Per Instance Algorithm Configuration for Continuous Black Box Optimization

Nacim Belkhir

NNT : 2017SACLS455

THESE DE DOCTORAT
DE
L'UNIVERSITÉ PARIS SACLAY
PRÉPARÉ À L'UNIVERSITÉ PARIS-SUD

Ecole doctorale n°580
Sciences et technologies de l'information et de la communication
Spécialité de doctorat: Informatique

Par

Nacim Belkhir

Per Instance Algorithm Configuration for Continuous Black Box Optimization

Thèse présentée et soutenue à Gif-sur-Yvette, le 20/11/2017:
Composition du Jury:

M	SÉBASTIEN VEREL	Professeur Université du Littoral Côte d'Opale	Président
M	FRÉDÉRIC SAUBION	Professeur Université d'Angers	Rapporteur
M	THOMAS STÜTZLE	Maître de Recherches Université Libre de Bruxelles (ULB)	Rapporteur
M	AMIR NAKIB	Maître de Conférence Université Paris-Est Créteil	Examinateur
M	EMMANUEL VAZQUEZ	Maître de Conférence Centrale-Supelec	Examinateur
M	MARC SCHOENAUER	Directeur de Recherches Inria Saclay, LRI/TAO, France	Directeur de thèse
M	JOHANN DRÉO	Ingénieur-Chercheur Thales Research & Technology	Co-encadrant de Thèse
M	PIERRE SAVÉANT	Ingénieur-Chercheur Thales Research & Technology	Invité

Abstract

Evolutionary Algorithms (EAs) have been widely investigated in the literature and in many industrial contexts to solve black box optimization problems. They have been demonstrated to be able to solve a wide range of optimization problems. However, despite many successful results, it is widely acknowledged in the optimization community that the quest of a general algorithm that would solve any optimization is vain.

This PhD thesis focuses on the automated algorithm configuration that aims at finding the 'best' parameter setting for a given problem or a class of problem, where the notion of 'best' is related to some user-defined performance measure, usually some balance between the computational cost of the optimization and a domain-specific measure of quality (e.g., the precision of the solution in the continuous domain). The Algorithm Configuration problem thus amounts to a meta-optimization problem in the space of parameters, whose meta-objective is the performance measure of the given algorithm at hand with a given parameter configuration. However, in the continuous domain, such method can only be empirically assessed at the cost of running the algorithm on some problem instances, hence making the task of a direct meta-optimization immensely costly, either for one or a set of problems.

More recent approaches rely on a description of the objective functions in some *features space*, and try to learn a mapping from this feature space onto the space of parameter configurations of the algorithm at hand, based on examples of the behavior of several configurations on a training set of objective functions. Along these lines, this PhD thesis focuses on the Per Instance Algorithm Configuration (PIAC) for solving continuous black box optimization problems, where only a limited budget of function evaluations is available.

We first survey Evolutionary Algorithms for continuous optimization, with a focus on two algorithms that we have used as target algorithm for PIAC, DE and CMA-ES. Next, we review the state of the art of Algorithm Configuration approaches, and the different features that have been proposed in the literature to describe continuous black box optimization problems.

Because, in the continuous domain, features are computed from a sample of objective function values, we investigate their computation when only a small budget is available, and we propose a novel methodology based on surrogate modelling in order to artificially augment the sample set. The resulting features allow to reduce the computation of the sub-sampled features and slightly improve the efficiency of the features for solving a classification task of optimization problem.

We then introduce a general methodology to empirically study PIAC for the continuous domain, so that all the components of PIAC can be explored in real-world conditions. To this end, we also introduce a new continuous black box test bench, distinct from the famous BBOB benchmark, that is composed of a several multi-dimensional test functions with different problem properties, gathered from the literature.

The methodology is finally applied to two EAs. First we use Differential Evolution as target algorithm, and explore all the components of PIAC, such that we empirically assess the best. Second, based on the results on DE, we empirically investigate PIAC with Covariance Matrix Adaptation Evolution Strategy (CMA-ES) as target algorithm. Both use cases empirically validate the proposed methodology on the new black box testbench for dimensions up to 100.

Résumé

Les Algorithmes évolutifs (AEs) ont été largement étudiés dans la littérature et dans de nombreux contextes industriels afin de résoudre des problèmes d'optimisation boîte noire. Ils ont démontré qu'ils étaient capables de résoudre un large éventail de problèmes d'optimisation. Cependant, malgré de nombreux résultats positifs, il est largement reconnu dans la communauté de l'optimisation que la quête d'un algorithme général qui résoudrait tous les problèmes d'optimisation est vaine.

Cette thèse porte sur la configuration automatisée des algorithmes qui vise à trouver le meilleur paramètrage à pour un problème donné ou une catégorie de problèmes, où la notion de « meilleur » est liée à une mesure de performance définie par l'utilisateur, généralement un équilibre entre le coût de calcul de l'optimisation et une mesure de qualité spécifique au domaine (par exemple, la précision de la solution dans le domaine continu). Le problème de configuration de l'algorithme revient donc à un problème de méta-optimisation dans l'espace des paramètres, dont le méta-objectif est la mesure de performance de l'algorithme donné avec une configuration de paramètres donnée. Cependant, dans le domaine continu, une telle méthode ne peut être évaluée empiriquement qu'au prix de l'exécution de l'algorithme sur certaines instances problématiques, ce qui rend la tâche d'une méta-optimisation directe extrêmement coûteuse, soit pour un problème soit pour un ensemble de problèmes.

Des approches plus récentes reposent sur une description des fonctions objectives dans certains em features space, et essayent d'apprendre une cartographie à partir de cet espace caractéristique sur l'espace des configurations de paramètres de l'algorithme en question, basé sur des exemples de comportement de plusieurs configurations sur un ensemble d'apprentissage de fonctions objectives. Dans le même ordre d'idée, cette thèse de doctorat porte sur le CAPI (Configuration d'Algorithme Par Instance) pour résoudre des problèmes d'optimisation de boîte noire continus, où seul un budget limité d'évaluations de fonctions est disponible.

Nous étudions d'abord les algorithmes évolutionnaires pour l'optimisation continue, en mettant l'accent sur deux algorithmes que nous avons utilisés comme algorithme cible pour CAPI, DE et CMA-ES. Ensuite, nous passons en revue l'état de l'art des approches de configuration d'algorithme, et les différentes fonctionnalités qui ont été proposées dans la littérature pour décrire les problèmes d'optimisation de boîte noire continue.

Parce que, dans le domaine continu, les caractéristiques sont calculées à partir d'un échantillon de valeurs de fonctions objectives, nous étudions leur calcul quand un petit budget est disponible, et nous proposons une nouvelle méthodologie basée sur la modélisation de substitution pour augmenter artificiellement l'échantillon. Les caractéristiques résultantes permettent de

réduire le calcul des caractéristiques sous-échantillonnées et d'améliorer légèrement l'efficacité des fonctionnalités pour résoudre une tâche de classification de problème d'optimisation.

Nous introduisons ensuite une méthodologie générale pour étudier empiriquement le CAPI pour le domaine continu, de sorte que toutes les composantes du CAPI puissent être explorées dans des conditions réelles. À cette fin, nous introduisons également un nouveau banc d'essai de boîte noire continue, distinct du célèbre benchmark BBOB, qui est composé de plusieurs fonctions de test multidimensionnelles avec différentes propriétés problématiques, issues de la littérature.

La méthodologie proposée est finalement appliquée à deux AEs. Premièrement, nous utilisons Differential Evolution comme algorithme cible, et nous explorons toutes les composantes du PIAC, de manière à ce que nous puissions évaluer empiriquement les meilleurs. Deuxièmement, sur la base des résultats de DE, nous étudions empiriquement le CAPI avec la stratégie d'évolution de l'adaptation de la matrice de covariance (CMA-ES) en tant qu'algorithme cible. Les deux cas d'utilisation valident empiriquement la méthodologie proposée sur le nouveau banc d'essai de la boîte noire pour des dimensions allant jusqu'à 100.

Contents

Part I

Introduction

CHAPTER 1

Introduction

1.1 Motivation

In spite of the end of Moore's Law, the computational power of computers, high performance computing infrastructures or processing units, is still steadily growing. This growth is assessed by a permanent need of computational power for more and more complex real-world tasks, e.g., simulations of natural or artificial systems with increasing accuracy. Furthermore, the optimization of such simulated models adds yet another level of computational complexity. In this context, the use of sophisticated and robust optimization algorithms still remains a challenge. In particular, in many cases, the analytic model is too complex or the problem cannot be expressed by a mathematical formulation, and the problem to be optimized is seen as a black box.

In the literature [Cag+00; Pad12; TR04; CST17] and in many industrial contexts, Evolutionary Algorithms (EAs) have been widely investigated to solve continuous black box optimization problems. EAs are based on the evolution of a population of candidate solutions, and have demonstrated to be well performing on different types of problems, without any prior knowledge of the structures of the objective function (e.g. convexity, multimodality, …) in contrast with well known, classical methods such as gradient based methods.

