differentiable and too complex to be analytically described.

Solving the radar network configuration problem therefore relies on *black-box optimization*, where the problem is not explicitly modeled and can be accessed only through a simulator. The quality of a configuration is assessed through simulations which assign a value to each solution candidate x. In black-box optimization, this assignment is called a *fitness function* f.

In the literature and in industrial context, metaheuristics have been widely investigated and used to solve numerical black-box optimization problems. Metaheuristics are stochastic methods that do not need *a priori* information on the function to solve. Even if these methods have no optimality guarantee, they have been well performing on different types of problems, from traffic congestion [BSF+21] to RNA design [MS21].

Over the years, many metaheuristics have been developed but, a complementarity between algorithms have been observed, *i.e.*, no algorithm will perform best on all possible optimization problems. Local search algorithms and methods that estimate the gradient typically perform well

on convex problems, but may perform very badly on more complex problem types. Therefore, choosing the right algorithm to solve an optimization problem is a challenging task. As solvers have complementary strengths and weaknesses, finding the good algorithm to solve the optimization problem at hand is crucial.

1.2 Automated Algorithm Selection

Metaheuristics are classically introduced as frameworks within which a user can gather some modules to instantiate an algorithm. For instance, the *design* of an evolutionary algorithm requires to choose the population size, the variation and selection operators in use, the encoding structure, fitness function penalization weights, etc. This highly flexible design of metaheuristics allows for efficient abstractions but comes at the burden of having to solve an additional (meta-)optimization problem.

The impressive advances of machine learning (ML) techniques are currently shaking up literally every single scientific discipline, often in the function to support decisions previously requiring substantial expert knowledge by recommendations that are derived from automated data-processing techniques. Computer science is no exception to this, and an important application of ML is the selection and configuration of metaheuristics [HKV19, KHNT19, SM09], where automated techniques have proven to yield tremendous efficiency gains in several classic optimization tasks, including SAT solving [XHHL11], AI planning [VHCM15], and Traveling Salesperson Problem solving [KKB$^+$18].

In the context of numerical optimization, supervised learning approaches are particularly common for the automated selection of algorithms [KT19a, BDSS17, MSKH15]. These methods often build on features developed in the context of *fitness landscape analysis* [ME13, PA12], which aims at quantifying the characteristics of an optimization problem through a set of features. More precisely, a *feature* maps a function (the optimization problem) to a real number. Such a feature could measure, for example, the *skewness* of f, its *multi-modality*, or its similarity to a quadratic function. In black-box optimization, the feature values need to be approximated from a set of $(x, f(x))$ pairs. The approximation of feature values through such samples is studied under the notion of *exploratory landscape analysis* (ELA [MBT$^+$11]). ELA has been successfully applied, for example, in per-instance hyperparameter optimization [BDSS17] and in algorithm selection [KT19a].

1.3 Our Key Findings

In this thesis, we define six properties that feature should satisfy to be efficient:

- *stability*, *i.e.*, ability of a feature to keep the same values with independent samples of same size;
- *robustness to noise*, *i.e.*, ability of a feature to keep the same values with when uniform noise is applied to original samples;
- *robustness to the sampling strategy*, *i.e.*, ability of a feature to keep the same values with different sampling strategies;
- *expressiveness*, *i.e.*, ability of a feature to distinguish between different optimization problems;
- *robustness*: ability of a feature to keep the same values with different number of samples;

- *transformation invariance*, *i.e.*, ability of a feature to keep the same values when transformations are applied to the fitness function or to the samples.

We found that none of the feature analyzed fully satisfies the six properties. One of the six properties is the invariance to the sampling strategy. We found that, surprisingly to what was recommended in the literature, the sampling strategy actually matters. We found important discrepancies in the feature values computed from different sampling strategies. In particular, distributions of nearest better clustering feature values computed with different sampling strategies are non-overlapping. Overall, feature values based on Sobol´ sampling have a better expressiveness.

We analyze the performance of 13 algorithms on two radar use-cases. Among the 13 algorithms, we consider five variants of the CMA-ES [vRWvLB16], Particle Swarm Optimization (PSO) [KE95] or the Nelder-Mead [NM65]. The use-cases are solved for two budgets of function evaluations. A small budget of 500 function evaluations and a large budget of 2,500 function evaluations. In our analysis, we found a complementarity of algorithms depending on the budget, *i.e.*, different algorithms are performing best depending on the number of function evaluations. Considering CMA-ES variants, we found that elitist variants are more suitable for low budget settings while non elitist variants are performing better when the number of function evaluations available is larger.

On the two use-cases, we found that the difference between the single best solver (SBS[1]) and the virtual best solver (VBS[2]) is actually quite small, and thus, it is difficult to obtain better performances than those of the SBS. Yet, we found that the landscape-aware algorithm selection has similar performances to the single best solver (SBS).

1.4 Thesis Outline

This thesis is composed of four parts. Part I introduces the context of the thesis.

Part II surveys the background on black-box optimization and landscape-aware selection of metaheuristics.

Part III presents and analyzes the properties of landscape features. These landscape features and their properties constitutes the central part of a landscape-aware selection of metaheuristics.

In Part IV, we describe the optimization of radar networks. We define here the main use-cases of the thesis, we describe the results of different metaheuristics and compare them to the performance of a landscape-aware selection of algorithms.

We conclude this thesis in Part V with an overview of results from this thesis and an outlook for promising research directions.

Within the four parts, this thesis is composed of ten chapters. Chapter 2 introduces black-box optimization and presents a list state-of-the-art algorithms for numerical black-box optimization problems. The algorithms in this chapter are those that will later be used in this thesis.

In Chapter 3, the algorithm selection problem is introduced. We also introduce a method to tackle this problem, the landscape-aware algorithm selection. The landscape-aware algorithm selection allows to link data obtained by algorithms presented in Chapter 2 and feature data presented in Chapter 4 with the objective to find the best suited algorithm.

Chapter 4 gives an introduction of continuous optimization problems' properties and ways to describe them. This chapter introduces both high-level properties that defines optimization problems and also low-level, cheap methods to gain insight on an optimization problem at hand. We focus particularly in this chapter on the *exploratory landscape analysis* technique. This technique

[1] Best solver on training instances
[2] Perfect selector, *i.e.*, select the best solver on each instance

will be used in the thesis in order to characterize problems characteristics. In this chapter, we also introduce new features based on existing ones in the Principal Component Analysis feature set (see Section 4.2.6).

Chapter 5 focuses on the computation of cheap problem features in order to efficiently characterize the optimization problems. This chapter defines six properties of features that are important to take into account when features are computed in order to maximize their efficiency. We also analyze in this chapter to what degree landscape features satisfy these properties.

Chapter 6 describes the basic principles of radar operations. This chapter contains some key radar equations and processing that will influence the objective function.

Chapter 7 introduces the components of the radar network configuration problem such as radar parameters and the target model. Chapter 7 also introduces the use-cases and the objective function that will be used in this thesis. We also introduce here the key functionalities of our Ægis framework: aggregation of radars into networks, constraint handling, handling of geographical data, etc.

Chapter 8 presents the results of a portfolio of optimization algorithms on the use-cases. This chapter also compares the performances of optimization algorithms to manually designed solutions. We show in this chapter that human-designed solutions are worse than the best performing algorithms and are close to the performances of random search.

Chapter 9 empirically investigates the application of landscape-aware algorithm selection on the radar network configuration problem and compares it to classical state-of-the-art solvers. This approach has similar performances to the single best solver on our use-cases.

We conclude this thesis with Chapter V with a summary of our results. We also share some promising research directions such as dynamic algorithm selection or artificially created benchmark instances for real-world applications.

Part II

Landscape-Aware Selection of Metaheuristics

Chapter 2

Continuous Black-Box Optimization Algorithms

Contents

In this chapter, we present the general context of this thesis, *i.e.*, black-box optimization. We also present a list of algorithms that are often used to solve black-box optimization problems. These algorithms will be used later in this thesis in order to solve a real-world radar configuration problems (see Chapter 8).

In Section 2.1, we present the context of black-box optimization. Section 2.2 introduces the paradigm of evolutionary computation while Sections 2.3 to 2.7 describe the different optimization algorithms that will be used later in this thesis.

2.1 Black-Box Optimization

The general context of this work is that of *black-box optimization* where the goal is to minimize an objective function on a continuous search space

$$f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R}.$$

No information is known about the characteristics of the objective function f (e.g., separability, differentiability, smoothness, ...) in black-box optimization. The only information available is the value of $f(x)$ given a solution $x = (x_0, x_1, \ldots, x_d) \in \Omega \subseteq \mathbb{R}^d$. Through the optimization process, the goal is to find one or multiple as good as possible solutions using as few evaluations as possible.

Many complex problems or real-world problems do not have any known analytic expression of their objective function. The values of the objective function are only reachable through experimentation or simulation. Black-box optimization is used to find sufficiently good solutions within a reasonable number of evaluations.

In order to find these solutions, metaheuristics were developed throughout the years. Within metaheuristics, evolutionary computation techniques are widely used and proven effective. In the next sections, the paradigm of evolutionary computation is described along with all the algorithms that will be used in this thesis.

2.2 Evolutionary Computation

2.2.1 History

The idea of Evolutionary Computation (EC) came up first in the 1940s as Turing proposed a "genetical or evolutionary search". In the 1960s, three different interpretations of this idea were simultaneously developed: Fogel in the US in the 90s introduced Evolutionary Programming [Fog98] while Holland in the 70s presented Genetic Algorithm (GA) [Hol73]. Meanwhile, in Germany, Rechenberg and Schwefel introduced in the 60s Evolution Strategies (ES) [RTE65, Sch65]. In the 1990s, these three interpretations were merged into one technology called Evolutionary Computation while a fourth inspiration had emerged: Genetic Programming (GP) [Koz94].

2.2.2 Metaphor from Biology

As the name Evolutionary Computation suggest, bio-inspired algorithm are developed with the main inspiration coming from the natural evolution as described by Darwin. Solution candidates of a *population* are generated in an *environment* and endeavor to survive. Their ability to survive, *i.e.*, their *fitness* is determined by the environment. The process of solving a problem results in a stochastic trial-and-error where a population is evolved in order to fit the environment best. This phenomenon is coined *survival of the fittest*. Fitter solution candidates survive and reproduce when the others die. As in the Darwinian theory, some modifications can occur through reproduction.

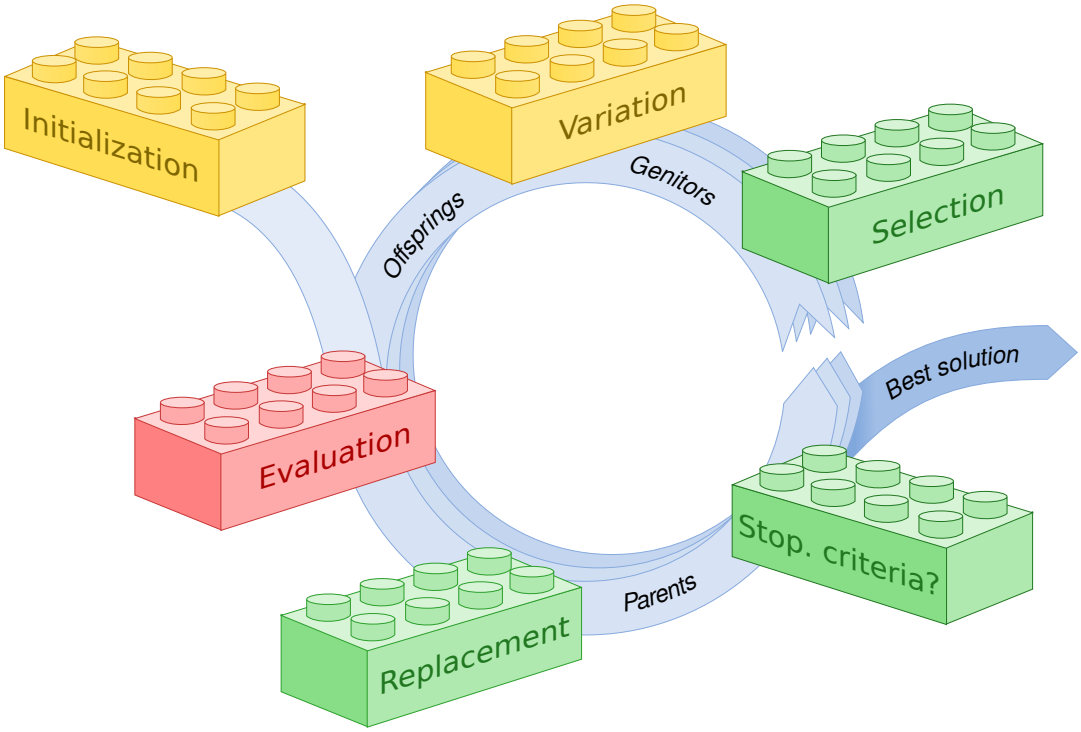


Figure 2.1: General diagram of an Evolutionary Algorithm, taken from [DLV+21]. Red and yellow blocks are encoding-dependent while green blocks are encoding-independent.

As the evolution process goes by, the constitution of the population is modified towards the fittest solution candidates. This process is captured by the analogy of a *landscape*. On this landscape, d dimensions represents the biological traits of the solution candidates when an additional dimension represents the fitness of each candidate solution. Hence, each peak in the landscape represents an area of fit trait combinations.

These analogies correspond to classical optimization concepts. Each solution candidates are a candidate solution and the population represents a set of these candidates solutions. The environment of these solution candidates is represented by the search space $\Omega \subseteq \mathbb{R}^d$. The quality of the solution candidates $x \in \Omega$ is assessed by the fitness function $f : \Omega \to \mathbb{R}$ which is named the objective function in classical problem solving.

2.2.3 Canvas of an Evolutionary Algorithm (EA)

There are multiple branches in EC but it is possible to define a general scheme that correspond to every evolutionary algorithm. A general diagram of an EA can be found in Figure 2.1. By definition and construction, EAs are are stochastic and population-based, *i.e.*, they process a collection of candidate solutions concurrently. The following sections present the main building blocks of an EA.

2.2.3.1 Initialization

In most EAs, the initialization is kept simple as a population is often generated at uniformly random. Quasi-random initialization procedure are also possible such as latin hypercube sampling [MBC79] or low-discrepancy sequences such as Sobol´ [Sob67] or Halton [Hal64] sequences. If one possesses knowledge on the underlying problem to solve, a heuristic could be applied in order to start with a population with greater fitness.

2.2.3.2 Evaluation

The evaluation part is where the fitness function is considered. The candidate solutions are evaluated. This part forms the basis for all the following blocks as all they will all need the results from the evaluation part.

2.2.3.3 Replacement

The replacement operator permits to update the actual population. At this stage, quite often, the number of solution candidates may exceed the authorized population size. Hence, the solution candidates that will form the next generation have to be chosen in both the offsprings and/or parents solution candidates. The ones not selected will die while the others become the parents of the next generation. Different replacement strategies have been introduced such as generational replacement (*e.g.*, no parents survive), steady-state replacement (*e.g.*, one offspring replace the worst parent in the population), and all possibilities between these two strategies.

2.2.3.4 Stopping Criterion

Choosing a stopping criterion may depend on the knowledge available on the problem at hand. If there is a known global optimum x^*, a stopping condition may be obtaining a solution x with fitness[1] value $f(x) \leq f(x^*) + \varepsilon$ for some user-defined $\varepsilon > 0$. This stopping criterion is coined *fixed-target*. If no global optimum is known, which is often the case for real-word problems, there exist two main categories of stopping criteria: one based on time, the other based on fitness. The classical criteria used are:

- cap the CPU computation time;
- cap the number of fitness evaluation;
- fitness improvement is under a threshold for too many generations;
- diversity of the population drops under a threshold;
- the population distribution is ill-formed.

Capping the CPU computation time or the maximum number of fitness evaluation is called *fixed-budget*.

2.2.3.5 Selection

Selection is a mechanism that permits to define the solution candidates of a population that will become the genitors of the next generation. That will determine the distribution from which the next solution candidates are sampled. This selection is often based on quality, *i.e.*, fitter solution

[1] to be minimized

candidates stand better chances to get selected than low quality solution candidates. Allowing to select low quality solution candidates prevent the algorithm getting stuck in local optimum. Different selection mechanisms can be used such as elitism (*i.e.*, only fittest candidate solutions are selected), deterministic or stochastic tournaments, random selection, rank-based selection (*i.e.*, solution candidates are selected using their rank, not their directly their fitness values) or roulette wheel (*i.e.*, the selection probability of a solution candidate is proportional to its relative fitness).

2.2.3.6 Variation

Variation operators aim at creating new solution candidates, *i.e.*, offsprings, from parents. These operators are of two types: *mutation* that are unary operators and *recombination* that are n-ary operators where n represent the number of solution candidates used for recombination.

Mutation A mutation is stochastic and unbiased transformation of a candidate solution. This operator has also a theoretical role as mutation can guarantee the ergodicity of the algorithm, *i.e.*, that the algorithm can visit the whole search space. However, it is important to note that this property is realized only when the mutation operator allows to jump anywhere in the search space with a non-zero probability which is not always the case for all mutation implementations. Mutation for continuous variables is often achieved by modifying randomly some candidate solutions in the population. Any probability distribution can be chosen to perform the modifications such as the uniform distribution or the normal distribution.

Recombination A binary operator is called recombination or crossover and merges two or more parents to create offsprings. As mutation, this operator can be deterministic or stochastic and the parts inherited from one or another parent are random. Different recombination operators are available such as one or two points recombination. One cutting point k is defined uniformly at random and applied to two parents x, y, *i.e.*, parents are divided in two $x = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_d)$ and $y = (y_1, \ldots, y_k, y_{k+1}, \ldots, y_d)$, d being the dimension of the problem. The new offspring z is created by taking elements from one or the other parent, *i.e.*, $z = (x_1, \ldots, x_k, y_{k+1}, \ldots, y_d)$ Another possibility is the uniform recombination, *i.e.*, the offspring is created by taken a coordinates of one or the other parent at uniformly at random.

2.2.4 Evolution Strategies

Evolution Strategies (ES) are probably one of the simplest form of EA. They are mainly used in continuous optimization and has one variation operator: a mutation operated by a multivariate Gaussian. As an example, Algorithm 1 shows the pseudo-code of one of the simplest evolutionary algorithm: the $(1+1)-ES$ where there is one parent and one offspring.

Algorithm 1 $(1+1)-ES$ for minimizing a black box problem $f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R}$.

```
Generate the first solution x randomly.
Set σ the step size and C the covariance matrix used in the normal distribution.
while Stopping criterion do
    Create y ← x + σ𝒩(0, C).                          ▷ Mutation
    if f(y) ≤ f(x) then                               ▷ Selection
        Replace x by y.
    end if
end while
```

Some parameters are involved in the multivariate Gaussian as the standard deviation or step-size σ and the covariance matrix $C \in \mathcal{M}_{d\times d}(\mathbb{R})$, d being the dimension of the problem. Algorithm 1 has a constant population of one candidate solution but more generic algorithms can have more parents, noted μ and/or more offsprings, noted λ. In ES, there can be two types of selection that can be described by a particular notation:

- $(\mu/\rho + \lambda) - ES$ where the μ new parents are selected among the best offsprings only. In this case, the fitness of the population is allowed to decrease;
- $(\mu/\rho, \lambda) - ES$ where the next generation of parents is created with the μ best solution candidates contained in the full population, *i.e.*, parents and offsprings. This configuration is called an *elitist* selection as the fitness values in the population can only increase. The $(1 + 1) - ES$ is the simplest example of an elitist ES where the best solution is kept for the next iteration.

The parameter ρ represents the number of parents used for recombination, *i.e.*, $\rho \leq \mu$ parents are used for recombination.

2.3 The Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

In this section, we present the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [HO01] algorithm and many of its variants. Section 2.3.1 introduces the algorithm while Section 2.3.2 presents the different variants.

2.3.1 Vanilla CMA-ES

The $(\mu/\mu_w, \lambda) - ES$ or more commonly called CMA-ES [HO01] is an evolutionary algorithm that was proposed by Hansen *et al.* in 2001. The parameter μ represents the number of parents, μ_w the number of parents used in recombination and λ the number of offsprings. CMA-ES uses a multivariate normal distribution $\mathcal{N}(m, C)$ to sample candidate solutions with m the mean of the distribution and C, the covariance matrix. In the algorithm, the solution candidates are ranked by fitness value, hence $x_{i:\lambda}$ is the i-th solution such that $f(x_{1:\lambda}) \leq \cdots \leq f(x_{\lambda:\lambda})$. The new candidate solutions, the λ offsprings, are generated with $x_{i:\lambda} \sim m + \sigma\mathcal{N}(0, C)$[2] where

- $m \in \mathbb{R}^d$;
- $\sigma \in \mathbb{R}^+$, the standard deviation or step-size;
- $C \in \mathcal{M}_{d\times d}(\mathbb{R})$ the covariance matrix, with d the dimension of the problem.

The goal of CMA-ES is to adapt the mean, the step-size and the covariance matrix in order to guide the search, which takes the form of a sequence of sampling that evolves toward best solutions.

2.3.1.1 Evolution Paths

In general, the information about the evolution of parameters are lost over several generations. The evolution path is a way to keep track of the evolution of a parameter over multiple generations. Evolution path is a sequence of the successive steps that gives information about the search.

[2] $\sim$ means "drawn" from

The length of the evolution path can be measured: a long path results in consecutive steps in the same directions whereas short paths results in steps that cancel out. This can be used to update parameters, consecutive steps in the same directions could be replaced by a longer step whereas steps that cancel out could be replaced by a shorter step [HO01]. The evolution path can also be expressed as the sum of the length of its consecutive steps. This process is called *cumulation* [OGH94].

2.3.1.2 Covariance Matrix Adaptation

Rank-one Update Rank-one is used to update the covariance matrix. The rank-one update adaptation modifies the covariance matrix by adding a rank one matrix. To do so, a weighted sum of the $y_{i:\lambda}$ is computed $y_w = \sum_{i=1}^{\mu} w_i y_{i:\lambda}$ with $y_i \sim \mathcal{N}(0, C)$, $x_i = m + \sigma y_i$, and using the μ best candidate solutions. The weights w_i are defined such as $w_1 \geq \cdots \geq w_\mu > 0$ and $\sum_{i=1}^{\mu} w_i = 1$.

The mean and the covariance matrix are then updated using y_w and a learning rate c_1 [AH21]

$$m \leftarrow m + \sigma y_w \tag{2.1}$$

$$C \leftarrow (1 - c_1)C + c_1 \mu_w y_w y_w^T, \text{where } \mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \geq 1. \tag{2.2}$$

An evolution path p_c can be computed and used to update the covariance matrix

$$p_c \leftarrow (1 - c_c)p_c + \sqrt{1 - (1 - c_c)^2}\sqrt{\mu_w} y_w \tag{2.3}$$

$$C \leftarrow (1 - c_1)C + c_1 p_c p_c^T. \tag{2.4}$$

Rank-μ Update Rank-μ update is an extension preferred for large population sizes and uses the weighted empirical covariance matrix C_μ to update the matrix C and the learning rate c_μ

$$C_\mu = \sum_{i=1}^{\mu} w_i y_{i:\lambda} y_{i:\lambda}^T \tag{2.5}$$

$$C \leftarrow (1 - c_\mu)C + c_\mu C_\mu. \tag{2.6}$$

The adaptation of the matrix have several properties [AH21]

- adaption of the covariance matrix learns pairwise dependencies between variables;
- adaption of the covariance matrix performs an iterative Principal Component Analysis (PCA) where the principal components are the eigenvectors of the covariance matrix;
- adaption of the covariance matrix updates of the mean and covariance matrix can be interpreted as gradient descent;
- the rank-μ update can increase the learning rate for large populations.

The main difference between rank-one and rank-μ updates is the number of vectors used to update the covariance matrix. In the algorithm, both rank-one and rank-μ updates can be combined as in Algorithm 2.

2.3.1.3 Step-Size Adaptation

On top of the adaptation of the covariance matrix, the step-size σ is also adapted during the search. The adaptation is done using an evolution path to decide if the step-size should increase or decrease based on its length. In order to decide if the evolution path is either too long or too short, the path length is compared to its expected length. The expected length is computed as every selection of solution candidate was made uniformly at random.

When the path is longer than expected, it means that most of the steps are in the same direction and so σ will be increased. Conversely, if the path is shorter, σ will be decreased. The evolution path p_σ and the update of the step-size are computed as [AH21]:

$$p_\sigma \leftarrow (1 + c_\sigma)p_\sigma + \sqrt{1 - (1 - c_\sigma)^2}\sqrt{\mu_w}y_w \tag{2.7}$$

$$\sigma \leftarrow \sigma \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma\|}{\mathbb{E}(\|\mathcal{N}(0, I)\|)} - 1\right)\right), \tag{2.8}$$

with I the identity matrix, $c_\sigma < 1$ such that $1/c_\sigma$ is the backward time horizon of the evolution path and d_σ the damping parameter.

2.3.1.4 Pseudo-Code of the CMA-ES Algorithm

By combining the evolution paths, rank-one and rank-μ updates and the step-size adaptation, the pseudo-code of the $(\mu/\mu_w, \lambda)$-CMA-ES algorithm is given by Algorithm 2.

Algorithm 2 CMA-ES algorithm [AH21] for minimizing a black-box problem $f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R}$.

Initialize $C = I$ and $p_c = p_\sigma = 0$.
Set $c_c, c_\sigma, c_\mu, c_1, d_\sigma$ and $w_{i=1,\dots,\lambda}$.
repeat
  for $i = 1, \dots, \lambda$ **do**
    $x_i \leftarrow m + \sigma y_i$, $y_i \sim \mathcal{N}_i(0, C)$ ▷ Sampling
  end for
  $m \leftarrow m + \sigma y_w$ ▷ Mean update
  $p_c \leftarrow (1 - c_c)p_c + \sqrt{1 - (1 - c_c)^2}\sqrt{\mu_w}y_w$ ▷ Covariance cummulation
  $p_\sigma \leftarrow (1 + c_\sigma)p_\sigma + \sqrt{1 - (1 - c_\sigma)^2}\sqrt{\mu_w}y_w$ ▷ σ cummulation
  $C \leftarrow (1 - c_1 - c_\mu)C + c_1 p_c p_c^T + c_\mu C_\mu$ ▷ C update
  $\sigma \leftarrow \sigma \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma\|}{\mathbb{E}(\|\mathcal{N}(0,I)\|)} - 1\right)\right)$ ▷ σ update
until Stopping criterion

For more details, the interested reader can refer to this tutorial [Han16].

2.3.2 Modular CMA-ES Framework

Over the years, many variants of the CMA-ES algorithm were introduced. The Modular CMA-ES framework [vRWvLB16] (ModCMA) comes with the idea to unify in the same framework most of the CMA-ES variations introduced in the literature. In the framework, these variations are called *modules* and compose the different variants of the CMA-ES algorithm. The general scheme of the algorithm presented in Section 2.3 is the same but the way some components behave might be modified, *i.e.*, a different way to perform selection, to initialize parameters, to mutate or recombine solution candidates, etc.

The authors described 11 modules with two or three options available for each which results in $4,608$ possible variants. These variants are labeled using a bit string of length 11, each bit representing one module. Hence, the representation 00000000000 labels the vanilla CMA-ES (Algorithm 2) while 11111111122 labels the variant with every module activated and the second choice for the two last modules. The 11 modules are the following:

1. *Active Update:* [JA06] instead of updating the covariance matrix C with only the most successful mutations, this module updates the matrix by also considering the least successful solution candidates;

2. *Elitism:* the common selection strategies: comma-selection (μ, λ) and plus-selection $(\mu + \lambda)$ are available. The default option is the non-elitist comma-selection;

3. *Mirror Sampling:* [BAH$^+$10] originally, all new samples are drawn from the normal distribution. When this module is activated, a number of vectors corresponding to half of the population is drawn from a Gaussian distribution. These vectors are then added or subtracted to the parents to create symmetric offsprings;

4. *Orthogonal Sampling:* [WEB14] this module was later added to mirror sampling. First, the samples are drawn from a normal distribution (or using mirror sampling). Then, the Gram-Schmidt process is applied to the samples. This method is used for orthonormalizing a set of vectors.

5. *Sequential Selection:* [BAH$^+$10] instead of evaluating all λ offsprings, this module evaluates sequentially the new solution candidates and stops when any improvement has been found;

6. *Threshold Convergence:* [PER$^+$15] in order to prevent the search from ending in a local optimum, this module forces the algorithm to stay in the exploration phase by defining a length threshold that new vectors need to reach. This threshold is then decreased after every generation to transition in a exploitation phase;

7. *Two-Point Step-Size Adaptation (TPA):* [Han08] in the two-point adaptation variant, two solution candidates are evaluated right after the mean has been updated using a test width parameter. Usually these solution candidates are chosen to be symmetrical to the mean. Depending on their relation, the step-size is either increased or decreased;

8. *Pairwise Selection:* [ABH11] in mirror sampling, there can be a bias as two mirrored vectors can cancel each other in recombination. To avoid this bias, pairwise selection was introduced and select the best offspring out of each mirror pairs;

9. *Recombination Weights:* [vRWvLB16] instead of the traditional weight vector w, the arithmetic mean is used $w_i = 1/\mu$;

10. *Quasi-Gaussian Sampling:* [AJT05] instead of drawing samples from a Gaussian distribution, new solutions are generated by a quasi-random uniform sequence and then transformed into a normal distribution. The two alternatives proposed are a Sobol´ sequence or a Halton sequence;

11. *Increasing Population Size:* during a restart, using the remaining budget more effectively can be achieved by increasing the population size (IPOP) [AH05]. Then, a bi-population (BIPOP) [Han09] alternative was proposed which alternates with larger and smaller population sizes.

2.4 Differential Evolution (DE)

Differential Evolution was first proposed by Storn and Price [SP97] in the late 1990s, especially for continuous optimization. DE evolves a population of n solution candidates, denoted NP. The population is represented by d-dimensional vectors, d being the dimension of the problem. The main idea in DE is to create a new mutation operator based on vector differences in order to perturb the population.

2.4.1 Initialization and Mutation

An initial population p_0 of $k \geq 4$ solutions is generated uniformly at random. Its i-th candidate solution is denoted $x_i = (x_1, \ldots, x_d)$ A mutant solution x' is then generated by adding a perturbation vector τ where $\tau = F \times (y - z)$, with $F \in [0, 2]$ the scaling factor, and y, z two solutions in the population different from x. Different variants have been proposed to do this perturbation, some variants propose to choose the best candidate solution found so far to be the mutant instead of a random one, other variants will use five solution candidates instead of three. Put differently, four candidate solutions will be used to create the mutant instead of two. In DE algorithms these choice are coined a *strategy* which is denoted by *DE/x/y/z* where:

- x is the way the mutant vector is chosen, *i.e.*, randomly or using the best found so far;
- y is the number of candidate solutions used in the mutation, *i.e.*, *1* stands for three candidate solutions and *2* for five candidate solutions;
- z denotes the crossover scheme, e.g., binomial (bin) or exponential (exp). For the binomial crossover, each coordinate of the offspring may be coming from one or the other parent. For the exponential crossover, blocks of consecutive coordinates of the offspring belong to the first or second parent. See below for further explanations.

2.4.2 Crossover

In DE algorithms, the crossover probability is noted $CR \in [0, 1]$. To avoid that the result of the crossover creates just a copy of the second parent, at least one coordinate, chosen at random, is modified with probability 1. Other coordinates are modified with probability CR. This crossover scheme is named *binomial crossover* in DE framework and is the classical uniform crossover in evolutionary algorithms [Zah06]. Algorithm 3 presents the pseudo-code of this crossover type.

Algorithm 3 Binomial Crossover [Zah06].

```
Perform crossover between x and y.
Choose k randomly in {1, ..., d}.
for j = 1, ..., d do
    if rand(0, 1) < CR or j = k then
        z_j ← y_j
    else
        z_j ← x_j
    end if
end for
```

Another type of crossover is often used in DE, the *exponential crossover*. Exponential crossover can be seen as a two-point crossover [Zah06]. The offspring is first composed of the elements of the

first parent. Starting from a random point $k \in \{1, \ldots, d\}$, elements of the second parents replace the elements of the first parent with probability CR. Algorithm 4 displays the pseudo-code of this crossover type where $\langle j+1 \rangle_n$ is $j+1$ if $j < n$ and is 1 if $j = n$.

Algorithm 4 Exponential Crossover [Zah06].

Perform crossover between x and y.
Choose k randomly in $\{1, \ldots, d\}$, $z = x$, $j = k$, $L = 0$.
$z \leftarrow x$
repeat
  $z_j \leftarrow y_j$
  $j \leftarrow \langle j+1 \rangle_n$
  $L \leftarrow L + 1$
until $rand(0,1) > CR$ or $L = d$

2.4.3 Template of a Differential Evolution Algorithm

Using the definition of mutation and crossover above, Algorithm 5 displays a general template for a DE algorithm. The keyword *strategy* denotes the mutation type whereas *crossover* defines the crossover type.

Algorithm 5 Differential Evolution [SP97]for minimizing a black-box problem $f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R}$.

Choose $F \in [0, 2]$, $CR \in [0, 1]$.
Choose a mutation strategy and a crossover type.
Generate randomly a population P of NP solution candidates, $g = 0$.
repeat
  for $i = 1, \ldots, NP$ **do**
    $V_{i,g} \leftarrow$ strategy ▷ Mutation
  end for
  for $i = 1, \ldots, NP$ **do**
    $U_{i,g} \leftarrow$ crossover ▷ Crossover
  end for
  for $i = 1, \ldots, NP$ **do**
    if $f(U_{i,g}) \leq f(X_{i,g})$ **then**
      $X_{i,g+1} \leftarrow U_{i,g}$ ▷ Selection
    end if
    if $f(U_{i,g}) \leq f(X_{\text{best},g})$ **then**
      $X_{\text{best},g} \leftarrow U_{i,g}$
    end if
  end for
  $g \leftarrow g + 1$
until Stopping criterion

2.5 Particle Swarm Optimization (PSO)

Particle Swarm Optimization is a population-based, stochastic search algorithm introduced in the late 1990s by Kennedy, Eberhart, and Shi [KE95, SE98]. PSO was first developed to solve con-

tinuous, multi-dimensional, boundary constrained and single objective problems. The population is represented by a swarm of particles where each particles is a candidate solution. The position updates of particle i at a time step $t+1$ is computed as follows

$$x_i(t+1) = x_i(t) + v_i(t+1), \tag{P}$$

where v is the *velocity* or the *step size* and the first particles are generated uniformly at random.

2.5.1 Parameters

The performance of PSO is sensitive to the different control parameters of the algorithm [EC21].

The velocity is the mechanism that drives the optimization process, velocity is updated by dimension j [SE98]

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)], \tag{V}$$

with $w \in \mathbb{R}$ is called the inertia weight, $c_1, c_2 \in \mathbb{R}_{>0}$ are positive acceleration coefficients, and $r_{1j}(t), r_{1j}(t) \sim U(0,1)$. $y_i(t)$ is the best position for particle i whereas $\hat{y}_i(t)$ is the best position in the neighborhood of $y_i(t)$. The best position in the neighborhood depends on the neighborhood topology (see Section 2.5.2). The previous velocity $wv_i(t)$ has an inertia component and represents the memory of previously chosen direction. The inertia component also prevents the swarm from changing completely of direction [EC21].

The inertia weight w is used to control the step sizes and is usually used to balance the trade-off between exploration and exploitation, *i.e.*, large values will favor exploration whereas small values will encourage exploitation.

The acceleration weights c_1 and c_2 are called the *cognitive* and *social components*, respectively. The cognitive component represents the nostalgia as it reflects the memory of previous best positions. The social component reflects the envy as it is related to the performances of relative neighbors. Depending the relation between these two components, some behavior of the algorithm are known [EC21]:

- $c_1 > 0, c_2 = 0$: particles are hill-climbers and perform a local search;
- $c_1 = 0, c_2 > 0$: the swarm is a hill-climber;
- $c_1 = c_2 > 0$: particles are attracted towards $\hat{y}_i$;
- $c_1 > c_2$: promotes exploration;
- $c_1 < c_2$: promotes exploitation.

2.5.2 Neighborhood Topologies

In PSO, the neighborhood topologies are components that guide the search [Ken99]. They determine the parts of the search space that is used to determine best positions and control the speed of the information given to the swarm [EC21]. These neighborhood are based on particle indices and not spatial information. While many topologies have been proposed throughout the years, the three most popular are the star topology, the ring topology and the Von Neumann topology (see Figure 2.2 for a graphical representation of these topologies).

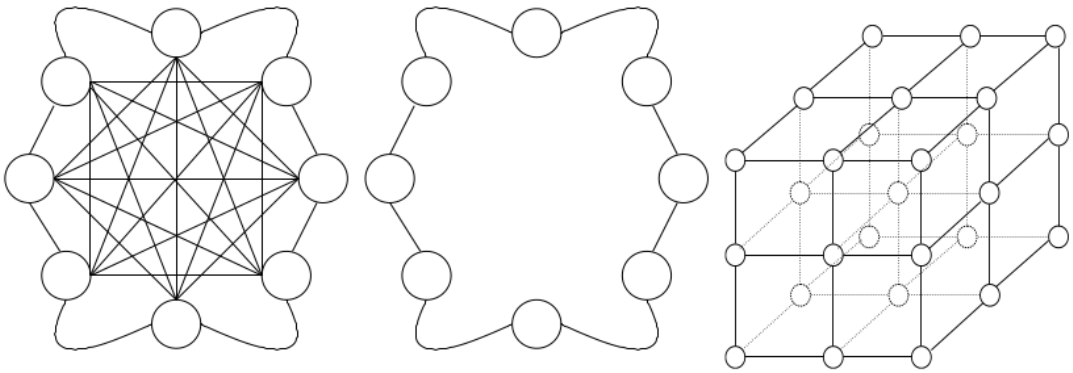


Figure 2.2: Popular neighborhood topologies: star (left), ring (center) and Von Neumann (right), taken from [EC21]. Topologies define neighborhood based on particle indices and not spatial information.