Historically, EAs are bio-inspired algorithms, that rely on the genetic variation framework. By analogy with real biological evolution, two main types of operators are distinguished: the blind variation operators (mutation and crossover), and the natural selection operator. These operators are themselves controlled by several parameters, which directly influence their behavior and thus the behavior of the algorithm. These parameters are defined by the expert during the initialization, but can also be adapted during the optimization process controlled by other (meta-)parameters.

The choice of the value of these parameters is critical, in particular to address the *exploration* vs *exploitation* tradeoff: The *exploration* of the search space aims at finding yet unexplored regions where better solutions than the current best

might exist; The *exploitation* aims at looking around the most promising regions already discovered, where slightly better solutions might still exist. Too much exploration (e.g., pure random search) will result in a never-converging algorithm, whereas too much exploitation will most probably only discover local optima, missing more promising ones. Unfortunately, there is no analytic way to balance *exploration* and *exploitation*, and, thus, tuning the algorithm parameters is crucial

The different aforementioned aspects directly influence the ability of Evolutionary Algorithms to solve a given optimization problem. EAs are a priori able to tackle a wide range of optimization problems, involving different global structures (e.g. multimodality, plateaus, convexity, separability, …), in low or high dimensional search spaces, with or without noise, with a static or dynamic objective function. Therefore, the Evolutionary Computation community has proposed a wide variety of approaches to tackle such variety of optimization problems. These approaches have been empirically validated on analytical benchmark functions (analytically defined, see for example the CEC test suite [Sug+05] or the BBOB test bench [Han+10]) as well as on a variety of real-world problems [Han09a].

It is widely acknowledged, today, in the optimization community at large, that the issue of designing a general optimization algorithm, that would solve all problems at best, is vain: It was theoretically proved by different works on the *No Free Lunch theorem* [WM97; AT10]. Hence, tackling an unknown optimization problem first amounts to choose the most appropriate algorithm, among a given set of possible algorithms (also known as the *Algorithm Selection* problem), and to choose the best parameter setting for the chosen algorithm (the *Algorithm Configuration* problem).

Focusing on the Algorithm Configuration problem, the choice of the best parameter setting can be considered itself as an optimization process in the space of parameters, thus pertaining to the *Programming by Optimization* (PbO) paradigm [Hoo12]: Given a problem instance (i.e., an objective function), a set of algorithms with domain for their parameters, and a performance measure, the PbO approach aims at finding the best algorithm and/or parameter setting to solve this new problem instance. The performance measure generally involves time-to-solution (CPU time, or number of function evaluations) and precision/accuracy of the solution returned by the algorithm (precision of the solution for continuous optimization problems, number of constraints violated for Constraint Satisfaction problems, …).

But such a meta-optimization problem is in general difficult to solve (mainly due to hierarchical search space, multimodal properties, …), hence requiring to run different algorithms with different parameter settings, where each of these runs will be a full optimization of the objective function, thus calling the objective

function a large number of times. In total, the overall number of calls of the objective function is huge, making such an approach intractable in most real-world problems with costly objective functions.

Nonetheless, another approach to the PbO paradigm is to target entire classes of objective functions: The best algorithm or parameter setting can be learned *offline*, once and for all for a given class of functions (the performance measure has to be modified accordingly), strongly relying on the hypothesis that the optimal setting (algorithm and parameters) can be applied to all members of that class. This kind of approach can be applied in an operational context, where the same type of problems but with slightly different settings, has to be solved recurrently. But, in the general case of black-box optimization problems, little domain knowledge is known (e.g. type and dimension of the search space, possibly relevant variables, …). Such *elementary* characteristics are not sufficient to accurately and reliably guarantee that a problem is part of a given problem class, and even less so to determine the appropriate algorithm and its parameters.

When the "per class" approach to Algorithm Configuration sketched above is often not possible in an operational context, for expensive objective functions, and with few (or none) examples of the same class, another approach might prove efficient, the *Per Instance Algorithm Configuration* (PIAC) [Hut+06]. PIAC aims at using some characteristics of objective functions, aka *features*, that can be computed without any domain knowledge, to derive an *empirical performance model* for a given algorithm, that maps the features and the parameters to the performance of the algorithm. Learning such a model requires a large example base of algorithms performances on known objective functions, but these can be acquired once and forall, offline. When a new problem is to be solved, its features can be computed, and the algorithm parameters that maximizes the empirical performance can be derived.

Well-known successes have been obtained in the SAT domain [Xu+08; XHL10] and in mixed integer programming domain [Xu+11]. These successes are mainly due to the number of features that have been proposed in the literature during many decades to describe SAT and MIP problems, and the effort to understand what makes a SAT or MIP problem hard for an algorithm [Nud+04; Hut09]. Unfortunately, these examples of a successful PIAC approach is to-date quite unique, and appeals for research regarding the design of features in other domains.

By contrast to the SAT or the MIP domain, beyond which the features can be directly derived from the problem description, in the continuous domain (the search space is a subset of $\mathbb{R}^d$ for some d), in particular in a black box context (where the only known information of the problem often is the dimension of the search space d), other methods have been proposed to describe problem prop-

erties. Several works (see [LW06; Mer+11; M+15; MG12] among many others) have proposed many different features in order to try to understand the global properties of continuous problems. However, solving the Algorithm Selection and/or Configuration problem is only a long-term goal in these works.

A large body of mathematical programming algorithms exists, and are proved to be optimal for specific classes of objective functions, e.g. Linear Programming should be used if the objective function and the constraints are linear; gradient-based algorithm should be used when the objective function is convex and differentiable (and well-conditioned). However, in real-world applications, the general Algorithm Selection and Configuration problem remains open. In such a context, the features-based approach seems a promising research direction, in particular regarding the first results obtained in [MKH12; Mer+11; M+15; Bis+12]. Nevertheless, the computation of all proposed features relies on the computation of the value of the objective function for many sample points (generally randomly chosen) in the search space. In real-world situations, where the objective function is expensive and the computational budget limited, the computation of the features as proposed in [Mer+11] might simply be intractable. Along these lines, the contribution presented in this thesis can be divided into two main parts.

1.2 Main Contributions

- In a first part, we introduce a PIAC methodology for the continuous domain, focusing on the computation of problem features to efficiently characterize problems. In particular, we analyze how a degraded accuracy of the feature computation can affect the quality of the *empirical performance model* and hence, the efficiency of the best parameters derived from this model. We investigate the PIAC methodology on Differential Evolution and four of its parameters. We also propose a novel approach to learn the relationship between problem features and parameter settings, based on the ranking of parameter settings on problems, and rank-based statistical machine learning [HGO99; Joa02; Xia+08];

- The second part is dedicated to the validation of the PIAC methodology in an expensive black-box optimization context; we extend the 'standard' BBOB benchmark suite used in the first part, and we apply the proposed PIAC methodology, to Differential Evolution [Pri97] and CMA-ES [HO01b] algorithms.

1.3 Thesis Outline

Chapter 2 formally introduces continuous black box optimization, and gives a historical overview of Evolutionary Computation focusing on state-of-the-art algorithms for unconstrained (single-objective) black box problems.

In Chapter 3, the Algorithm Configuration problem is formally introduced. The Chapter continues with a survey of different approaches proposed in the literature for the offline tuning of metaheuristics, e.g. SMAC [HHL11b], ParamILS [Hut+09d], Irace [Bir+02; Lóp+11], or REVAC [NE07].

In order to address the limitations of the offline methods, Chapter 3 introduces the Per Instance Algorithm Configuration approach, and gives a historical overview of this approach for different domains, with a specific attention to the continuous domain.

Chapter 4 gives an introduction to the problem properties in continuous optimization, giving both the mathematical definition and the empirical computation of related quantities from function samples. In addition, it introduces well-known problem features — measures derived from a sample of candidate solutions of the objective function— and their relationship with problem properties.

Chapter 5 focuses on the computation of problem features to efficiently characterize continuous optimization problems. A general methodology and an experimental protocol to investigate the PIAC methodology is proposed, which will be used in all further experiments.

Chapter 6 introduces the first complete case study, with Different Evolution as target algorithm for PIAC. We empirically investigate the general methodology and experimental protocol introduced in Chapter 5, and give some hints about experimental settings to learn and use an *empirical performance model* when the practitioner must cope with limited experimental budget.

Chapter 7 empirically investigates the PIAC methodology with CMA-ES as the target algorithm, in the same context of limited budget.

Chapter 8 introduces the first preliminary results on an original online parameter control mechanism that relies on the PIAC methodology, and discusses different research paths to incorporate an *empirical performance model* into a parameter control mechanism.

Chapter 9 concludes the thesis and summarizes our contributions toward a better understanding of an efficient Per Instance Algorithm Configuration methodology for the continuous domain.

Part II

Background

CHAPTER 2

Evolutionary Computation for Continuous Black-Box Optimization

In many fields, researchers are faced with optimization problems for which finding an optimal solution or even a sufficiently good solution is difficult. Among a wide variety of methods, Evolutionary Algorithms are bio-inspired algorithms that are known to be particularly efficient when the objective function is costly to compute or unknown. First, Section 2.1 defines a continuous black-box optimization problem. Section 2.1 gives a brief overview of Evolutionary Computation and main methods that lie into the Evolutionary Computation techniques. This thesis focus on the parameter tuning of Evolutionary Algorithms, then as examples of optimizers, Section 2.3 introduces the CMA-ES algorithm known for its robust parameter setting to solve a wide range of continuous black-box optimization problems; and Section 2.4 introduces Differential Evolution a well-known optimizer thanks to its performance and its simple mechanism. Finally, in Section 2.5, we conclude the Chapter and discuss open questions of the field.

2.1 Continuous Black-Box Optimization

The general context of this work is that of the black-box continuous optimization scenario, i.e., in which the goal is to minimize (without loss of generality) an *objective function* defined on a continuous search space

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

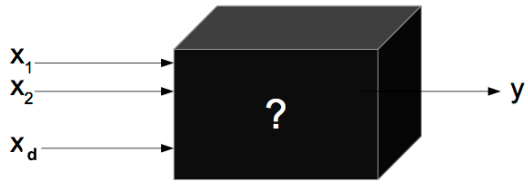


Figure 2.1: black-box Optimization.

More precisely, the goal is to find one (or several) *optimal solutions* x^* such that $f(x^*) \leq f(x)$ for all $x \in \Omega$, while minimizing the overall computation time taken by the optimization process.

The term *black-box optimization* (see Figure 2.1) refers to the fact that the only available information about the objective function f[1] are values of $f(x)$ given $x \in \Omega$. The black-box continuous optimization problem can then be recast as "Find a sequence $\{x_i\}_{i=1,n}$ of points of Ω with objective values $\{f(x_i)\}_{i=1,n}$ that allow to compute a good estimate of some optimal solutions. In particular, no assumption is made on the function itself (e.g., smoothness, differentiability, convexity, ...).

In this context, for most optimization algorithms, and even more so if the cost of one evaluation of f gets higher and higher, the number n of function evaluations needed to get a good estimate of x^* is generally used as a proxy for the overall cost of the optimization itself. This is the point of view adopted in this thesis.

[1] with the notable exception of the dimension d of the search space $\Omega \subset \mathbb{R}^d$.

2.2 Evolutionary Algorithms: Historical Overview

Evolutionary Computation (EC) was born as a research field of Artificial Intelligence, using a crude model of natural evolution to solve real-world optimization problems. Evolutionary Algorithms (EAs) are part of the larger family of bio-inspired algorithms. From the modern optimization point of view, Evolutionary Algorithms are meta-heuristics algorithms [HS04; Tal09; Dré+06]. In this work (and hence in this Chapter), we will focus on continuous optimization, and after a brief historical perspective, we will only introduce and discuss bio-inspired algorithms that can be used for continuous optimization[2]

Furthermore, we will adopt the bio-inspired point of view on EAs, and also describe other bio-inspired algorithms even though they do not perfectly fit the evolutionary paradigm, but because they are (or were originally) proposed among the same community (e.g., Differential Evolution and Ant Colony Algorithms).

2.2.1 History

The origins of Evolutionary Computation can be traced back to the 50's when Friedberg [Fri58; Fri59] described algorithms inspired by the evolution for *automatic programming*, for example when the task aims at finding a program that calculates a function. Shortly after, Bremermann [Bre62] proposed to use simulated evolution methods to solve numerical optimization problems, involving convex and linear optimization and the solution of nonlinear simultaneous simulations. This was one of the precursors of the Evolutionary Algorithms paradigm [BRS65].

Box [Box57] and Box and Draper [BD69] proposed the first attempt to use evolutionary methods for the design and analysis of industrial experiments within the concept of *Evolutionary operation* (EVOP), which inspired Spendley, Hext, and Himsworth [SHH62] to use this method as a basis, for their so-called *simplex design* method. Other approaches [Sat59a; Sat59b] proposed to introduce the randomness into the EVOP concept.

Although first attempts were met with non-negligible skepticism, in the mid 60's the first bases of Evolutionary Computation were established, into what is considered until today as the three main roots of Evolution Computation. Evolutionary Programming (EP) was introduced by Larry Fogel [Fog66] and Genetic Algorithms (GA) by Holland [Hol67], both in the United States, while in Europe,

[2] We will also avoid to discuss any of the recent "new" bio-inspired metaheuristics that have been proposed , and nicely discussed by Sörensen [Sör13].

initial work on Evolution Strategies (ES) were proposed by Rechenberg [Rec65] and Schwefel [Sch65].

Each of these branches developed with little (if any) communication with one another, until the late 80's. Only in the early 90's the different Evolutionary Computation research communities started to interact, and the term *Evolutionary Computation* was coined in the mid 90's [Fog95].

2.2.2 The Evolutionary Paradigm

The common characteristic of Evolutionary Algorithms is to rely on a crude computational model of biological evolution grossly following Darwin's theory. The main idea is that species adapt to their environment thanks to the synergetic action of both *natural selection* (popularized under the moto *survival of the fittest*) and *blind variations* (i.e., offspring inherit their parents' genetic material with some random modifications that are independent of any idea of fitness).

When it comes to the computational model of EA for optimization, the fitness function is the objective function (to be maximized without loss of generality in this Chapter[3]), and a set of candidates solutions in the search space (aka a *population* of *individuals*) undergoes a succession of *generations*. During one generation, some individuals in the current population (the *parents*) are selected, favoring the best-performing individuals w.r.t. the fitness function (*parental selection*), some *variation operators* are applied to the parents and give "birth" to *offspring*. These offspring are *evaluated* (the value of the objective function for these points of the search space is computed), and a final selection step, called *survival selection* is applied to the joint population made of parents and offspring to close the loop and build the starting population for next generation.

The different flavors (*dialects*) of EAs differ by the type of search spaces they can handle, and the corresponding variation operators, and the way the selection steps are implemented. Furthermore, several variants of EAs do not exactly fit in the above framework, but are nevertheless considered part of the EC domain, as we shall see in the following.

2.2.3 Genetic Algorithms

Genetic Algorithms are the most widely known dialect of Evolutionary Computation, probably due to their simplicity and apparent universality. They have their

[3] though in the optimization community (and the other Chapters of this dissertation), the default is to minimize the objective function, the evolutionary paradigm generally aims at maximizing the fitness.

origins in Holland [Hol67]. They perform optimization on a binary search space ($\{0,1\}^n$ for some $n > 0$). Historically, the parental selection is proportional to the individuals' fitness (roulette wheel selection), or achieved with limited comparisons of some parents (tournament selection), and the survival selection is a bare replacement of all parents by all offspring. The variation operators are the so-called "binary" operators (one- or multi-point crossover, point mutation). GAs consider that the main drive for variation is the crossover operator, and that the mutation is a background operator only useful to restore some diversity and ensure global exploration of the search space.

In the early 90s, a common belief was that GAs could solve any problem provided you can encode it in a binary search space. This explains why the early attempt to handle continuous optimization problems with GAs started by discretizing the continuous search space, projecting it onto the binary search space. With the advent of problem-specific encodings [MJ91], several works proposed to use real-coded GAs for continuous problems (see e.g., Wright [Wri+91]). However, it rapidly became clear that neither binary-coded GAs nor real-coded could perform well on continuous problems [JM91], in particular compared to Evolution Strategies (see below). Interestingly, some work by [SR96] proved that in the limit (number of bits going to infinity), binary-coded GAs with some generalized one-point crossover and bit-flip mutation were equivalent to standard Evolution Strategies with Gaussian mutation – though intractable in practice.

2.2.4 Evolution Strategies

The origins of Evolution Strategies can be traced back to the joint work between Ingo Rechenberg [Rec73] and Hans-Paul Schwefel [Sch65] in the mid 60's, in which they addressed the continuous optimization problem through a discretization of the search space, and binomially distributed mutations centered around the parent's position – an algorithm that later became known as the (1+1)-ES (see Section 2.3.1 for an explanation of the notation). The multi-membered Evolution Strategies, in which a whole population is evolved, was later introduced by [Sch81].

Today, the baseline ES is an EA that works on the continuous search space (subspace of $\mathbb{R}^d$ for some $d > 0$). There is no parental selection, and a deterministic survival selection. But the characteristics of ES is that they rely mainly on Gaussian mutation (though some crossover operators were also used in popular variants in the 90's [Sch81]) as the main variation operator.

Another important characteristic of ESs in the context of this thesis is related to parameter tuning: ESs were the first EA that proposed some adaptive parameter tuning mechanisms (more in Section 2.3.1), originally based on theoretical

studies on two simple functions, the corridor function (a linear function with bounds on all coordinates but one) and the sphere function [Rec73]. De Jong [De 06; De 07] also claims that these adaptive mechanisms of ESs are the only example of successful online parameter tuning. In any case, as will be discussed in Section 2.3.1, the state-of-the-art algorithm for continuous optimization today is, in many contexts, the CMA-ES algorithm [HO01b], one of the central algorithms studied in this work that will be presented in detail in Section 2.3.1.