Algorithm 6 Particle Swarm Optimization for minimizing a black-box problem $f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R}$.

Generate a swarm of $x \in \mathbb{R}^d$.
Set the best known position for each candidate solution y_i to the initial solution x_i.
while Stopping criterion **do**
  for $i = 1, \ldots, n$ **do**
    if $f(x_i) \leq f(y_i)$ **then**
      $y_i \leftarrow x_i$
    end if
    for particles $\hat{i}$ with i in its neighborhood **do**
      if $f(y_i) \leq f(\hat{y}_{\hat{i}})$ **then**
        $\hat{y}_{\hat{i}} \leftarrow y_i$
      end if
    end for
  end for
  for $i = 1, \ldots, n$ **do**
    Update velocity using (V).
    Update position using (P).
  end for
end while

2.5.3 Algorithm Pseudo-Code

Given the choices of parameters (inertia, cognitive and social components) and the choice of neighborhood topology, the pseudo-code of a Particle Swarm Optimization algorithm is shown on Algorithm 6.

2.6 Direct Search Methods

2.6.1 Nelder-Mead Downhill Simplex

The Nelder-Mead algorithm was published in 1965 [NM65] and is a geometrical method to find a minimum of a function without derivatives. The algorithm is transforming a simplex S until the simplex is small enough to pass a termination criterion and outputs the best fitness value of one vertex of the simplex.

The Nelder-Mead algorithm is a generalization of the method proposed by Spendley *et al.* [SHH62] where only two transformations were available: reflection and shrinkage which allowed the simplex to change size but not shape Nelder and Mead then allowed two other possible transformations: expansion and contraction which also allow the sample to have another shape.

Algorithm 7 Nelder-Mead Downhill Simplex [NM65] for minimizing a black-box problem $f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R}$.

```
Generate a simplex S of d + 1 vertices x_i around x_0.
while Stopping criterion do
    Order fitness values and label the worst point x_h, the second worst x_s and the best point x_l.
    Compute the centroid c of the best side: c = (∑_{j≠h} x_j)/n.
    if Contraction, reflection or expansion gives a better point than x_h then
        Replace x_h by the new point.
    else
        Perform shrinkage towards x_l and create d new vertices.
    end if
end while
```

The algorithm is controlled by four parameters, one for each transformation: α (reflection), β (contraction), γ (expansion) and δ (shrinkage). These parameters should satisfy some constraints: $\alpha > 0, 0 < \beta < 1, \gamma > 1, \gamma > \alpha, 0 < \delta < 1$. The transformations mentioned in Algorithm 7 are defined below. In each figure representing the transformation: Figure 2.3a for reflection, Figure 2.3b for expansion, Figure 2.4 for contraction and Figure 2.5 for shrinkage, the initial simplex S in $\mathbb{R}^2$ is represented in red whereas the new simplex formed by the transformation is displayed in blue.

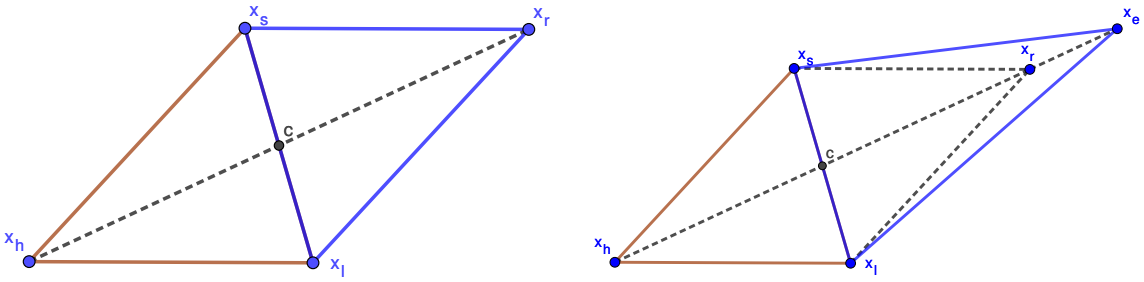
Reflection: compute $x_r = c + \alpha(c + x_h)$, if $f(x_r) < f(x_s)$, accept x_r as new vertex;

Expansion: compute $x_e = c + \gamma(x_r - c)$, if $f(x_e) < f(x_r)$, accept x_e as new vertex;

Contraction: if $f(x_r) \geq f(x_s)$, compute x_c with the best point between x_h and x_r, $x_c = c + \beta(x_{h,r} - c$. If $f(x_c) \leq f(x_r)$, accept x_c, if $f(x_c) < f(x_h)$, accept x_c, otherwise perform shrinkage;

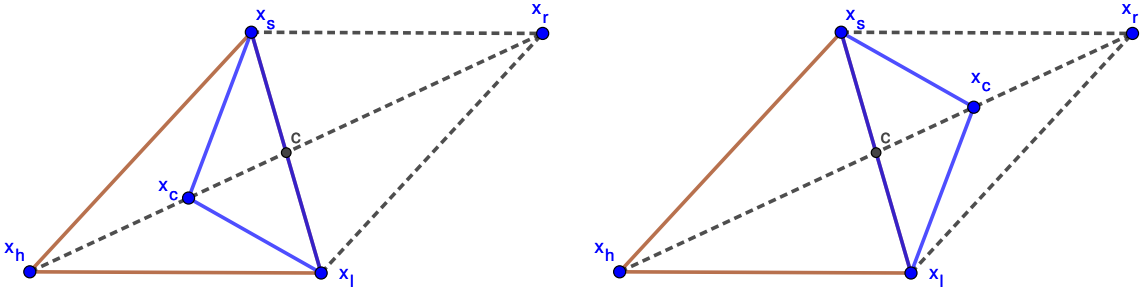
Shrinkage: compute d vertices x such that $x_j = x_l + \delta(x_j - x_l)$.

The Nelder-Mead algorithm usually stops when almost no improvement is done between two iterations of the algorithm or when the simplex is too small. It is very common to keep this



(a) Reflection (b) Expansion

Figure 2.3: Example of simplex reflection and expansion.



(a) Contraction inside initial simplex (b) Contraction outside initial simplex.

Figure 2.4: Example of simplex contraction.

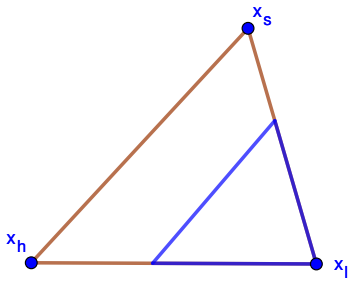


Figure 2.5: Example of simplex shrinkage.

termination criterion even if the total budget of the search is not spent as direct search methods usually finish the search in a basin of attraction of a local or global optimum. The strategy to spend efficiently the whole budget is to perform random restarts of the algorithm, *i.e.*, when no improvement is found, the algorithm starts again with a new simplex sampled randomly in the search space.

2.6.2 Powell's Method

Powell's conjugate direction method, abbreviated in Powell's method is a direction set method introduced in 1963 [Pow64] to find local minimum in functions with no assumption on the function. The basic idea of the algorithm is to perform successive one-dimensional optimizations along well chosen directions. If we denote e_i the vectors of the orthonormal basis, i.e, $e_i = \{0, \ldots, \underbrace{1}_{i}, 0, \ldots, 0\}$, the first directions are vectors of the basis and new directions are chosen using the current best point. Different choices can be made for the one-dimensional optimization but usually this optimization is performed using either the Golden-section search [Kie53] or Brent's method [Bre71]. The pseudo-code of the method is displayed in Algorithm 8.

Algorithm 8 Powell's method [Pow64] for minimizing a black-box problem $f : \Omega \subseteq \mathbb{R}^d \to \mathbb{R}$.

```
Choose a random candidate solution x_0.
Set current solution x ← x_0.
Set the cycle of direction C with c_i = e_i, i = 0, ..., d.
while Stopping criterion do
    for i = 1, ..., d do
        Perform a one-dimensional optimization along the direction c_i starting from x.
        x ← x_i
    end for
    Remove the direction c_j that lead to the best fitness improvement.
    Add the direction x_d − x_0 at the end of the cycle C.
end while
```

As the Nelder-Mead algorithm, when no improvement is found, a restart strategy can be done by starting the method with another random point.

2.7 L-BFGS-B Algorithm

The L-BFGS-B algorithm is a variant of the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) [Fle87] for limited computer memory and box constrained problems. L-BFGS-B is an algorithm from the quasi-Newton methods that aims at finding the minimum of objective functions.

The algorithm works by performing a gradient descent in order to estimate the Hessian matrix to guide the search towards promising areas. As we are in a black-box optimization context, the function f may not be differentiable and no information on the gradient is known. To cope with this non-differentiability, the BFGS algorithm is estimating the gradient using forward finite differences, a numerical method often used to approximate derivative and solve differential equations.

2.8 Automated Algorithm Configuration

2.8.1 Motivation

The main motivation for automatic algorithm configuration (AC) comes from the idea that, in various domains, algorithms generally have parameters and humans do not have good intuitions to find good parameter settings. Algorithms presented above in this section are no exceptions as they possess different design choices. Three different types of design choices and parameters can be defined [SL21]:

Numerical: integer or real valued parameters such as the population size, the mutation and the crossover probability in Differential Evolution 2.4;

Categorical: parameters that describe different design choices. For instance, in Differential Evolution (Section 2.4), the crossover type is a categorical parameter, either binomial or exponential crossover;

Ordinal: parameters that have ranks but are not numerical, *i.e.*, different sizes of neighborhoods such as small, medium or large.

Some of these parameters may be *conditional*, *i.e.*, activated only if needed. Hence, as the parameter space can be large, finding a set of parameters that achieve good performances may be difficult. The traditional approaches to tune parameters of an algorithm are of three kinds: trial-and-error, theoretical based or grid search. The trial-and-error is mostly guided by one's expertise or intuition and thus, is limited in the exploration of the parameter space. Tuning an algorithm based on theoretical results may not be preferable for complex or real-world problems. Theoretical analyses are restricted to synthetic benchmarks. The transfer of these results to complex real-world problems is not straightforward. Grid search is in between manual and automatic configuration but is often limited when the parameter space is large. Instead of doing the search manually, it is worth exposing parameters and design choices and let automatic procedures search the parameter space to find good configurations.

2.8.2 Automated Algorithm Configuration Methods

The automated algorithm configuration operates as showed in Figure 2.6 regardless of the parameter type. First the configurator samples a configuration for a target algorithm. The configuration is evaluated on a set of instances of the problem at hand and returns a solution cost. The cost can be the average running time of the algorithm, the average solution quality obtained or any cost function that fits the purposes of the user. Given this cost, the configurator will sample a new configuration to be evaluated. Over the years, many methods based on different underlying ideas were introduced to implement the configurator part of the workflow. Algorithm configuration is itself an offline stochastic black-box optimization problem which can be discrete, continuous, or mixed discrete-continuous. Among the common methods for algorithm configuration are iterated local searches in the parameter space such as ParamILS [HHLS09], methods based on surrogate models to predict performances such as SMAC [HHL11], racing methods like F-race [BSPV02] or irace [LDLP$^+$16] where worst algorithm are discarded based on statistical testing, and bandit-based approaches such as hyperband [LJD$^+$17]. More details about configurators can be found in [SL21].

In this thesis, automated algorithm configuration is used in Chapter 8 to create instance specific solvers. Surprisingly, we found difficult to obtain configured algorithms that were performing better than their default configuration.

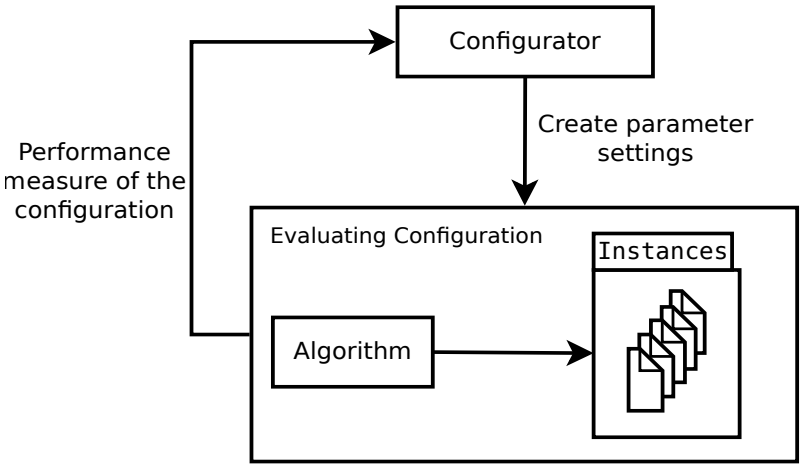


Figure 2.6: Algorithm Configuration workflow.

Chapter 3

Landscape-Aware Algorithm Selection

Contents

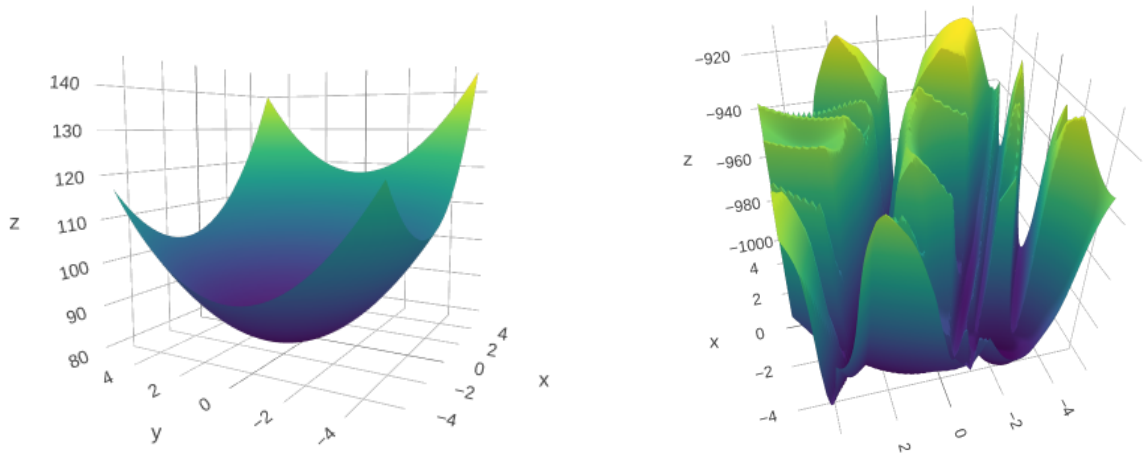
This chapter present the background on landscape-aware algorithm selection, also called *Per-Instance Algorithm Selection* (PIAS). In Section 3.1, we motivate the use of a landscape-aware algorithm selection. Section 3.2 presents the algorithm selection problem introduced by Rice [Ric76] while Section 3.3 describes the different components required to perform landscape-aware algorithm selection. Finally, Section 8.2.2 introduces automatic algorithm configuration and how can algorithm configuration can interconnect with landscape-aware algorithm selection.

3.1 Motivation

Automated algorithm selection is a broad domain composed of several techniques such as landscape-aware algorithm selection. Landscape-aware algorithm selection is at the boundary between black-box optimization and Machine Learning. Landscape-aware algorithm selection aims to learn a mapping between features measures and the performances of algorithms.

The idea of this approach is based upon the performance complementarity of solvers. This complementarity has been observed for most of NP-hard optimization problems such as satisfiability [XHHL11], planning [VHCM15], traveling salesperson problem [KKB$^+$18] and also continuous optimization problems [KHNT19].

Hence, it is likely that the best performing algorithm on the sphere function (Figure 3.1a) which is unimodal, highly symmetric and convex will not be the same as the best algorithm solving the Gallagher's Gaussian 21-hi Peaks Function (Figure 3.1b). The latter is composed of 21 local optima randomly chosen where only one is global and located in a basin with high conditioning, *i.e.*, points around the optimum are solutions with high fitness values.



(a) Shpere function

(b) Gallagher's Gaussian 21-hi Peaks Function

Figure 3.1: Example of two objective functions from the Black-Box Optimization Benchmark (BBOB), taken from [HAR+21].

3.2 Landscape-Aware Algorithm Selection

The algorithm selection problem was first introduced in 1976 by Rice [Ric76]. Given a problem, algorithms to solve this problem, and a specific instance of this problem to solve, the goal is to determine which algorithm can be expected to solve the instance.

To formulate the landscape-aware algorithm selection problem in a black-box optimization context, we need:

- a set of instances of an optimization problem $\mathcal{I} \subset \mathcal{P}$;
- a set of k algorithms solving the problem, $\mathcal{A} = \{A_1, \ldots, A_k\}$;
- a set of l features characterizing the problem, $\Phi = \{\varphi_1, \ldots, \varphi_l\}$ (see Chapter 4);
- a performance measure providing the quality of an algorithm on a particular instance, $m : \mathcal{A} \times \mathcal{I} \to \mathbb{R}$.

Using these elements, a selector S is constructed. The selector is a function that outputs an algorithm given a problem instance, $S : \mathcal{I} \to \mathcal{A}$. For each instance $i \in \mathcal{I}$, the selector outputs an algorithm to solve the instance by minimizing the performance measure m. To choose the appropriate algorithm, the selector relies on features. These features help the selector to discriminate between different problem instances. The goal is to find a selector that has the closest possible performances to the perfect selector.

In practice, the selector does not have a perfect efficiency. To measure the quality of a selector, two bounds are defined [KHNT19]. The lower bound is defined by the performances of the perfect selector, also called the *oracle selector* or *Virtual Best Solver (VBS)* [KHNT19] This selector chooses the best performing algorithm in the portfolio given any instance. The upper bound is established by the performances of the *Single Best Solver (SBS)* [KHNT19], *i.e.*, the algorithm $A' \in \mathcal{A}$ with the best performances among the k algorithms in $\mathcal{A}$. Worse performances than the

SBS implies that there is no need to perform an automated selection as a single algorithm obtain better results.

Moreover, the ratio between performances of the SBS and VBS gives the amount of gain that can be achieved by performing an automated algorithm selection. This ratio is linked to the performance complementarity of the algorithms in the portfolio. This ratio is also called the *VBS-SBS gap* [KHNT19]. Put differently, a small gap indicate that performances of the SBS are close to those of the VBS and thus suggests that there is only a small complementarity in the portfolio. The performance of the selector P_s can be evaluated using a measure of *merit* [CDL+21] that compares the performance of the selected solvers to both the SBS P_{SBS} and VBS P_{VBS}. First the difference between the recommended solver performance and the VBS performance as such as the difference between the SBS performance and the VBS performance. the merit computes the ratio between these two differences

$$merit = \frac{P_s - P_{\text{VBS}}}{P_{\text{SBS}} - P_{\text{VBS}}}.$$

3.3 Landscape-Aware Algorithm Selection Pipeline

The landscape-aware algorithm selection pipeline is divided into two distinct parts:

1. a *Train* part which correspond to the phase where the selector is built;
2. a *Test* part where the selector is actually used to solve an unknown instance of the problem.

Figure 3.2 regroups all the steps needed in order to perform an automated algorithm selection procedure. In the following sections, we will describe practically the steps of this pipeline for both the *Train* (Section 3.3.1) and *Test* 3.3.2) parts.

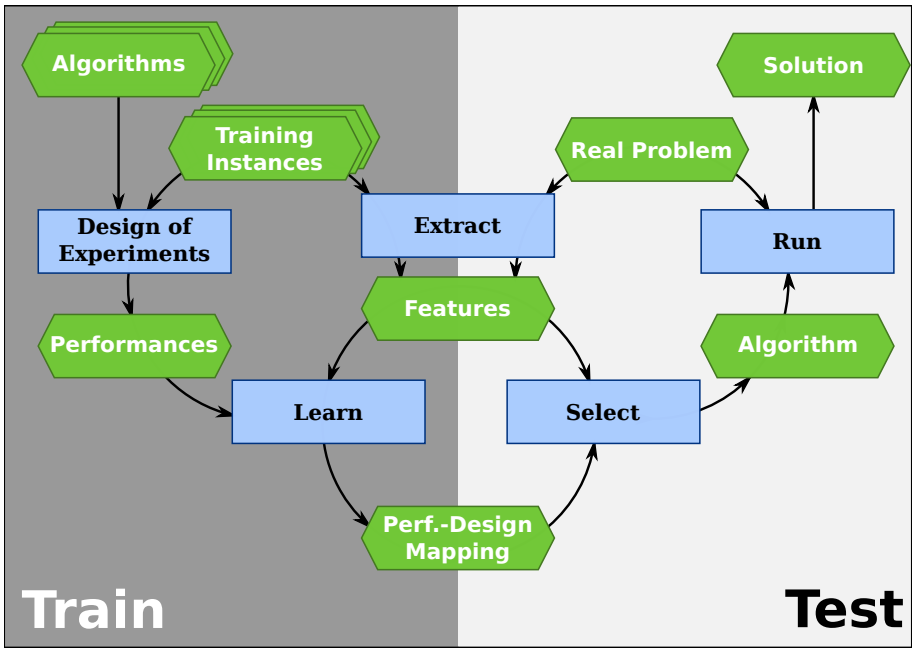


Figure 3.2: Landscape-aware algorithm selection pipeline.

3.3.1 Training Phase

3.3.1.1 Algorithms

To define the landscape-aware algorithm selection pipeline, we need to select which algorithms are going to compose the portfolio of available algorithms. In general, we can expect from a diverse portfolio of algorithms favor better performances of the approach than a portfolio composed of similar algorithms.

3.3.1.2 Training Instances

Creating a set of training instances is a crucial and often difficult part, especially when real-world problems are involved. In order to be able to generalize to test instances, the training instances have to be both diverse and representative [BDB+20]. When the training instances have no diversity, then the automated algorithm selection is no use as the selection process is done on similar instances and might not be able to generalize to the test instances.

There are techniques that permits to visualize the diversity of training instances [BDB+20]. The first way is to look at the feature space of the training instances, *i.e.*, when the distribution of instances in the feature space is dense, the set is not diverse. Another possibility is to look at the performance space of the portfolio. Different algorithms performing best on different instances might indicate that the benchmark set is diverse. Finally, one can look at the instance space [ST12] which combines the information of the feature space and the performance space as regions of the feature space are associated with the algorithm performing best on these regions.

Moreover, the instances have to be representative of the type of instances that can be found in the test set. In general, the application or the problem is known in advance. For example, before solving the problem, we know that the instance to solve will be a Traveling Salesperson Problem (TSP) instance or radar network configurations instance. Therefore, we can build a training set by instantiating some instances of the appropriate problem type.

3.3.1.3 Performance Measures

The choice of the performance measure can have a huge impact on the selection and should be chosen carefully to match the user's goal. Commonly, performance is measured by the solution quality (*fixed-target*), or related to time (*fixed-budget*).

Fixed-target performance measures are mostly used when the global optimum is known or when the end user have some criteria that has to be met, to match a regulation for example. Different measures can be designed such as the proportion of runs reaching the target or the average time an algorithm takes to reach the target.

Fixed-budget performances are usually measured based on CPU time or number of function evaluations. The CPU time makes comparisons difficult between different hardware configuration but is easier to grasp. For some particular cases, the user might have time restriction, such as, "an algorithm should run no more than 10 minutes". In this case, CPU time is a well suited measure. The number of function evaluations is more standard when we want to compare different algorithms. As metaheuristic solvers are stochastic, multiple runs are performed to gather their performances. Different methods may be used to gather performances as Expected Running Time (ERT) [HAR+21], Empirical Attainment Function (EAF) [LPS10] or Expected Target Value. The way these performances are aggregated may have an impact on the selection, *i.e,* an expected target value taking the median or the 2% quantile of the runs may differ a lot.

3.3.1.4 Features

Features characterize problem instances. They can be of three types:

1. *Measurements:* real numbers coming from generic computations such as exploratory landscape analysis. As samples that are used to compute features are not deterministic, multiple independent runs are performed. See Chapter 4 for more details;
2. *Modeled:* problem specific values. This type of feature represents information about the problem instances such as the dimension or the computational resources to solve the problem;
3. *Indirect:* features from the analysis of the source code of software [PK20].

3.3.1.5 Mapping between Features and Algorithm Performance

The goal of the mapping is to learn the dependencies between feature vectors and algorithm data on each instance. The algorithm data can refer to the best performing algorithm by instance or to the performance of algorithms on each instance. The mapping is often inferred through machine learning and represent the core part of the selector. Building the mapping can be seen as learning a function where the inputs are information on an instance, *i.e.*, feature vectors and algorithm performance. The output of this function is the best performing algorithm on this instance.

Over the years, different techniques have been used to create this mapping as classification, regression, and clustering [LHHS15, Tan21a].

Classification: Usually, a multiclass classification model [KHNT19, BMTP12] is built in order to predict the best algorithm $A^* \in \mathcal{A}$ for a given instance. The instance is represented by a feature vector.

We can also possible perform a pairwise classification [XHHL11] where models for each pair of algorithms are built. The output of the selector is the algorithm that was the most selected in all pairs for the instance to solve. Nevertheless, the major drawback of using classification is that the ranking of the non selected solvers and their performances are not considered.

Regression: One way to cope with the drawback of classification is to perform regression to construct the mapping [XHHLB08, BDSS17, JPED21]. By building k models, one by algorithm, one can select the algorithm that has the best predicted performance. Using regression, we know the ranking of best performing algorithms by instance. Knowing this ranking, the selector can recommend the $a, a \leq k$ first algorithms instead of only one.

As for classification, pairwise regression models that predict the performance differences between pairs of solvers on one instance can be constructed. The best algorithm is chosen based on the sum of the predicted performances [KHNT19].

Clustering: Clustering is used in [KMST10]. Clusters of training instances are created based on feature values using a *k-means* algorithm [HE03]. The best performing algorithm is selected based on the performances of algorithms on each cluster of instances.

3.3.2 Testing Phase

3.3.2.1 Real Problem

The "real" problem is a new unknown instance to solve for which we need to find the appropriate solver.

3.3.2.2 Features

Landscape features are computed on the optimization problem at hand. The vector of features of the real problem to solve is then fed to the mapping in order to compare it to the instance vectors of features in the benchmark. In contrast to the learning phase, computing features based on one sample is common to gain computation time.

3.3.2.3 Algorithm and Solution

The algorithm is the output of the mapping, *i.e.*, it is the algorithm that the mapping supposes best suited to solve the problem. The solution obtained is the result of the run of the algorithm given by the mapping or the result of the best run if multiple runs can be performed.

Chapter 4

Characterizing Problem Instances via Landscape Features

Contents

In this chapter, we present methods introduced in the literature that characterize problem instances.

Section 4.1 describes the general concept of *fitness landscape analysis* and some techniques that can be used to characterize problem instances.

Section 4.2 presents one particular technique, *i.e.*, *exploratory landscape analysis* (ELA). This technique is widely used for characterizing black-box optimization problems via features. In this section, we present a list of most features from ELA. These features will be studied in Chapter 5 and used to solve the radar network configuration problem in Chapter 9.

4.1 Fitness Landscape Analysis

4.1.1 Motivation

The concept of a *fitness landscape* was first introduced in the 30's by Wright in biology [Wri32] to study the evolution of species. It has been extended to other numerous domains such as statistical physics, molecular evolution, and ecology. The idea behind fitness landscape is to visualize the distribution of genotypes and to describe how easily one genotype is reached from another one.

In our context of black-box optimization, a genotype represents a candidate solution. A fitness landscape combines three elements [Sta02]:

1. a set $X \subseteq \Omega$ of candidate solutions x;
2. a notion of distance or neighborhood on Ω;
3. a fitness function $f : \Omega \to \mathbb{R}$.

For instance, in the case of continuous optimization problems, the set $\Omega \subseteq \mathbb{R}^d$, where d is the dimension of the problem, represents the search space and is commonly equipped with the Euclidean distance. The fitness function f is represented by the objective function to minimize. Here, we took the example of numerical optimization but the same approach is also valid with combinatorial optimization, search spaces of programs (as in Genetic Programming) and so on. Fitness landscape has also been extended from the original definition to multiobjective landscapes, violation landscapes (*i.e.*, landscapes characterizing constraints applied on objective functions), dynamic landscapes, and many more [Mal21].

4.1.2 Fitness Landscape Characteristics

The goal of fitness landscape analysis is to find characteristic properties of optimization problems that could influence the performance of algorithms solving these problems. While some characteristics are linked to the objective function only (e.g., number of local optima, separability), others may directly influence the performance of algorithms (*i.e.*, localization of optima in the search space). Some characteristics are presented in sections below and more can be found in [ME13].

4.1.2.1 Local Optima

A candidate solution $\bar{x}$ is said to be a local optimum when in a neighboring set of candidate solutions $\bar{x} = \{x \in \Omega \mid f(x) \leq f(y) \, \forall y \in N_\delta(x)\}$ with $N_\delta(x) = \{z \in \Omega \mid \Delta(x, z) \leq \delta\}$. An optimum x^* is said to be global when in the set of local optima $f(x^*) \leq f(\bar{x})$. While *unimodal* functions possess only one local optimum, which is also the global optimum, *multimodal* functions have more than one local optima. The number and localization of local optima in the search space can have an impact on the search.

The distribution of these local optima is referred as *ruggedness*: a rugged landscape consists in neighboring points with very different fitness. As mentioned in [MF04], ruggedness of a landscape may affect the search as it is possible for an algorithm to get stuck in a local optimum which can slow down or compromise the search for global optimum.

4.1.2.2 Basins of attraction

The notion of basin of attraction is related to an attractor. In our case, we will focus on the basins of attraction where the attractor is a local optimum. That is, the set of solutions for which there

exists a sequence of neighbors starting from a local optimum, that is monotonic in the objective function. Both size and depth of the basin can affect the search. A needle-in-a-haystack problem will have one global optimum with a very small and deep basin that will be hard to reach, whereas problems with large basins around the global optimum are easier to solve.

4.1.2.3 Separability

A function f is called *separable* when it can be additively or multiplically decomposed as such

$$f(x_1, \ldots, x_n) = \begin{cases} \sum_{k=1}^{n} f_k(x_k), \\ \text{or,} \\ \prod_{k=1}^{n} f_k(x_k). \end{cases} \tag{4.1}$$

When a function f is separable, the global optimum can be reached by optimizing each coordinate independently. Thus, separable functions are easier to optimize since line search is an easy task that can be performed for instance with a Newton's method or the Nelder-Mead method [NM65].

In real-world problems, fully separable functions are quite rare. Nevertheless, it is possible to encounter a partially separable function. Partially separable functions can be written as a sum or multiplication of sub-functions that depend on disjoint sets of variables. Hence, finding the optimum can be achieved by optimizing independent coordinates separately.

4.1.2.4 Neutrality/Plateaus

Neutrality corresponds to the degree to which a landscape contains areas of equal fitness that are connected. Neutrality was first introduced by Kimura [Kim68] in biology. If a considerable number of mutations have no effect on fitness values, the result is a neutral landscape represented by plateaus.

4.1.2.5 Deception

Deception is known as the presence of misleading information in the landscape [ME13]. Deception is related to the structure of the distribution of optima and is related to the fact that misleading information can be present and guide the search in an unsuccessful direction. Deception is linked to a particular algorithm, *i.e.*, a problem can be deceptive for some algorithms but not for others.

4.1.3 Fitness Landscape Techniques

Fitness landscape techniques aim at measuring via *features* the fitness landscape characteristics. Fitness landscape techniques were developed in order to create features that can measure different characteristics of an optimization problem. For instance, the *correlation length* [Wei90] seeks to measure the ruggedness of the landscapes. *Fitness distance correlation* [JF95] is designed to measure the deception property. *Local Optima Networks* [OTVD08] (LON) are designed to give an insight on the global structure of the landscape.

The most common drawbacks of these techniques are the following:

- Some of these techniques such as fitness distance correlation, require the knowledge of the global optima, which is unlikely in a black-box optimization context;
- some techniques (e.g., LONs) require the evaluation of the whole search space;

- the code to compute these features is often not available: most of the proposed features lack an open-source implementation to compute them.

Different fitness landscape techniques have been introduced over the years and most of them are described in [ME13] or [Mal21].

In continuous single-objective optimization, several properties are relevant to characterize a problem such as its *separability*, its *multimodality*, its *global structure* or *plateaus*. However, these *high-level* properties [MBT+11] are difficult to quantify as some require knowledge of the entire problem a requirement typically not met in a black-box optimization context. To overcome this difficulty, Mersmann *et al.* [MBT+11] introduced *low-level* properties. We compute low-level properties using sample observations of the underlying problem. Low-level properties are expressed by numerical values computed using sample observations of the underlying problem.

4.2 Exploratory Landscape Analysis

Exploratory Landscape Analysis (ELA) is a practical way to characterize and quantify problem instance properties. ELA features represent black-box optimization problems by means of numerical values.

To characterize optimization problems, several landscape *features* were introduced to such as *dispersion* [LW06], ELA features [MBT+11], *information content* [MKH15], *nearest better clustering* [KPWT15], *principal component analysis* [KT19b], and *SOO-based features* [DLV+19].

To be computed, these features relies on n sample points x of dimension d. Some of them may also rely on the fitness values associated with these sample points $f(x)$.

A lot of these features can be computed via the R package *flacco* [KT19b]. Recent development of landscape analysis methods can be found in [Mal21]. In the following we describe the most relevant landscape features for our study.

Features presented in Sections 4.2.1 to 4.2.6 will be further investigated while features presented in Sections 4.2.7 to 4.2.10 will be disregarded.

4.2.1 y-Distribution Feature Set (yD)

This feature set was introduced by Mersmann *et al.* in [MBT+11] to measure the degree of peakedness of the fitness values distribution. All features from this set are computed using the fitness values only, *i.e.*, features from this set ignore explicit consideration of the search points that have been used to sample these fitness values. y-distribution feature set is composed of three features, kurtosis, skewness and number of peaks.

4.2.1.1 Kurtosis

The *kurtosis* is a measure of sharpness of a distribution. Kurtosis is often measured using the excess of kurtosis. In *flacco*, the excess of kurtosis is computed by the following formula, originally suggested in [JG98]

$$b_2 = \frac{m_4}{s^4} - 3 = \frac{g_2 + 3}{(1 - 1/n)^2} - 3,$$

where s is the standard deviation of the fitness values, m_i are the i^{th} order moments of the fitness values and

$$g_2 = \frac{m_4}{m_2^2} - 3.$$

The meanings of these computations are

$$\begin{cases} b_2 = 0 & \text{Misokurtic distribution such as normal distribution;} \\ b_2 < 0 & \text{Platykurtic distribution, } i.e., \text{ the distribution has thinner tails} \\ & \text{than the normal distribution (example : uniform distribution);} \\ b_2 > 0 & \text{Leptokurtic distribution, } i.e., \text{ the distribution has fatter tails} \\ & \text{than the normal distribution (example : Student t-distribution).} \end{cases}$$

4.2.1.2 Skewness

The *skewness* is a measure of asymmetry of a distribution.

In *flacco*, the skewness is computed by [JG98]:

$$b_1 = \frac{m_3}{s^3} = g1 \left(\frac{n-1}{n} \right)^{3/2},$$

where

$$g_1 = \frac{m_3}{m_2^{3/2}},$$

s is the standard deviation of the fitness values, m_i are the i^{th} order moments of the fitness values.

When skewness $= 0$, then the distribution is symmetric. When skewness > 0 (resp. skewness $<$ 0), the central tendency of the distribution is concentrated on the left (resp. on the right).

4.2.1.3 Number of peaks

The *number of peaks* feature is an indicator for multi-modality. The number of peaks are computed by estimating the distribution density of fitness values using a Kernel Density Estimation [Par62]. Potential peaks are represented by the masses in valleys within the estimated distribution. These potential peaks are then accepted if they are greater than a threshold.

4.2.2 Meta Model Feature Set (mm)

The *meta model* feature set aims at fitting a linear and a quadratic model to the data respectively. The meta model feature set was introduced by Mersmann *et al.* in [MBT+11]. In this section, let $f^{(i)}$ be the fitness values of the point $x^{(i)}$, $i \in [1, n]$, n being the sample size and d the dimension of the samples. Let $y^{(i)}$ be the predicted fitness values of one of the models below and let $\bar{f}^i$ be the mean of the computed fitness values.

In this set, features are extracted using four regression models: a simple linear model (LM), a linear model with a two-way interaction (LMI), a quadratic model (QM), and a quadratic model with a two-way interaction (QMI). $\beta_j, j \in [0, 3d]$ are the coefficients of linear and quadratic models.

$$
\begin{aligned}
y^{(i)} &= \beta_0 + \beta_1 x_1^{(i)} + \cdots + \beta_d x_d^{(i)}, && (LM)\\
y^{(i)} &= \beta_0 + \beta_1 x_1^{(i)} + \cdots + \beta_d x_d^{(i)} + \beta_{d+1} x_1^{(i)} x_2^{(i)} + \cdots + \beta_{2d-1} x_{d-1}^{(i)} x_{d-1}^{(i)}, && (LMI)\\
y^{(i)} &= \beta_0 + \beta_1 x_1^{(i)} + \cdots + \beta_d x_d^{(i)} + \beta_{d+1} (x_1^{(i)})^2 + \cdots + \beta_{2d} (x_d^{(i)})^2, && (QM)\\
y^{(i)} &= \beta_0 + \beta_1 x_1^{(i)} + \cdots + \beta_d x_d^{(i)} && (QMI)\\
&\quad + \beta_{d+1} (x_1^{(i)})^2 + \cdots + \beta_{2d} (x_d^{(i)})^2\\
&\quad + \beta_{2d+1} x_1^{(i)} x_2^{(i)} + \cdots + \beta_{3d-1} x_{d-1}^{(i)} x_{d-1}^{(i)}.
\end{aligned}
$$

Multiple features are defined below using the coefficient of determination R^2. This coefficient of determination is computing using the following formulas

$$
\begin{aligned}
SS_{\text{res}} &= \sum_{i=1}^{n} (y^{(i)} - f^{(i)})^2, \text{ the sum of squares of the residuals,}\\
SS_{\text{reg}} &= \sum_{i=1}^{n} (y^{(i)} - \bar{f}^{(i)})^2, \text{ the regression sum of squares,}\\
SS_{\text{tot}} &= SS_{\text{res}} + SS_{\text{reg}}, \text{ the total sum of squares,}\\
R^2 &= 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}.
\end{aligned}
$$

From this definition of the R^2, the adjusted R^2, denoted $\bar{R}^2$, is computed as follow

$$\bar{R}^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1},$$

where p is the number of explanatory variables in the model, *i.e.*, the number of independent variables which correspond to $p = d$ in this case.

4.2.2.1 Adjusted R^2

This feature computes the adjusted R^2 of the model (LM), (LMI), (QM) and (QMI), respectively.

4.2.2.2 Intercept coefficient of (LM)

This feature return the value of the *intercept coefficient* β_0.

4.2.2.3 Minimum coefficient of (LM)

This feature return the smallest coefficient of the model (LM) in absolute value:

$$c_{\min} = \min_{i, i \neq 0} |\beta_i|.$$

4.2.2.4 Maximum coefficient of (LM)

This feature return the greatest coefficient of the model (LM) in absolute value:

$$c_{\max} = \max_{i, i \neq 0} |\beta_i|.$$

4.2.2.5 Ratio of the maximum and minimum coefficients of (*LM*)

This feature computes the ratio between the greatest and smallest coefficients:

$$c_r = \frac{c_{\max}}{c_{\min}}.$$

4.2.2.6 Condition of (*QM*)

This feature computes the condition (*QM*), *i.e.*, condition is define as the ratio between the absolute biggest and smallest coefficients of (*QM*).

4.2.3 Nearest Better Clustering Feature Set (nbc)

Nearest better clustering features were introduced by Kerschke *et al.* in [KPWT15] in order to detect *funnels*. Funnels are peaked structures containing a global optimum and several local optima such as the Rastrigin function Nearest better clustering features extract information based on the comparison of the sets of distances that are computed from:

1. all observations towards their nearest neighbors;
2. their nearest better neighbor.

Neighbors are defined as the set of the p closest points in the sample. These features are based on a *kd*-tree to find the p closest neighbors for each points. *kd*-trees are binary trees that are used for organizing points in a k-dimensional space, in our case $k = d$. The quick version of the algorithm that finds closest neighbors in *flacco* computes the $0.05 \times d$ closest neighbors.

In [KPWT15], Kerschke *et al.* define $\mathcal{P}$ as the population of sampled points. They define the distance to the nearest neighbor of a search point x as

$$d_{nn}(x, \mathcal{P}) = \min(\{\text{dist}(x, y) \,|\, y \in \mathcal{P} \setminus \{x\}\}),$$

and the distance to the nearest better neighbor as

$$d_{nb}(x, \mathcal{P}) = \min(\{\text{dist}(x, y) \,|\, f(y) \leq f(x) \wedge y \in \mathcal{P} \setminus \{x\}\}).$$

Then, they defined the sets of nearest neighbors distances $\mathcal{D}_{nn} = \{d_{nn}(x, \mathcal{P}) \,|\, x \in \mathcal{P}\}$, the set of nearest-better distances $\mathcal{D}_{nb} = \{d_{nb}(x, \mathcal{P}) \,|\, x \in \mathcal{P}\}$ and the set of quotient of nearest neighbor and nearest-better neighbor distances:

$$\mathcal{Q}_{nn/nb} = \left\{ \frac{d_{nn}(x, \mathcal{P})}{d_{nb}(x, \mathcal{P})} \,\middle|\, x \in \mathcal{P} \right\}.$$

4.2.3.1 Mean and standard deviation ratios

This two features compute ratios of the average values (resp. the standard deviation) of the nearest points and the nearest better points

$$\frac{\text{mean}(\mathcal{D}_{nn})}{\text{mean}(\mathcal{D}_{nb})} \quad \text{(resp. sd)}.$$

These features aim at recognizing multimodal problems from unimodal ones. In case of multimodal problems, $\text{sd}(\mathcal{D}_{nb})$ should be larger than $\text{sd}(\mathcal{D}_{nn})$ whereas the ratio should be close to 1 for unimodal problems [KPWT15].

4.2.3.2 Correlation

This feature computes the Pearson correlation between the distance of the closest neighbors and the closest better neighbors

$$\mathrm{cor}(\mathcal{D}_{nn}, \mathcal{D}_{nb}).$$

The intuition behind this feature is that the correlation on landscape composed of random peaks should be much lower than the correlation of landscape with a funnel [KPWT15].

4.2.3.3 Coefficient of variation

This feature computes the following ratio:

$$\frac{\mathrm{sd}(\mathcal{Q}_{nn/nb})}{\mathrm{mean}(\mathcal{Q}_{nn/nb})}.$$

This represent the coefficient of variation of the set $\mathcal{Q}_{nn/nb}$ which should be larger for random peaked landscape [KPWT15].

4.2.3.4 Fitness correlation

To compute this feature, let us consider the directed graph of the nearest better points. The vertices are the search points and the edges represents the nearest better points, *i.e.*, there is an edge from point x to point y if y is the nearest better neighbor of x. Let $\deg^-(x)$ be the indegree of vertex x. Then the fitness correlation feature computes

$$-\mathrm{cor}(\{(\deg^-(x), f(x)) \,|\, x \in \mathcal{P}\})).$$

In a funnel landscape, the global optimum should have more incoming edges than a random peak landscape, thus a greater indegree [KPWT15].

4.2.4 Dispersion Feature Set (disp)

The features presented in this section were introduced by Lunacek and Whitley in [LW06]. Subsets of the fitness values are created with predefined thresholds. These thresholds represents the best X% of the search space.

For each threshold, the mean $\bar{m}$ and the median M of all distances among the points of the subset are computed and compared to the distances of all the points of the sample.

Fig. 4.1 is an example of the dispersion features for a given threshold. Distances between round points, under the red line, are computed and then compared with distances computed between triangle point, above the threshold.

The features below are available for the following thresholds: 2%, 5%, 10% and 25% and both for mean and median:

Ratio: The comparison of the two subsets is done using the ratio $\bar{m}_{\mathrm{subset}}/\bar{m}_{\mathrm{all}}$ and $M_{\mathrm{subset}}/M_{\mathrm{all}}$.

Difference: The comparison of the two subsets is done using the difference $\bar{m}_{\mathrm{subset}} - \bar{m}_{\mathrm{all}}$ and $M_{\mathrm{subset}} - M_{\mathrm{all}}$

The combination of all possibilities results in 16 features composing this set.

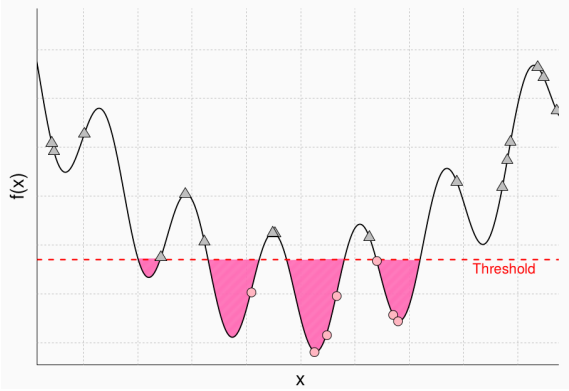


Figure 4.1: Example of the dispersion features [LW06].

4.2.5 Information Content Feature Set (ic)

The features shown in this section were first introduced by Vassilev *et al.* [VFM00] and then redefined and enhanced by Munõz *et al.* [MKH15].

First, the sequence of the search points $(x_1, \ldots, x_n)$ is sorted. The sequence can be sorted at random or by nearest neighbors. In *flacco*, the default sorting is by nearest neighbors. A path through the landscape is constructed starting with an initial observation (usually the last observation) and walking (greedily) from an observation to its nearest not-yet-visited neighbor.

Assuming that $(x_1, \ldots, x_n)$ are already sorted this way, let $y_i = f(x_i)$, $S = \{y_1, \ldots, y_n\}$, and $\Phi(\varepsilon) = \{\Phi_1, \ldots, \Phi_{n-1}\}$, where $\Phi_i \in \{\bar{1}, 0, 1\}$. Φ_i is defined as

$$\Phi_i = \begin{cases} \bar{1} & \text{if } y_{i+1} - y_i < -\varepsilon \\ 0 & \text{else if } y_{i+1} - y_i \leq \varepsilon \\ 1 & \text{if } y_{i+1} - y_i > \varepsilon \end{cases}.$$

Information Content (IC) is defined as a measure of the variety of the objects in the landscape as

$$H(\varepsilon) = -\sum_{a \neq b} p_{ab} \log_6 p_{ab} \,;\, a, b \in \{\bar{1}, 0, 1\}; H(\varepsilon) \in [0, 1].$$

with p_{ab} the probability of finding the block of symbols ab in the sequence. The $\log_6$ is used because there are $3! = 6$ possibilities of blocks ab where $a, b \in \{\bar{1}, 0, 1\}$.

By defining Φ' as a part of Φ removing the zeros and the consecutive identical symbols, we can define the Partial Information Content (PIC) by

$$M(\varepsilon) = \frac{|\Phi'(\varepsilon)|}{n-1}.$$

The main hypothesis here is to assume a statistically isotropic fitness landscape. In order to do that, we must have an unbiased sample. Moreover, Φ' represents the change of concavity during the random walk.

4.2.5.1 $H_{\max}$ and $\varepsilon_{\max}$

$H_{\max}$ feature gives us the maximum number of rugged elements

$$H_{\max} = \max_{\varepsilon} H(\varepsilon).$$

$\varepsilon_{\max}$ is the value of ε where

$$H(\varepsilon_{\max}) = H_{\max}.$$

4.2.5.2 $\varepsilon_{\mathbf{s}}$

ε_{s} gives us the maximum change of fitness found during the sequence of observations. At this point, the sequence is almost composed of 0 only

$$\varepsilon_{\mathrm{s}} = \log_{10}\left(\min_{\varepsilon}\{\varepsilon | H(\varepsilon) < 0.05\}\right).$$

4.2.5.3 M_0

This feature represents the maximum inflexion points normalized over the number of sample points

$$M_0 = M(\varepsilon = 0).$$

4.2.5.4 $\varepsilon_{\mathbf{ratio}}$

This feature corresponds to the ratio of partial information

$$\varepsilon_{\mathrm{ratio}} = \log_{10}\left(\max_{\varepsilon}\{\varepsilon : H(\varepsilon) > rM_0\}\right).$$

Usually, we compute this feature for $r = 0.5$. At $\varepsilon_{r=0.5}$, the number of inflexion points is half the number at M_0.

4.2.6 Principal Component Analysis Feature Set (pca)

Features of this set are computed using a Principal Component Analysis (PCA) to the data. They try to describe the variable scaling of a continuous problem.

The PCA is performed using either the covariance or the correlation matrix. In this context, *data* can mean the matrix of sample points and associated fitness values or the matrix of the sample points only.

4.2.6.1 Default Features

Hence, four different settings are available to perform the PCA: covariance/correlation matrix on the full design/sample points only.

On these four different settings, two values are computed as features: the relative amount of principal components that are required to explain 90% of the variance and the importance of the first principal component. Hence, with all the possibilities, this set regroups 8 features.

4.2.6.2 Additional Features

In addition of the existing features in this set, we add new features applying the same procedure as for the Dispersion features described in Section 4.2.4, *i.e.*, by filtering the data based on the best fitness values. We add three thresholds at which the sample points are filtered based on their fitness values, *i.e.*, we compute the existing features but for X% of the best points only. We originally experimented as threshold 25%, 50%, and 75% of the best points of the samples. We found that these three thresholds contained similar information and that we can consider only one of them: we kept the 25% threshold.

4.2.7 Level Set Feature Set (ls)

Level sets features were introduced by Mersmann *et al.* [MBT+11]. The features in this set are based on Linear, Quadratic or Mixture Discriminant Analysis (LDA, QDA or MDA) that are used to predict if the distribution of the objective values are below or above a calculated threshold.

The properties of these features were not investigated for two main reasons:

1. their computation time. Even if level sets features are coined "cheap"[1] features, their computation time exceed by far the computation time of every other set of features [Tan21b];
2. their low robustness. The implementation of some features of this set output *NaN* results even with the recommended number of search points, i.e., $50d$ [KPWT16]. When we reduce the sampling budget further, most of the features output *NaN* results.

Both the computation time and the low robustness of these features lead us to discard them and do not analyze their properties.

4.2.8 Cell-Mapping Feature Sets

Cell mapping feature sets [KPH+14] were introduced by Kerschke *et al.*. The idea behind all cell mapping features is to divide the search space into hypercubes. Then, some measures are computed in each hypercube and aggregated over all the cells. Three sets of features use cells: *cell mapping*, *generalized cell mapping*, and *barrier trees*.

An example of such a feature is the *angle* feature which computes for each cell the angle between the worst and the best point of the cell. This value is then aggregated with other cells using the mean or the standard deviation.

Nevertheless, there are some drawbacks to the use of these feature sets. First, it is not easy to decide the size of the discretization a priori. Moreover, the number of cells have a direct impact on the number of search points needed to compute features as sufficient number of points need to be in each cell. In the example of the angle feature, one needs to ensure at least two points by cell.

Finally, the authors of [KPH+14] point out that these techniques may not be suited to represent complex problems, especially in high dimension. For all these reasons, cell mapping features are not widely used and are therefore not included in this study.

4.2.9 SOO Feature Set

SOO features are based on the Simultaneous Optimistic Optimization algorithm [Mun11]. The idea behind this algorithm is a divide-and-conquer method that iteratively expands a tree. A more detailed explanation of this algorithm can be found in [Mun11, DLV+19].

The idea of SOO features is to extract information from the tree shape in order to characterize optimization problems. For the sake of reasonable computation time, only simple statistics are extracted from the tree such as the number of nodes, number of leaves, the average heights of leaves, etc.

Overall, Derbel *et al.* introduce 214 tree-based features divided in five groups: tree shape, tree fitness, global deviation, derivative and Lipschitz and tree dynamics features [DLV+19]. Since there is no open-source code for these features and because these features are not based on sample points, we rule them out of our studies.

[1] Being computed on a fixed sample, without additional calls to the objective function, see Section 4.2.10.

4.2.10 Expensive Feature Sets

Features introduced by Mersmann *et al.* [MBT+11] have been divided into two types of sets in [Bel17, BDSS17], *cheap* and *expensive* sets.

The main difference between cheap and expensive sets resides in the number of search points that are needed. Cheap sets requires only any fixed number of search points to be computed. Expensive sets may also require an additional number of point to be computed For instance, *convexity* features requires 1000 repetitions of the features computation or features from the *local search* set require multiple runs of a Nelder-Mead algorithm.

The extra number of points needed in order to compute the features are a major obstacle to the use of these sets in a black-box optimization context, especially when the evaluation of the objective function is quite expensive.

This may explain why in many applications, only cheap features are used and expensive ones are rejected. In this study, as previously done in other studies [BDSS17, JPED21, KPWT16, SEK20], we will not use expensive features and concentrate only on cheap feature sets.

Part III

Analysis of Landscape Features

Chapter 5

Exploratory Landscape Features Properties

Contents

ELA features have been successfully used for many applications [Bel17, XHHL11, KKB+18, KHNT19] but our understanding of features properties is rather weak.

As landscape features are computed using sample points, the numerical value obtained as an output of the computation is not the true value of the feature but only a sample of the feature values' distribution.

The distribution of features values may be impacted by several factors:

- the randomness of different samples. Given n sample points, feature values may differ between two samples as the search space is sampled slightly differently;
- the method used to sample search points. Different ways of sampling points may result in different part of the search space sampled;
- the number of search points. An increasing or decreasing number of search point may impact the feature values. More or fewer parts of search space could be sampled;
- transformations of the fitness function or transformations of the search space. Rotations and translations of the fitness function as long as a normalization of the search space could impact the feature values;
- the addition of noise in samples. Features values may be impacted if the original sample points are slightly modified from their original coordinates.

In this chapter, we define six properties features should satisfy. Five are based on the factors that could impact the distributions of feature values and one corresponds to the ability to differentiate optimization problems. We also study the distributions of feature values in order to determine to what degree features satisfy the six properties.

As most properties can be visualized directly using the distributions of feature values, we do not quantify to what degree a feature satisfies a property. Nevertheless, we give a measure of the ability of features to differentiate optimization problems. The measure is based on classification as features are often used in a machine learning context to differentiate optimization problems.

Section 5.1 presents the way features are computed while Sections 5.2 to 5.7 define and study the different properties. Section 5.8 discusses the results of previous sections.

5.1 Design of Experiments

In this section, we describe the different computations performed in order to analyze the different properties presented below. We selected as test suite the noiseless functions of the Black-Box Optimization Benchmark (BBOB) [FHRA10] from the COCO (Comparing Continuous Optimizers) platform [HAR+21]. Landscape features need to be approximated using sample points. For their value approximation, we sample for each of the 24 functions f a number n of points $x^{(1)}, \ldots, x^{(n)} \in [-5, 5]^d$ [1], and we evaluate their function values $f(x^{(1)}), \ldots, f(x^{(n)})$. The set of pairs $\{(x^{(i)}, f(x^{(i)})) \mid i = 1, ..., n\}$ is then fed to the *flacco* package[2] [KT19b], which returns a vector of features. For each function, we create 100 independent samples of n search points which results in 100 feature vectors for each of the 24 BBOB functions. We repeat the same procedure for each of the first five instances of BBOB functions.

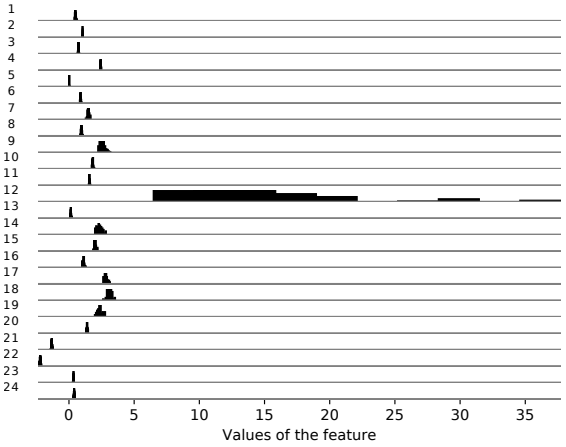
The *flacco* package covers a total number of 343 features [KHNT19], which are grouped into 17 feature sets. However, some of these features are often omitted in practice because they require adaptive sampling, see [BDSS16, KT19a, MG14, PRH19] for a discussion. Thus, we compute 7 feature sets from the *flacco* package: *dispersion* [LW06], *information content* [MKH15], *nearest better clustering* [KPWT15], *level set* [MBT+11], *meta model* [MBT+11], *y-distribution* [MBT+11], and *principal component analysis* [KT19b]. Overall, we compute 62 features: 46 from *flacco* and 16 new PCA features introduced in Section 4.2.6.

The value of each feature may be normalized between 0 and 1 where 0 (resp. 1) correspond to the smallest (resp. largest) value encountered in the approximated feature values.

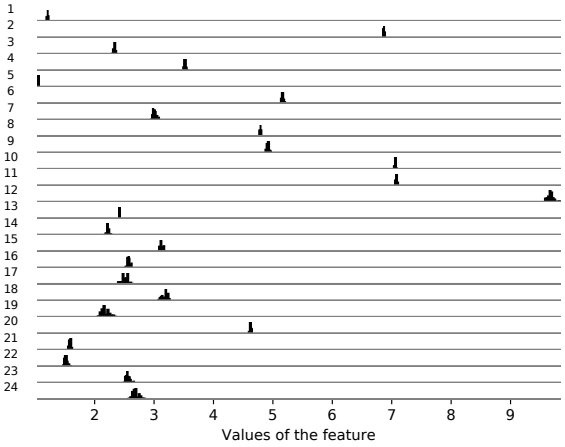
Most of the data presented in this paper can be found in [RDDD20a] or [RDDD21b].

[1] The BBOB functions are traditionally studied over the search domain $[-5, 5]^d$

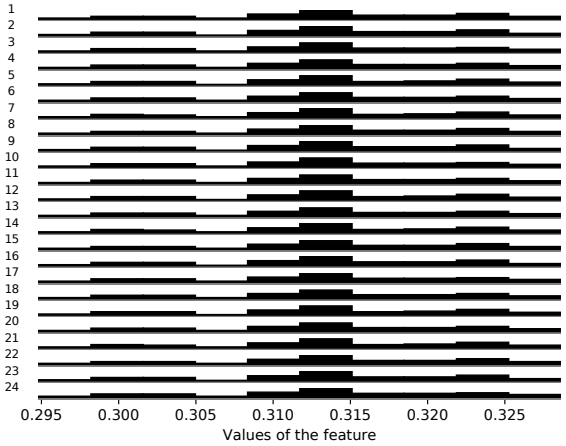
[2] version 1.8



(a) y-distribution skewness feature



(b) Information Content ε_s feature



(c) PCA covariance of the first principal component of the samples feature

Figure 5.1: Distributions of a y-distribution, an Information Content, and a PCA features for 3,125 search points in $d = 5$.

5.2 Stability

5.2.1 Definition

We define the *stability* of a feature as its ability to keep the same values with independent samples of same size. Put differently, we would like the distributions of feature values to have a small variance when the n search points are sampled multiple times. Hence, a stable feature will have a narrow distribution contrary to very unstable features.

5.2.2 Graphical visualization

Figure 5.1 presents the distributions of three features on the 24 BBOB functions. Each row correspond to function of the test suite. Feature values are not normalized for these visualizations.

Of the three features presented in Figure 5.1, two of them display a great stability, the information content feature (Figure 5.1b), and the PCA feature (Figure 5.1c). Even if Figure 5.1c seems to show a large dispersion, the minimum and maximum values of the feature are separated by 0.03 which make this feature stable. This stability is reflected by a low average standard deviation across all functions of the benchmark.

Conversely, the y-distribution feature displays a huge variation of values between the 100 runs. This can be easily seen on Figure 5.1a, especially on function 12 where the standard deviation of the distribution on this function is the greatest.

Overall, most feature are quite stable. Only some Meta-model and y-distribution features exhibit a large variance and thus are unstable.

5.3 Influence of Sampling Strategy

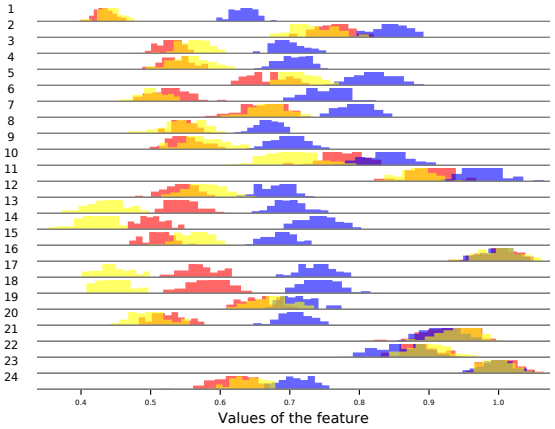
5.3.1 Definition

As mentioned previously, *exploratory landscape analysis* [MBT+11] relies on sampled points in the search space. To analyze whether the sensitivity of the random feature value approximations depend on the strategy, we investigate a total number of five different sampling strategies from three categories: uniform sampling, Latin Hypercube Sampling and extracting samples from a low-discrepancy sequence.

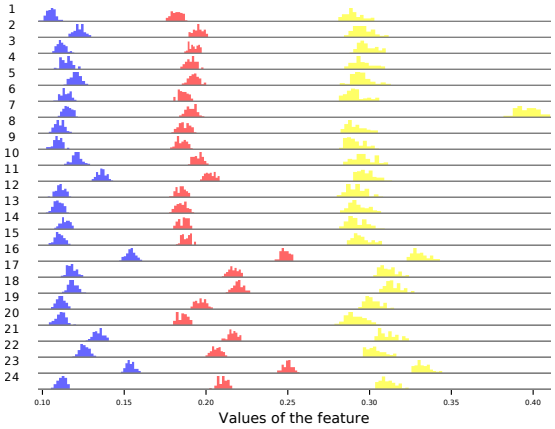
Uniform Sampling We compare uniform random sampling based on two different pseudo-random number generators:
- **random:** We report under the name *random* results for the Mersenne Twister [MN98] random number generator. This generator is commonly used by several programming languages, including Python, C++, and R. It is widely considered to be a reliable generator.
- **RANDU:** we compare the results to those for the linear congruential number generator RANDU. This generator is known to have several deficits such as an inherent bias that results in the numbers falling into parallel hyper-planes [Knu98]. We add this generator to investigate whether the quality of the random sampling has an influence on the feature value approximations.

Latin Hypercube Sampling (LHS) LHS [MBC79] is a commonly used quasi-random method to generate sample points for computer experiments. In LHS, new points are sampled avoiding the coordinates of the previously sampled points. More precisely, the range of each coordinate is split into n equally-sized intervals. From the resulting $n \times \ldots \times n$ grid the points are chosen in a way that each one-dimensional projection has exactly one point per interval.
- **LHS:** Our first LHS designs are those provided by the *pyDOE* Python package (version 0.3.8). We use the centered option, which takes the middle point of each selected cube as sample.
- **iLHS:** The *"improved"* LHS (**iLHS**) designs available in *flacco*. This strategy builds on work of Beachofski et Grandhi [BG02]. Essentially, it implements a greedy heuristic to choose the next points added to the design. At each step, it first samples a few random points, under the condition of not violating the Latin Hypercube design. From these candidates the algorithm chooses the one whose distance to its nearest neighbor is closest to the ideal distance $n/\sqrt[d]{n}$.

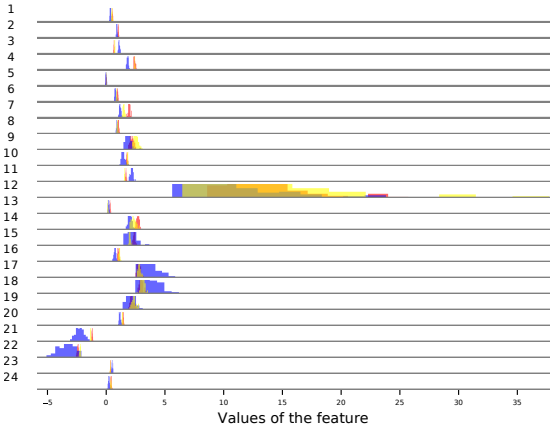
Sobol´ low-discrepancy sequence We add to our investigation a third type of sampling strategies, the sequences suggested by **Sobol´** in [Sob67]. Sobol´ sequences are known to have small *star discrepancy*, a property that guarantees small approximation errors in several important numerical integration tasks. They are also commonly used in *Design of Experiment* (DoE) tasks and in



(a) [Dispersion ratio of the mean with a 2% threshold feature



(b) Nearest Better Clustering coefficient of variation feature



(c) y-distribution skewness feature

Figure 5.2: Distributions of a Dispersion, a Nearest Better Clustering, and a y-distribution features for 3,125 search points in $d = 5$. The sampling strategies are Sobol´ (yellow), uniform sampling (red) and Latin Hypercube sampling (blue).

the initialization of Bayesian Optimization techniques [SWN03]. The interested reader is referred to [DP10, Mat09] for an introduction to these important families of quasi-random sampling strategies. For our experiments we generate the Sobol´ sequences from the Python package *sobol_seq* (version 0.1.2), with randomly chosen initial seeds.

5.3.2 Graphical visualization

Figure 5.2 presents the distributions of three features on the 24 BBOB functions. Each row correspond to function of the test suite. Feature values are not normalized for these visualizations. We compare on this figure three sampling strategies: Sobol´ low discrepancy sequence, uniform sampling and Latin Hypercube sampling.

On Figure 5.2, two features are sensitive to the sampling strategy. The nearest better clustering feature (Figure 5.2b) seems to be the more sensitive as there is no overlapping between distributions of each sampling strategy. Though, some kind of pattern seems to emerge as it looks

like distribution of values seems to be translated from one sampling strategy to another. The dispersion feature (Figure 5.2a) is also sensitive to the sampling strategy. Nevertheless, not in the same way as the nearest better clustering one. Here, for most functions, the three distributions are overlapping. For instance, distribution are almost the same for functions 16 and 23 or are widely overlapping for functions 2 or 19. For this feature, distribution coming from Sobol´ and uniform samples often overlap which is not the case for distributions coming from LHS samples .

The y-distribution feature (Figure 5.2c) is not much influenced by the sampling method as almost all distributions fully coincide for all functions.

Overall, the only features not impacted by the sampling strategy are those based on fitness values only, i.e., y-distribution features. Most of the features are impacted as the Dispersion feature presented in Figure 5.2a is. Only Nearest Better Clustering features present massive differences between sampling strategies. We also observe in [RDDD20b] that sampling strategies does not give the same levels of performance when used with machine learning. The Sobol´ low-discrepancy sequence seems to give better performances than the others in the machine learning context.

5.4 Expressiveness

5.4.1 Definition

We define the *expressiveness* of a feature as its ability to distinguish between several optimization problems [RDDD19]. Hence, the more expressive a feature is, the more problems it can distinguish.

5.4.2 Measure

In this thesis, we measure the expressiveness of a feature as the accuracy of classification of the 24 BBOB functions. We perform this analysis in dimension $d = 5$ and we use $50d = 250$ search points to compute the feature values. We sample these points from Sobol´ sequence. This is different from our previous work [RDDD19], where we have used Halton points instead.

For each feature, we use 80 uniformly chosen feature value (per function) out of 100 independent runs for training a classifier that, given a previously unseen feature value, shall output which of the 24 functions it is faced with. We test the classifier with all 20 feature values that were not chosen for the training, and we record the average classification accuracy, which we measure as the fraction of correctly attributed function labels. We apply 20 independent runs of this uniform random sub-sampling validation, i.e., we repeat the process of splitting the 24×100 feature values into 24×80 training instances and 24×20 test instances 20 independent times.

We use the *scikit learn* [PVG+11, version 0.21.3] implementation of a K **Nearest Neighbors (KNN)** (we use $K = 5$) classifier to perform the classification.

5.4.3 Graphical visualization

Figure 5.1 can also be used to visualize expressiveness.

For every function in figure 5.1c, the distribution is the same which makes impossible to distinguish one function from another which is reflected by a really low accuracy of classification of 1.8%.

Conversely, both features in figures 5.1a and 5.1b displays more scattered distributions across the possible feature values. This makes it easier to distinguish optimization problems even with a graphical visualization. Nevertheless, the information content feature seems to be more expressive than the y-distribution feature. The ability to distinguish optimization problems seen with

the distributions is reflected with the accuracy of classification. The y-distribution feature, i.e., skewness, has an accuracy of classification of 40.3% while the ε_s feature reaches 72.4% accuracy.

Overall, almost all features show some expressiveness excepted PCA features based on samples only. The most expressive features belong to the meta-model and the information content feature sets. In the information content set, the most expressive feature is ε_s. The most expressive feature of the meta-model set is also the most expressive feature overall. It is the intercept feature with 97.3% of accuracy.

5.5 Robustness

5.5.1 Definition

We define the *robustness* [RDDD19] of a feature as its ability to keep the same values with different number of search points. Put differently, a robust feature has distributions of feature values extracted with different number of search points that are close. It is quite natural that the distributions' variances should increase as the number of samples is reduced but robust feature should display small increase of their variance.

5.5.2 Graphical visualization

Figure 5.3 presents the distributions of three features on the 24 BBOB functions. Each row correspond to function of the test suite and the three columns by feature represents the three different sample sizes. Feature values are not normalized for these visualizations.

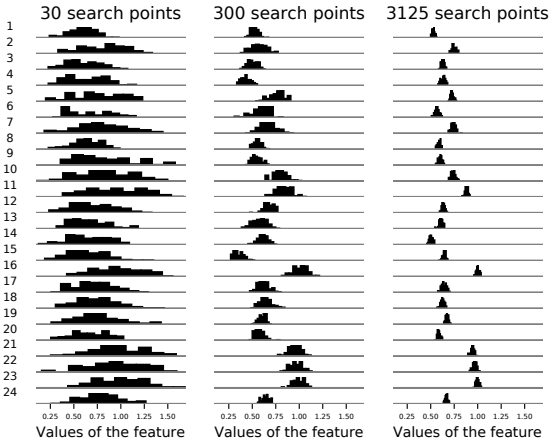
As expected, nearly all features suffer from a standard deviation increase when the number or search points decreases. This subsampling estimation error behavior was expected as it gets more and more difficult to evaluate precisely one fitness function property when less samples of search space are available, i.e., fewer parts of the search space are explored and might not be the same for two independent runs.

Nevertheless, some features are less sensitive to this phenomenon. For instance, the PCA feature (Figure 5.3b) displays a low variation of its distribution standard deviation. On most BBOB functions, the standard deviation increases only slightly. The larger growths can be found on functions 21, 22 and 23 and even if larger standard deviation can be observed, the center of mass of the distribution is still roughly the same.

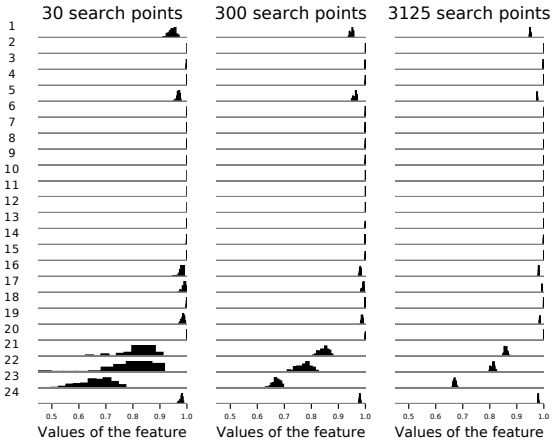
Contrariwise, both dispersion (Figure 5.3a) and meta-model (Figure 5.3c) features have growing standard deviations and a moving center of mass for nearly all functions. The only exception can be found for function 5 on the meta-model feature. This function is the linear slope and the adjusted R^2 of (LM) feature can recognize this function whatever the number of search points used.

The main difference between the dispersion and the meta-model features is the growth of the standard deviation. In the case of the ratio of the median with a 5% threshold feature, distributions for every functions on the lowest setting are almost as wide as the range of possible values taken by the functions in the BBOB benchmark which is not the case on the meta-model feature.

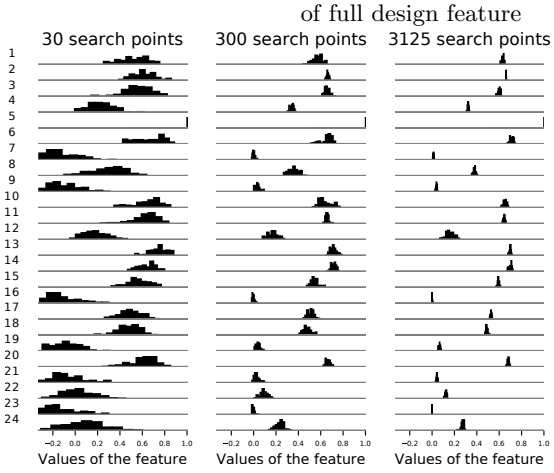
Some features show no increase in the dispersion of feature values and thus are really robust: the PCA features. For most features, we observe an increasing variance as the number of points decreases. Surprisingly, on some functions, y-distribution features show a decreasing variance as the number of points decreases. For some of them, the center of mass of the distribution is also moving and can be quite different for two different sample sizes. It is in particular the case for all dispersion features.



(a) Dispersion ratio of the median with a 5% threshold feature



(b) PCA covariance of the first principal component of full design feature



(c) Meta-model adjusted R^2 of (LM) feature

Figure 5.3: Distributions of a Dispersion, a Principal Component Analysis, and a Meta-Model features for 30, 300 and 3,125 search points in $d = 5$.

5.6 Invariance to Transformations

In this section, we differentiate two types of transformations. The first type is directly related to the BBOB test bed where multiple instances of each function can be generated. These instances are generated using transformation of the fitness function such as translation, rotation and/or scaling. In this section, we both study the impact of these transformations and the impact of the search space normalization on feature values.

5.6.1 Transformation of the fitness function

5.6.1.1 Definition

To visualize the impact of the transformation of the fitness function, we compute feature values for the first five instances of the BBOB test bed and compare them. In [SEK20], the authors also look at the impact of scaling and translations on feature values but applied these transformations directly without taking BBOB instances.

5.6.1.2 Measure

The measure of invariance is computed as follow. We record the standard deviation of distributions for the first five instances of each 24 BBOB functions in dimension $d = 25$ for a budget of 2,500 search points. For each instance, we compute the distance to the median value of the first instance distribution. The measure is given by the average distance of instance two to five to the distribution of the first instance. A greater value will imply that instances two to five are quite different from the first.

5.6.1.3 Graphical visualization

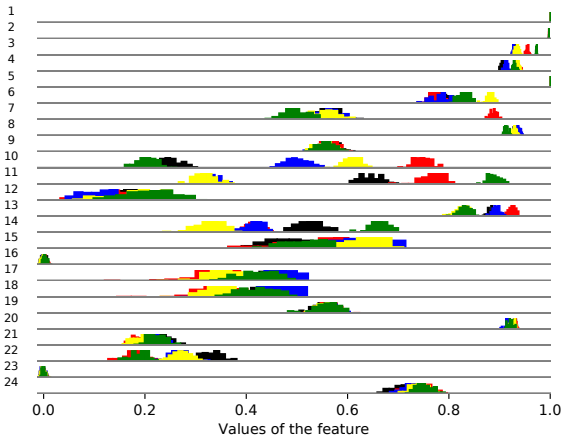
Figure 5.4 presents the distributions of three features on the 24 BBOB functions. Each row correspond to function of the test suite. We compare on this figure the first five BBOB function instances.