2.2.5 Evolutionary Programming

Evolutionary Programming (EP) was initiated with the original work by Larry Fogel [Fog62; Fog66] on the use of simulated evolution of finite state machines for the forecasting of non-stationary time series. EP was then used as a general purpose optimizer, being the first EC dialect to argue for problem-dependent representation and variation operators, but problem independent evolution engine. Furthermore, EP always argued that mutation was the main variation operator, crossover being at best useless, at worst harmful. It used no parental selection (all parent produce one offspring, by mutation only), and some specific form of survival selection close to GA tournament selection involving both parents and offspring populations.

When it comes to continuous optimization, EP used real representation, as expected, and naturally turned to Gaussian mutation, also (and independently) proposing online adaptation of the mutation parameters very similar to the ones from Evolution Strategies. However, in spite of an active community in the early 90s, EP did not ”survive” as a stand-alone dialect, and its community was seamlessly merged in the wider community of Evolutionary Algorithms.

2.2.6 Genetic Programming

Genetic Programming (GP) is a late root of EC, born as a subset of GAs devoted to the evolution of programs. The term Genetic Programming was first explicitly used by Cramer [Cra85] during the first ICGA conference, and was later popularized by Koza [Koz90; Koz92].

The main characteristics of GP is that it operates in the space of programs, originally encoded in parse trees, though later other representations for programs were proposed (linear [BB07], cartesian [MT00]). The fitness function is computed by running the program in a specific context and estimating how well it performed in achieving a target task. The selections are similar to those of GAs with some emphasis on tournament selection. The variation operators are specific

crossover (exchange of sub-trees) and mutation (local variation of small parts of the tree operators).

In any case, the goals of GP are rather far from continuous optimization, and despite its successes to evolve human-competitive programs [Koz+96], GP will not be mentioned anymore here.

2.2.7 Differential Evolution

Differential Evolution was originally proposed by Storn and Price [SP97] for continuous optimization. Its principles are close to those of EAs, and it gained a large popularity due to its simplicity and efficiency.

DE relies on the evolution of a population, every member of the population undergoes some kind of mutation followed by a crossover operator. Natural selection is at stake too in the choice of the way mutation operates. The characteristic feature of DE is that the mutation depends on the whole population: this allows DE to realize an implicit adaptation of the mutation strength during the course of the algorithm [DB01]. Because DE is the other algorithm on which the main ideas of this thesis have been implemented and validated (with CMA-ES), a comprehensive description of DE will be given in Section 2.4.

Related to the central topic of this work, DE is known to have only a few parameters, but to be particularly sensitive to their setting. Furthermore, DE does not suffer from the main defect of, e.g., PSO, the sensitivity to non-separability (at least when the crossover parameter is small), which explains our choice of DE for the first experiments with PIAC (see Section 6).

It is also known for experimental studies investigating adaptive mechanisms for the online tuning of is parameters [Fia+10]. Several adaptive variants of DE has been proposed [LL05; Liu+02; Bre+09; Bre+06; ZS09], such that the main parameters of DE are updated during the run by a parameter control mechanism like JADE [ZS09] that uses an adaptive mutation operator to update the mutation and crossover operators with respect to an external archive of best individuals.

2.2.8 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) [LL02] do not evolve a population but a parameterized probability distribution over the search space. This distribution is sampled, and updated according to the values of the best-performing samples in terms of objective function value. In some sense, EDAs adopt a point of view that is orthogonal to that of the generic EA described above, as

the population is only an ephemeral instantiation of the underlying probability distribution on the search space.

The first EDAs were handling binary search spaces [Müh97], evolving Bernouilli distributions. EDAs were also rapidly used for continuous optimization, evolving Gaussian distribution over the continuous search space [SD98]. The boundary has become very slim today between EDAs and continuous EAs: for instance, CMA-ES (see Section 2.3.1) can be viewed as an EDA evolving a Gaussian mutation with adaptive parameters.

2.2.9 Ant Colony Optimization

Ant Colony Optimization (ACO) was introduced by Dorigo [Dor92], and imitates the behavior of ants, in order to solve combinatorial optimization problems that can somehow be cast to routing problems.

Crudely mimicking natural ants, the algorithm aims at finding the optimal path between the colony and the source of food. The "ants" lay down some pheromone that in turn attracts other ants, gradually reinforcing the attraction of the most used route, hopefully the shortest path to the food. Evaporation reduces pheromone values of all trails over time, thus preserving some diversity in optimal path seeking.

ACO is widely used in the combinatorial domains [DS03], considering problem specific approaches, in particular specific representations, for example the Traveling Salesman Problem [DG97]. However, despite encouraging results, ACO could not compete with state of the art algorithms for solving TSP problems. Nevertheless, it opened a research path to new algorithmic variants and other applications (see [DS09] for an overview).

Regarding the continuous domain, several works [DS02; DS04; SD08; Che+05; Yan+03; CCW06; WW03; Dua+07] adapt the original concept of ACO to the continuous domain. While first attempts consider a discretization of the search space. Socha and Dorigo [SD08] propose to use a Gaussian kernel probability density function expressed as a distribution model of pheromones.

Zlochin and Dorigo [ZD02] and Zlochin et al. [Zlo+04] revealed considerable structural similarities between ACO, stochastic gradient descents and EDAs. Hence they proposed to regroup these methods under the term *model-based search* metaheuristics, as they rely on a probabilistic model. Therefore, it opened a research path to a theoretical approach of ACO, yet difficult to be used in order to propose new and efficient variants of ACO.

2.2.10 Particle Swarm Optimization

Particle Swarm Optimization (PSO) was initially proposed by Kennedy and Eberhart [KE95] for continuous optimization, after the simulation of some simplified social model of bird flocks or fish schools.

Though the vocabulary differs (coming from another natural paradigm), here again, the algorithm evolves a population –*swarm*– of candidate solutions –*particles*–, that move around the search space according to a simple strategy. The speed of a particle is modified at each time step by the best position the particle encountered before, and by the position of the best particle the whole population ever encountered. Initially, the algorithm was elitist, only considering the current best solution for the next iteration, hence favoring a premature convergence on multimodal functions. Then, different variants were proposed, using a set of best particles at each iteration [Lia+06a].

Like DE, PSO exposes simplicity and performances, at least on separable functions: PSO is proven to be scale invariant [WKG07], but Hansen et al. [Han+11b] empirically demonstrated that PSO is not rotation invariant. As a result, Bonyadi and Michalewicz [BM14] proposed a rotation invariant version of PSO ensuring local convergence. More recent variants (e.g. [CM11; Zha+09]), propose some adaptation mechanisms to deal with complex problems (see Bonyadi and Michalewicz [BM16] for a more precise overview of recent methods).

2.3 CMA-ES

2.3.1 Evolution Strategies

The main concept of Evolution Strategies was introduced by Rechenberg [Rec73] and Schwefel [Sch68]. However, as said above, an Evolution Strategy (ES) today is an Evolutionary Algorithm for continuous optimization whose main variation operator is a multivariate Gaussian mutation that follows the generic EA evolution template. A comprehensive overview of modern ES is given in [BFK13], but only a very small part will be covered here, in relation to the core topic of this work, the optimal setting of hyper-parameters.

A population of μ individuals is generated at random (usually uniformly on the search space Ω), and undergoes a series of generations. At each generation, λ offspring are generated, without parental selection, linear recombination or coordinate-wise crossover, followed by multivariate Gaussian mutation, and a deterministic survival selection.

The Gaussian mutation of parents x giving offspring y works by drawing a sample from a Gaussian mutation centered on x with (positive definite) covariance matrix $C \in \mathcal{M}_{d\times d}(\mathbb{R})$ and so-called step-size σ, i.e.,

$$y_i = x + \sigma\mathcal{N}(\mathbf{0}, \boldsymbol{C}), i = 1, \ldots, \lambda \tag{2.1}$$

There are two variants of the (deterministic) survival selection, as first described in Rechenberg [Rec78]:

- In the $(\mu + \lambda) - ES$, the μ best individuals from (parents+offsprings) are selected to become the next population. This is an elitist selection, i.e., the fitness value of the best individual in the population can only increase.
- In the $(\mu, \lambda) - ES$, the μ best offspring are selected to become the next population. In this variant, the fitness value of the best individual in the population can decrease from one generation to the next.

The original algorithm designed by Rechenberg and Schwefel involved a single parent, that was undergoing Gaussian mutation, and the best individual out of parent and offspring was selected to become the next parent: this is a $(1+1)-ES$.

Crossover could be either a linear convex recombination of both parents with randomly drawn weights, or a uniform exchange of coordinates between both parents. These crossovers will not be discussed here as they have been abandoned in recent works, in particular in CMA-ES, as discussed in Section 2.3.3[4].