In Figure 5.4, two features are not invariant to transformations. Both the dispersion feature in Figure 5.4c and the meta-model feature in Figure 5.4a exhibit non overlapping distributions on all instances for some functions. Interestingly, it seems that this result also depends on the objective function. In Figure 5.4c, while no distribution are overlapping for function 8, distributions for the five instances are almost identical for functions 5, 6, 9, 13, 16, 19, 23 and 24. On other function, at least one instance is different from the others.

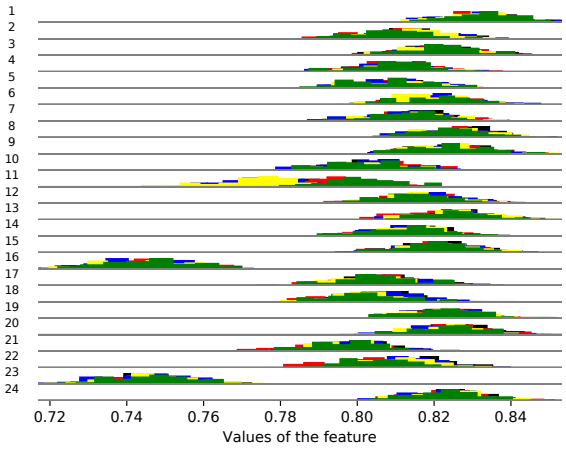
We observe the same behavior in Figure 5.4a but for different functions. Feature values are invariant on functions 1, 2, 5, 9, 16, 19, 20 and 23. On the contrary, distributions are far apart on functions 10 and 11.

Distributions displayed in Figure 5.4b are all overlapping and thus, this feature is invariant to the transformations for all functions.

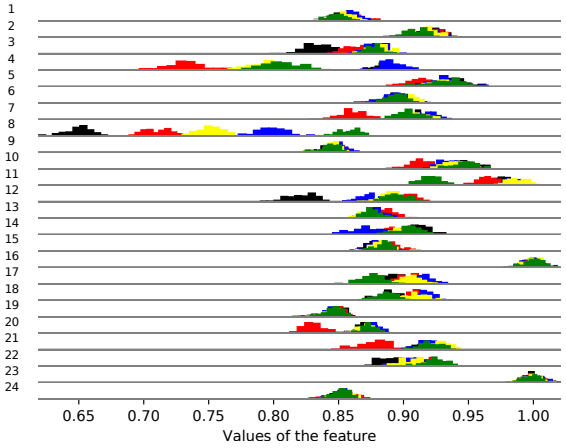
Overall, most of the features have similar behavior as the dispersion and meta-model features. These features are often invariant to transformations for some functions but not all of them. Only Nearest Better Clustering features seems to be invariant for all functions.



(a) Meta-model adjusted R^2 of (QM) feature



(b) Nearest Better Clustering correlation feature



(c) Dispersion ratio of the mean with a 5% threshold feature

Figure 5.4: Distributions of a Meta-Model, a Nearest Better Clustering, and a Dispersion features for 2,500 search points in $d = 25$. Colors represent the BBOB instance: black for instance 1, red for instance 2, blue for instance 3, yellow for instance 4 and green for instance 5.

5.6.2 Normalization of the search space

5.6.2.1 Definition

In this part, we study the impact of the normalization of the search space. Here, the fitness function remains the same but we normalize the search space.

To do so, we generate n search points $x^{(1)}, \ldots, x^{(n)} \in [-5, 5]^d$ and compute the fitness value $f(x^{(1)}), \ldots, f(x^{(n)})$ associated to every points. Then, we normalize the $x^{(i)}$ to obtain $\bar{x}^{(1)}, \ldots, \bar{x}^{(n)} \in [0, 1]^d$ and we keep the same fitness values $f(x^{(1)}) = f(\bar{x}^{(1)}), \ldots, f(x^{(n)}) = f(\bar{x}^{(n)})$. Thus, we compute feature values using $\bar{x}^{(i)}$ as sample points and investigate the differences with feature approximations obtained with the $x^{(i)}$.

5.6.2.2 Graphical visualization

Figure 5.5 presents the distributions of three features on the 24 BBOB functions. Each row correspond to function of the test suite. We compare on this figure the distribution obtained with normalized samples to the original distribution.

We can observe in Figure 5.5c and 5.5b that the normalized samples have almost no effect on these two features. It is nearly impossible to differentiate the distributions coming from one set of points from another.

Conversely, the dispersion feature in Figure 5.5a exhibits a huge sensitivity to normalization as the distributions are quite far apart. Moreover, we also find that the variance of distribution was reduced when the normalization was applied.

Overall, most feature are insensitive to normalization. The only features that have different distributions of feature values when we applied normalization are Dispersion features based on differences and not on ratios. Information Content features based on ε (*i.e.*, $\varepsilon_{\max}$, ε_{ratio}, ε_s) are also sensitive to the normalization.

5.7 Sensitivity to Noise

5.7.1 Definition

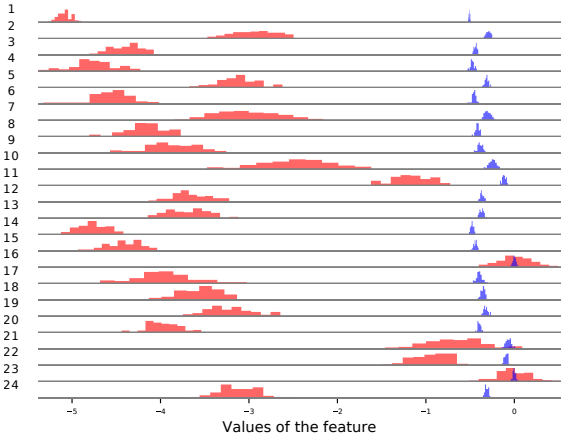
In this part, we examine the feature approximations when noise is applied to sample points. Noisy inputs are generated using the points from the Sobol′ sampling and by adding a uniform noise $U(-0.5, 0.5)$ on each coordinate: $\tilde{x_i} = x_i + a, a \sim U(-0.5, 0.5)$. These new sample points are associated with the corresponding fitness values, i.e., $f(\tilde{x_i}) = f(x_i)$ As in Sec. 5.1, we generate 100 independent feature vectors by function in order to compare their distribution with the true feature value.

We expect that feature non-sensitive to this noise to have distributions close to the initial one.

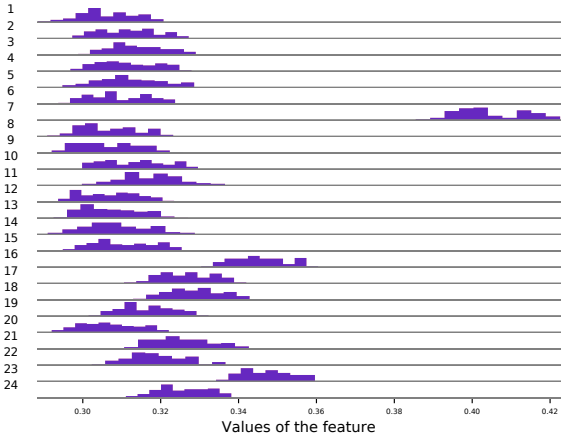
5.7.2 Graphical visualization

Figure 5.6 presents the distributions of three features on the 24 BBOB functions. Each row correspond to function of the test suite. We compare on this figure the distribution obtained with noisy samples to the true distribution.

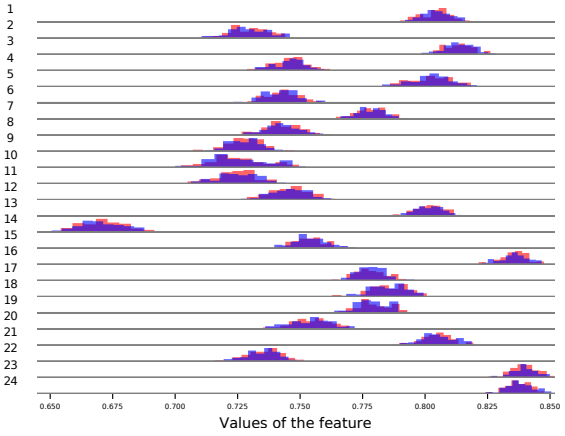
As seen in Figure 5.6, the addition of noise does not have a strong impact on feature values' distributions. For both features, distributions are widely overlapping with medians relatively unchanged. While the variance of distributions coming from noisy samples are similar to the original samples in Figure 5.6b, it is not the case for Figure 5.6a on some functions. The variance



(a) Dispersion feature difference of the mean with a 2% threshold

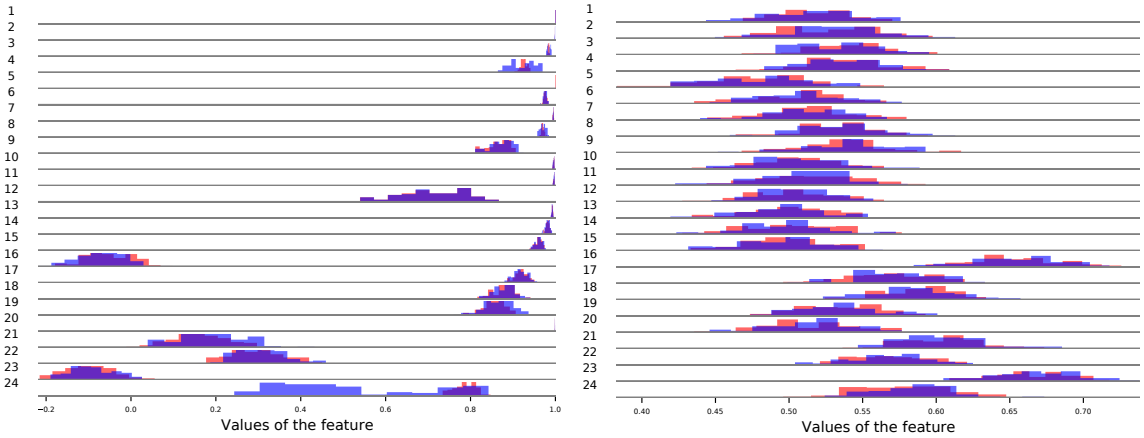


(b) Nearest Better Clustering coefficient of variation feature



(c) Information content $H_{\max}$ feature

Figure 5.5: Distributions of a Dispersion, a Nearest Better Clustering, and an Infomation Content features for 3,250 search points in $d = 5$. Red distributions correspond to the original data while blue distributions correspond to normalize data.



(a) Meta-model adjusted R^2 of (QMI) feature.

(b) Information content M_0 feature.

Figure 5.6: Distributions of a Meta-Model and an Information Content features for 3,250 search points in $d = 5$. Noisy samples are generated using random uniform variation. Red corresponds to the original sample while blue corresponds to noisy samples.

of noisy distributions is greater than the original ones for functions 4, 8 and 24. Moreover, function 24 exhibits feature values that are very different when computed with noisy samples.

Overall, most features have the same behavior as the Information Content feature presented in Figure 5.6b.

5.8 Discussion

We found that no feature fully satisfy the six properties. Table 5.1 shows the features and if they satisfy the different properties. Out of the six properties, one is key: expressiveness, as the property it focuses on is distinguishing between different optimization problems. As the main goal of ELA features is distinguishing between different optimization problems, a feature not satisfying this property will be hard to use in any context.

The other five properties characterize the sensitivity of landscape features to its inputs: the sample points and fitness values. The characterization of the sensitivity to the inputs helps the user to know when to use a certain type of features. In any applications, one would favor expressive features but other properties are not always necessary:

- stability is useful when few independent samples are used;
- invariance to noisy samples is only important when for some reasons sample points may be affected by some minor transformations;
- the influence of the sampling strategy should be kept in mind in order to not change the chosen sampling method or if the user decides to use the points sampled during the run of an optimization algorithm;
- robustness should be disregarded if a large number of sample points can be afforded. When fewer sample points are required by the time consuming evaluation of the fitness function, the robustness becomes a key property for a feature;

- the invariance to transformation of the fitness function is not needed when the user only have one instance. The same holds if we are not performing any normalization.

For example, in a case where we have no transformation of the fitness function, no noisy sample, an identical sampling strategy for all samples, 100 independent runs and an expensive evaluation of the objective function, we should only look at robust features. In this case, we need to have features that are still expressive when the number of sample points is low.

Overall, the practitioner should use feature properties regarding what is needed for her application.

Table 5.1: Exploratory Landscape Analysis feature properties. Expressivenes is given with the accuracy of classification. Other properties are marked with ✓when a feature satisfies a property and ✗otherwise.

Set	Feature	Expressiveness	Stability	Robustness	Invariance to transformations	Invariance to normalization	Invariance to noise	Invariance to sampling strategy
yD	Kurtosis	22.4	✗	✓	✓	✓	✓	✓
yD	Skewness	40.3	✗	✓	✓	✓	✓	✓
yD	# of peaks	13.6	✗	✓	✓	✓	✓	✓
mm	(LM) R^2	36.9	✓	✗	✗	✓	✓	✗
mm	(LMI) R^2	44	✓	✗	✗	✓	✓	✗
mm	(QM) R^2	48.4	✓	✗	✗	✓	✓	✗
mm	(QMI) R^2	55	✓	✗	✗	✓	✓	✗
mm	Intercept (LM)	97.3	✗	✗	✗	✗	✓	✗
mm	Minimum (LM)	35.7	✗	✗	✓	✗	✓	✗
mm	Maximum (LM)	75.1	✗	✗	✗	✗	✓	✗
mm	Ratio max/min (LM)	17.2	✗	✗	✗	✗	✓	✗
mm	Conditionning (QM)	22.9	✗	✗	✓	✗	✓	✗
nbc	Mean ratio	10.5	✗	✗	✓	✓	✓	✗
nbc	sd ratio	13.4	✗	✗	✓	✓	✓	✗
nbc	Correlation	8.4	✗	✗	✓	✓	✓	✗
nbc	Coeff of variation	8.8	✗	✗	✓	✓	✓	✗
nbc	Fitness correlation	25.1	✗	✗	✓	✓	✓	✗
disp	Ratio mean 2%	10.8	✗	✗	✗	✓	✓	✗
disp	Ratio mean 5%	16.6	✗	✗	✗	✓	✓	✗
disp	Ratio mean 10%	16	✗	✗	✗	✓	✓	✗
disp	Ratio mean 25%	20.7	✗	✗	✗	✓	✓	✗
disp	Ratio median 2%	10	✗	✗	✗	✓	✓	✗
disp	Ratio median 5%	14.2	✗	✗	✗	✓	✓	✗
disp	Ratio median 10%	16.3	✗	✗	✗	✓	✓	✗
disp	Ratio median 25%	16.5	✗	✗	✗	✓	✓	✗
disp	Diff mean 2%	10.7	✗	✗	✗	✗	✗	✗
disp	Diff mean 5%	15.4	✗	✗	✗	✗	✗	✗
disp	Diff mean 10%	16.9	✗	✗	✗	✗	✗	✗
disp	Diff mean 25%	18.8	✗	✗	✗	✗	✗	✗
disp	Diff median 2%	9.6	✗	✗	✗	✗	✗	✗
disp	Diff median 5%	15.2	✗	✗	✗	✗	✗	✗
disp	Diff median 10%	15.7	✗	✗	✗	✗	✗	✗
disp	Diff median 25%	13.8	✗	✗	✗	✗	✗	✗
ic	$H_{\max}$	13.7	✗	✗	✓	✗	✓	✗
ic	$\varepsilon_{\max}$	48.75	✓	✗	✗	✗	✓	✗
ic	ε_{s}	72.4	✓	✓	✗	✗	✓	✗
ic	M_0	10.6	✗	✗	✓	✗	✓	✗
ic	$\varepsilon_{\mathrm{ratio}}$	55.8	✓	✓	✗	✗	✓	✗
pca	Cov x	4.3	✓	✓	✓	✓	✓	✗
pca	Cor x	4.2	✓	✓	✓	✓	✓	✗
pca	Cov *init*	12.5	✓	✓	✓	✓	✓	✗
pca	Cor *init*	5.3	✓	✓	✓	✓	✓	✗
pca	PC1 cov x	1.8	✓	✓	✓	✓	✓	✗
pca	PC1 cor x	1.7	✓	✓	✓	✓	✓	✗
pca	PC1 cov *inti*	87.2	✓	✗	✗	✓	✓	✗
pca	PC1 cor *init*	18.3	✓	✗	✗	✓	✓	✗
pca	Cov x 25%	8.3	✓	✓	✗	✓	✓	✗
pca	Cor x 25%	8.1	✓	✓	✗	✓	✓	✗
pca	Cov *init* 25%	27.3	✓	✗	✗	✓	✓	✗
pca	Cor *init* 25%	11.4	✓	✗	✗	✓	✓	✗
pca	PC1 cov x 25%	26.5	✓	✗	✗	✓	✓	✗
pca	PC1 cor x 25%	25.8	✓	✗	✗	✓	✓	✗
pca	PC1 cov *inti* 25%	86.8	✓	✗	✗	✓	✓	✗
pca	PC1 cor *init* 25%	27.8	✓	✗	✗	✓	✓	✗

Part IV

Optimization of Radar Networks

Chapter 6

Background on Radar Operation

Contents

6.1 Introduction

Part II of this thesis presented the background on landscape-aware algorithm selection and contributions on exploratory landscape analysis features.

This part of the thesis focus on the application of the landscape-aware approach on a real-world problem. The problem to solve is a radar network configuration problem. The goal is to find a good set of parameters for radars in order to maximize some criterion.

This part first introduces in Chapter 6 the basic principles of radar processing. In Chapter 7, we describe the radar modeling and the use-cases to solve in this thesis. The objective function in these use-cases is directly based on the basic principles presented in Chapter 6.

Chapter 8 and Chapter 9 focus on the resolution of the problem. The former presents the performances of optimization algorithms on the use-cases while the latter presents the results of the landscape-aware approach.

6.2 History of Radar Development

The term RADAR is an acronym for RAdio Detecting And Ranging. Radar working principle with electromagnetic waves is very similar to sound waves reflection. When a pulse is sent by the system, a small part is sent back by the target: the echo. Radars use the echo in order to find the direction and location of a target.

The development of the radar technology was done by several researchers and inventions over many years. The first experiments were conducted by Heinrich Hertz in 1886 visualizing Maxwell's theory of electromagnetism by using an antenna and actually demonstrating the presence of magnetic fields. The first detection of an object using electromagnetic waves was performed by Christian Hülsmeyer in 1904 following Tesla's suggestion to detect objects using electromagnetic waves in 1900. His *Telemobiloskop* permitted to detect a metal boat in the sea. Technological breakthroughs that appeared right after World War I lead researchers from the Naval Research Laboratory (USA) to detect a wooden boat in 1922 and an aircraft in 1930. Many improvements on sensing and tracking targets were done with the upcoming World War II and the Cold War. These improvements led to the deployment of many radars, especially in Europe around Germany borders. Nowadays, radars are assets for both military defense and civilian applications such as flight control, weather forecasting, topography or even geology.

6.3 Basic Principle

As mentioned before, a radar can detect objects by using electromagnetic waves and can infer from these waves some information such as the direction, *i.e.*, the object radial speed, and the distance of an object. This process can be divided into three parts [Bri17]:

1. an electromagnetic wave is sent in a scanning direction;
2. a small part of the wave is propagated back to the antenna: the echo;
3. the radar processes the echo and estimates the distance and the radial speed of the object.

However, the signal is often polluted with noise that can come from other objects or from the environment. The signal can also have some ambiguity is distance or in speed which can make the detection harder. An aspect of the electronic warfare is to make the target create these ambiguities in order to perturb the detection system. For the sake of simplicity, we will not explain in detail these ambiguities but the interested reader can take a look at [Car19, Sko01, Bri17].

6.4 Radar Equation

The radar equation describes the relationship between range, the wave propagation and radar characteristics. It is given by the relation [Sko01]

$$R^4 = \frac{P_r G^2 \sigma \lambda^2}{P_t (4\pi)^3}, \tag{6.1}$$

with:

- R the distance antenna-target in meters (m);
- P_t the transmitted power in watts (W);
- G the antenna gain in decibel (dB);
- σ the radar cross section in square meters (m^2);
- λ the signal wavelength in meters (m);
- P_r the power received in watts (W).

In this equation, the power transmitted P_t, the antenna gain G and the wavelength λ depends on the radar type and can be replaced by a constant V.

In order to compare the power of the signal with the power of the noise, we can compute the *signal-to-noise* ratio. This value directly impacts the ability of a radar to detect a target.

The signal-to-noise ratio is obtained by introducing the power of the noise P_n in (6.1). The power of noise depends only on the radar and is constant. We can regroup all constants in (6.1) and the power of the noise under K to obtain

$$\frac{S}{N} = \frac{P_r}{P_n} = K\frac{\sigma}{R^4}. \tag{6.2}$$

6.5 Radar Cross Section

The Radar Cross Section (RCS), denoted as σ, is a property of a target that is included in the radar equation. It is measured in square meters (m^2). The RCS represents the amount of signal returned to the antenna by the target. Put differently, it is a measure of how detectable a target is. Its a priori computation is quite complicated and only possible for simple targets. In practice, for complex targets, the value of the RCS is estimated when the radar detect the target.

The RCS depends on four characteristics:

1. the target size and geometry: bigger targets will present a bigger detectable surface and hence a bigger RCS;
2. the direction of the radar beam: it is harder for a radar to sense a plane facing it than a plane on the side at the same distance. This is mainly because the plane exhibits more fuselage when it is on the side and, is thus easier to detect;
3. the radar frequency;
4. the target materials: reflective materials will increase the RCS as they propagate more echo.

Figure 6.1 represents an example of a plane RCS. At each angle, a radar beam is sent to the plane and the value of the cross section is plotted in red. On this figure, we can see that the cross section is very sensitive to noise but also to the shape of the plane, *i.e.*, the RCS is bigger for beams coming to the sides of the plane than for those coming to the front or to the back.

In this thesis, in order to keep the model simple, we define the RCS of the target as a simple function of its direction and radar locations.

6.6 Swerling Models

As a target moves, the reflected signal may fluctuate depending on the RCS. Peter Swerling developed five statistical models [Swe54] to describe the different properties of RCS of complex surfaces and target dynamics:

- Swerling 0 or Swerling V is a toy model where the RCS is constant, *i.e.*, a sphere with no fluctuations;
- Swerling I describes slow fluctuations, *i.e.*, the RCS is assumed constant during a scan where a scan is composed of several radar beam pulses;
- Swerling II is similar to Swerling I but describes faster fluctuations, from pulse to pulse this time. Pulses are components of a radar scan;

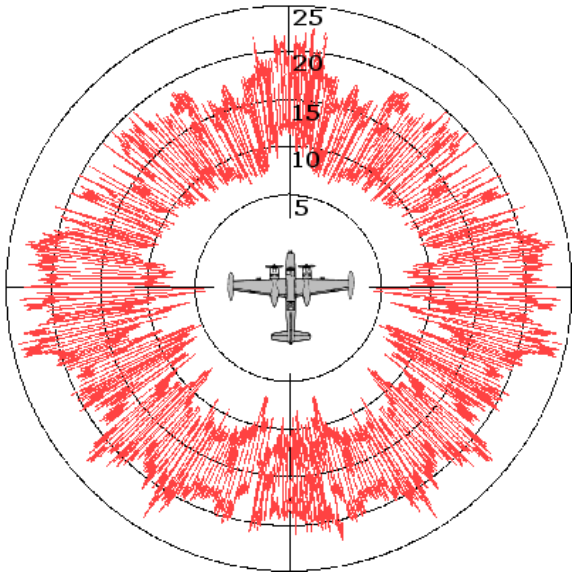


Figure 6.1: Example of RCS of a plane [Sko01].

- Swerling III is also similar to Swerling I but assumes that targets are more complicated objects such as one big reflective area and multiple small ones;
- Swerling IV is similar to Swerling III but for faster fluctuations.

Swerling I and II usually represents aircraft where Swerling III and IV may represents ships. For a more detailed presentation of Swerling models, the interested reader may consult [Sko01, Car19].

In this work, we wanted a more realistic target than a sphere but without any fluctuations. We made the choice to have a target in between Swerling 0 and Swerling I, i.e, a shape more complex than a sphere but excluding all noises that come with Swerling I. A complete description of the target used in this thesis can be found in Section 7.2.

6.7 Probability of Detection

Given that the signal is polluted with noise and/or ambiguities, detecting a target is not a binary task, *i.e.*, this is not a situation where the target is either detected or not detected. The detection relies on a performance indicator which is called the *probability of detection*.

The instantaneous probability of detection P_d depends on five characteristics:

1. the target RCS σ;
2. the target Swerling model, *i.e.*, its fluctuation law;
3. the signal-to-noise ratio;
4. a false alarm probability P_{fa}, *i.e.*, a false alarm corresponds to a fake detection by the radar due to noise;
5. radar processing.

Ultimately, the probability of detection depends also on the position of the target and its radial speed. In general, the computation of the probability of detection is very complex and depends very much on the radar. In a case of a non-fluctuating target, *i.e.*, Swerling 0, the probability of detection looks like the curve in Figure 6.2 depending on the signal-to-noise ratio.

In our model, we consider no ambiguities and the probability of detection is a function of the signal-to-noise ratio, the false alarm probability and the RCS.

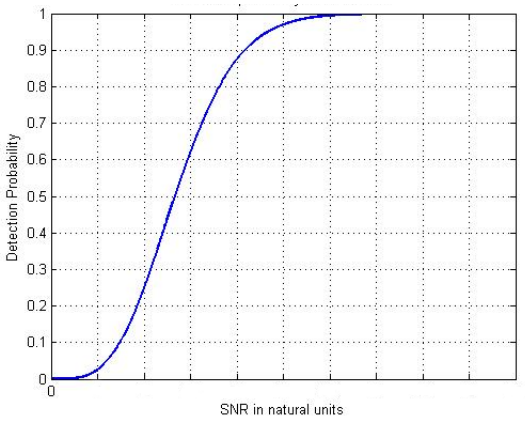


Figure 6.2: Example of a probability detection curve for a Swerling 0 target.

Chapter 7

Radar Network Modeling

Contents

We summarize in this section the modeling of radars and the use-cases that we considered to train and to test our approach. Section 7.1 presents the framework used to model radars in this thesis while Sections 7.2 and 7.3 summarize the modeling of the target and radars. Section 7.4 presents the use-cases of this thesis. Finally, Section 7.6 presents the geographical data used to place radars.

7.1 Ægis: Radar Network Modeling Framework

Ægis is a framework developed within Thales in order to easily simulate radar networks. Thales possesses more elaborated simulators but these ones are too costly in evaluation time.

The purpose of Ægis is to produce a quick and approximated version of Thales' simulators. Part of this thesis was to expand this framework to add new radar models and new functionalities. In order to have an intuition of how much this framework was enhanced during the thesis, we give the evolution of the framework in regard to the number of code lines. At the beginning, the number of code lines was around 5,000 to reach more than 17,000 at the end.

The implementation of these 12,000 new lines of code concerns mainly:

- the definition of the target for this thesis (see Section 7.2);
- the radar models of this thesis (see Section 7.3);
- new ways to aggregate radars in networks (see Section 7.4.2);
- handling of constraints (see Section 7.4.4);
- modifications in the handling of geographical data (see Section 7.6);
- some visualizations that are used in this thesis.

7.2 Target Characteristics

For the sake of simplicity, we consider a simple target model to be detected by radars. The speed of the target is supposed to be constant. The target altitude above ground level h is also supposed to be constant, *i.e.*, if the ground altitude above sea level is z, then the target is flying above the ground at altitude $z + h$.

The RCS of the target does not belong to any Swerling models but stays relatively simple. It is modeled with a cross section (Figure. 7.1) that takes into account a simple geometry of a target, *i.e.*, the target has a lower RCS when it is facing the radar and a higher RCS when it is perpendicular to the radar beam. Nevertheless, an assumption of a Swerling 0 target is made for the computation of the probability of detection, *i.e.*, we consider no noise and no fluctuations.

Moreover, we define the target angle θ as the angle between the target direction and the North. This angle is called the azimuth.

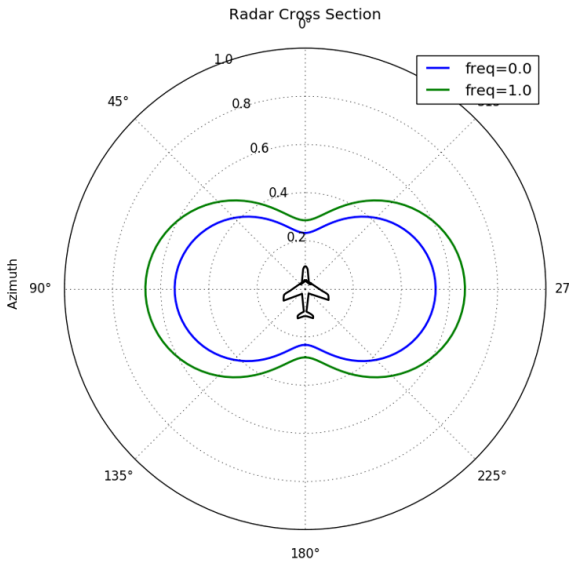


Figure 7.1: RCS model of the target.

7.3 Radar Models and Parameters

In this thesis, multiple radars are gathered into a *network*. A network is used to aggregate the probabilities of detection for each radars. Possible ways of aggregation are further discussed in Section 7.4. The network can be composed by two types of radars: rotating or staring radars. These two types only differ by the angles that the radar can sense. We make the hypothesis that radars with rotating antennas can sense all around them at any time, whereas radars with staring antennas can only sense predefined regions around their *staring direction* (see Figure 7.2).

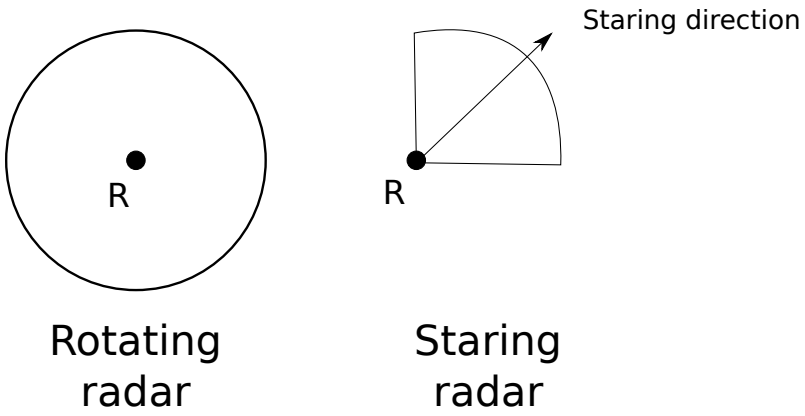


Figure 7.2: Diagram of a rotating and a staring radar.

7.3.1 Radar Tunable Parameters

The number of tunable parameters depends on the types of radars. In our model, staring radars have four parameters that can be modified:

1. the x ($\approx$ longitude) location in the domain;
2. the y ($\approx$ latitude) location in the domain;
3. the *tilt*, *i.e.*, the angle between the antenna and the horizontal plane;
4. the *staring angle*, *i.e.*, the direction the radar will look to.

For rotating radars, the staring angle is not needed and we therefore have three parameters to tune.

7.3.2 Radar Varying Parameters

In addition to the tunable parameters, radars have settings that can change depending on the target position and its *elevation angle*, *i.e.*, the angle between the antenna and the target. This elevation angle has an impact on several parameters:

- ranges can be affected and modify the distance to which radars can sense;
- a radar constant affecting the detection probability can be changed;
- the transmitted waveform of the radar can be modified;

- the radial speed coverage can be modified which implies that slower targets might not be sensed.

All these parameters have an impact on a target detection probability. This probability can be seen in an example for one radar in Figure. 7.4 on a landscape in Brasil. Figure. 7.3 represents the $2d + \theta$ visualization where the axis Oxy represent the position x, y on the landscape and axis Oz is the target direction θ, *i.e.*, it is the angle between the longitudinal axis of the target and the radar-to-target direction. The coiled shape of the cylinder is a direct effect of the target RCS as for some angles, the target exhibits more fuselage and thus a bigger RCS. If we fix θ, we obtain a slice of the cylinder that we can project onto the digital elevation model.

The holes in the probability of detection are caused by a relief in front of the radar, or by the radar altitude as it cannot sense lower than its altitude.

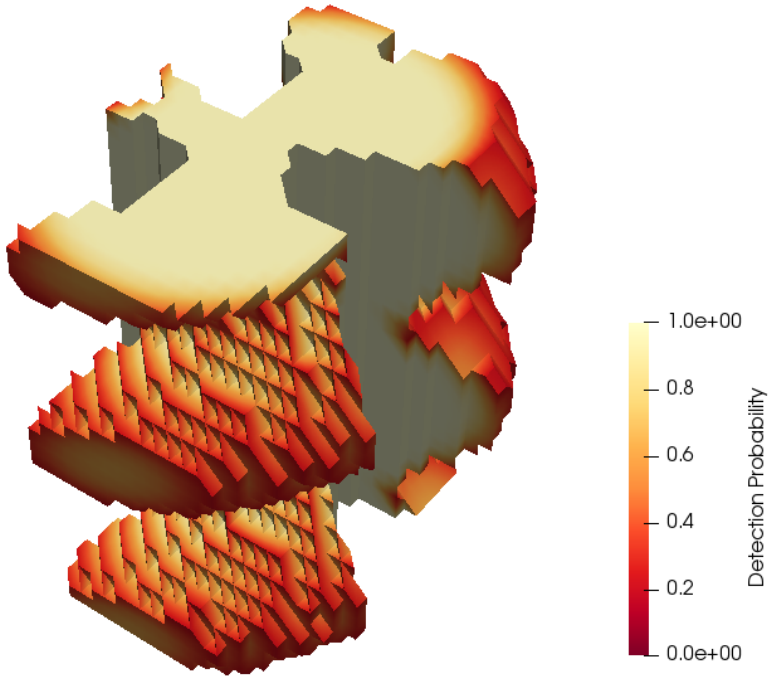


Figure 7.3: $2d + \theta$ visualization, θ (vertical axis) is the azimuth.

7.4 Radar Network Use-Cases

7.4.1 Domain Definition

The terrain to cover is a square of $\delta \times \delta$ kilometers and for each point, the target can face in any direction. Therefore, the domain Δ to cover is represented by (x, y, θ), where θ is the target azimuth. Each axis of Δ is then split into D equal parts resulting in D^3 voxels.

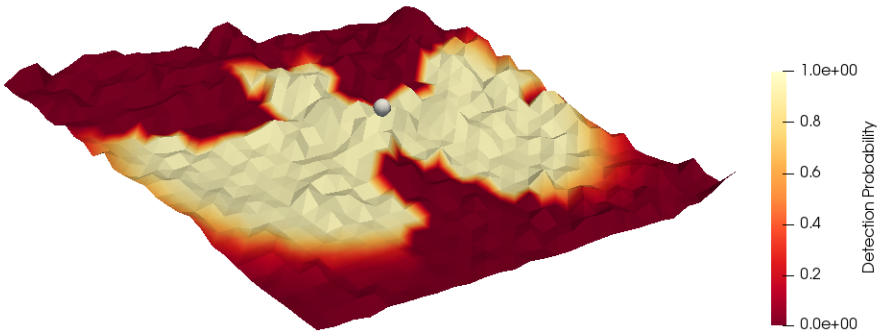


Figure 7.4: Probability of detection for a fixed target azimuth θ.

7.4.2 Radar Network

To cover this domain Δ, k radars R are available: r rotating and s staring, hence the dimension d of the problem, based on Section 7.3, is $d = 4r + 3s$. These radars compose a network ξ in the set of possible networks Ξ. A network aggregates radars' probability of detection in one of the following ways:

Additive: the probability of detection of the network on a location is the addition of the probability of detection for each radar on this point, capped at 1:

$$P_d^{\xi}(x, y, \theta) = \min\left(1, \sum_{i=1}^{k} P_d^{R_i}(x, y, \theta)\right);$$

Multiplicative: the probability of detection of the network on a location is the multiplication of the probability of detection for each radar on this point, capped at 1:

$$P_d^{\xi}(x, y, \theta) = \min\left(1, \prod_{i=1}^{k} P_d^{R_i}(x, y, \theta)\right);$$

Minimum: the probability of detection of the network on a location is the minimum probability of detection of a radar on this point:

$$P_d^{\xi}(x, y, \theta) = \min_{i=1,\dots,k}\left(P_d^{R_i}(x, y, \theta)\right);$$

Maximum: the probability of detection of the network on a location is the maximum probability of detection of a radar on this point:

$$P_d^{\xi}(x, y, \theta) = \max_{i=1,\dots,k}\left(P_d^{R_i}(x, y, \theta)\right);$$

Binary: the probability of detection of the network is 1 if one of the radar probability of detection if above a threshold, 0 otherwise:

$$P_d^{\xi}(x, y, \theta) = \begin{cases} 1 & \text{if } \min_{i=1,\dots,k}\left(P_d^{R_i}(x, y, \theta)\right) \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases};$$

Union: the probability of detection of the network is given by multiplying the probabilities of non-detection for each radar and returning the corresponding probability of detection. We assume that each radars are independent and compute

$$P_d^{\xi}(x, y, \theta) = 1 - \prod_{i=1}^{k} \left(1 - P_d^{R_i}(x, y, \theta)\right).$$

Union networks can be used when probabilities are involved whereas *additive* networks may be used in use-cases considering coverage.

7.4.3 Objective Function: Coverage Scenario

The objective function aims at maximizing the coverage of Δ. For a given network ξ, we want to find the location and configuration of the radars such that the number of covered voxels is as large as possible. We consider a voxel covered if the probability of detection p is greater or equal to a predefined fixed threshold. The objective function for the coverage scenario is defined as follow: $f : \Xi \to [0, D^3]$ (see Section 7.4.2). We want to find ξ^* such

$$\xi^* = \arg\max_{\xi \in \Xi} f(\xi).$$

7.4.4 Constraints: Definition and Handling

On top of this problem, two sets of constraints can be added simultaneously or separately: areas to defend and exclusion areas.

An *area to defend* $\Gamma \subset \Delta$ is a region where at least one radar needs to be located. It is a polygon that can be anywhere in the domain and of any shape (Figure 7.5). Moreover, multiple areas to defend can be added as long as their number is lower or equal to the number of radars available.

An *exclusion area* $\Lambda \subset \Delta$ is a region where no radar should be located. Similarly to areas to defend, they are polygons of any shapes but their number can be greater than the number of radars (Figure 7.5). Both areas to defend and exclusion areas can be combined in the domain.

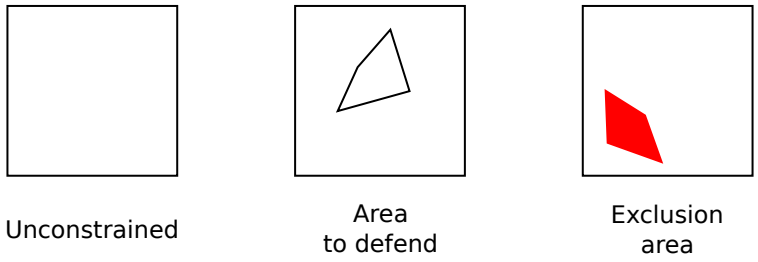


Figure 7.5: Diagrams of the problem and possible constraints.

In Ægis, constraints are handled following this procedure:

1. for each radar, we define its distance to the border of the constraint. This distance is 0 if the radar does not violate the constraint;
2. all radar constraints are aggregated in one constraint either by summing, multiplying or taking the minimum of each constraints;
3. the aggregated value is added to the objective function as penalty.

7.5 Thesis Use-Cases

In this thesis, we focus on two particular use-cases: an unconstrained and a constrained one, both defined in the following sections.

7.5.1 Unconstrained Use-Case

We consider a terrain of 50×50km. The discretization of the domain is fixed to 30 pixels. On each pixel, we consider a target facing successively in all directions. The angle of the target is also discretized using 30 pixels. The domain Δ to cover is hence composed of $30^3 = 27{,}000$ voxels.

In order to cover this domain, four radars are available: one rotating radar and three staring radars. The dimension of the problem is thus $d = 3 + 3 \times 4 = 15$. In this use-case, radars are gathered into a network using the *union* aggregation defined in Section 7.4.2.

This use-case remains unconstrained, *i.e.*, there is no area to defend nor exclusion areas, so radars can be placed anywhere in the domain. The target has the characteristics described in Section 7.2 and is flying at a constant altitude above the ground h. The goal of the problem is to find a configuration that maximizes the number of voxels covered of the network. Since most off-the-shelf optimization algorithms are implemented for minimization problems, we seek to minimize the number of voxels that are not covered.

7.5.2 Constrained Use-Case

The constrained use-case has the same characteristics as the unconstrained one presented above. In addition of these characteristics, the problem has an area to defend where at least one radar of the network should be located.

The area to defend is a square located in the center of the domain Δ of dimension $2 \times 2km$. The aggregation used for constraints is the minimum, *i.e.*, the minimum distance of the radars to the border is added to the value of the objective function.

7.6 Geographical Data

In order to place and configure radar networks, we need to have geographical data to actually locate radars. In this thesis, we use data from the NASA Shuttle Radar Topography Mission [HKP+01] (SRTM). SRTM data were sampled at 3 arc-seconds, which is 1/1200th of a degree of latitude and longitude, or about 90 meters at the equator. These Digital Elevation Models (DEM) cover the entire globe and are provided in mosaics of 5 degrees by 5 degrees tiles. The mission provided digital elevation models for 80% of the globe with a vertical error reported to be less than 16 meters.

The original data from NASA contains data voids due to water (lake and rivers), areas with snow or heavy shadow which prevented the quantification of elevation. For instance, the Himalayas region is the one containing the most data voids. In order to use complete data without holes, we used digital elevation models that were post-processed with interpolation in [JRNG08]. The resulted files have the same sizes and properties as the original data but without holes. As a result of post-processing, peaks in high-mountain areas are interpolated which might result into peaks slightly lower than they are in reality.

Figure 7.6 and Figure 7.7 show an example of data recovered from the CGIAR website[1] that is for a region close to Calgary, in Canada.

[1] Data available at https://srtm.csi.cgiar.org.

The objective function defined in Section 7.4.3 will be applied on a collection of DEMs to create different instances of the radar network configuration problems.

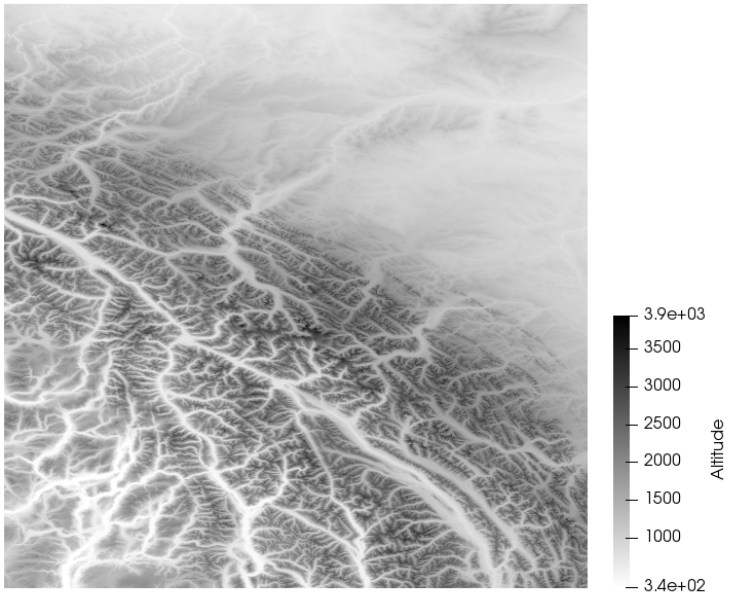


Figure 7.6: 2D Digital Elevation Model tile in Canada.

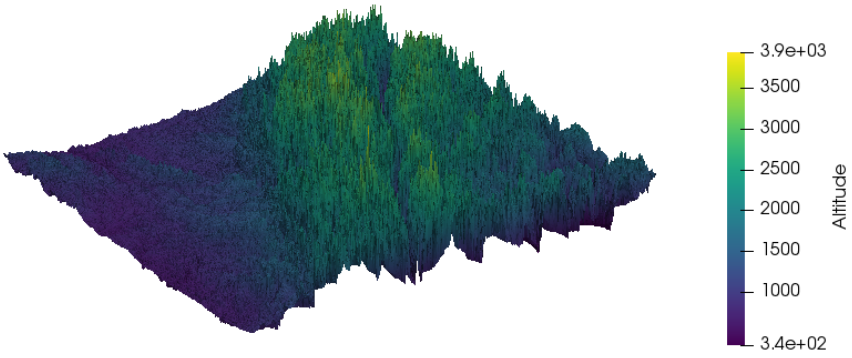


Figure 7.7: 3D Digital Elevation Model tile in Canada.

Chapter 8

Solving the Radar Network Configuration Problem

Contents

In this chapter, we present the optimization results on the radar network configuration problem. We first describe in Section 8.1 how the different instances of use-cases (see Section 7.5) are produced. Section 8.2 presents the different optimization algorithms used to solve the use-cases. Sections 8.4 and 8.5 show the performances of the algorithms on both use-cases. Finally, in Section 8.6 we compare the results of algorithms to human-designed solutions.

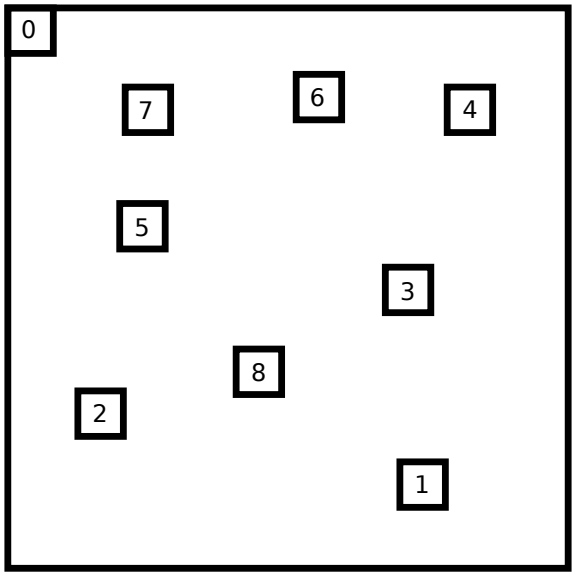


Figure 8.1: Uniform subsampling of DEM tiles. The large area correspond to the DEM tile while smaller numbered squares represent the samples.

8.1 Problem Instances

The network of radars computed by our algorithms will be evaluated on several instances. Each instance is defined by an individual terrain (see Section 7.6). Overall, we consider 17 different tiles of digital elevation model (DEM) from all around the world in order to have a large variety of terrains. The tiles represent large areas of geographical data.

The size of the use-cases presented in Section 7.5 are smaller than the area of the DEM tiles. To match the dimensions of the use-cases, we subsample the original tiles with domains of 50×50 kilometers. As there is no particular reason to favor one area over another, we perform the downsampling via the mask presented in Figure 8.1. Eight subsquares were chosen uniformly at random and added to the condition that they are non-overlapping. These eight sets represent together with the top left square the subsamples chosen for each terrain. The result is a total number of $9 \times 17 = 153$ instances.

The instances are labeled by the region name of the DEM tile and by the number of its sample in the mask, *i.e.*, the first sample in the mask for the Brasil region has the instance name *brasil0*.

The difference between the highest and the lowest points in each instance can be found in Table 8.1. This information gives a hint of the actual landscape of the terrain. It may also give an an idea of which problems might be easier to solve than others as flat areas may be easier to cover than mountainous areas.

We define *flat instances* to be instances where the difference between the highest and the lowest point is below 100 meters. As the target to detect is always 50 meters above ground level (see Section 7.5, it implies that it might be visible from almost any point in the domain. *Mountainous instances* have a difference between the highest and the lowest points greater than 1,000 meters.

Table 8.1: Difference between the highest and lowest altitudes (in meters) of each instance. Columns represent the subsample in the DEM tile and rows corresponds the region of the DEM tile.

	Instances								
	0	1	2	3	4	5	6	7	8
afghanistan	1616	1266	1152	1161	1534	1177	1085	1144	1023
argentina	24	24	24	24	27	22	26	26	27
australia	189	181	117	67	376	93	203	384	90
belarus	166	196	145	158	135	187	154	166	147
brasil	61	79	85	108	89	90	129	58	107
canada	1494	1571	1419	1452	1528	1194	1474	1515	1387
chile	1408	1311	1555	2625	1252	2469	1869	1330	2523
china	1417	1439	1462	893	1398	1039	924	1271	956
congo	66	70	68	84	64	74	88	64	87
france	161	329	300	325	331	177	189	186	307
india	3828	2111	3821	2224	1983	3195	3175	2070	3755
iran	914	487	841	929	788	788	986	319	979
moldavia	111	245	241	235	257	166	170	187	229
nepal	1439	1753	1232	1656	1723	1367	1529	1644	1264
russia	341	302	341	343	302	327	355	330	322
sahara	25	25	24	43	20	28	25	27	29
usa	1161	1486	1518	1300	1321	1049	1173	1393	1228

8.2 Algorithm Portfolio

In this section, we present the algorithms used to solve the use cases (see Section 8.1. Section 8.2.1 present the portfolio of default algorithms used while Section 8.2.2 present the automatic configuration of some algorithms.

8.2.1 Default Algorithms

The portfolio of algorithms is composed of 12 of the algorithms presented in Chapter 2:

- Differential Evolution [VGO+20][1] with default *scipy* parameters;
- Nelder-Mead [VGO+20] with default *scipy* parameters;
- Powell's method [VGO+20] with default *scipy* parameters;
- L-BFGS-B [VGO+20] with default *scipy* parameters and a finite differences step of 0.01;
- PSO with global best topology [Mir18][2] and default *pyswarms* parameters;
- Random Search with uniform sampling;
- Quasi-Random Search with Sobol´ low-discrepancy sequences;
- Five CMA-ES variants from the ModCMA framework [vRWvLB16]:

[1] version 1.5.4
[2] version 1.3.0

- Vanilla CMA-ES [HO01] (denoted 00000000000 in ModCMA),
- CMA-ES with mirror sampling [BAH+10] and pairwise selection [ABH11] (denoted 00100001000 in ModCMA),
- CMA-ES with elitism (denoted 01000000000 in ModCMA),
- CMA-ES with active update [JA06] (denoted 10000000000 in ModCMA),
- CMA-ES with active update, elitism and BIPOP increasing population [Han09] (denoted 11000000002 in ModCMA).

8.2.2 Specifically Configured Algorithms

We expand our portfolio of algorithms by performing algorithm configuration (see Section 2.8) on some instances in order to create specific solvers. This expansion is done to increase the complementarity in the algorithms from our portfolio. All configured algorithms that may perform better than their default configuration will be added in our portfolio.

We use the *irace* [LDLP+16][3] configurator with default parameters and the lowest budget possible of runs: 180. This choice of number of runs is motivated by the expensive computation time of the objective function (see Chapter 7).

The configuration is done on four sets of instances for the large budget of 2,500 function evaluations:

- one flat instance, *belarus0*;
- three flat instances, *belarus0, congo0 and sahara0*;
- one mountainous instance,*chile0*;
- three mountainous instances, *chile0, canada0 and india0.*

The algorithms and parameters used for the configuration were:

- Differential Evolution: population size (between 4 and 100), the mutation (between 0 and 2) and recombination constants (between 0 and 1);
- PSO: population size (between 1 and 100), acceleration coefficients c_1, c_2 (both between 0 and 4) and the inertia weight w (between 0 and 1);
- vanilla CMA-ES: population sizes λ (between 4 and 100) and μ (a proportion of λ between 0 and 1).

Most of the configurations given by irace performed similarly to the default version of algorithms. But, one stands out by outperforming its default configuration: Differential Evolution tuned on the *chile0* instance. Parameters of both default and tuned DE algorithm can be found in Table 8.2.

The main differences between the two set of parameters resides in a population size reduced more than twice in the tuned version and lower mutation constant. In the default algorithm, for each generation the mutation constant is taken uniformly at random between the bounds. These bounds are much larger than the mutation constant of the tuned version.

As the tuned Differential Evolution seems to have good performance, it was added to our portfolio of algorithms so that the **total number of algorithms in our portfolio is 13**.

[3] version 3.4.1

Table 8.2: Differential Evolution population size, mutation and recombination parameters. Comparison of default algorithm parameters with tuned algorithm parameters.

	DE	DE_2500_chile
popsize	15	6
mutation	$\mathcal{U}(0.5,1)$	0.1
recombination	0.7	0.58

8.3 Experimental Setup

The experimental setup presented in this section is used to solve the two use-cases.

For each algorithm, 30 runs are performed on each of the problem instances. We log the whole performance trajectory, i.e, we log all the improvements achieved by the algorithms up to a budget of 2,500 function evaluations. But, for the presented analyses in this thesis, we will focus on one **low-budget setting with 500 evaluations** and one **large-budget setting with 2,500 evaluations**.

We look at the 30 individual runs separately in Section 8.4.1. In Section 8.4.3, we assess and compare the performances of the algorithms using the median of the best objective function values found in each run runs only.

8.4 Results for the Unconstrained Use-Case

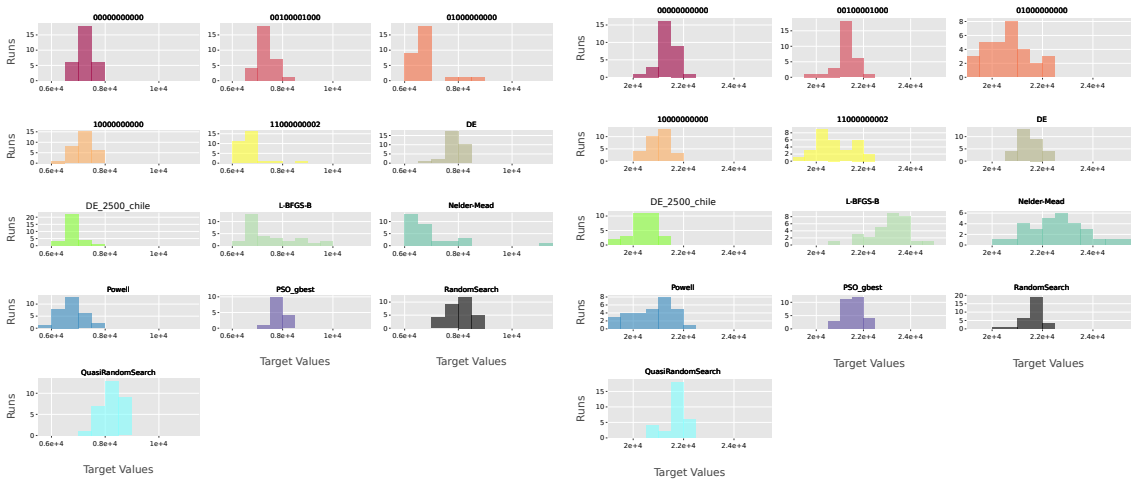
In this section, we will focus on the performances of the algorithms from our portfolio applied to the unconstrained use-case presented in Section 7.5.1.

8.4.1 Individual Runs

Figures 8.2 and 8.3 show the histograms of the objective values of the 30 runs for all algorithms. Figure 8.2 shows the results for 500 function evaluations while Figure 8.3 are for 2,500 function evaluations The bins used in the histograms and the scaling of the x axis are the same for each algorithm.

We observe that on the low budget, the algorithm reaching most often the best quality of solution found on both instances is Powell's method. On the *sahara0* instance (see Figure 8.2a), Powell's method is the only algorithm reaching this quality of solution. Nevertheless, given the dispersion of fitness values on the runs, Powell's method is not the best performing algorithm in median on either instances. On the *sahara0* instance (see Figure 8.2a), it is the fourth best performing algorithm in median after 11000000002 (best performing in median), Nelder-Mead (second best performing in median), and 01000000000 (third best performing in median). For the *nepal3* instance, it is also the fourth best performing algorithm in median after DE_2500_chile (best performing in median), 01000000000 (second best performing in median), and 11000000002 (third best performing in median).

For the larger budget, on the *sahara0* instance (see Figure 8.3a), 00100001000 is reaching the best objective value found among all algorithms once on both instances. 00000000000 is reaching the best performance twice on the *nepal3* but never on the *sahara0* instance. The best performing algorithm in median is also the 00100001000 CMA-ES variant. Even if 00000000000 reaches the best objective value twice on *nepal3* instance, given its dispersion, it is only the fourth best performing algorithm in median behind 00100001000, DE_2500_chile and 10000000000.



(a) *sahara0* (b) *nepal3*

Figure 8.2: Histograms of the values reached by the 30 runs, for *sahara0* and *nepal3* instances, for the 13 algorithms of our portfolio for 500 function evaluations on the unconstrained use-case.

Looking at the dispersion of objective values, we observe that overall and especially on these two instances, the Nelder-Mead algorithm is the one with the greatest variance. This behavior may be due to the mechanisms of the algorithm. The Nelder-Mead acts as local search algorithm and very much depends on the starting point. Moreover, as mountainous instances are more peaked than flat instances, they may contain more local optima. The dispersion of reached objective values by the algorithm tends to often be larger on mountainous instances. In contrast, the algorithms that have, overall, the smallest dispersion are both DE_2500_chile and DE.

Tables of algorithms performing best on the instances in median and for the 2% quantile as well as the best performance reached in median and for the 2% quantile can be found in appendix B. With 30 runs, the 2% quantile of the distribution represent the best run achieved.

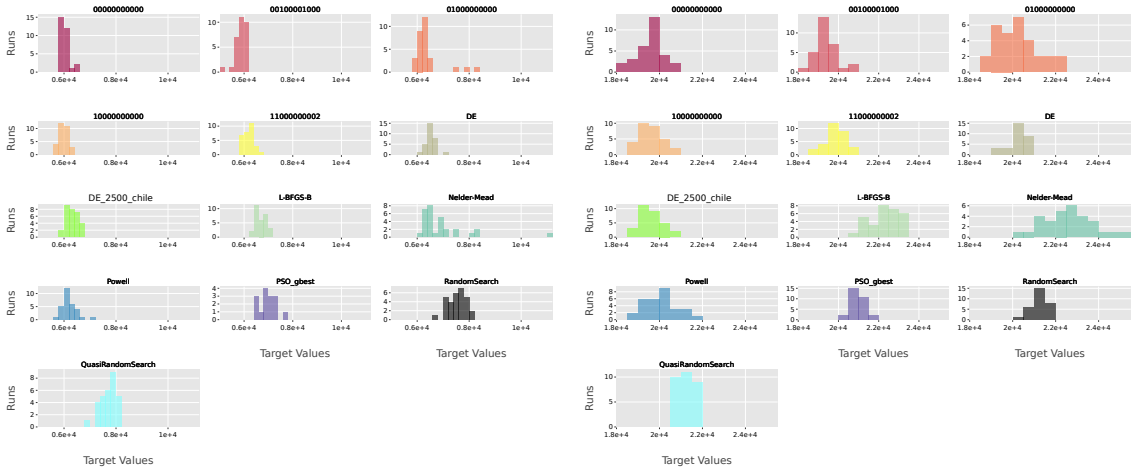
8.4.2 Statistical Significance

All algorithms seem to have similar performances. To confirm this behavior, we perform a Kolmogorov-Smirnov test to determine if one algorithm outperforms all others on each instance. We perform this test for each pair of algorithms with a confidence level $\alpha = 0.01$.

The results are presented as a network in Figure 8.4 and Figure 8.5 for *sahara0* and *nepal3* respectively. In these plots, if an arrow goes from an algorithm A to an algorithm B, it indicates that A is significantly better than B.

In the *nepal3* instance, for a budget of 500 function evaluations, the best performing algorithm, *i.e.*, *DE_2500_chile* is statistically better than the majority of algorithms except for *Powell*, and the *11000000002* and *01000000000* CMA-ES variants (Figure 8.5a).

For a larger budget of 2,500 function evaluation, *Powell*, *DE_2500_chile* and all CMA-ES variants perform better than other algorithms but are performing statistically the same between each other (Figure 8.5b).



(a) *sahara0* (b) *nepal3*

Figure 8.3: Histograms of the objective values reached by the 30 runs, for *sahara0* and *nepal3* instances, for the 13 algorithms of our portfolio for 2,500 function evaluations on the unconstrained use-case.

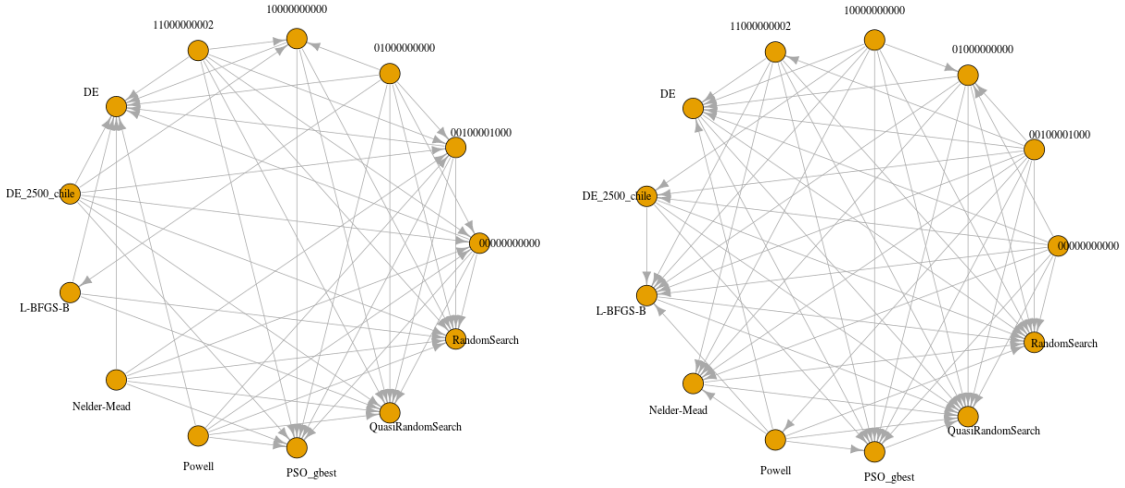
This tendency of multiple algorithms having similar performances can also be seen for *sahara0* instance (Figure 8.4a and Figure 8.4b) and all other instances in the benchmark.

8.4.3 Algorithm Median Objective Value

Figure 8.6 presents convergence plots of the algorithms from our portfolio. Figure 8.6a shows the results for a flat instance *sahara0*, while Figure 8.6b presents results for a mountainous instance *nepal3*.

These plots represent the median objective value of the 30 runs for each algorithm. On both instances, the CMA-ES variant *00100001000* seems to perform best after 2,500 function evaluations. Nevertheless, the difference among the algorithms from the portfolio on the *sahara0* instance is small, *i.e.*, around a 300 voxels difference between the best performing algorithm and the fifth best.

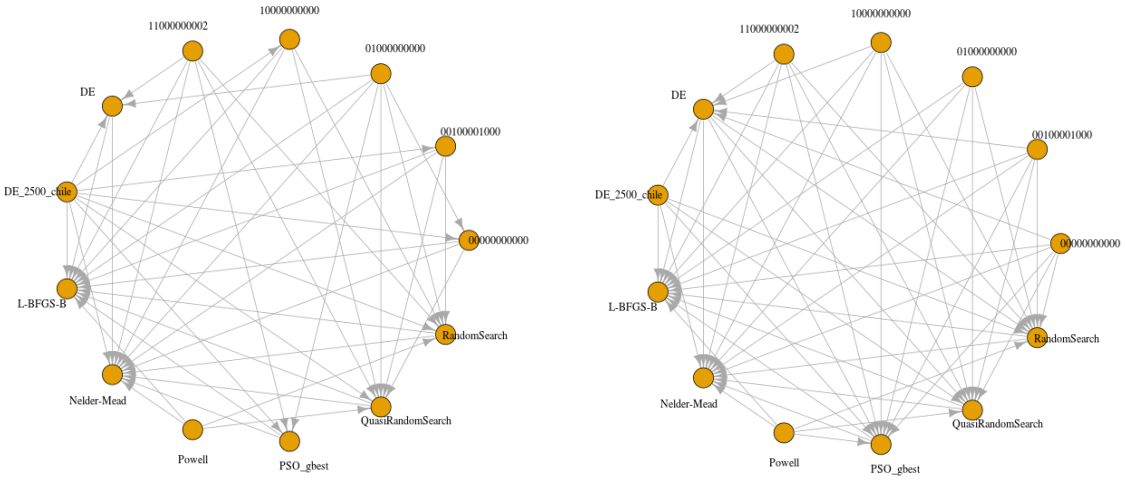
All algorithms perform better than random search for the flat *sahara0* instance. This is not the case on the *nepal3* instance. On this instance, both the Nelder-Mead and the L-BFGS-B perform far worse than random search. This may be explained by the characteristics of the algorithms and the instances. The Nelder-Mead and the L-BFGS-B perform a local search around the current solution. As flat instances reveal some convexity towards the center of domain, *i.e.*, the area where radars can see the most, local search often performs well on this type of instance. On the contrary, mountainous instances may be are composed of lots of peaks caused my mountains. This implies that local search algorithms may get stuck in these peaks.



(a) 500 function evaluations

(b) 2,500 function evaluations

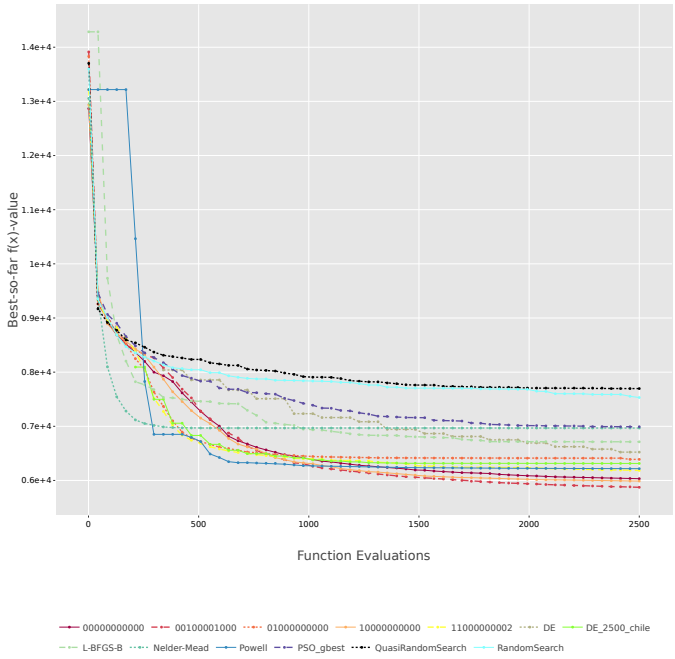
Figure 8.4: Partial order resulted from the Kolmogorov-Smirnov statistical test visualized as a network on *sahara0* instances. The tests are for low and large budget with a significance level $\alpha = 0.01$. An arrow from algorithm A to B indicates A is significantly better than B.



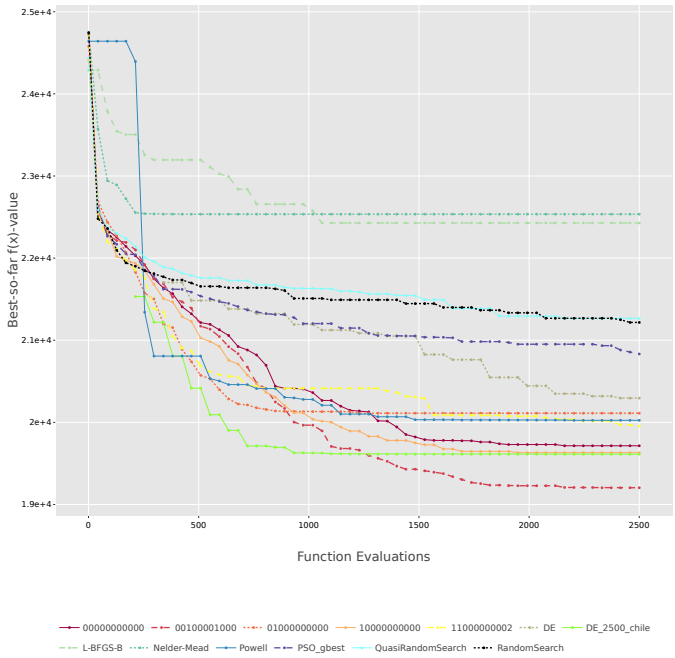
(a) 500 function evaluation

(b) 2,500 function evaluation

Figure 8.5: Partial order resulted from the Kolmogorov-Smirnov Statistical test visualized as a network on *nepal3* instances. The tests are for low and large budget with a significance level $\alpha = 0.01$. An arrow from algorithm A to B indicates A is significantly better than B.

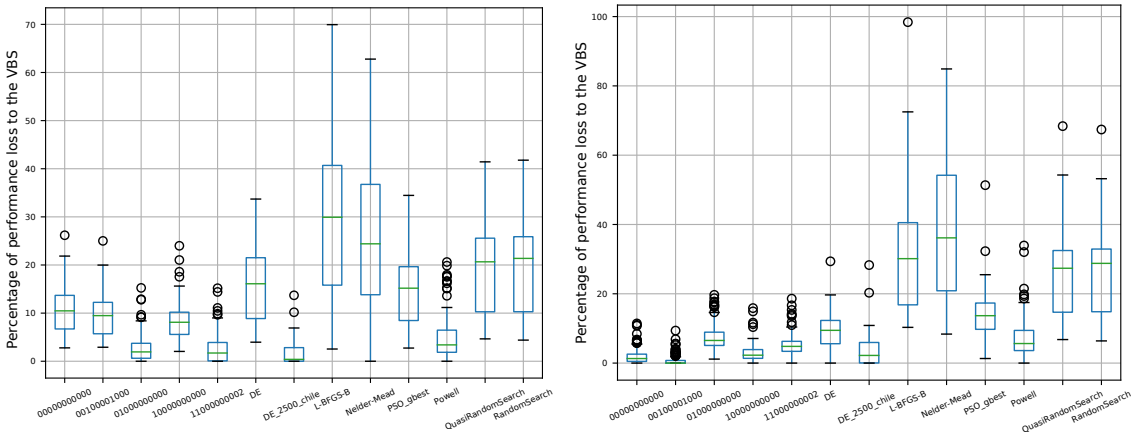


(a) *sahara0*



(b) *nepal3*

Figure 8.6: Median objective values over time, for 30 runs, for *sahara0* and *nepal3* instances, for the 13 algorithms of our portfolio.



(a) 500 function evaluations

(b) 2,500 function evaluations

Figure 8.7: Boxplots of median performance loss on the best performing solver on all problem instances for low and large budget of function evaluations.

8.4.3.1 Single Best Solvers (SBS)

We look at our two budgets of interest, *i.e.*, 500 and 2,500 function evaluations to extract the best performing solver on all instances. The best performing solver for the low budget is denoted SBS_{500} whereas the one for the large budget is denoted $SBS_{2,500}$

To extract the two SBS, we record for each instance the best performing algorithm and its median objective performance for both budgets. We compare all other algorithm median objective performances to the best one and record the difference in percentage.

Figure 8.7 aggregates this data and plots the difference in percentage to the best algorithm for each problem instances. In the boxplots, a median value of an algorithm illustrates that this algorithm performs $X\%$ worse than the best algorithm for half of the instances.

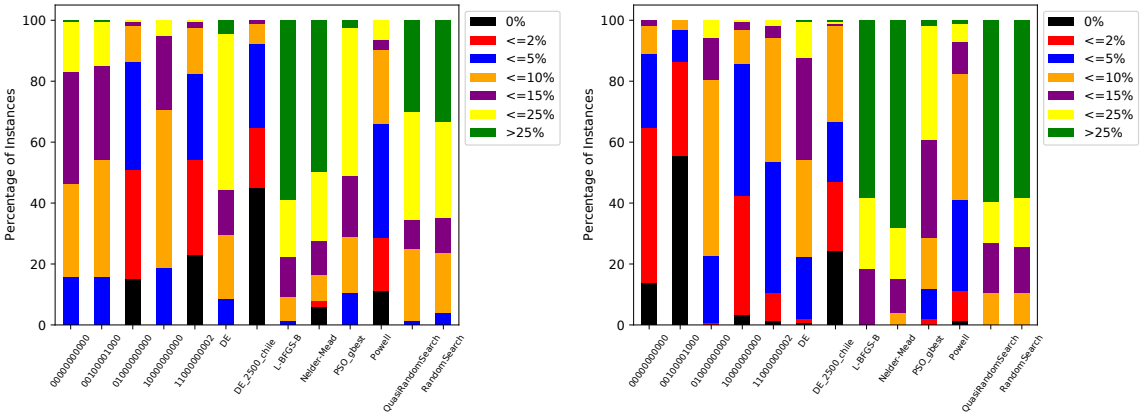
Figure 8.8 presents the number of instances that are solved within $X\%$ of the best algorithm.

Figure 8.7a presents the results for the low budget. The best performing algorithm, and thus the SBS_{500} is *DE_2500_chile* with a median difference with the best algorithms for each instance of 0.39%. On top of that, it is also the best algorithm for 69 out of the 153 problems and never has performances worse by more than 15% than the instances best solver.

For the low budget setting, four algorithms outperform the others significantly in median performances and are also the only to reach the best performance found on some instances. Alongside *DE_2500_chile* , *11000000002* CMA-ES variants is best on 35 instances, *01000000000* on 23 and *Powell* on 17.

One of the worst performing algorithms in median is the *Nelder-Mead*, performing 24.4% worse than the best solver. However, even if it performs worse than random search in median, the Nelder-Mead algorithm exhibits one of the largest complementarity on the instances. It performs best on 9 instances but also performs worse than 25% of the best algorithm on 76 instances.

When the budget is larger (see Figure 8.7b), algorithms from our portfolio start to have similar performances. Six algorithms have a median difference to the best algorithm around 5% and seven of them are the best algorithm on at least one instance. The $SBS_{2,500}$ is the CMA-ES variant *00100001000* with a median performance loss of 0% and performing best on 86 of the 153 instances.



(a) 500 function evaluations

(b) 2,500 function evaluations

Figure 8.8: Percentages of instances solved within $X\%$ of the best algorithm performance.

8.4.3.2 Low Budget Performances versus Large Budget Performances

Figure 8.9 presents the evolution of the median performance on all instances for each algorithm in our portfolio. Most algorithms improve their performance by around 10% with the 2,000 supplementary evaluations. That is the case for DE_2500_chile , L-BFGS-B, PSO, Powell and quasi random search. DE achieves one of the biggest improvements with 19.5% while the Nelder-Mead algorithm does not improve at all.

Concerning CMA-ES variants, algorithms that were performing well on the low budget only improve by 6%. Other variants really benefit from these 2,000 additional evaluations by improving their performances up to 20%.

8.4.4 Discussion

In this section, we study in Section 8.4.4.1 the impact of the budget on the performances of the elitist module in CMA-ES variants. In Section 8.4.4.2, we observe the impact of the physical landscape on algorithm performances. Finally, in Section 8.4.4.3, we study the benefit of automated algorithm configuration over default parameters of algorithms.

8.4.4.1 Budget and Elitism

The results presented in Section 8.4.3.2 seem to show that some CMA-ES variants are preferable for small budgets whereas others perform well on large budgets.