2.3.2 Parameter Adaptation in Evolution Strategies

The main variation operator of ES is the Gaussian mutation described by Equation (2.1). However, nothing is said about how the mutation parameters σ and C are set. Considering two simple functions (the corridor and the sphere functions), Rechenberg [Rec73] analyzed the (1+1)-ES with identity covariance matrix and showed that linear convergence can be achieved if the mutation step-size is adapted in such a way that the success probability of the mutation (offspring better than parent) is close to 0.2. Accordingly, he designed the first known adaptive rule for online parameter tuning, the well-known $\frac{1}{5}$ success rule. This rule gives very good results on some functions, but can sometimes fail to reach the global optimum of the objective function even in very simple case (e.g., with simple constraints on Ω).

[4] though the averaging over the μ best sampled points can be seen as a generalized crossover.

A further step in parameter adaptation was proposed by Schwefel [Sch81], and extensively studied by Beyer and Schwefel [BS02]. It is called *self-adaptive* parameter tuning and relies on the idea that each individual should carry its own mutation parameters (e.g., σ and C for the full multivariate model). These mutation parameters are themselves mutated (nevertheless ensuring the positivity of σ and the positive definiteness of C), then the individual itself is mutated using the new values of σ and C. The rationale is that, even though selection applies only to the individual (and not to the mutation parameters), an individual with poor mutation parameters w.r.t. the region of the search space it is exploring (e.g., small σ in regions of low gradient) will rapidly be overpassed by other individuals with better-fitted parameters (large σ that will allow it to make larger steps). Great successes were obtained using this self-adaptive mechanism, but at the price of a high computational cost, as the adaptation in particular of the full covariance matrix ($d \times (d-1)/2$ parameters) requires a lot of iterations.

Based on these experiences, and on a careful study of the actual adaptation of the parameters, the CMA-ES algorithm was proposed by Ostermeier, Gawelczyk, and Hansen [OGH94], Hansen and A [HA96], Hansen and Ostermeier [HO01a], and Hansen, Müller, and Koumoutsakos [HMK03], and some of its variants are considered today the state-of-the-art off-the-shelf optimization algorithm for continuous optimization, beyond the evolutionary or even the stochastic optimization communities.

2.3.3 The CMA-ES Algorithm

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [HMK03] is one of the most efficient algorithms for real valued single-objective optimization problems. Thanks to its invariance properties [Han+11a], some default parameter values could be tuned using a rather small set of test functions [HMK03], and nevertheless provide robust performances on a large variety of problems, from analytical benchmark functions [Han09b] to many real-world applications (see, among many others, [Han+09b]). The code is available in many languages on the main author's Web page at https://www.lri.fr/~hansen/cmaesintro.html.

2.3.3.1 The Algorithm

CMA-ES [HMK03] evolves a Gaussian distribution $\mathcal{N}\left(\boldsymbol{m}^t, (\sigma^t)^2 \boldsymbol{C}^t\right)$ on $\mathbb{R}^d$ with mean $\boldsymbol{m}^t$ (the current estimate of the optimal solution) and Covariance matrix $(\sigma^t)^2 \boldsymbol{C}^t$, where the step-size σ^t is isolated from the covariance direction $\boldsymbol{C}$ so they can be adapted separately.

The original $(\mu/\mu_w, \lambda)$-CMA-ES (Algorithm 1) works as follows: at iteration

t, the current distribution $\mathcal{N}\left(\boldsymbol{m}^t, (\sigma^t)^2\,\boldsymbol{C}^t\right)$ is sampled, generating λ candidate solutions (line 5), whose fitness is computed (line 6). The new mean $\boldsymbol{m}^{t+1}$ is computed line 7 as the weighted sum of the best μ individuals according to f. The adaptation of the step-size σ^t is controlled by the evolution path $\boldsymbol{p}_\sigma^{t+1}$, that stores, with relaxation factor c_σ, the successive mutation steps $\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$ (line 8). The step-size is increased (resp. decreased) in the case of the length of the evolution path $\boldsymbol{p}_\sigma^{t+1}$ is longer (resp. smaller) than the expected length it would have under a random selection (line 9).

The Covariance matrix is updated using both a rank-one update term, computing the evolution path $\boldsymbol{p}_c^{t+1}$ of successful moves of the mean $\frac{\boldsymbol{m}^{t+1}-\boldsymbol{m}^t}{\sigma^t}$ of the distribution in the original coordinate system (line 11) and the rank-μ update, a weighted sum of the covariances of successful steps of the best μ individuals (using the weights of the update of the mean – line 12). Two weights are used for this last update (line 13), c_1 for the rank-one term, and c_μ for the rank-μ term, hence c_1 and c_μ must be positive with $c_1 + c_\mu \leq 1$.

2.3.3.2 CMA-ES Invariances

Like all Evolution Strategies (and like all comparison -based algorithms, e.g., DE, PSO, and some variants of GAs), CMA-ES is invariant by monotonous transformations of the fitness function: optimizing f or $g \circ f$, for a monotonous $g : \mathbb{R} \mapsto \mathbb{R}$, are equivalent. But CMA-ES exhibits a unique and very important invariance, with respect to rigid transformations of the search space (e.g., any change of orthogonal coordinates). Hence CMA-ES has exactly the same behavior on a function and any of its rotated variants. In particular, CMA-ES is by design totally insensitive to the non-separability of the objective function, which is not the case for the base versions of PSO or ACO for instance. However, any optimization algorithm can be made coordinate-independent by encapsulating it into the CMA-ES mechanism [Han08].

2.3.3.3 CMA-ES Parameters

The default values of the parameters of the algorithm [HMK03] are set in line 1, but are hidden to the user in the standard CMA-ES distributions, except for the population size λ, the initial step-size $\sigma^{t=0}$, and the number of selected parents μ. Though the already-mentioned invariance properties of CMA-ES [Han+11a] ensure some robustness of the default setting, several improvements could be reached using off-line tuning of some of these parameters, namely λ (or more precisely the coefficient of λ as a function of d) and the ratio $\frac{\mu}{\lambda}$ [Tey10], as well as the parameters c_σ and d_σ for the adaptation of σ [Hut+09a; SE10b].

Note that some additional parameters related to the stopping criterion are not presented in Algorithm 1, and have a large impact on the restart versions of CMA-ES [AH05]. These were also tuned using IRACE in [LS13b; LS13a; LLS12]. However, to the best of our knowledge, the parameter setting for the adaptation of the Covariance matrix c_c (line 11), c_1 and c_μ (line 13) has only been addressed within a parameter control mechanism (such that parameters are updated during the optimization process) in [Los+14], that will be detailed in the next Section.

2.3.3.4 Self-CMA-ES

Loshchilov et al. [Los+14] proposed Self-CMA-ES, a novel variant of CMA-ES, that consider the on-line adaptation (during the optimization process) of c_c , c_1 , c_μ Self-CMA-ES relies on the hypothesis that the best parameter configuration at time t is the one that would have maximized at time $t-1$ the likelihood of generating the best individuals selected at time t.

At every iteration t, an auxiliary optimization algorithm (another CMA-ES, denoted CMA-ES$_{aux}$) is hence used to estimate the optimal configuration for $t+1$. After generating and evaluating the λ offspring at time t (lines 4-5 of Algorithm 1), the state of the algorithm at time $t-1$ is restored, and the optimization of parameters c_c , c_1 , c_μ proceeds as follows: for each triplet value (c_c , c_1 , c_μ), the virtual distribution parameters σ and $\boldsymbol{C}$ are computed (lines 8-13) from state $t-1$, and the performance of (c_c , c_1 , c_μ) is the likelihood of generating the best μ of the actual λ offspring at time t from this virtual distribution. The triplet (c_c , c_1 , c_μ) that maximizes this likelihood is returned and is then used, at time t, to complete the actual update of the actual mutation parameters of CMA-ES (lines 8-13).

A first issue is that computing the log-likelihood of generating μ given points of $\mathbb{R}^d$ from a given Gaussian is costly and numerically unstable. It was hence replaced by a proxy, that works as follows. λ points are sampled from the virtual Gaussian, their virtual mean is computed (as in line 7), and the Mahalanobis distance between the actual μ best offspring at time t and this mean is computed. The sum of ranks of these distances is used as a proxy for the likelihood. The detailed formal description of this proxy for the likelihood is given in [Los+14], together with the global Self-CMA-ES algorithm.