The main differences between these variants are their respective selection mechanism. Variants that perform well for the low budget are elitist (see Section 2.3.2), *i.e.*, they use plus-selection, whereas variants that perform well for the larger budget are non-elitist, *i.e.*, they use comma-selection.

Figure 8.10 illustrates this behavior on the *sahara0* (Figure 8.10a) and the *nepal3* instance (Figure 8.10b) of two CMA-ES variants. One variant is the non-elitist vanilla CMA-ES (00000000000) and the other variant uses the elitism module (0100000000). On both instances, we observe that the elitist variant performs better on lower budgets. In contrast, the non-elitist variant is performing better for larger budgets. On all instances, there is a number of function evaluations where the non-elitist performance curve crosses the elitist one.

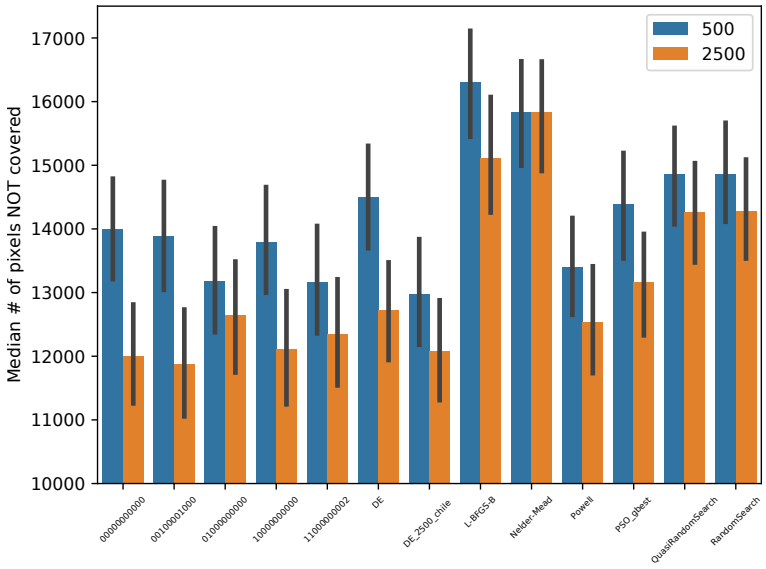


Figure 8.9: Median number of voxels not covered on all instances for low and large budget. Black bars represent standard deviations over instances.

The crossing point occurs around the 900 evaluations on the *sahara0* and around 1,300 evaluations on *nepal3*. Overall, this crossing occurs on all instances between 500 and 1,650 evaluations. Nevertheless, the crossing seems not to depend of the landscape of the terrain, *i.e.*, the variants performances may cross each other later on some flat instances than mountainous instances and vice versa. As an example, the crossing occurs around 1,300 evaluations on the *brasil1* instance which is rather flat and around 750 evaluations on the *chile6* instance.

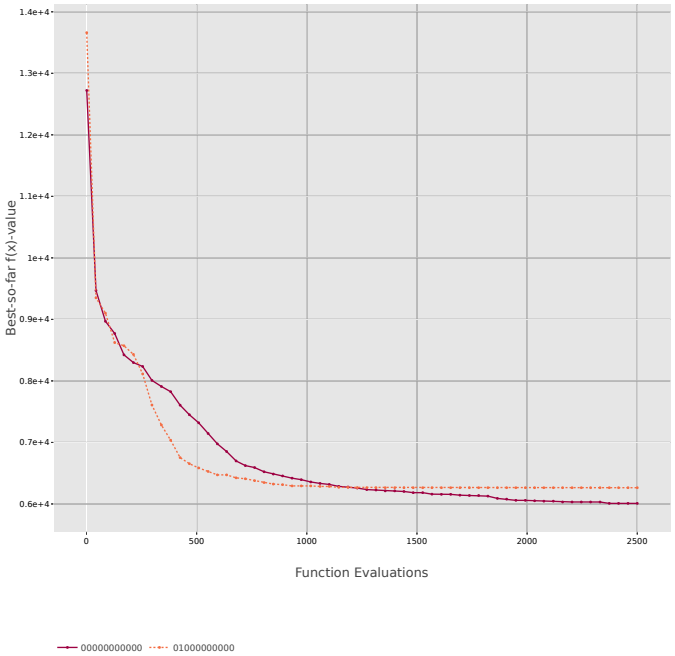
8.4.4.2 Flat Instances versus Mountainous Instances

An easy division of the instances of our benchmark is to split them into flat instances and into mountainous instances. Of course, some instances are neither entirely flat, nor particularly mountainous but have just some minor elevations. We focus in this section only on the performance of solvers on the two extreme types of instances: flat and mountains.

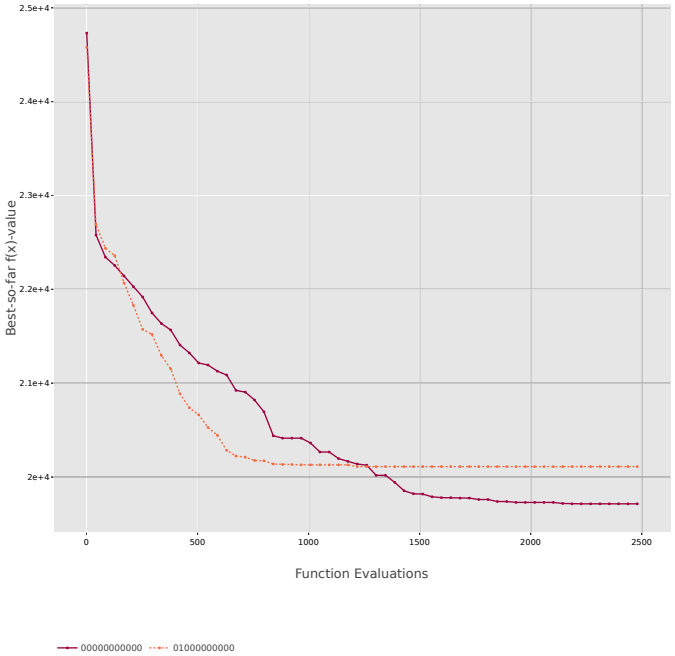
Figure 8.11 shows the median performances for each solver on each instance for the lower budget (Figure 8.11a) and the larger budget (Figure 8.11b). For both budgets, we can clearly see that the number of voxels not covered is lower on flat instances than mountain instances. This could be expected before any optimization as mountains are more difficult to cover as they block the radar view.

The convergence between flat and mountainous instances is also different. We compare the solutions obtained after 500 and 2,500 evaluations for 27 flat instances (Argentina, Congo and Sahara instances) and 27 mountain ares (Nepal, India and USA instances).

For both flat and mountainous instances, the Nelder-Mead algorithm has already converged after 500 evaluations and does not improve with more evaluations. Overall, for the other algorithms from our portfolio, we observe a bigger gain on flat instances. The median gain on flat instances from letting the algorithms go to 2,500 function evaluations is 9.1%. This gain is only 4.3% on mountainous instances. Again, this may be explained by the rugged landscape of the terrain and

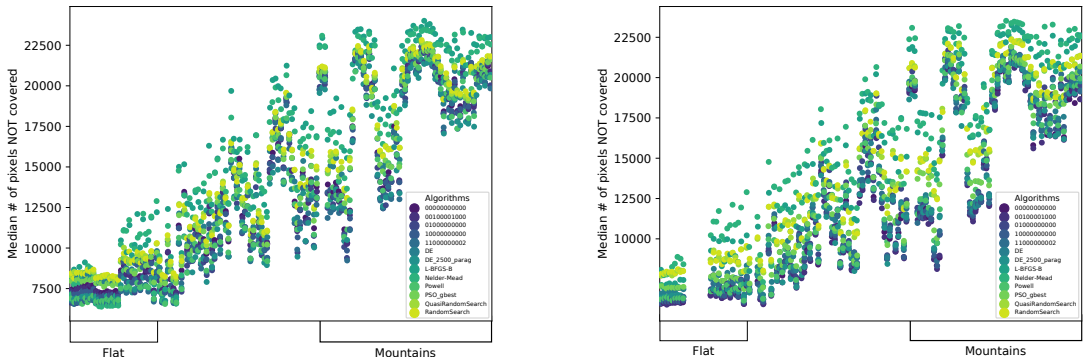


(a) *sahara0*



(b) *nepal3*

Figure 8.10: Median objective values over time, for 30 runs of vanilla (red) and elitist (orange) CMA-ES variants with different selection mechanism.



(a) 500 function evaluations

(b) 2,500 function evaluations

Figure 8.11: Median performances of solvers on each problem instance for low and large budget of function evaluations. Instances are sorted by the elevations in Table 8.1. Flat instances on the left, mountainous instances on the right.

thus, it is more difficult to improve the current solution.

8.4.4.3 Contribution of Algorithm Configuration

Algorithm Configuration has a huge impact on the performances of the performances of the DE algorithm. The tuned version of DE is statistically performing better than the default algorithm in a great number of instances, especially on low budgets. As a matter of fact, DE_2500_chile is even the SBS for the low budget setting.

Even if the default DE gains a lot in performance with a larger budget, DE_2500_chile is still performing best on more instances and is statistically better

8.5 Results for the Constrained Use-Case

The number of instances for the constrained use-case was reduced in order to save computation time. We used only the instances 0 and 3 of every region. Moreover, there is also two fewer algorithms in the portfolio:

- we do not consider the quasi random search as it has the same performances as random search for all instances on the unconstrained use-case;
- DE_2500_chile was not considered as it was specially tuned for the unconstrained use-case.

Hence, our portfolio is composed of **11 algorithms**.

8.5.1 Individual Runs

Analogous to the unconstrained use-case, Figures 8.12 and 8.13 show the histograms of the reached objective values on the 30 runs for the constrained use-case. These histograms concern 11 algorithms of our portfolio for a low budget of 500 function evaluations and a large budget of 2,500 function evaluations. The bins used in the histograms and the x axis are the same for each algorithm.

On the lower budget, as for the unconstrained use-case, Powell's method is reaching more often the best objective found on both instances. Powell's method is reaching the best objective value found on one run on the *sahara0* instance and on eight runs on the *nepal3* instance. On top of reaching the best objective value found, Powell's method is also the best performing algorithm in median on both instances before the 11000000002 CMA-ES variant on both instances.

For larger budgets, Powell's method is still reaching the best objective found value five times on the *sahara0* instance and once on the *nepal3* instance. On the *nepal3* instance, Powell's method is the only algorithm reaching the best objective value reached On the *sahara0* instance, the 11000000002 CMA-ES variant along with Powell's method is reaching the best objective value reached once.

Powell's method is also the best performing algorithm in median on the *nepal3* before the PSO algorithm. On the *sahara0* instance, given its dispersion Powell's method is not the best performing algorithm in median. The 00100001000 CMA-ES variant is not achieving the best objective reached but obtain a good performance with a low variance over the runs. Thus the 00100001000 CMA-ES variant is the best performing algorithm on the *sahara0* instance before vanilla CMA-ES and Powell's method.

Overall, as for the unconstrained use-case, the Nelder-Mead and BFGS algorithms have the wider range of objective values attained on 30 runs.

The addition of the constraint tends to worsen the objective value obtained for all instances. The median value obtained by 00100001000 on the constrained use-case is 14% worse than the median value obtained on the unconstrained problem on the *sahara0* instance. Powell's method has a median value 2.5% worse on the *nepal3* instance when the constraint is added.

In the rest of the study of this use-case, we choose to aggregate runs using the median objective value of the 30 runs.

Tables of algorithms performing best on the instances in median and for the 2% quantile as well as the best performance reached in median and for the 2% quantile can be found in appendix B.

8.5.2 Algorithm Median Objective Value

As in Section 8.4.3, Figure 8.14 presents the convergence plot of the portfolio on the *sahara0* instance (Figure 8.14a) and the *nepal3* instance (Figure 8.14b).

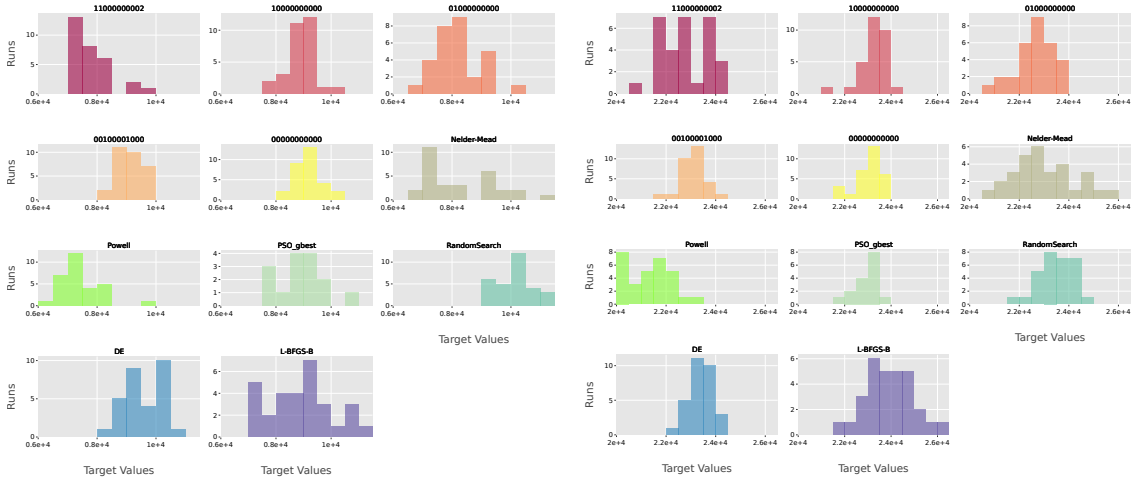
Powell's method performs best for the low and the large budget on the *nepal3* instance. Powell's method is also performing better on the low budget for the *sahara0* instance. On the large budget for the *sahara0* instance, the 00100001000 CMA-ES variant is performing better but by a small margin, *i.e.*, by around 100 voxels. Even if the difference with other algorithms is significant on the *nepal3* instance for the low budget, it is not the case for the other instance or with larger budgets.

Compared to the performances on the unconstrained use-case, algorithms tend to perform worse when the constraint is added. The main factor to explain this behavior is the location of the area to defend. Depending on the landscape of the terrain, this location forces to have bad solutions, *i.e.*, if the area to defend have an high altitude, then the radar located there will sense almost nothing.

8.5.2.1 Statistical Significance

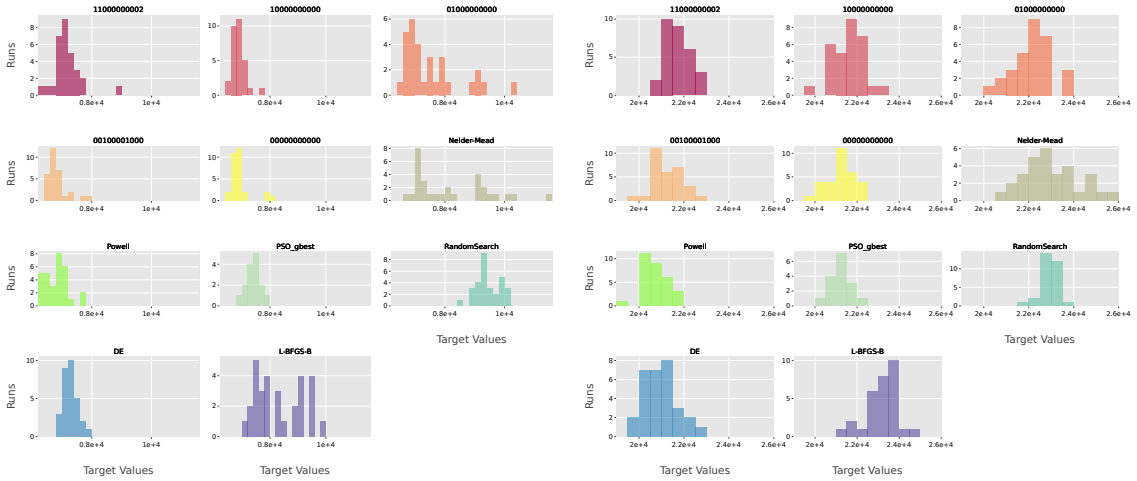
In the constrained use-case, algorithms tend to have similar performances more often than in the unconstrained use-case. This is especially the case for the lower budget of evaluations. This can be seen in the networks, as Figure 8.15a and Figure 8.16a are more sparse. Again, this is a direct effect of the area to defend as it might take several evaluations to put a radar in this area.

Otherwise, we observe the same behavior for both use-cases.



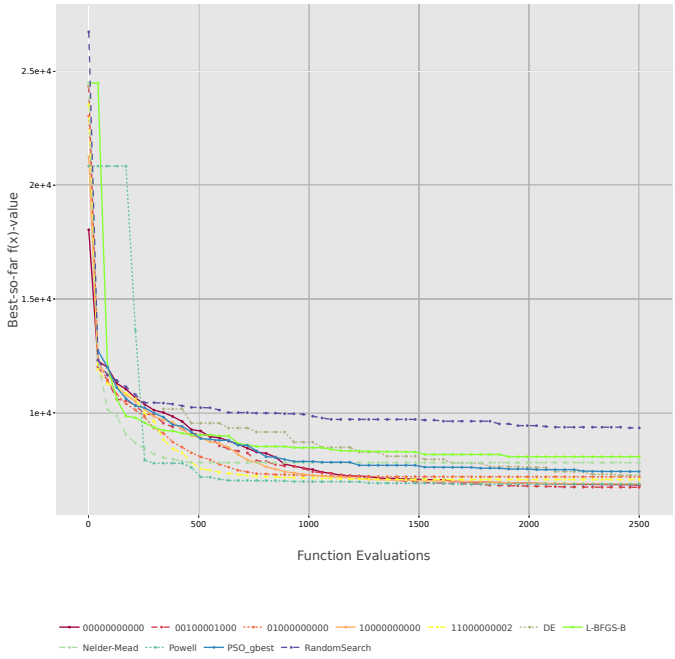
(a) *sahara0* (b) *nepal3*

Figure 8.12: Histograms of the values reached by the 30 runs, for *sahara0* and *nepal3* instances, for the 11 algorithms of our portfolio for 500 function evaluations on the constrained use-case.

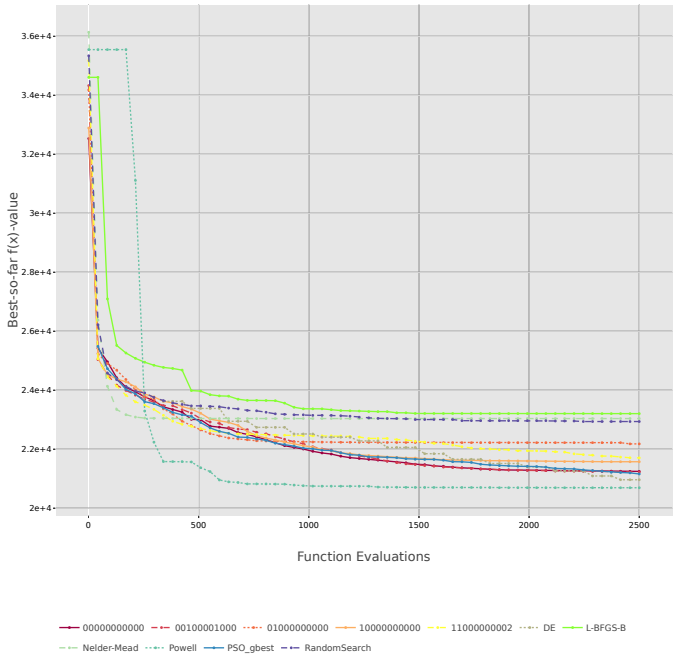


(a) *sahara0* (b) *nepal3*

Figure 8.13: Histograms of the values reached by the 30 runs, for *sahara0* and *nepal3* instances, for the 11 algorithms of our portfolio for 2,500 function evaluations on the constrained use-case.

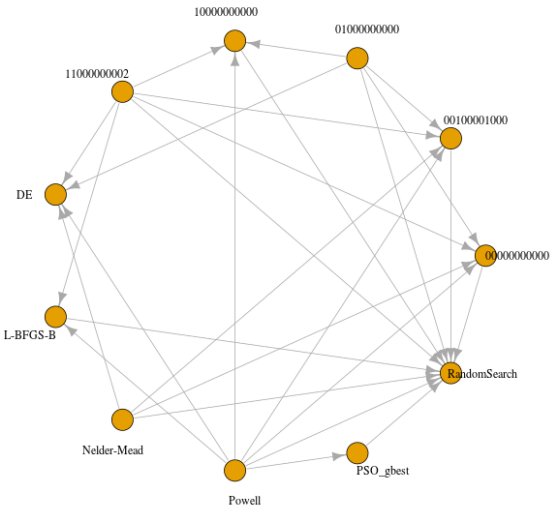


(a) *sahara0*

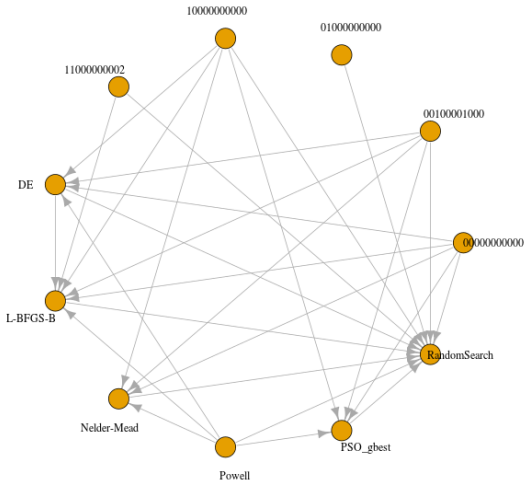


(b) *nepal3*

Figure 8.14: Median objective values over time, for 30 runs, for *sahara0* and *nepal3* instances, for the 11 algorithms of our portfolio.

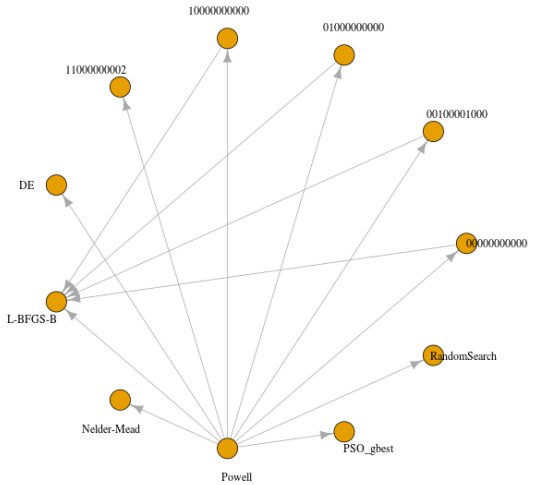


(a) 500 function evaluation

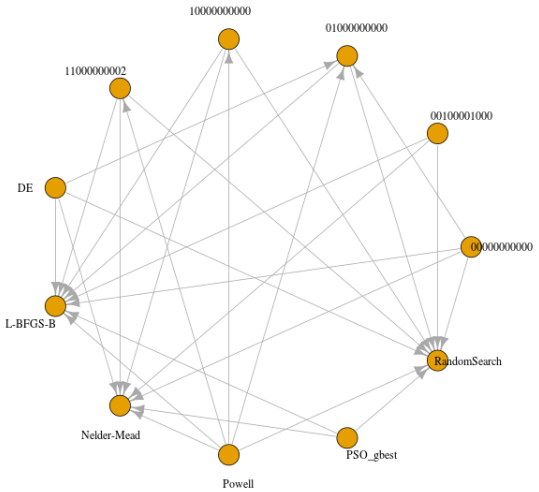


(b) 2,500 function evaluation

Figure 8.15: Partial order resulted from the Kolmogorov-Smirnov statistical test visualized as a network on *sahara0* instances on the constrained use-case. The tests are for low and large budget with a significance level $\alpha = 0.01$. An arrow from algorithm A to B indicates A is siginicantly better than B.

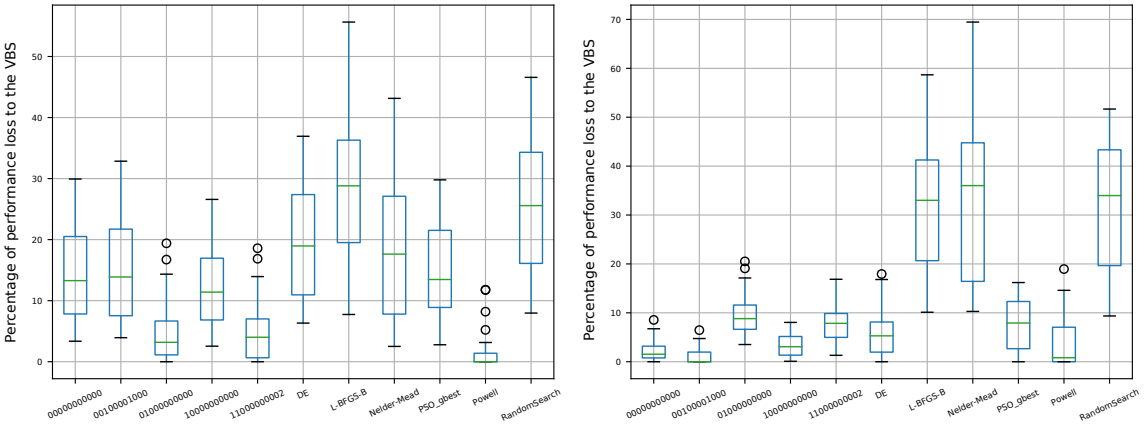


(a) 500 function evaluations



(b) 2,500 function evaluations

Figure 8.16: Partial order resulted from the Kolmogorov-Smirnov statistical test visualized as a network on *nepal3* instances on the constrained use-case. The tests are for low and large budget with a significance level $\alpha = 0.01$. An arrow from algorithm A to B indicates A is significantly better than B.



(a) 500 function evaluations

(b) 2,500 function evaluations

Figure 8.17: Boxplots of all algorithm performances on all problem instances for low and large budget of function evaluations on the constrained use-case.

8.5.2.2 Single Best Solvers (SBS)

As for the unconstrained use-case, we can determine the SBS for each budget of function evaluation.

As can be seen in Figure 8.17a, when we consider 500 evaluations, the SBS_{500} is the *Powell* algorithm. The median difference to the best solver by instance of the SBS_{500} is 0%. Powell's method also performs best on 21 out of 34 instances evaluated (Figure 8.18a).

When the number of evaluations is larger, the SBS is the same as for the unconstrained use-case, *i.e.*, the CMA-ES variant *00100001000* (Figure 8.17b). The median difference to the best solver by instance is also 0%. This variant performs best on 19 instances while the second best algorithm, Powell's method solves 10 instances (Figure 8.18b).

8.5.3 Discussion

The discussion in Section 8.4.4 on the unconstrained use-case still holds when the area to defend is added as algorithm performances have a similar behavior.

Elitists variants of CMA-ES still outperform non-elitist variants for lower budgets (Figure 8.17)

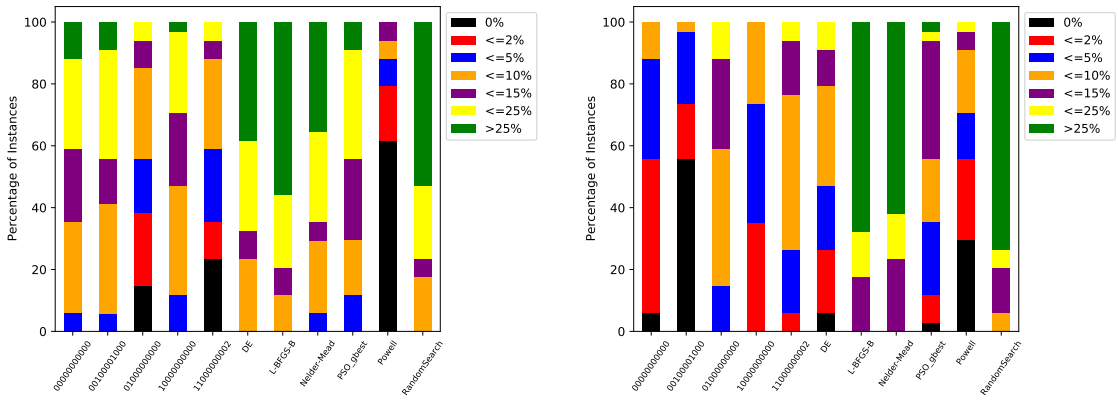
We can also still observe that flat areas are easier to cover than mountainous areas. Furthermore, it is also easier to gain performance on flat areas with more function evaluations than it is for mountainous areas.

8.6 Comparison with Manual Optimization

Configuration of radar networks is often hand-designed by experts with the help of a simulator. The objective of this section is to compare hand-designed solutions with those of the 13 algorithms from our portfolio.

The evaluation of hand-designed configurations also allows to assess how difficult the problem is to solve and to evaluate how good are solutions computed by algorithms.

The following sections present the instance selected to be solved by hand, the performances of the portfolio on this particular instance and a comparison of human-generated results to those of the algorithms.



(a) 500 function evaluations

(b) 2,500 function evaluations

Figure 8.18: Percentages of instances solved within $X\%$ of the best instance algorithm for the portfolio on the constrained use-case.

8.6.1 Problem Instance

The problem considered to be solved by hand here is the *canada0* instance which is situated in Canada, near the region of Calgary. This instance has its lowest point at 627 meters and its highest point at 2,121 meters. The shape of the landscape is given by Figure 8.19. The landscape can be divided into two areas: a rather flat area on the left and a peaked area with mountains on the right.

8.6.2 Algorithm Performances on *canada0*

The best performing algorithm on this problem is vanilla CMA-ES (00000000000) with a median fitness of 15,320 voxels not covered and a best run of 14,125 voxels not covered for 2,500 function evaluations. This solution can be observed on Figure 8.20 with the white mesh representing the covered areas.

For a smaller budget of 500 function evaluations, the CMA-ES variant 11000000002 (variant with active update, elitism and BIPOP activated, see Section 2.3.2) achieve the best median performance with a fitness of 16,518. Powell's method performs the best run with a value of 15,841.

As a comparison, the median value of random search for a large budget is 17,865 and 18,545 for a small budget. Median objective values over time for 30 runs for the 13 algorithms in the portfolio can be seen in Figure 8.21.

8.6.3 Radar Network Configuration Contest

In order to gather a sufficient amount of data to compare hand-designed solutions to algorithms, we launched a contest to solve the *canada0* instance.

8.6.3.1 Panel

The panel was composed of the following 13 people:

- 2 interns;

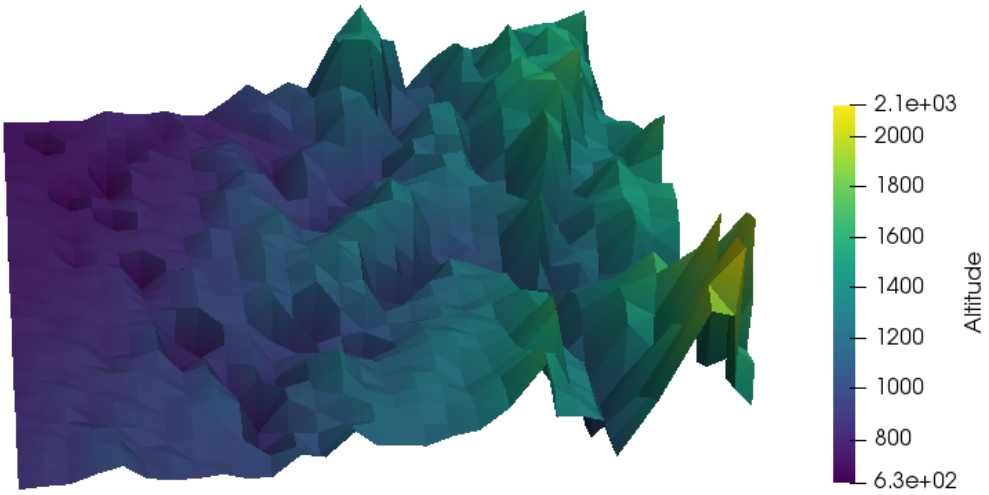


Figure 8.19: Altitude profile of the *canada0* instance.

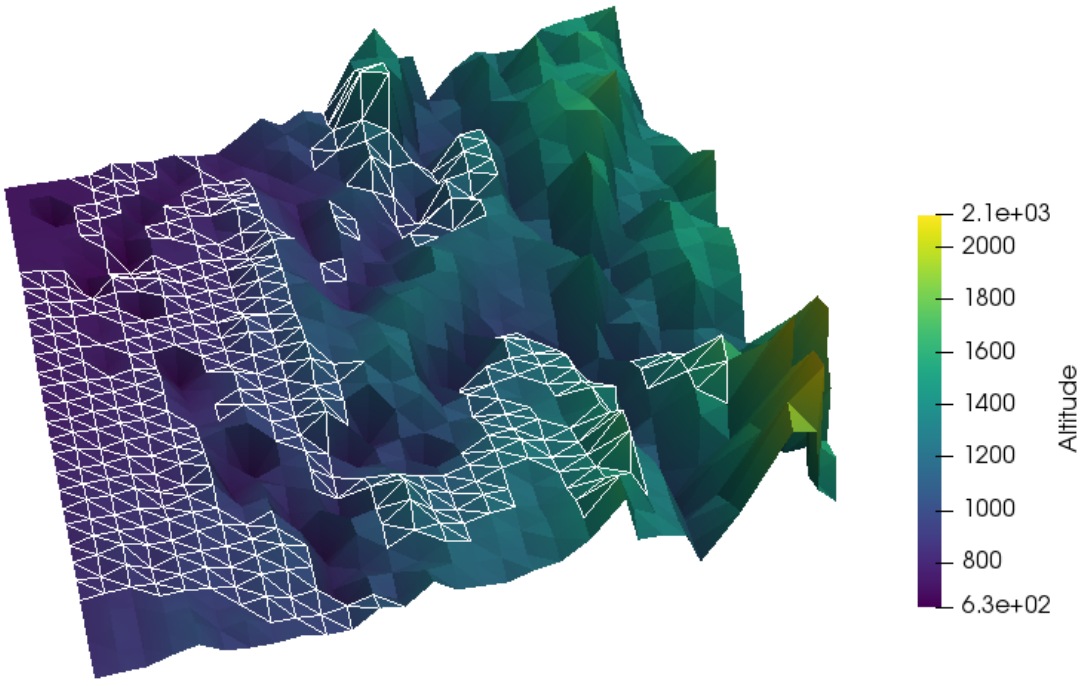


Figure 8.20: Coverage of the best solution found by the vanilla CMA-ES algorithm for 2,500 function evaluations on the *canada0* instance. The white mesh represents the area covered.

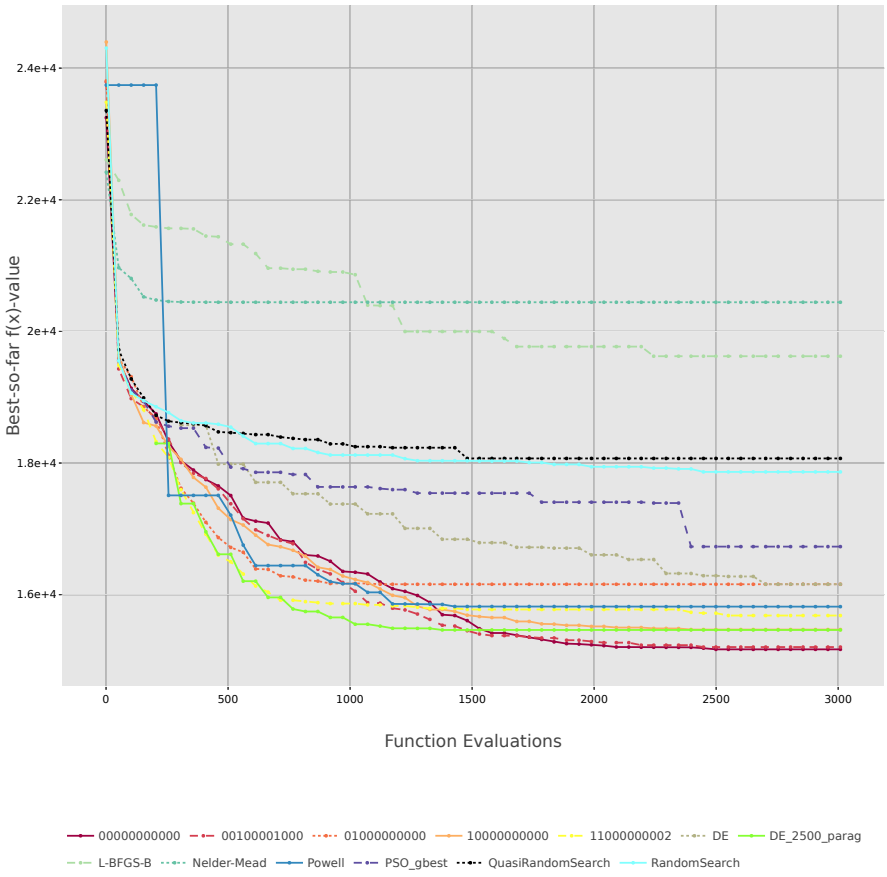


Figure 8.21: Median objective values over time, for 30 runs, for *canada0* instance, for the 13 algorithms of our portfolio.

Table 8.3: Difference of performance between solvers and human results in percentage. *fevals* refers to function evaluations. Negative values indicate that human was less efficient.

	Human best	Human median
Best algorithm run (500 fevals)	-4%	-10%
Best algorithm run (2500 fevals)	-16%	-24%
Median of best algorithm (500 fevals)	+0.6%	-6%
Median of best algorithm (2500 fevals)	-8%	-15%

- 3 PhD students with one having some knowledge on optimization;
- 1 Postdoctoral student having some knowledge on optimization;
- 7 engineers, one having a radar expertise, two having some knowledge on optimization and one with both.

8.6.3.2 Solving Conditions

At first, we presented the landscape of the instance to the contestant. Then, we introduced them the goal, *i.e.*, covering as much space as possible and the resources to reach that goal, *i.e.*, the four radars and their corresponding parameters.