A second issue is the possible overfitting of the parameters (c_c , c_1 , c_μ) due to a single and limited sampling of the actual offspring at time t. And a third issue is the computational cost of running a full CMA-ES$_{aux}$ inside every iteration of the master CMA-ES: even though no additional fitness computation of the main CMA-ES is required, and even though the dimension of the auxiliary optimization problem is only 3, sampling the virtual Gaussian distribution to evaluate the

Algorithm 1 The $(\mu/\mu_w, \lambda)$-CMA-ES (from [HMK03])

given $n \in \mathbb{N}_+$, $\lambda = 4 + \lfloor 3 \ln n \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, $w_i = \frac{\ln(\mu+\frac{1}{2}) - \ln i}{\sum_{j=1}^{\mu} (\ln(\mu+\frac{1}{2}) - \ln j)}$ for $i = 1 \dots \mu$, $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$, $c_\sigma = \frac{\mu_w+2}{n+\mu_w+3}$, $d_\sigma = 1 + c_\sigma + 2\max(0, \sqrt{\frac{\mu_w-1}{n+1}} - 1)$, $c_c = \frac{4}{n+4}$, $c_1 = \frac{2}{(n+1.3)^2+\mu_w}$, $c_\mu = \frac{2\,(\mu_w - 2 + 1/\mu_w)}{(n+2)^2+\mu_w}$
initialize $\boldsymbol{m}^{t=0} \in \mathbb{R}^d, \sigma^{t=0} > 0, \boldsymbol{p}_\sigma^{t=0} = \mathbf{0}, \boldsymbol{p}_c^{t=0} = \mathbf{0}, \boldsymbol{C}^{t=0} = \mathbf{I}, t \leftarrow 0$
repeat
  for $k = 1, \dots, \lambda$ **do**
    $\boldsymbol{x}_k = \boldsymbol{m}^t + \sigma^t \mathcal{N}(\mathbf{0}, \boldsymbol{C}^t)$
    $\boldsymbol{f}_k = f(\boldsymbol{x}_k)$
  $\boldsymbol{m}^{t+1} = \sum_{i=1}^{\mu} \boldsymbol{w}_i \boldsymbol{x}_{i:\lambda}$
  $\boldsymbol{p}_\sigma^{t+1} = (1 - c_\sigma)\boldsymbol{p}_\sigma^t + \sqrt{c_\sigma(2 - c_\sigma)}\sqrt{\mu_w}(\boldsymbol{C}^t)^{-\frac{1}{2}} \frac{\boldsymbol{m}^{t+1} - \boldsymbol{m}^t}{\sigma^t}$
  $\sigma^{t+1} = \sigma^t \exp(\frac{c_\sigma}{d_\sigma}(\frac{\|\boldsymbol{p}_\sigma^{t+1}\|}{\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|} - 1))$
  $h_\sigma = \mathbb{1}_{\|p_\sigma^{t+1}\| < \sqrt{1-(1-c_\sigma)^{2(t+1)}}(1.4 + \frac{2}{n+1})\,\mathbb{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|}$
  $\boldsymbol{p}_c^{t+1} = (1 - c_c)\boldsymbol{p}_c^t + h_\sigma \sqrt{c_c(2 - c_c)}\sqrt{\mu_w} \frac{\boldsymbol{m}^{t+1} - \boldsymbol{m}^t}{\sigma^t}$
  $\boldsymbol{C}_\mu = \sum_{i=1}^{\mu} w_i \frac{\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}^t}{\sigma^t} \times \frac{(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}^t)^T}{\sigma^t}$
  $\boldsymbol{C}^{t+1} = (1 - c_1 - c_\mu)\boldsymbol{C}^t + c_1 \underbrace{\boldsymbol{p}_c^{t+1} \boldsymbol{p}_c^{t+1\,T}}_{\text{rank-one update}} + c_\mu \underbrace{\boldsymbol{C}_\mu}_{\text{rank}-\mu\text{ update}}$
  $t = t + 1$
until *stopping criterion is met*

proxy likelihood of many triples (and here the dimension is d) has a non-negligible cost.

However, both issues can be resolved simultaneously. First, the CMA-ES$_{aux}$ is not restarted from scratch at every iteration t of the main CMA-ES, but restarts from the state of the CMA-ES$_{aux}$ at the end of iteration $t-1$; Second, only a small number of iterations of CMA-ES$_{aux}$ is actually run, avoiding possible overfitting.

Loshchilov et al. [Los+14] empirically demonstrated that Self-CMA-ES can outperform CMA-ES on a set of test functions taken from the BBOB testbench [Han+10] and for different dimensions n. Figure 2.2 shows typical results of Self-CMA-ES on BBOB, such that at each iteration a new parameter setting of c_c , c_1 , c_μ is applied, hence outperforming CMA-ES in terms of the number of function evaluations needed to reach a given target numerical precision.

Algorithm 2 The Self-CMA-ES

$t \leftarrow 1$
$\theta_f^t \leftarrow$ InitializationCMA() {CMA-ES}
$\theta_h^t \leftarrow$ InitializationCMA() {CMA-ES$_{aux}$}
$\theta_f^{t+1} \leftarrow$ GenerationCMA(f, θ_f^t)
$t \leftarrow t + 1$
repeat
  $\theta_f^{t+1} \leftarrow$GenerationCMA($f, \theta_f^t$)
  $\theta_h^{t+1} \leftarrow$GenerationCMA($h_t, \theta_h^t$)
  fill θ_f^{t+1} with the mean of the distribution (explanation in subsection 2.3.3.1) $\theta_h^{t+1}.m$
  $t \leftarrow t + 1$
until stopping criterion is met

Algorithm 3 Objective function $h_t(\theta)$

Input: θ, θ_f^{t+1},θ_f^{t-1}, θ_f^t, μ, $w_{sel,i}$ for $i = 1, \ldots, \mu$
$\theta_f^{'t-1} \leftarrow \theta$
$\theta_f^{'t} \leftarrow$ReproduceGenerationCMA($f, \theta_f^{'t-1}$) injecting already evaluated $\theta_f^t.\boldsymbol{x}_{i:\lambda}$
$d_i \leftarrow \left\| \theta_f^{'t}.\sqrt{C^{-1}} \cdot (\theta_f^{t+1}.\boldsymbol{x}_i^t - \theta_f^{'t}.m) \right\|$; for $i = 1, \ldots, \theta_f^{t+1}.\lambda$
$p_i \leftarrow$ rank of $d_i, i = 1 \ldots \lambda$ sorted in decreasing order
$h(\theta) \leftarrow \sum_{i=1}^{\mu} w_{sel,i} p_{i:\lambda}$ { $i : \lambda$ denotes the rank of $\theta^{t+1}.\boldsymbol{x}_i$ }
Output: $h(\theta)$

2.4 Differential Evolution

Differential Evolution (DE) was originally proposed by Storn and Price [SP97] for solving global optimization problems in the continuous domain.

Differential Evolution evolves a population of NP individuals, d-dimensional vectors, corresponding to the decision variables of the objective function. One generation of DE goes as follows: for each individual in the population, a mutant vector is built from the rest of the population, adding to one vector the weighted difference between two others – the choice of these three vectors is governed by a given **strategy**. A crossover operator is then applied between the initial individual and the mutant vector, and the resulting individual replaces the initial individual if it performs better. The algorithm is described more formally in Algorithm 4, and source code for different programming languages is available at http://www.icsi.berkeley.edu/ storn/∼code.html.

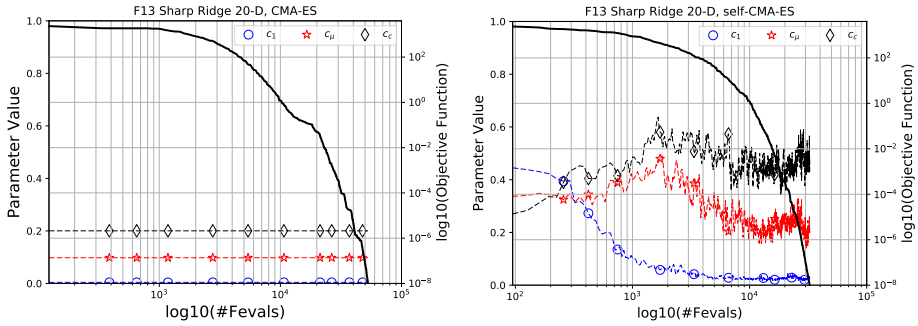


Figure 2.2: Evolution of learning rates c_1, c_μ, c_c (lines with markers, left y-axis) and log10(objective function) (plain line, right y-axis) of CMA-ES(left) and Self-CMA-ES(right) on 20-dimensional Sharp Ridge from [Han+10]. The medians of 15 runs are shown..

The original version of DE is proposed with three main parameters controlling the selection, the recombination and the mutation of individuals: the population size NP, the crossover probability CR and the scaling factor F. Furthermore, the mutation operator is the main operator in DE. Hence different variants for the mutation strategy were proposed. Tuning DE thus amounts to choosing these four parameters.

2.4.1 Initialization

Because DE is a global optimization algorithm, the initial population $\boldsymbol{P}_0$ should cover as uniformly as possible the search space, an hyper-rectangle in $\mathbb{R}^d$ delimited by the bounds on the variables: The initial individuals in the population are generated by uniformly sampling $\Pi_i[x^i_{min}, x^i_{max}]$.