We set the resolution time at 30 minutes with as many solution evaluations as required. As a comparison, the average algorithm wallclock time for is around five minutes. For each contestant, we log the best value found and the improvements done during the resolution.

8.6.4 Comparison of Hand-Designed Configurations with Algorithms

The median value obtained as a group is 17,495 voxels not covered when the human best designed configuration reached the value of 16,425. The comparison between the algorithm performances and human-designed solutions can be found in Table 8.3.

Overall, only the best configuration found by the panel can outperformed solutions proposed by the best algorithms. Moreover, when the human-designed configuration outperforms the optimization algorithm, it is only by a small margin, i.e, less than 1%.

In these results, the median performance of the human-designed configuration has a difference around 2% with the random search. Given these results, optimization algorithms tend to perform better than humans at designing radar network configurations. Moreover, the *canada0* instance seems hard to solve manually as human performances are close to those of the random search.

Chapter 9

Landscape-Aware Algorithm Selection on the Unconstrained Use-Case

Contents

In this chapter, we describe the different steps and design choices to perform a landscape-aware algorithm selection on the two radar network configuration use-cases introduced in Section 9.1. We present the characteristics of the unconstrained use-case problems in Section 9.2. Results for this use-case are summarized in Section 9.4. A discussion of the results is offered in Section 9.5.

9.1 Design of the Selector

Our landscape-aware algorithm selection pipeline is composed of the following components:

- creation of training and testing sets (Section 9.1.1);
- features extraction (Section 9.1.2);
- algorithm performances presented in Chapter 8;
- creating the mapping between feature data and algorithm performances (Section 9.1.3).

9.1.1 Training and Testing Sets

Training and testing sets are created randomly using all 153 instances available. The training set is composed of 80% of the instances, the remaining 20% compose the testing set.

As a random selection of the benchmark instances can be biased and not representative of the full set of instances, five independent runs are performed to assess the selector.

9.1.2 Feature Computation

We describe the two ways that we used to compute features. One is traditional and computes features on the objective function (Figure 9.1a), while the other computes features on the DEM (Figure 9.1b). These two approaches lead to two different selectors that will be compared in the following sections.

9.1.2.1 Objective Function

The feature computation on the objective function is done as described in Section 4.2. All cheap features available in *flacco* are computed. We excluded the PCA features as most of them do not provide sufficient information (see Section 5.4 for a discussion). This leaves us with 38 features in total. The Sobol´ low discrepancy sequence (defined in Section 5.3) is used to sample the radar network configurations. Following common recommendation from the literature, we base the computation of features on $50d = 750$ samples. To compensate for the randomness in the sampling, we perform this feature extraction step 100 independent times. The 100 runs of the features computed on the objective function are aggregated using their *median value*. Hence, for each problem, one vector corresponding to the median values of each feature is used to construct the mapping.

9.1.2.2 Digital Elevation Model (DEM)

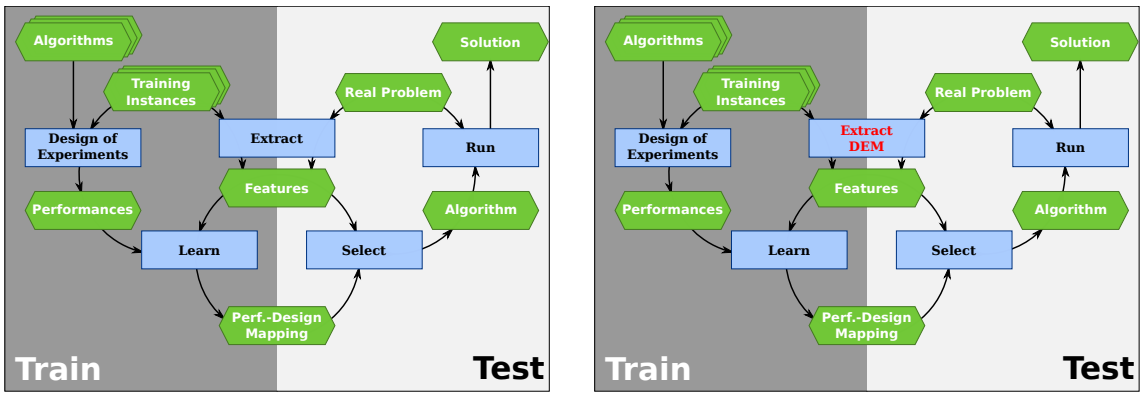
Instead of computing features on the objective function, they also can be computed on the DEM. This comes with the idea that the structure of the physical landscape is the factor that has the biggest impact on the objective function.

In order to compute features, we need to have an objective function. In this case, we take the altitude as a function, *i.e.*, we define a function $g(x, y) = z$, where z is the altitude of the point (x, y) in the domain. Hence, unlike the radar network configuration problem, the dimension of this problem is $d = 2$. On top of that, the domain is discretized, which implies that the number of different altitudes that we can use for the feature computation is at most the number of pixels which is equal to $30 \times 30 = 900$.

This approach has two main *benefits*: first, it avoids the rather expensive evaluation of the objective function. Another advantage, as the domain is discretized, we can fully sample the search space. Doing so, we can remove the randomness of the sampling.

The *drawback* of this approach is the loss of information. Even though the altitude seems to be the factor that has the biggest impact on the objective function, it is not the only one. While computing features using the objective function consider all radar parameters (*tilt*, *staring angle*, and internal processing), computing features on DEM focus on the elevation profile of the instance. Disregarding radar parameters may imply some loss of information on the problem to solve.

The second loss of information resides in the number of features we can use. Nearest Better Clustering features (see Section 4.2.3) are sensitive to grid sampling and cannot be computed in this case. Having less features available may imply having less information to distinguish between instances.



(a) Radar use-case

(b) DEM

Figure 9.1: Landscape-aware algorithm selection pipelines: feature extraction on DEM or on the radar use-case.

In the following, the features computed on the DEM are referred by DEM features.

9.1.3 Building the Mapping between Feature Data and Algorithm Performances

The mapping aims at identifying a function $f : \mathbb{R}^{|\Phi|} \to \mathbb{R}$ that link the performances of algorithms with feature data.

One vector of feature is representing an instance by using the median value of the 100 runs on the objective functions (Section 9.1.2.1) or directly the DEM features (Section 9.1.2.2). The algorithms performances are represented for each problem by the median value of the 30 independent runs for a given budget. This setting corresponds to the Per-Instance Algorithm Selection introduced in Chapter 3.

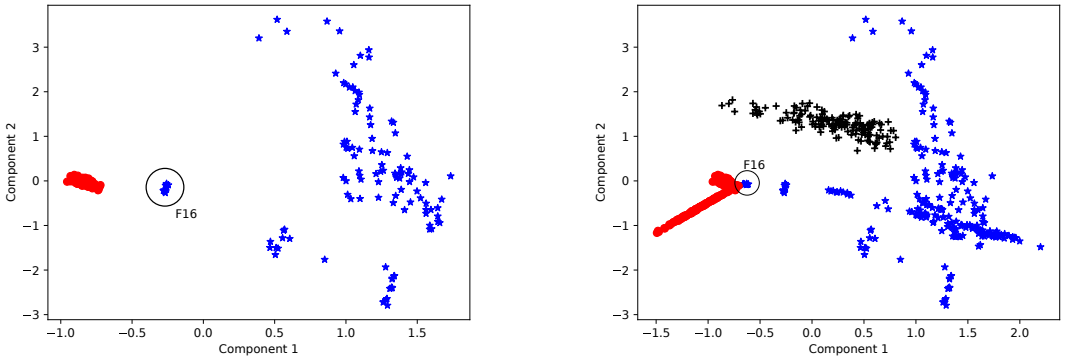
We denote with S_{radar} the selector built with features computed on the objective function and by S_{DEM}, we denote the selector built with features extracted from the DEM.

As previously done in the literature [Bel17, JD20], we build the mapping between feature data and algorithm performances with default scikit-learn Random Forests [PVG+11] using *regression*. For each algorithm, we create one Random Forests regression model.

Each regression model learns the performances of the associated algorithm for a given feature vector. The selector is composed of all the regression models. When the selector has an unknown feature vector as input, every regression model predicts the performance of its associated algorithm, and the selector then ranks the algorithms by their predicted performance. Hence, the selector can be used to recommend one or multiple algorithms.

9.2 Problem Characteristics

Figure 9.2 is a projection in two dimensions with a Principal Component Analysis [Pea01]. Figure 9.2a is a projection of 38 features extracted from the 24 BBOB function and the radar use-case only. Figure 9.2b is a projection of 33 features excluding nearest better clustering features. The features are computed on the 24 BBOB functions, the radar use-cases, and the DEM features.



(a) Projection of 38 features extracted from 24 BBOB functions (blue stars) and from the radar objective function (red dots).

(b) Projection of 33 features extracted from 24 BBOB functions (blue stars), from the radar objective function (red dots), and from the DEM. (black crosses).

Figure 9.2: Projection of features extracted from 24 BBOB functions (blue stars), from the radar objective function (red dots), and from the DEM (black crosses). Nearest better clustering features are ruled out for the projection using the DEM.

Even if all three sets of functions are apart from each other, it is important to note that the cluster of both radar and DEM functions set are more dense than the BBOB function set, *i.e.*, elements from radar clusters have a smaller distance between each other than elements from the BBOB functions cluster. This suggests that radar configuration problems are more similar to one another than functions from the BBOB benchmark suite. This may impact the performances of the selector as similar instances may be difficult to differentiate.

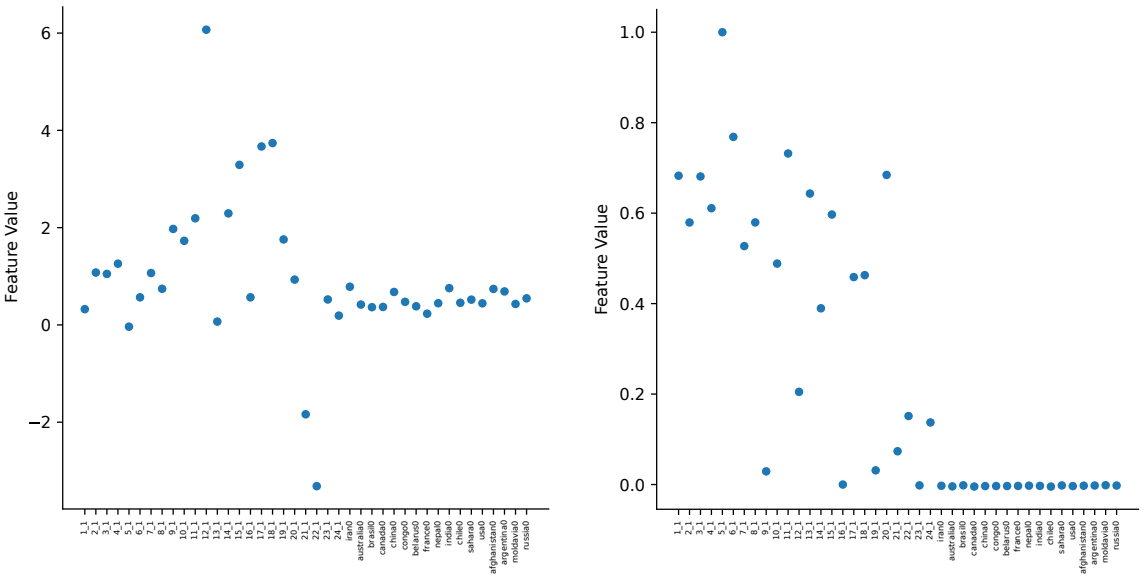
The 24 BBOB functions that have features close to the radar-use are the five instances of function 16, the Weierstrass function.

Figure 9.3 shows feature values for a y-distribution and a meta-model feature on the first instance of the 24 BBOB functions and the radar instances 0. In Figure 9.3a, we can observe that radar problem features and BBOB functions features are not so different. Radar problem features are similar to 13 out of the 24 BBOB function features. Nevertheless, the general trend in the one observed in Figure 9.3b, *i.e.*, radar problem features and BBOB functions features forming two different sets. Occasionally, some BBOB functions feature values are close to radar instances feature values such as functions 16 and 23 in Figure 9.3b

9.3 Definition of the SBS

To perform a landscape-aware algorithm selection, we need to split our 153 instances into a training and a testing set (see Section 9.1.1). The splitting can have an impact on the definition of the SBS. Training splits composed of a majority of mountainous instances may have a different SBS than training splits composed of a majority flat instances.

To visualize the impact of the splits, we create 1,000 independent splits of training instances and record the SBS on both budgets. On the lower budget of evaluations, three algorithms were SBS on the splits. 11000000002 was SBS on 503 splits, DE_2500_chile was SBS on 394 splits, and 01000000000 on 103 splits. On the use-cases, we will consider as SBS_{500} the 11000000002 CMA-ES



(a) y-distribution skewness feature

(b) Meta-model R^2 of (LM) feature

Figure 9.3: Feature values of a y-distribution and a meta-model feature. Values for the first instance of the 24 BBOB functions and the radar instances 0.

variant as this algorithm was more often SBS on the different splits. This choice is different from the global SBS_{500} from Section 8 where DE_2500_chile was the best performing algorithm.

Concerning the larger budget, two algorithms were SBS on the splits. The CMA-ES variant 00100001000 on 986 runs and DE_2500_chile on the remaining 14 runs. The $SBS_{2,500}$ is the same for the different splits and for all the instances.

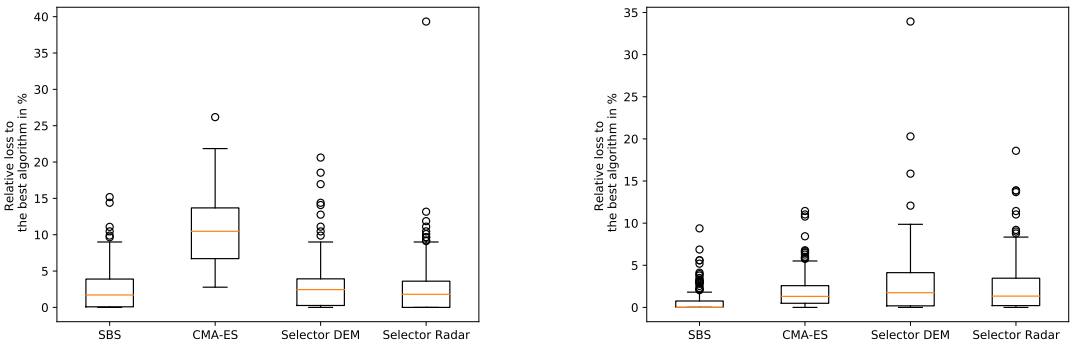
When two algorithms are proposed, the SBS correspond to the best pair of algorithms, *i.e.*, the pair of algorithms that has the best complementarity on all instances. The overall SBS pair for this problem is given by the couple of algorithms *DE_2500_chile , 11000000002* for the low budget. Concerning the large budget, the overall SBS pair is composed of *DE_2500_chile , 00100001000*. For both budgets, the overall pairs are also the best pairs for each splits.

9.4 Selector Performances

We will show in this section the results of our two selectors and how they compare to the VBS, the SBS, and a baseline: the vanilla CMA-ES (from the ModCMA framework [vRWvLB16]). We have chosen the vanilla CMA-ES as baseline as we expect that vanilla CMA-ES will be tried in the industrial context.

From Section 9.2, we know that radar problem instances have similar feature data which can harden the task of distinguishing between instances. Moreover, we have seen in Section 8.5 that algorithms have similar performances. This implies that the VBS-SBS gap is small and that the margin to perform landscape-aware algorithm selection is small.

Figures 9.6 and 9.7 show the results of the regression selector for both low and large budget of function evaluations with features computed on the radar use-case and on the DEM respectively. The results are given in function of the number of predicted algorithms by the selector.



(a) 500 function evaluations

(b) 2,500 function evaluations

Figure 9.4: Relative loss in percentage of the SBS, vanilla CMA-ES, and the selectors with respect to the VBS, assuming that only the best algorithm is proposed.

Figure 9.4 illustrates the results when one algorithm is proposed while Figure 9.5 shows the results for two recommendations of algorithm.

9.4.1 Recommending One Algorithm

On low budget, the selectors outperform the baseline vanilla CMA-ES by around 7% in median performance. On larger budget, selectors and vanilla CMA-ES have equivalent performances, *i.e.*, at around 2.5% of the VBS in median performance (see Figure 9.4).

This could be expected from what was mentioned in Section 8.4. Vanilla CMA-ES was not performing well for low budget settings and needed more function evaluations to converge. On the larger budget, it was one of the best performing algorithm so we expected its performances to be close to those of our selectors.

Overall, the selector perform worse than the SBS for both budgets at 2.6% from the VBS in median We recall that for this use-case, the median gap between the SBS and the VBS is 0.39% on low budget and 0% for the large budget.

9.4.2 Recommending Multiple Algorithms

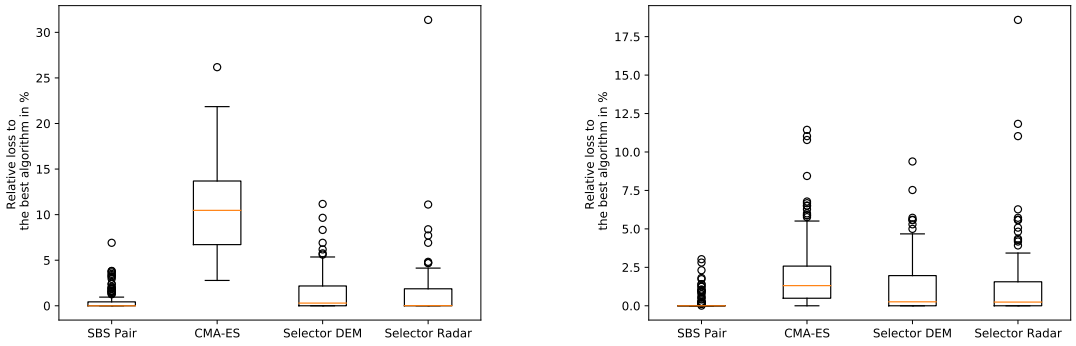
When two algorithms are proposed by the selectors, performances are within 1.2% of the VBS performances in median. Given that the SBS pair always has a median performance within 0% of the VBS performance on both budgets, there is no gap for improvement.

As expected, Figure 9.6 and Figure 9.7 show the performances of the selectors converging to the VBS performances as the number of recommended algorithms increases. We obtain for both selectors a median performance equal to the VBS when four algorithms are proposed.

9.4.3 Performances of $S_{\mathbf{radar}}$ versus $S_{\mathbf{DEM}}$

The performances of S_{DEM} and S_{radar} are similar. The median performance of S_{DEM} is within 0.2% of the median performance of S_{radar} on both budgets of function evaluations.

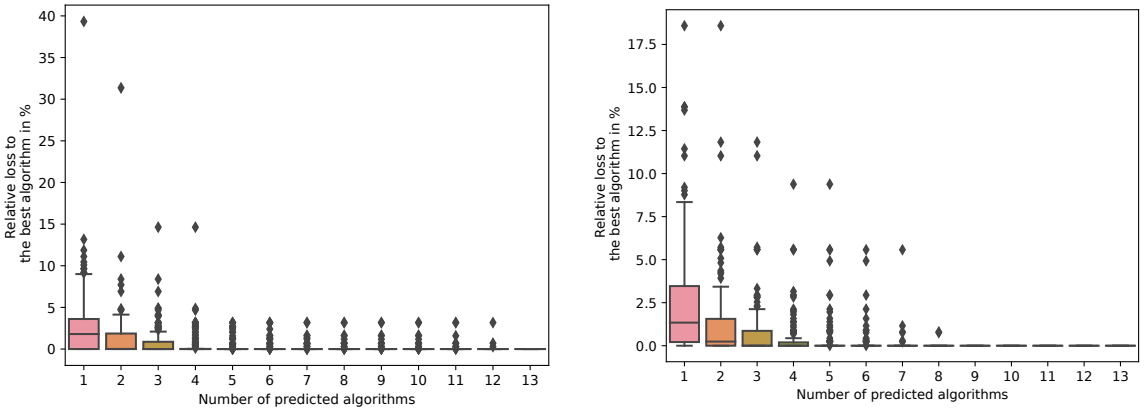
Given the small difference in performances, the DEM selector can be preferred to S_{radar} as its building and using are almost free in computation time. More precisely, the computation time



(a) 500 function evaluations

(b) 2,500 function evaluations

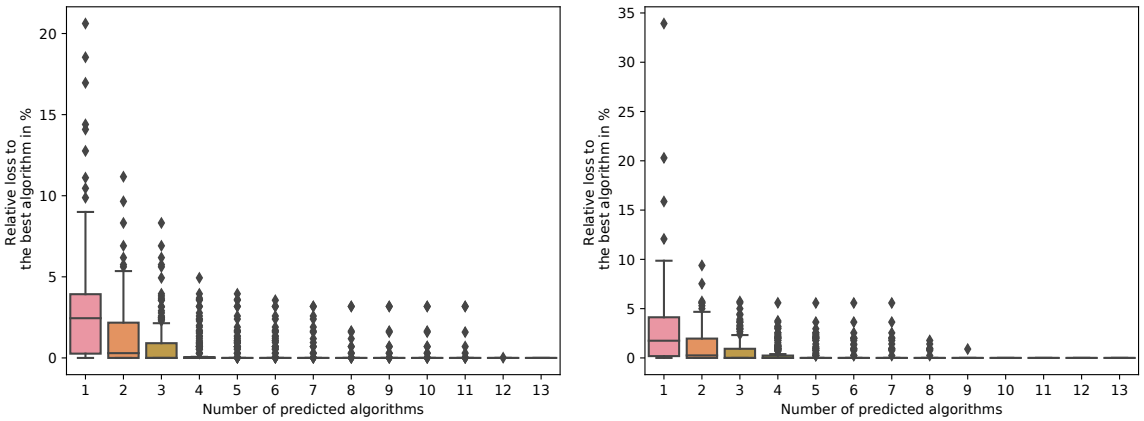
Figure 9.5: Relative loss in percentage of the SBS, vanilla CMA-ES, and the selectors with respect to the VBS, assuming that only the best pair of algorithms are proposed.



(a) 500 function evaluations

(b) 2,500 function evaluations

Figure 9.6: Relative loss in percentage of the selector compared with the VBS. The selector is built using objective function feature data and the 13 algorithms from our portfolio.



(a) 500 function evaluations

(b) 2,500 function evaluations

Figure 9.7: Relative loss in percentage of the selector compared with the VBS. The selector is built using DEM feature data. the 13 algorithms from our portfolio.

of the selector S_{radar} is around 2.5 minutes when it only takes one or two seconds to use S_{DEM}. This difference is a consequence of the evaluation of the objective function samples and the feature computation.

9.5 Discussion

Performances of our selectors are slightly worse than the SBS. Nevertheless, their performances are quite close to the VBS, *i.e.*, around 2.5% in median for both budgets.

Performance of selectors may be improved by tuning the Random Forest used for mapping feature data and algorithm performances. Nevertheless, even if the performances of the selectors increase, the possible gain is relatively low.

Given the performances of the selectors compared to those of the SBS, it would be beneficial to use only the SBS to solve this use-case. Selectors are not performing badly but the VBS-SBS gap is too small and does not benefit an automated algorithm selection procedure.

When there is time to run multiple algorithms, it could be beneficial to run both the SBS and the algorithm recommended by the selector. For some instances, the algorithm predicted by the selector may have better performances than the SBS.

Part V

Conclusions

Conclusions

10.1 Summary of Contributions

In this thesis, we have investigated Per Instance Algorithm Selection (PIAS) on a real-world continuous black-box optimization problem namely a *radar network configuration problem*.

The contributions of this thesis can be summarized in two parts. A first part that focuses on the analysis of ELA (Exploratory Landscape Analysis) features and a second part that investigates the efficient optimization of two radar use-cases with different computational budgets.

10.1.1 Investigation of Exploratory Landscape Analysis Features

The contributions on landscape features were presented in Chapter 5 and focus on the properties of ELA features. We defined six properties that features should possess.

We empirically studied these properties and found that even if most features are invariant to transformations and noise, some of them are not expressive or not robust. We also found that the sampling strategy matters as most features are sensitive to it and that Sobol´ low-discrepancy sequence may be outperforming others with respect to expressiveness. Concerning stability, we found that only a few features are stable to independent samples. Overall, we found that no feature satisfies the six defined properties.

We also introduced new features that refine the PCA features. The analysis of the properties of the new introduced features demonstrate that they are more expressive than the PCA features they are based on.

10.1.2 Optimizing Radar Networks

To optimize radar networks, we extended the Ægis framework developed in Thales. This framework of sensor modeling has now more than 17,000 code lines. This is more than three times bigger than the 5,000 code lines from the beginning of this thesis. New functionalities were added, including new radar models, new ways to aggregate networks, the handling of constraints, a more flexible handling of geographical data, and an interface for users to visualize the use-cases.

The Ægis framework also acts as the objective function for the radar network use-cases that we solve in this thesis. The use-cases are radar network configuration problems where we want to find radar locations and radar parameters in order to maximize the coverage of an area.

Usually, such radar network configuration problems are solved by experts with the help of simulators. In this thesis, we proposed multiple optimization techniques to solve this problem.

The first technique is optimizing the coverage using an optimization algorithm. We showed that, for most algorithms, the performances of the optimization procedure is often better than human-designed solution.

Given that we can easily categorize some instances of the use-cases as *flat instances* or *mountainous instances*, we created specific solvers by tuning their parameters. Using this procedure, we found one configuration of Differential Evolution that is better than its default configuration at solving the use-cases.

As we compare different algorithms, we also found the Single Best Solver (SBS) of our portfolio to solve the use-cases.

We went one step further by performing a landscape-aware algorithm selection with the goal to find the best suited algorithm for any given instance. This selection has performances similar to the SBS which suggests that using the SBS instead of the landscape-aware selection may be sufficient on the use-cases investigated in this thesis. Nevertheless, we show that the landscape-aware selection had performances close to the Virtual Best Solver (VBS) and thus, the approach is still performing well on the use-cases.

Moreover, the use-cases are purposely simplified models of radar network configuration problems. Highly realistic models are more complex and have more parameters that influence the search. Consequently, this complexity may increase the difference between the VBS and the SBS and thus, favor a landscape-aware selection of algorithm.

10.2 Perspectives

In the light of our results, we have identified several promising research directions.

At the center of landscape-aware algorithm selection are ELA features. New features were developed by other research groups during the thesis time and will probably be developed in the coming years. The study of the properties of these features is crucial in order to be able to use them correctly. Moreover, the study of their interconnection is also relevant. There are more than 200 SOO features [DLV$^+$19] and from now on, their interconnection between each other and other ELA features is unknown.

In this thesis, we perform an *offline* landscape-aware algorithm selection. A promising research direction would be to look at a *dynamic* algorithm selection, *i.e.*, be able to switch algorithm during the search. Switching algorithms during the search may be done using hybrid metaheuristics [Tal13] or using memetic algorithms [COLT11]. While memetic algorithms apply a local search after the mutation operator, hybrid metaheuristics often add a local search algorithm after the run of an evolutionary algorithm [Tal13]. The addition of the local search is classically done after the run of the evolutionary algorithm in a static way, *i.e.*, after a fixed number of function evaluations or when there is no further improvement in new of solutions. In the dual annealing [VGO$^+$20], a local search is applied after simulated annealing.

One first step would be to consider hybrid algorithms that either favor exploration or exploitation given the landscape of fitness function as in [RY21]. While the algorithm is switched during the search, the choice of the switching point is done *a priori*. A more ambitious direction would be to switch algorithms during the search given the local structure of the landscape [VWBD20]. While dynamic selection has been done for CMA-ES variants [VvRBD19], a broader picture would be to be able to switch different types of algorithms during the search. In [VvRBD19], the authors switched the CMA-ES variant once per run. Allowing multiple changes given the local landscape could improve the performance of the dynamic selection. In [JED21], the authors studied the use of ELA features computed with the algorithm samples in order to predict the algorithm runtime with the objective to do dynamic algorithm selection.

There is also a promising research direction in training instances creation, especially for real-world use cases. In this thesis, we are limited by the number of instances for two main reasons, the expensive computation time of the simulator and the amount of geographical data available. In order to avoid the computation time needed by the simulator, one could use a similar approach as proposed in [MS20], *i.e.*, generating analytic versions of real-world problems based on their feature vectors. Moreover, this approach could also be used to generate new instances that have similar feature vectors to real radar use-cases and thus expand artificially the training instances available. Increasing the number of training instances implies having more data to construct a selector. Compared to classical machine learning or deep learning approaches, the amount of data used in this thesis is limited. The additional data created may improve and add robustness to the performance of the selector as more different training instances will be available.

Appendix A

Summary of Papers and Industrial Achievements

A.1 Academic Papers and Presentations

The academic contributions focus on ELA features that characterize optimization problems and in particular their properties. In this thesis, we exhibit properties that features should possess and we perform an analysis of the already existing features.

The results presented in this thesis built upon the papers listed below. [RDDD20b] [RDDD19] [RDDD21a]

A.1.1 International Conferences

- **Q. Renau**, J. Dreo, C. Doerr, and B. Doerr. Expressiveness and robustness of landscape features. In *Proceedings of the Genetic and Evolutionary Computation Con- ference (Companion), GECCO '19*, pages 2048–2051. ACM, 2019;
- **Q. Renau**, C. Doerr, J. Dreo, and B. Doerr. Exploratory landscape analysis is strongly sensitive to the sampling strategy. In *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Proceedings, Part II*, volume 12270 of *Lecture Notes in Computer Science*, pages 139-153. Springer, 2020;
- T. Eftimov, G. Popovski, **Q. Renau**, P. Korošec, and C. Doerr. Linear Matrix Factorization Embeddings for Single-objective Optimization Landscapes. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI), 2020*, pages 775-782. IEEE, 2020;
- **Q. Renau**, J. Dreo, C. Doerr, and B. Doerr. Towards Explainable Exploratory Landscape Analysis: Extreme Feature Selection for Classifying BBOB Functions. In *Applications of Evolutionary Computation - 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Proceedings*, volume 12694 of *Lecture Notes in Computer Science*, pages 17-33. Springer, 2021.

A.1.2 Data Sets

As part of the PhD project, we also produced two open source data sets:

- **Q. Renau**, J. Dreo, C. Doerr, and B. Doerr. Experimental Data Set for the study "Exploratory Landscape Analysis is Strongly Sensitive to the Sampling Strategy". In Zenodo, 2020;
- **Q. Renau**, J. Dreo, C. Doerr, and B. Doerr. Exploratory Landscape Analysis Feature Values for the 24 Noiseless BBOB Functions. In Zenodo, 2021.

A.2 Industrial Achievements

Industrial achievements are linked to the radar network configuration problem.

To model the use-cases considered in this thesis, we have significantly extended the Ægis framework by adding new functionalities and 12,000 lines of code. Ægis is a C++framework for sensor network modeling and is the core of the industrial part of this thesis as it permits to create and simulate the radar networks. Ægis can be interfaced as a black-box with optimization algorithms in order to evaluate the possible configurations.

Usually, the radar network configuration problem is solved by experts manually with the help of a simulator. Different methods presented and codes developed in this thesis can be used in order to solve this problem:

- *using an optimization algorithm.* As the problem is usually solved by hand, using an off-the-shelf optimization algorithm is already an upgrade. As no information is known on the problem or on the behavior of the algorithm on this problem, the result can either be better or worse than human-designed solutions;
- *using Algorithm Configuration.* Off-the-shelf optimization algorithms may not be adapted to the problem at hand and may require some configuration of its parameters. Using algorithm configuration, the user can expect better results than from an algorithm with no configuration. Nevertheless, if the algorithm is not well adapted to the problem, human-designed solutions can still be better than the results from the optimization algorithm;
- *finding the single best solver (SBS) in a portfolio.* Instead of using one off-the-shelf algorithm, we can find the best performing algorithm in a portfolio of off-the-shelf or custom algorithms. This solution will find a well adapted algorithm on the use-case if one is present in the portfolio. As a result, this solution should produce better results than the previous two and is likely to perform better than human-designed solutions;
- *performing a landscape-aware algorithm selection on a portfolio of algorithms.* The user can expect at least equivalent performances with the SBS. When there is a good algorithm's complementarity in the portfolio, the user may expect even better performances than the SBS as algorithms are chosen on the instances particularities;
- *creating instance specific solvers.* Instead of tuning an algorithm on all instances, we tune it on some instances only. Creating specific solvers increases the complementarity between algorithms in a portfolio. As such, one could expect better performances of a landscape-aware algorithm selection with a portfolio containing specific solvers than the other approaches.

Appendix B

Best Performing Algorithm for each Instance

B.1 Best Performing Algorithm in Median

Instance	Algorithm 500 evaluations	Median value 500 evaluations	Algorithm 2,500 evaluations	Median value 2,500 evalutaions
afghanistan0	11000000002	11855	11000000002	10983
afghanistan1	11000000002	12196	11000000002	11348.5
afghanistan2	11000000002	12386.5	00100001000	11269
afghanistan3	11000000002	13055	00100001000	11310
afghanistan4	11000000002	12160	00100001000	10922.5
afghanistan5	1000000000	12311	00100001000	11076.5
afghanistan6	1000000000	10064	00100001000	8725.5
afghanistan7	11000000002	9233	00100001000	8127.5
afghanistan8	11000000002	11104.5	00000000000	8839.5
arabia0	11000000002	11533	00100001000	10102.5
arabia1	DE_2500_chile	9467.5	DE_2500_chile	8444
arabia2	DE_2500_chile	11192	00000000000	10001.5
arabia3	DE_2500_chile	13380.5	10000000000	12287.5
arabia4	1000000000	10400.5	00100001000	9209.5
arabia5	DE_2500_chile	11551.5	00100001000	10111
arabia6	11000000002	13001.5	00000000000	11268
arabia7	DE_2500_chile	10034.5	00100001000	8789.5
arabia8	DE_2500_chile	12080	00100001000	11064.5
argentina0	1000000000	6619.5	00100001000	5965
argentina1	11000000002	6548	00000000000	6012.5
argentina2	1000000000	6608	00100001000	5918.5
argentina3	11000000002	6633.5	00100001000	6008
argentina4	Powell	6559	00100001000	6031
argentina5	1000000000	6726	00100001000	6031.5
argentina6	Powell	6659.5	00100001000	6072.5

Table B.1: Best performing algorithm in median by instance.

argentina7	11000000002	6809.5	00100001000	6020
argentina8	1000000000	6771.5	00000000000	6063.5
australia0	1000000000	8547.5	10000000000	7202
australia1	11000000002	7141	00100001000	6242
australia2	11000000002	6606	00100001000	5941.5
australia3	Nelder-Mead	6579.5	00100001000	5910.5
australia4	11000000002	10261	10000000000	9087
australia5	1000000000	6645	00100001000	5962.5
australia6	1000000000	7307	00100001000	6466.5
australia7	Powell	10672.5	DE_2500_chile	9886
australia8	Powell	6552	00100001000	6025
brasil0	Powell	7639	00100001000	6527
brasil1	11000000002	7313	00100001000	6391.5
brasil2	Powell	7895	00000000000	6978.5
brasil3	Powell	8470.5	00100001000	7386
brasil4	11000000002	6698	00000000000	5958.5
brasil5	DE_2500_chile	8184	00100001000	7085
brasil6	1000000000	9167	00100001000	8080
brasil7	DE_2500_chile	7302.5	DE_2500_chile	6668.5
brasil8	DE_2500_chile	8503.5	00100001000	7297
canada0	11000000002	16518	00000000000	15169
canada1	Powell	20877	DE_2500_chile	19833
canada2	DE_2500_chile	21387	00100001000	20502.5
canada3	DE_2500_chile	21159	00100001000	20345
canada4	DE_2500_chile	20102.5	DE_2500_chile	19346.5
canada5	DE_2500_chile	18823.5	DE_2500_chile	17832.5
canada6	DE_2500_chile	21213	DE_2500_chile	20381
canada7	DE_2500_chile	18137	DE_2500_chile	17000
canada8	DE_2500_chile	18909	DE_2500_chile	18184.5
china0	DE_2500_chile	19427	DE_2500_chile	18616
china1	DE_2500_chile	19903	00000000000	18654
china2	1000000000	20302.5	00100001000	19140.5
china3	DE_2500_chile	19555.5	DE_2500_chile	18643
china4	DE_2500_chile	20007.5	00000000000	18421.5
china5	DE_2500_chile	20545.5	00100001000	19270.5
china6	DE_2500_chile	19908	DE_2500_chile	18955.5
china7	11000000002	19843	00000000000	18658.5
china8	DE_2500_chile	19776	DE_2500_chile	18834
congo0	11000000002	7346.5	00100001000	6474.5
congo1	DE_2500_chile	8142	00000000000	6814
congo2	11000000002	7241	00100001000	6455
congo3	Powell	7622.5	00100001000	6534.5
congo4	DE_2500_chile	7261.5	00000000000	6324.5
congo5	1000000000	7261	00100001000	6419
congo6	1000000000	7781.5	00100001000	6758.5
congo7	1000000000	7403.5	00100001000	6348.5
congo8	Powell	7646.5	DE_2500_chile	6822

Table B.1: Best performing algorithm in median by instance.

estonia0	DE_2500_chile	9442.5	DE_2500_chile	8274.5
estonia1	DE_2500_chile	9772.5	00100001000	8604.5
estonia2	11000000002	8736.5	DE_2500_chile	7501
estonia3	1000000000	9116	00100001000	7797.5
estonia4	DE_2500_chile	7909.5	00100001000	6963.5
estonia5	Powell	10674.5	Powell	9579
estonia6	1000000000	8628	00100001000	7553
estonia7	1000000000	9329	00100001000	7913.5
estonia8	DE_2500_chile	11164.5	00100001000	9181.5
france0	11000000002	9652.5	00100001000	8621
france1	Powell	12292	00000000000	11323
france2	Powell	11093.5	00100001000	10255
france3	11000000002	9474	00000000000	8403
france4	DE_2500_chile	14913.5	DE_2500_chile	13937
france5	Powell	13048	DE_2500_chile	11983
france6	DE_2500_chile	12462	DE_2500_chile	11361.5
france7	DE_2500_chile	13149	DE_2500_chile	11913
france8	1000000000	10909	00100001000	9770.5
moldavia0	DE_2500_chile	11097.5	DE_2500_chile	9739
moldavia1	DE_2500_chile	12062	DE_2500_chile	10747
moldavia2	11000000002	9187	00000000000	8220.5
moldavia3	DE_2500_chile	10013	00100001000	9150.5
moldavia4	DE_2500_chile	13244	00100001000	11895.5
moldavia5	Powell	10230	00100001000	9266.5
moldavia6	1000000000	9553.5	00100001000	8043.5
moldavia7	DE_2500_chile	9095	DE_2500_chile	8002.5
moldavia8	DE_2500_chile	8948	00100001000	8230.5
mongolia0	DE_2500_chile	13963	00100001000	12217.5
mongolia1	DE_2500_chile	14641.5	00100001000	13015.5
mongolia2	DE_2500_chile	16752.5	DE_2500_chile	15445.5
mongolia3	DE_2500_chile	14830	DE_2500_chile	13588
mongolia4	Powell	15973	DE_2500_chile	14613
mongolia5	DE_2500_chile	13940	00100001000	12405
mongolia6	DE_2500_chile	17839.5	00100001000	16245
mongolia7	Powell	14596	DE_2500_chile	13406
mongolia8	DE_2500_chile	13626.5	00100001000	11895.5
nepal0	11000000002	19420	00100001000	18490
nepal1	DE_2500_chile	18796	00100001000	17998
nepal2	DE_2500_chile	20307.5	00100001000	19007
nepal20	11000000002	21099.5	00000000000	20376.5
nepal21	DE_2500_chile	21334.5	DE_2500_chile	20512
nepal22	DE_2500_chile	20728.5	DE_2500_chile	19806
nepal23	DE_2500_chile	20485	DE_2500_chile	19715.5
nepal24	DE_2500_chile	21616	DE_2500_chile	20894.5
nepal25	DE_2500_chile	21355	00100001000	20602.5
nepal26	DE_2500_chile	21417	DE_2500_chile	20775.5
nepal27	DE_2500_chile	21463.5	00000000000	20795

Table B.1: Best performing algorithm in median by instance.

nepal28	11000000002	21213	00100001000	20453
nepal3	DE_2500_chile	20416	00100001000	19204
nepal4	1000000000	20251	00100001000	19450
nepal5	11000000002	19464.5	Powell	18144
nepal6	DE_2500_chile	17060.5	00100001000	15610.5
nepal7	DE_2500_chile	19154.5	00100001000	18229.5
nepal8	DE_2500_chile	18049	DE_2500_chile	17132
chileuay0	11000000002	14416	00100001000	13323.5
chileuay1	11000000002	12926.5	00100001000	11491
chileuay2	1000000000	12444.5	00100001000	11077.5
chileuay3	1000000000	12637.5	00100001000	11684
chileuay4	DE_2500_chile	13114.5	DE	12090.5
chileuay5	11000000002	13024	10000000000	12283.5
chileuay6	DE_2500_chile	16221	10000000000	14402
chileuay7	DE_2500_chile	14785	DE_2500_chile	13822
chileuay8	11000000002	13011	00100001000	12031.5
sahara0	11000000002	6580.5	00100001000	5886.5
sahara1	Nelder-Mead	6409.5	00100001000	5881.5
sahara2	Nelder-Mead	6385.5	00000000000	5892
sahara3	Nelder-Mead	6524	00100001000	5921.5
sahara4	Nelder-Mead	6442.5	00100001000	5899.5
sahara5	Nelder-Mead	6449	00100001000	5897.5
sahara6	Nelder-Mead	6441.5	00100001000	5837.5
sahara7	Nelder-Mead	6490	00100001000	5913
sahara8	Nelder-Mead	6451	00100001000	5793
usa0	DE_2500_chile	17709.5	00100001000	15968.5
usa1	DE_2500_chile	19126	00100001000	17735
usa2	11000000002	17351	00000000000	16445
usa3	DE_2500_chile	17565.5	00000000000	16403
usa4	DE_2500_chile	19858	DE_2500_chile	18887.5
usa5	DE_2500_chile	17781	00100001000	16348
usa6	DE_2500_chile	17666	DE_2500_chile	16505
usa7	DE_2500_chile	19450.5	DE_2500_chile	18378
usa8	DE_2500_chile	17483	00100001000	16141.5

B.2 Best Performing Algorithm for the 2% Quantile

Instance	Algorithm 500 evaluations	2% quantile 500 evaluations	Algorithm 2,500 evaluations	2% quantile 2,500 evalutaions
afghanistan0	01000000000	10535.98	00000000000	9378.4
afghanistan1	Powell	11140.56	00100001000	9737.76
afghanistan2	Nelder-Mead	10784.02	00100001000	9534.96
afghanistan3	Powell	11676.4	10000000000	10709.02
afghanistan4	11000000002	10345.16	00000000000	9473.5
afghanistan5	Powell	10081.84	00000000000	9562.38
afghanistan6	Powell	9204.86	00100001000	8365.9

Table B.2: Best performing algorithm at the 2% quantile by instance.

afghanistan7	01000000000	8435.02	00100001000	7586.54
afghanistan8	11000000002	9286.76	00100001000	8342.58
arabia0	Powell	10185.22	Powell	9483.96
arabia1	11000000002	7973.16	DE_2500_chile	7292.9
arabia2	Powell	9801.96	10000000000	8922.34
arabia3	11000000002	12165.5	00000000000	11335.58
arabia4	01000000000	9134.12	00000000000	7914.02
arabia5	01000000000	10059.56	00100001000	8670.52
arabia6	11000000002	11611.56	00100001000	10725.42
arabia7	11000000002	8906.8	00000000000	7749.96
arabia8	11000000002	11071.24	00000000000	9734.66
argentina0	Powell	5923.04	00000000000	5566.4
argentina1	Powell	6220.8	00100001000	5609.1
argentina2	Powell	5994.84	Powell	5650.72
argentina3	Powell	5981.58	Powell	5700.12
argentina4	Powell	5900.36	Powell	5553.04
argentina5	Powell	6205.68	10000000000	5725.74
argentina6	Powell	6006.68	Powell	5605.96
argentina7	Powell	5846.72	Powell	5662.14
argentina8	11000000002	6323.78	00000000000	5790.2
australia0	01000000000	7481.16	00100001000	6675.94
australia1	Powell	6123.5	00000000000	5839.82
australia2	Powell	6139.46	Powell	5627.84
australia3	Powell	5873.02	Powell	5590.92
australia4	Powell	9427.52	10000000000	8526.82
australia5	Nelder-Mead	5980.94	00100001000	5711.86
australia6	Nelder-Mead	6340.88	Powell	5797.16
australia7	Powell	9613.76	Powell	9070.56
australia8	Powell	5911.5	Powell	5476.18
brasil0	Powell	6817.6	00100001000	6068.5
brasil1	11000000002	6564.96	10000000000	6023.58
brasil2	Powell	6765.48	00000000000	6225
brasil3	Powell	7116.44	Powell	6537.78
brasil4	Powell	6033.98	Powell	5639.7
brasil5	Powell	6951.18	01000000000	6380.1
brasil6	11000000002	7889.06	00000000000	7046.08
brasil7	Powell	6649.1	00000000000	6109.56
brasil8	Powell	7022.24	DE_2500_chile	6623.12
canada0	11000000002	15600.64	00100001000	14259.28
canada1	Powell	19155.64	Powell	18832.1
canada2	11000000002	20648.84	DE_2500_chile	19917.06
canada3	DE_2500_chile	20261.58	00100001000	19273.44
canada4	01000000000	19185.5	DE_2500_chile	18641.62
canada5	DE_2500_chile	17735.06	00100001000	16936.26
canada6	Powell	20317.92	Powell	19487.16
canada7	Powell	17117.12	Powell	16360.76
canada8	Powell	18056.76	Powell	17373.44

Table B.2: Best performing algorithm at the 2% quantile by instance.

china0	DE_2500_chile	18706.78	00000000000	17714.36
china1	01000000000	18978.5	00000000000	17638.3
china2	11000000002	19335.6	00000000000	18281.68
china3	DE_2500_chile	18623.54	00000000000	17826.56
china4	Powell	18437.9	00000000000	17844.98
china5	DE_2500_chile	19508.98	00100001000	18258
china6	DE_2500_chile	19026.56	DE_2500_chile	18221.22
china7	11000000002	18588.72	00100001000	17732.9
china8	DE_2500_chile	18852.58	DE_2500_chile	18004.16
congo0	Powell	6478.4	Powell	5982.38
congo1	Powell	6905.74	00100001000	6098.52
congo2	Powell	6514.38	00100001000	5913.14
congo3	Powell	6264.46	00100001000	6088.9
congo4	Powell	6319	00000000000	5890.4
congo5	11000000002	6619.58	00100001000	6059.02
congo6	Powell	6859.86	00000000000	6128.18
congo7	11000000002	6629.62	00100001000	5932.16
congo8	Powell	6688.06	00000000000	6172.28
estonia0	Powell	7972.42	Powell	7272.56
estonia1	Powell	8280.72	11000000002	7567.58
estonia2	11000000002	7228.74	Powell	6632.3
estonia3	11000000002	7725.58	00000000000	7204.08
estonia4	Powell	6871.58	00100001000	6297.42
estonia5	DE_2500_chile	9530.38	DE_2500_chile	8700.02
estonia6	01000000000	7265.92	01000000000	6858.76
estonia7	DE_2500_chile	8141.5	00100001000	7143.42
estonia8	01000000000	9307.04	10000000000	8026.2
france0	Powell	8865.18	00100001000	7917.48
france1	01000000000	11100.06	10000000000	10544.68
france2	Powell	10059.5	00100001000	9481.22
france3	01000000000	8289.28	Powell	7674.72
france4	01000000000	13959.3	00100001000	13125.54
france5	DE_2500_chile	12581.38	DE_2500_chile	11103.04
france6	Powell	11067.34	00100001000	10424.74
france7	Powell	11805.18	Powell	11057.82
france8	11000000002	9909.16	00100001000	9030.04
moldavia0	Powell	9763.26	DE_2500_chile	8838.36
moldavia1	Powell	10591.88	Powell	9697.8
moldavia2	Powell	8263.46	00100001000	7572.12
moldavia3	11000000002	9071.12	10000000000	8459.28
moldavia4	DE_2500_chile	11481.26	00100001000	10770.54
moldavia5	Powell	8548.5	Powell	8082.12
moldavia6	11000000002	8367.78	00000000000	7474.44
moldavia7	Powell	7747.6	Powell	7195.4
moldavia8	DE_2500_chile	8000.32	DE_2500_chile	7607.96
mongolia0	Powell	12591	00100001000	11404.6
mongolia1	DE_2500_chile	12997.76	00100001000	11719.3

Table B.2: Best performing algorithm at the 2% quantile by instance.

mongolia2	DE_2500_chile	15560.34	DE_2500_chile	14422.3
mongolia3	DE_2500_chile	13893.7	00100001000	12746.66
mongolia4	Powell	14541.4	00000000000	12912.2
mongolia5	Powell	12380.04	00100001000	11540.62
mongolia6	Powell	16450.28	00000000000	15109.88
mongolia7	Powell	13337.04	Powell	12598.88
mongolia8	Powell	12288	00100001000	11225.88
nepal0	11000000002	18685.04	10000000000	17922.72
nepal1	11000000002	17782.46	00100001000	17372.42
nepal2	DE_2500_chile	19214.68	00100001000	18188.12
nepal20	DE_2500_chile	20022.44	10000000000	19478.68
nepal21	DE_2500_chile	20641.92	00100001000	19923.64
nepal22	01000000000	20083.44	00100001000	19166.56
nepal23	Powell	19246.18	DE_2500_chile	18519.04
nepal24	11000000002	20692.14	DE_2500_chile	20233.96
nepal25	11000000002	20416.8	DE_2500_chile	20031
nepal26	Powell	20809.92	00100001000	20027
nepal27	Powell	20685.62	00100001000	19673.78
nepal28	Powell	20379.3	Powell	19886.64
nepal3	01000000000	19201.32	00000000000	18286.14
nepal4	01000000000	19619.16	00100001000	18622.08
nepal5	11000000002	17779.64	00100001000	16936.78
nepal6	DE_2500_chile	15839.98	00000000000	14765.72
nepal7	01000000000	18059.36	00000000000	17349.1
nepal8	DE_2500_chile	17095.82	DE_2500_chile	16058.12
chileuay0	01000000000	13178.54	01000000000	11713.46
chileuay1	01000000000	11512.44	10000000000	10518.42
chileuay2	11000000002	11163.24	00100001000	10226.76
chileuay3	DE_2500_chile	11570.12	10000000000	10771.38
chileuay4	DE_2500_chile	11875.4	00100001000	10601.76
chileuay5	Powell	12450.76	11000000002	11724.32
chileuay6	01000000000	14295.9	10000000000	13216.54
chileuay7	DE_2500_chile	13735.3	DE_2500_chile	13007.14
chileuay8	11000000002	11793.08	00000000000	10958.66
sahara0	Powell	5971.98	00100001000	5375.32
sahara1	Nelder-Mead	5884.46	00100001000	5497.94
sahara2	Nelder-Mead	5857.72	00100001000	5621.32
sahara3	Nelder-Mead	5963.54	00100001000	5673.98
sahara4	Powell	5979.68	Powell	5560.56
sahara5	Nelder-Mead	5971.02	00100001000	5656.96
sahara6	Nelder-Mead	6021.56	00100001000	5499.1
sahara7	Powell	5848.82	Powell	5403.86
sahara8	Nelder-Mead	6035.76	00100001000	5395.6
usa0	Powell	15862.6	00100001000	15028.94
usa1	Powell	17399.8	00100001000	16433.56
usa2	Powell	16377.24	Powell	15582.3
usa3	Powell	15832.56	00100001000	15187.36

Table B.2: Best performing algorithm at the 2% quantile by instance.

usa4	DE_2500_chile	19019.76	00000000000	18010.68
usa5	11000000002	16266.48	00000000000	15371.74
usa6	01000000000	16178.24	10000000000	15244.48
usa7	Powell	18260.84	Powell	17424.56
usa8	Powell	16228.66	00100001000	15212.4

Appendix C

Radar Network Configuration Contest

C.1 Radar Network Configuration Contest Results

The table below presents the best result found for each contestant.

Table C.1: Best objective value found by contestants.

	Objective value
Contestant #1	16,585
Contestant #2	16,425
Contestant #3	16,839
Contestant #4	18,513
Contestant #5	17,731
Contestant #6	17,495
Contestant #7	17,945
Contestant #8	19,024
Contestant #9	16,430
Contestant #10	16,756
Contestant #11	18,282
Contestant #12	16,779
Contestant #13	18,785

C.2 Radar Network Configuration Contest Poster

Figure C.1: Poster of the radar network configuration contest.

Bibliography

[ABH11] A. Auger, D. Brockhoff, and N. Hansen. Mirrored sampling in evolution strategies with weighted recombination. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '11*, pages 861–868. ACM, 2011.

[AH05] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *IEEE Congress on Evolutionary Computation, CEC 2005*, pages 1769–1776. IEEE, 2005.

[AH21] Y. Akimoto and N. Hansen. Cma-es and advanced adaptation mechanisms. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21 (Companion)*, page 636–663. ACM, 2021.

[AJT05] A. Auger, M. Jebalia, and O. Teytaud. Algorithms (x, sigma, eta): Quasi-random mutations for evolution strategies. In *Artificial Evolution, 7th International Conference, Evolution Artificielle, EA 2005, October 26-28, 2005, Revised Selected Papers*, volume 3871 of *Lecture Notes in Computer Science*, pages 296–307. Springer, 2005.

[BAH+10] D. Brockhoff, A. Auger, N. Hansen, D. V. Arnold, and T. Hohm. Mirrored sampling and sequential selection for evolution strategies. In *Parallel Problem Solving from Nature - PPSN XI, 11th International Conference, September 11-15, 2010, Proceedings, Part I*, volume 6238 of *Lecture Notes in Computer Science*, pages 11–21. Springer, 2010.

[BDB+20] T. Bartz-Beielstein, C. Doerr, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P/ Kerschke, M. López-Ibáñez, K.M. Malan, J.H. Moore, B. Naujoks, P. Orzechowski, V. Volz, M. Wagner, and T. Weise. Benchmarking in optimization: Best practice and open issues. *CoRR*, abs/2007.03488, 2020.

[BDSS16] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer. Surrogate assisted feature computation for continuous problems. In *Learning and Intelligent Optimization - 10th International Conference, LION 10. Revised Selected Papers*, pages 17–31. Springer, 2016.

[BDSS17] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer. Per instance algorithm configuration of CMA-ES with limited budget. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17*, pages 681–688. ACM, 2017.

[Bel17] N. Belkhir. *Per Instance Algorithm Configuration for Continuous Black Box Optimization*. phdthesis, Université Paris-Saclay, November 2017.

[BG02] B. Beachkofski and R. Grandhi. Improved Distributed Hypercube Sampling. In *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. American Institute of Aeronautics and Astronautics, 2002.

[BMTP12] B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß. Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '12*, pages 313–320. ACM, 2012.

[Bre71] R. P. Brent. An algorithm with guaranteed convergence for finding a zero of a function. *The Computer Journal*, 14(4):422–425, 01 1971.

[Bri17] Y. Briheche. *Optimization of search patterns for fixed-panel tridimensional scanning radars*. Theses, École centrale de Nantes, November 2017.

[BSF+21] M. Böther, L. Schiller, P. Fischbeck, L. Molitor, M.S. Krejca, and T. Friedrich. Evolutionary minimization of traffic congestion. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21*, pages 937–945. ACM, 2021.

[BSPV02] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '02*, pages 11–18, 2002.

[Car19] R. Cariou. *Le traitement du signal radar*. Dunod, March 2019.

[CDL+21] R. Cosson, B. Derbel, A. Liefooghe, H.E. Aguirre, K. Tanaka, and Q. Zhang. Decomposition-based multi-objective landscape features and automated algorithm selection. In *Evolutionary Computation in Combinatorial Optimization - 21st European Conference, EvoCOP 2021, Held as Part of EvoStar 2021, Proceedings*, volume 12692 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2021.

[COLT11] X. Chen, Y-S. Ong, M-H Lim, and K.C. Tan. A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, 15(5):591–607, 2011.

[DLV+19] B. Derbel, A. Liefooghe, S. Vérel, H. Aguirre, and K. Tanaka. New features for continuous exploratory landscape analysis based on the SOO tree. In *Proceedings of Foundations of Genetic Algorithms (FOGA) '19*, pages 72–86. ACM, 2019.

[DLV+21] J. Dreo, A. Liefooghe, S. Verel, M. Schoenauer, J.J. Merelo, A. Quemy, B. Bouvier, and J. Gmys. Paradiseo: From a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of paradiseo. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21, (Companion)*, page 1522–1530. ACM, 2021.

[DP10] J. Dick and F. Pillichshammer. *Digital Nets and Sequences*. Cambridge University Press, 2010.

[EC21] A.P. Engelbrecht and C.W Cleghorn. Recent advances in particle swarm optimization analysis and understanding 2021. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21 (Companion)*, page 341–368. ACM, 2021.

[FHRA10] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-Parameter Black-Box Optimization Benchmarking 2010: Presentation of the Noiseless Functions. http://coco.gforge.inria.fr/downloads/download16.00/bbobdocfunctions.pdf, 2010.

[Fle87] R. Fletcher. *Practical Methods of Optimization; (2nd Ed.)*. Wiley-Interscience, USA, 1987.

[Fog98] D.B. Fogel. *Artificial Intelligence through Simulated Evolution*, pages 227–296. 1998.

[Hal64] J. H. Halton. Algorithm 247: Radical-inverse quasi-random point sequence. *Commun. ACM*, 7(12):701–702, dec 1964.

[Han08] N. Hansen. CMA-ES with two-point step-size adaptation. *CoRR*, abs/0805.0231, 2008.

[Han09] N. Hansen. Benchmarking a bi-population CMA-ES on the BBOB-2009 function testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '09 (Companion)*, pages 2389–2396. ACM, 2009.

[Han16] N. Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016.

[HAR+21] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tusar, and D. Brockhoff. COCO: a platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[HE03] G. Hamerly and C. Elkan. Learning the k in k-means. In *Advances in Neural Information Processing Systems 16 Neural Information Processing Systems, NIPS*, pages 281–288. MIT Press, 2003.

[HHL11] F. Hutter, H.H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization - 5th International Conference, LION 5. Selected Papers*, pages 507–523, 2011.

[HHLS09] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. Paramils: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[HKP+01] T. Hennig, J. Kretsch, C. Pessagno, P. Salamonowicz, and W. Stein. The shuttle radar topography mission. In *Proceedings of the First International Symposium on Digital Earth Moving*, DEM '01, page 65–77. Springer, 2001.

[HKV19] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019.

[HO01] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[Hol73] J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal of Computing*, 2:88–105, 1973.

[JA06] G.A. Jastrebski and D.V. Arnold. Improving evolution strategies through active covariance matrix adaptation. In *IEEE International Conference on Evolutionary Computation, CEC 2006, part of WCCI 2006*, pages 2814–2821. IEEE, 2006.

[JD20] A. Jankovic and C. Doerr. Landscape-aware fixed-budget performance regression for modular cma-es variants. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '20*, 2020. To appear.

[JED21] A. Jankovic, T. Eftimov, and C. Doerr. Towards feature-based performance regression using trajectory data. In *Applications of Evolutionary Computation - 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Proceedings*, volume 12694 of *Lecture Notes in Computer Science*, pages 601–617. Springer, 2021.

[JF95] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, 1995.

[JG98] D.N Joanes and C.A. Gill. Comparing measures of sample skewness and kurtosis. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 47(1):183–189, 1998.

[JPED21] A. Jankovic, G. Popovski, T. Eftimov, and C. Doerr. The impact of hyper-parameter tuning for landscape-aware performance regression and algorithm selection. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21*, pages 687–696. ACM, 2021.

[JRNG08] A. Jarvis, H.I. Reuter, A. Nelson, and E. Guevara. Hole-filled seamless SRTM data V4. *International Centre for Tropical Agriculture (CIAT)*, 2008.

[KE95] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[Ken99] J. Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. *Proceedings of the 1999 Congress on Evolutionary Computation-CEC 1999*, 3:1931–1938 Vol. 3, 1999.

[KHNT19] P. Kerschke, H.H. Hoos, F. Neumann, and H. Trautmann. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation*, 27(1):3–45, March 2019.

[Kie53] J. Kiefer. Sequential minimax search for a maximum. *Proceedings of the American Mathematical Society*, 4(3):502–506, 1953.

[Kim68] N. Kimura. Evolutionary Rate at the Molecular Level. *Nature*, 217(5129):624–626, February 1968.

[KKB+18] P. Kerschke, L. Kotthoff, J. Bossek, H.H. Hoos, and H. Trautmann. Leveraging TSP solver complementarity through machine learning. *Evolutionary Computation*, 26(4), 2018.

[KMST10] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. ISAC - instance-specific algorithm configuration. In *Proceedings of ECAI 2010 - 19th European Conference on Artificial Intelligence*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 751–756. IOS Press, 2010.

[Knu98] D. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley, 1998.

[Koz94] J.R. Koza. *Genetic programming 2 - automatic discovery of reusable programs*. Complex adaptive systems. MIT Press, 1994.

[KPH+14] P. Kerschke, M. Preuss, C. Hernández, O. Schütze, J-Q. Sun, C. Grimme, G. Rudolph, B. Bischl, and H. Trautmann. Cell Mapping Techniques for Exploratory Landscape Analysis. In *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, volume 288, pages 115–131. Springer, 2014.

[KPWT15] P. Kerschke, M. Preuss, S. Wessing, and H. Trautmann. Detecting Funnel Structures by Means of Exploratory Landscape Analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '15*, pages 265–272. ACM, 2015.

[KPWT16] P. Kerschke, M. Preuss, S. Wessing, and H. Trautmann. Low-Budget Exploratory Landscape Analysis on Multiple Peaks Models. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '16*, pages 229–236. ACM, 2016.

[KT19a] P. Kerschke and H. Trautmann. Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary Computation*, 27(1):99–127, 2019.

[KT19b] P. Kerschke and H. Trautmann. Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the r-package flacco. In *Applications in Statistical Computing: From Music Data Analysis to Industrial Quality Improvement*, pages 93–123. Springer, 2019.

[LDLP+16] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützl. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43 – 58, 2016.

[LHHS15] M. Lindauer, H.H. Hoos, F. Hutter, and T. Schaub. Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015.

[LJD+17] L. Li, K.G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research.*, 18:185:1–185:52, 2017.

[LPS10] M. López-Ibáñez, L. Paquete, and T. Stützle. Exploratory analysis of stochastic local search algorithms in biobjective optimization. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 209–222. Springer, 2010.

[LW06] M. Lunacek and D. Whitley. The dispersion metric and the CMA evolution strategy. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '06*, 2006.

[Mal21] K.M. Malan. A survey of advances in landscape analysis for optimisation. *Algorithms*, 14(2):40, 2021.

[Mat09] J. Matoušek. *Geometric Discrepancy*. Springer, 2nd edition, 2009.

[MBC79] M. D. McKay, R. J. Beckman, and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239–245, 1979.

[MBT+11] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, and G. Rudolph. Exploratory Landscape Analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '11*, pages 829–836. ACM, 2011.

[ME13] K. Malan and A.P. Engelbrecht. A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163, 2013.

[MF04] Z. Michalewicz and D.B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2 edition, 2004.

[MG14] R. Morgan and M. Gallagher. Sampling Techniques and Distance Metrics in High Dimensional Continuous Landscape Analysis: Limitations and Improvements. *IEEE Transactions on Evolutionary Computation*, 18(3):456–461, June 2014.

[Mir18] L.J.V. Miranda. Pyswarms: a research toolkit for particle swarm optimization in python. *Journal of Open Source Software*, 3(21):433, 2018.

[MKH15] M.A. Muñoz, 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, February 2015.

[MN98] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

[MS20] M.A. Muñoz and K. Smith-Miles. Generating new space-filling test instances for continuous black-box optimization. *Evolutionary Computation*, 28(3):379–404, 2020.

[MS21] N. S. C. Merleau and M. Smerlak. A simple evolutionary algorithm guided by local mutations for an efficient RNA design. In *Genetic and Evolutionary Computation Conference, GECCO '21*, pages 1027–1034. ACM, 2021.

[MSKH15] Mario A. Muñoz, Yuan Sun, Michael Kirley, and Saman K. Halgamuge. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences*, 317:224–245, 2015.

[Mun11] R. Munos. Optimistic optimization of a deterministic function without the knowledge of its smoothness. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011.*, pages 783–791, 2011.

[NM65] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 01 1965.

[OGH94] A. Ostermeier, A. Gawelczyk, and N. Hansen. Step-size adaption based on non-local use of selection information. In *Parallel Problem Solving from Nature - PPSN III, International Conference on Evolutionary Computation.*, volume 866 of *Lecture Notes in Computer Science*, pages 189–198. Springer, 1994.

[OTVD08] G. Ochoa, M. Tomassini, S. Vérel, and C. Darabos. A study of NK landscapes' basins and local optima networks. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '08*, pages 555–562. ACM, 2008.

[PA12] E. Pitzer and M. Affenzeller. A comprehensive survey on fitness landscape analysis. In *Recent Advances in Intelligent Engineering Systems*, volume 378 of *Studies in Computational Intelligence*, pages 161–191. Springer, 2012.

[Par62] E. Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065 – 1076, 1962.

[Pea01] K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[PER+15] A. Piad-Morffis, S. Estevez-Velarde, A. Bolufé Röhler, J. Montgomery, and S. Chen. Evolution strategies with thresheld convergence. In *IEEE Congress on Evolutionary Computation, CEC 2015*, pages 2097–2104. IEEE, 2015.

[PK20] D. Pulatov and L. Kotthoff. Opening the black box: Automatically characterizing software for algorithm selection (student abstract). In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020*, pages 13899–13900. AAAI Press, 2020.

[Pow64] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 01 1964.

[PRH19] Z. Pitra, J. Repický, and M. Holena. Landscape analysis of Gaussian process surrogates for the covariance matrix adaptation evolution strategy. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19*, pages 691–699, 2019.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RDDD19] Q. Renau, J. Dréo, C. Doerr, and B. Doerr. Expressiveness and robustness of landscape features. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19 (Companion)*, pages 2048–2051. ACM, 2019.

[RDDD20a] Q. Renau, C. Doerr, J. Dreo, and B. Doerr. Experimental Data Set for the study "Exploratory Landscape Analysis is Strongly Sensitive to the Sampling Strategy", June 2020.

[RDDD20b] Q. Renau, C. Doerr, J. Dréo, and B. Doerr. Exploratory landscape analysis is strongly sensitive to the sampling strategy. In *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part II*, volume 12270 of *Lecture Notes in Computer Science*, pages 139–153. Springer, 2020.

[RDDD21a] Q. Renau, J. Dréo, C. Doerr, and B. Doerr. Towards explainable exploratory landscape analysis: Extreme feature selection for classifying BBOB functions. In *Applications of Evolutionary Computation - 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Proceedings*, volume 12694 of *Lecture Notes in Computer Science*, pages 17–33. Springer, 2021.

[RDDD21b] Q. Renau, Johann Dreo, C. Doerr, and B. Doerr. Exploratory Landscape Analysis Feature Values for the 24 Noiseless BBOB Functions, 2021.

[Ric76] J.R. Rice. The algorithm selection problem. *Advance Computing*, 15:65–118, 1976.

[RTE65] I. Rechenberg, B.F. Toms, and Royal Aircraft Establishment. *Cybernetic Solution Path of an Experimental Problem:*. Library translation / Royal Aircraft Establishment. Ministry of Aviation, 1965.

[RY21] A. Bolufé Röhler and Y. Yuan. Machine learning for determining the transition point in hybrid metaheuristics. In *IEEE Congress on Evolutionary Computation, CEC 2021*, pages 1115–1122. IEEE, 2021.

[Sch65] H-P Schwefel. Kybernetische evolution als strategie der experimentellen forschung in der stromungstechnik. *Diploma thesis, Technical Univ. of Berlin*, 1965.

[SE98] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *IEEE International Conference on Evolutionary Computation Proceedings 1998*, pages 69–73, 1998.

[SEK20] U. Skvorc, T. Eftimov, and P. Korosec. Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis. *Applied Soft Computing*, 90:106138, 2020.

[SHH62] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4(4):441–461, 1962.

[Sko01] M.I. Skolnik. *Introduction to radar systems*. McGraw-Hill international editions. Electrical engineering series. McGraw-Hill, Boston, [Mass.] ;, 3rd ed. edition, 2001.

[SL21] T. Stützle and M. López-Ibáñez. Automated algorithm configuration and design. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21*, pages 959–982. ACM, 2021.

[SM09] K. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1), January 2009.

[Sob67] I.M. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, January 1967.

[SP97] R. Storn and K.V. Price. Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[ST12] K. Smith-Miles and T.T. Tan. Measuring algorithm footprints in instance space. In *IEEE Congress on Evolutionary Computation, CEC 2012*, pages 1–8. IEEE, 2012.

[Sta02] P.F. Stadler. Fitness landscapes. In *Biological Evolution and Statistical Physics*, volume 585, pages 183–204. Springer, 2002.

[Swe54] P. Swerling. *Probability of Detection for Fluctuating Targets*. RAND Corporation, 1954.

[SWN03] T.J. Santner, B.J. Williams, and W.I. Notz. *The Design and Analysis of Computer Experiments*. Springer Series in Statistics. Springer, 2003.

[Tal13] E.G. Talbi. *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*. Springer, 2013.

[Tan21a] R. Tanabe. Benchmarking feature-based algorithm selection systems for black-box numerical optimization. *CoRR*, abs/2109.08377, 2021.

[Tan21b] R. Tanabe. Towards exploratory landscape analysis for large-scale optimization: a dimensionality reduction framework. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21*, pages 546–555. ACM, 2021.

[VFM00] V. K. Vassilev, T. C. Fogarty, and J. F. Miller. Information characteristics and the structure of landscapes. *Evolutionary Computation*, 8(1):31–60, 2000.

[VGO+20] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C J Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[VHCM15] M. Vallati, F. Hutter, L. Chrpa, and T. McCluskey. On the effective configuration of planning domain models. In *ProcProceedings of International Joint Conferences on Artificial Intelligence (IJCAI) '15*. AAAI, 2015.

[vRWvLB16] S. van Rijn, H. Wang, M. van Leeuwen, and T. Bäck. Evolving the structure of evolution strategies. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*, pages 1–8. IEEE, 2016.

[VvRBD19] D. Vermetten, S. van Rijn, T. Bäck, and C. Doerr. Online selection of CMA-ES variants. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19*, pages 951–959. ACM, 2019.

[VWBD20] D. Vermetten, H. Wang, T. Bäck, and C. Doerr. Towards dynamic algorithm selection for numerical black-box optimization: investigating BBOB as a use case. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '20*, pages 654–662. ACM, 2020.

[WEB14] H. Wang, M. Emmerich, and T. Bäck. Mirrored orthogonal sampling with pairwise selection in evolution strategies. In *Symposium on Applied Computing, SAC 2014*, pages 154–156. ACM, 2014.

[Wei90] E. Weinberger. Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference. *Biological Cybernetics*, 63:325–336, September 1990.

[Wri32] S. Wright. *The roles of mutation, inbreeding, crossbreeding, and selection in evolution*, volume 1. 1932.

[XHHL11] L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *CoRR*, abs/1111.2249, 2011.

[XHHLB08] L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

[Zah06] D. Zaharie. A comparative analysis of crossover variants in differential evolution. pages 171–181, 05 2006.

DOCTORAL SCHOOL

Titre : Sélection de Métaheuriques Guidée par le Paysage de Recherche pour l'Optimisation de Réseaux de Radars

Mots clés : Optimisation, Métaheuristiques, Sélection Automatique d'Algorithmes, Réseaux de Radars

Résumé : Les réseaux de radars sont des systèmes complexes dont l'efficacité doit être optimisée. L'optimisation du réseau se fait par la maximisation de la couverture radar ou par la maximisation de la probabilité de détection d'une cible. Configurer un réseau de radars est une tâche complexe souvent réalisée par des experts à l'aide de simulateurs.
La résolution de ce type de problème s'appuie sur l'optimisation boîte noire. De nombreuses métaheuristiques ont été développés dans le but de résoudre des problèmes d'optimisation boîte noire. Ces métaheuristiques ont montré une certaine complémentarité dans leur performance qui est liée à la structure du problème à résoudre. Choisir l'algorithme le plus approprié pour résoudre un problème est donc crucial.
L'objectif de cette thèse CIFRE est de réaliser une sélection automatique de métaheuristiques guidée par le paysage de recherche dans le but d'optimiser un problème de configuration de réseau de radars. Les contributions de cette thèse sont doubles. Dans un premier temps, nous avons défini six propriétés que les mesures caractérisant un problème d'optimisation boîte noire devraient satisfaire. Nous avons également étudié à quel point les mesures satisfont ces propriétés. L'une de ces propriétés est la sensibilité à la méthode d'échantillonnage. Contrairement aux préconisations dans la littérature, nous avons remarqué que la méthode d'échantillonnage importait car une grande partie des mesures y est sensible. Nous avons trouvé d'importantes différences entre les distributions des mesures provenant de différentes méthodes d'échantillonnage. Globalement, aucune des mesures ne satisfait les six propriétés.
La résolution des problèmes radar avec différentes métaheuristiques montre ces dernières ont des performances similaires. Le gain d'une sélection automatique de métaheuristiques est par conséquent faible. Les performances de notre sélection automatique de métaheuristiques guidée par le paysage de recherche sont similaires aux performances du meilleur algorithme sur les instances d'entraînement (SBS).

Title : Landscape-Aware Selection of Metaheuristics for the Optimization of Radar Networks

Keywords : Optimization, Metaheuristics, Automated Algorithm Selection, Radar Networks

Abstract : Radar networks are complex systems that need to be configured to maximize their coverage or the probability of detection of a target. The optimization of radar networks is a challenging task that is typically performed by experts with the help of simulators. Alternatively, black-box optimization algorithms can be used to solve these complex problems. Many heuristic algorithms were developed to solve black-box optimization problems and these algorithms exhibit complementarity of performance depending on the structure of the problem. Therefore, selecting the appropriate algorithm is a crucial task.
The objective of this CIFRE PhD is to perform a landscape-aware algorithm selection of metaheuristics in order to optimize radar networks. The main contributions of this PhD thesis are twofold. In this thesis, we define six properties that landscape features should satisfy and we study to what degree landscape features satisfy these properties. One of the six properties is the invariance to the sampling strategy. We found that, surprisingly to what was recommended in the literature, the sampling strategy actually matters. We found important discrepancies in the feature values computed from different sampling strategies. Overall, we found that none of the features satisfy all defined properties. These features represent the core of a landscape-aware algorithm selection. We applied the landscape-aware algorithm selection of metaheuristics on the optimization of radar network use-cases. On this use-cases, algorithms have similar performances and the gain to perform an automated selection of algorithms is small. Nevertheless, the performance of the landscape-aware algorithm selection of metaheuristics is similar to the performance of the single best solver (SBS).

Institut Polytechnique de Paris
91120 Palaiseau, France