2.4.2 Mutation Strategies

The original mutation in DE constructs a set of NP of mutant vectors. The i^{th} mutant vector is built by uniformly picking up three different individuals in the population, $\boldsymbol{X}_{\text{rand}^i_1,g}$, $\boldsymbol{X}_{\text{rand}^i_2,g}$, $\boldsymbol{X}_{\text{rand}^i_3,g}$), and adding to the first one the relaxed difference of the other two. The crossover operator will later recombine each individual in the population with a different mutant vectors in turn. Multiple variants have been later proposed regarding the choice of the individuals that are used to construct the mutant vectors. Some involve the best individual $\boldsymbol{X}_{\text{best},g}$ encountered during the search up to iteration g, others use five individuals rather

Algorithm 4 Differential Evolution (from [SP97])

given $NP \in \mathbb{N}_+$, $F \in [0,1]$, $CR \in [0,2]$ and ***strategy***
initialization $g \leftarrow 0$
$\boldsymbol{P}_0 = \{\boldsymbol{X}_{i,g}, \ldots, \boldsymbol{X}_{NP,g}\}$, where $\boldsymbol{X}_{i,g} \equiv \{x^1_{i,g}, \ldots, x^d_{i,g}\} = rand(\Pi_i[x^i_{min}, x^i_{max}])$
repeat
  #### Mutation with the chosen strategy
  for $i = 1, \ldots, NP$ **do**
    $\boldsymbol{V}_{i,g} = \{\boldsymbol{v}^1_{i,g}, \ldots, \boldsymbol{v}^d_{i,g}\} \equiv$ ***strategy***(P_g)
  #### Binary Crossover
  for $i = 1, \ldots, NP$ **do**
    $j_{\text{rand}} = \lfloor rand[1,d]$
    for $j = 1, \ldots, d$ **do**
      $u^j_{i,g} = \begin{cases} v^j_{i,g} & \textbf{if } (rand(0,1) \leq CR) \text{ or } (j_{\text{rand}} = j) \\ x^j_{i,g} & \textbf{otherwise} \end{cases}$
  #### Selection
  for $i = 1, \ldots, NP$ **do**
    compute $f(\boldsymbol{U}_{i,g})$
    if $f(\boldsymbol{U}_{i,g}) \leq f(\boldsymbol{X}_{i,g})$ **then**
      $\boldsymbol{X}_{i,g+1} \leftarrow \boldsymbol{U}_{i,g}$
    if $f(\boldsymbol{U}_{i,g}) \leq f(\boldsymbol{X}_{\text{best},g})$ **then**
      $\boldsymbol{X}_{\text{best},g} \leftarrow \boldsymbol{U}_{i,g}$
until *stopping criterion is met*

than three (uniformly choosing $\boldsymbol{X}_{\text{rand}^i_4,g}$ and $\boldsymbol{X}_{\text{rand}^i_5,g}$) too. Note that all indices $\text{rand}^i_j, g, j = 1, \ldots, 5$ are all distinct, and different from index i.

Such choices are summarized into a **strategy**, and the most widely used strategies have been selected in this present work, and are described in the following. The scaling factor F is a key parameter that controls the scaling magnitude of the mutation.

- DE/rand/1

$$\boldsymbol{V}_{i,g} = \boldsymbol{X}_{\text{rand}^i_1,g} + F \times (\boldsymbol{X}_{\text{rand}^i_2,g} - \boldsymbol{X}_{\text{rand}^i_3,g})$$

- DE/best/1

$$\boldsymbol{V}_{i,g} = \boldsymbol{X}_{\text{best},g} + F \times (\boldsymbol{X}_{\text{rand}^i_1,g} - \boldsymbol{X}_{\text{rand}^i_2,g})$$

- DE/rand-to-best/1

$$\boldsymbol{V}_{i,g} = \boldsymbol{X}_{\text{best},g} + F \times (\boldsymbol{X}_{\text{best},g} - \boldsymbol{X}_{i,g}) + F \times (\boldsymbol{X}_{\text{rand}_1^i,g} - \boldsymbol{X}_{\text{rand}_2^i,g})$$

- DE/best/2

$$\boldsymbol{V}_{i,g} = \boldsymbol{X}_{\text{best},g} + F \times (\boldsymbol{X}_{\text{rand}_1^i,g} - \boldsymbol{X}_{\text{rand}_2^i,g}) + F \times (\boldsymbol{X}_{\text{rand}_3^i,g} - \boldsymbol{X}_{\text{rand}_4^i,g})$$

- DE/rand/2

$$\boldsymbol{V}_{i,g} = \boldsymbol{X}_{\text{rand}_1^i,g} + F \times (\boldsymbol{X}_{\text{rand}_2^i,g} - \boldsymbol{X}_{\text{rand}_3^i,g}) + F \times (\boldsymbol{X}_{\text{rand}_4^i,g} - \boldsymbol{X}_{\text{rand}_5^i,g})$$

2.4.3 Crossover

A second step corresponds to a crossover operation recombining each individual $\boldsymbol{X}_{i,g}$ with a mutant vector $\boldsymbol{V}_{i,g}$ in order to generate a *trial* vector $\boldsymbol{U}_{i,g} = \{\boldsymbol{u}_{i,g}^1, \ldots, \boldsymbol{u}_{i,g}^d\}$. The crossover operator (line 12) is applied for each coordinate with crossover probability CR, that controls the fraction of variable values copied from the mutation vector. Note that at least one coordinate is exchanged – this is the purpose of j_{rand} (line 10), all other coordinates are exchanged with probability CR.

The crossover described here can be viewed as the so-called uniform crossover in GAs (considering each coordinate as a gene), and is sometimes called binary crossover in DE framework. But, in other variants, the exponential crossover is used: introduced by Storn and Price [SP97], it is similar to a two-point crossover in which the first cutting point is randomly selected and the second cutting point is determined with respect to the L^{th} elements (counting in a circular manner). Zaharie [Zah07] gives an overview of the crossover operator with theoretical explanations, on both variants (binomial and exponential crossover), and its influence on the choice of the parameter control.

2.4.4 Deterministic Selection

The last step of the algorithm considers the selection of individuals, by first ensuring that each individual vector does not violate the boundary constraint defined by $\boldsymbol{X}_{min}$ and $\boldsymbol{X}_{max}$. In the case of a boundary constraint violation, the vector is reinitialized by uniformly randomizing a new individual respecting the constraint. All trial vectors are then evaluated and the selection is achieved, by replacing $\boldsymbol{X}_{i,g}$ by their corresponding $\boldsymbol{U}_{i,g}$ if $f(\boldsymbol{U}_{i,g}) \leq f(\boldsymbol{X}_{i,g})$.

2.4.5 DE and Adaptation of Parameters

Differential Evolution achieves good performances on a wide range of problems – see e.g. [PSL06]. Furthermore, it is a rather simple algorithm that can be implemented easily. On the other hand, it is very sensitive to its parameters: the population size, the mutation strategy, the scaling factor F and the probability of recombination CR.

High values of the scaling factor F will favor exploration of the search space, whereas a small value should be preferred to allow a differential in order to explore some rugged regions and still maintain a diversity. In addition to the scaling factor, more individuals help to reduce the mutation step size [Zah02], but large values of F or the population size may result in a premature convergence [L+00; Zah02; Zah03].

However, and in contrast with Evolution Strategies, even with a fixed value of F as in its original version [Pri97], DE exhibit some degree of adaptation to the landscape ruggedness, through the population variance, similarly to the case of Generic Algorithms with SBX crossover [DB99]. The population size hence also plays an important role in such adaptation – though at the price of a poor scale-up with the dimension (the original authors recommend that the population size increases linearly with the dimension).

CR has a direct influence of the population diversity. A high value of CR induces more variations in the new offspring, therefore increasing diversity and exploration [Zah03]. However a premature convergence of the population can be observed with high values of CR, such as $CR \in [0.3, 0.9]$ [GMK02]. But even more importantly, CR directly impacts the behavior of DE facing non-separable objective functions: if $CR = 0$, DE is invariant w.r.t. rigid transformations of the coordinate system, like CMA-ES. When CR increases, the algorithm is more and more sensitive to non-separability [Aug+09].

The choice of the parameter values hence highly depends on the properties of the problem to solve, and will be the object of some contributions from this work. Another approach to tackle this issue has been proposed through adaptive parameters, that are modified online (see [DS11] for an overview of many variants of Differential Evolution). The best known approaches are JADE [ZS09] and jDE [Bre+06], that adapts CR and F during the run of DE. In JADE, the adaptation strategy is based on the improvement of the fitness value in the population, and an external archive of individuals already evaluated. The goal is to prevent some premature convergence in favor of a better exploration when the algorithm is in a local optimum or a plateau.

JDE [Bre+06], on the other hand, implements a self-adaptive approach (see

Section 2.3.2) for the parameters F and CR and experimentally validate it on the benchmark of the CEC 2006 special session [Lia+06b]: the individuals do no only have decision variables but also carry the control of parameters F and CR. Each individual thus encodes the vector $\vec{x}_{NP,G}$ and the parameter values $F^k_{NP,G}$ and $CR^k_{NP,G}$ with $k \in 1, 2, 3$ that represent each candidate strategies $rand/1/bin$, $currenttobest/1/bin$, and $rand2/bin$. At every generation, the new values of these parameters are computed as follows

$$F^k_{i,G} = \begin{cases} F_l + rand_1 \times F_u & \text{if } rand_2 < \tau_1 \\ F^k_{i,G} & \text{otherwise} \end{cases}$$

$$CR^k_{i,G} = \begin{cases} rand_3 & \text{if } rand_4 < \tau_2 \\ CR^k_{i,G} & \text{otherwise} \end{cases}$$

where τ_1 and τ_2 are probabilities to adjust the control of the parameters, respectively F and CR.

Although, adaptive methods show improved performances on different continuous black-box benchmarks [BZM06; Bre+09], in the context of this thesis, where the focus is on algorithm configuration, the original version of Differential Evolution will be used, without comparison with its adaptive variants. More precisely, the proposed Per Instance Algorithm Configuration methods (see Chapter 3) sets parameters values at initialization, and the parameter setting found with PIAC will be compared to the recommended values in [SP97]

2.5 Discussion

In this chapter we introduced Evolutionary Computation to solve continuous black-box single-objective optimization problems, and presented a brief historical overview of Evolutionary Computation, showing that it is itself an evolution of successive approaches to optimization. The evolution of approaches is closely related to the complexity of the real-world models. While the diversity and complexity of these problems grow, the robustness and the ability to efficiently solve them with as few changes as possible within the optimization techniques has drawn a lot of attention. The CMA-ES algorithm is known to demonstrate such robustness, as a result of its invariance properties with respect to rank preserving transformations of the objective function and rigid transformations of the search space. But some empirical observations on Self-CMA-ES confirm that its mechanisms can still be improved, as its parameter setting is not optimal on all optimization problems. Hence there are still opportunities for further improvement.

While many optimizers exist in the literature (not covered in this thesis), no algorithm is known to outperform all others on all problems. But, for a given optimization algorithm, the open question is to know if some specific parameter settings can be found for each instance of an optimization problem, in particular when tackling a new problem for which no prior knowledge is available. Such crucial challenge is known in the literature as the Algorithm Configuration problem, and has already been partly successfully empirically tackled in the continuous domain for classes of problem instances, e.g. with CMA-ES[LS13b; LS13a; SE10b]. However, given a new continuous optimization instance in an operational context (with limited evaluation budget), finding specific parameterization of the algorithm at hand pertains to the Per Instance Algorithm Configuration problem and remains an open issue in the continuous domain.

CHAPTER 3

Algorithm Configuration

In the Section 3.1, we briefly introduce performance indicators used to compare and assess EAs and their parameter settings. It is followed in Section 3.2 by an overview of the different automated Algorithm Configuration methods, after the definition of the problem to solve. Finally, Section 3.3 introduces a novel approach of the AC based on problem description, the Per Instance Algorithm Configuration, such that, after a formal description, we give a historical overview of the method, and contributions to solve problems of the continuous domains.

3.1 Performance Indicators

With a steadily growing need of computational power for more and more complex real-world optimization problems, choosing the right algorithm and/or the right parameter setting becomes a high priority. In this direction, it is widely acknowledged that there is no algorithm nor parameter setting that will perform optimally on all optimization problems. Therefore, such algorithm and/or parameters setting choice, or the design of a new algorithm, mostly relies on empirical performance evaluations. The choice of the performance measure must reflect the specific needs of the practitioner.

In the context of continuous black box optimization, as mentioned earlier, the cost of the optimization itself is measured in terms of number of calls to the objective function - even more so when the objective function is considered expensive (in terms of time/money). However, another important target of the practitioner is the accuracy of the solution that the algorithm finally returns. Hence, when it comes to the comparison of different optimization algorithms, or

different parameter settings for a given algorithm, three types of performance indicators are at play:

- The *quality* of the solution found by a given algorithm on a given problem quantifies the distance to some known optimum or best known value of the objective function. Indeed, the result of the optimization will be the best solution returned by the algorithm. But this indicator has to be balanced by the cost of the optimization itself, and the quality is usually to be understood with a constraint on the overall cost, in terms of number of function evaluations in our context.

- Symmetrically, the **runtime** expresses the cost (in terms of number of function evaluations) of the optimization itself. Here again, the cost itself is meaningless if not related to the quality, and usually, one will report the number of function evaluations needed to reach a given quality.

- However, it can happen that some target quality is never reached, or not reached within the allocated budget. In such cases, the **probability of success** comes into play to describe in more detail such situations, and is empirically computed from the existing runs of the algorithm/parameter setting.

Of course, because we are dealing with stochastic algorithms, raw indicators are almost meaningless, and summary statistics of the quality or the runtime (e.g., median and quantiles, or mean and standard deviation) should be computed over a number of independent runs. In addition to these statistics, parametric or non-parametric statistical tests need to be performed when it comes to comparing the performance of several algorithms.

From these indicators, the robustness and stability of the algorithm can be investigated, with respect to the choice of the practitioner. On the one hand, a **robust** algorithm should perform well on a variety of problem classes that might be uncovered in the future. On the other hand, a **stable** algorithm should have statistically similar performances when multiple independent runs are done on the same problem.

The indicators described above require a test bench of optimization problem instances on which to run the different algorithms or settings. And a performance measure must be defined (from the indicators above) in order to assess and compare algorithms on the test bench instances. The Next Section introduces BBOB [Han+10], a well-known test bench that was built while keeping this in mind.

3.1.1 Black Box Optimization Test Bench

The COCO/BBOB[1] framework [Han+10] is today well-known in the numerical black-box optimization community for its careful design. It contains functions of controlled known difficulties, a common API to perform experiments, and some post-processing facilities that allow easy comparisons of two to many algorithms.

The BBOB test bench contains two set of instances, the noiseless and the noisy instances. Only the noiseless instances have been used in this work, and hence the noisy instances will not be mentioned any more here. The noiseless testbed is made of 24 functions, analytically defined on $[-5, 5]^d$, with known global optima and known characteristics (e.g. separability, multimodality, conditioning, …). In particular, some of these functions have been modified from their original definition in the optimization literature in order to exhibit given difficulties. These functions have been manually classified into five classes.

- 5 separable functions (F1 to F5)[2]
- 4 unimodal functions with low or moderate conditioning (F6 to to F9)
- 5 unimodal functions with high conditioning (F10 to F14)
- 5 multimodal function with a global structure (F15 to F20)
- 4 multimodal function without global structure (F21 to F24)

The dimension of the search space is indeed a relevant characteristic of the problem difficulty, hence all functions defined in the BBOB test bench can be instantiated in any dimension. In order to avoid any possible algorithm bias, all different (independent) runs on one function are actually done on different variants of the function, obtained from the original function by a translation of the position of the optimum and – for the non-separable functions – by a rotation of the coordinate system.

The COCO/BBOB framework aims at allowing an easy and sound comparison of optimization algorithms, taking into consideration the three aforementioned type of performance indicators: the quality, the runtime and the probability to reach a performance target. Given a target value f_{target}, the optimum value of the objective function f, and a numerical precision Δ_f, the Expected Run Time (ERT) defined by Equation (3.1) is based on multiple independent runs

[1] COCO (COmparing Continuous Optimizers) is the name of the platform. BBOB (Black-Box Optimization Benchmarking) is the name of the GECCO workshops (almost yearly since 2009) around the COCO platform. Both names are used indistinctly.

[2] all other functions are non-separable.

of the algorithm at hand optimizing f. In Equation (3.1), $\overline{RT_s}$ is the mean of the runtime (number of function evaluations) of successful run, i.e., runs that successfully reached $f_{target} + \Delta_f$ before exhausting the total function evaluation budget, and p_s is the empirical probability of success, i.e., the number of runs that did reach the target at the given precision divided by the total number of runs for this function. This performance measure integrates the time to reach a desired quality and the probability to reach it, and is therefore well suited to analyze and compare the robustness of algorithms.

$$ERT(f, \theta) = \frac{\overline{RT_s}}{p_s} \tag{3.1}$$

In order to aggregate this measure over different instances, possibly of different dimensions, Hansen et al. [Han+10] proposed to use the Empirical Cumulative Distribution Function (ECDF) of the proportion of solved instances (to the target precision Δ_f) vs the number of function evaluations normalized by the dimension. Figure 3.1 shows an example of ECDF, where 31 optimizers are compared on all BBOB noiseless functions. Such visualisation allows easy comparisons of the robustness of all algorithms, and their convergence to their respective optima for a given precision.

3.2 Algorithm Configuration

3.2.1 Motivations

Traditionally, algorithms for solving computationally hard and/or expensive problems are designed in an iterative, manual process, such that the designer introduces or modifies components or the mechanism gradually. With respect to these modifications, the performances are empirically assessed and evaluated on rather small-scale experiments (relatively small and easy set of benchmark problems). Then based on these experiments, some degrees of freedom are exposed as parameters.

As the iterative design process continues, the considered space of potential algorithm designs can quickly grow, with the risk to expose too many parameters than what the average practitioner can handle. For example, CPLEX has over 80 user controllable parameters in its 10.1 version. As a result of such approach, the choice of the most appropriate parameter configuration requires expert knowledge about the internal heuristics of the algorithm and their interactions, which is not necessarily expected from a typical practitioner, nor even from the algorithm designers themselves [Hut+07].

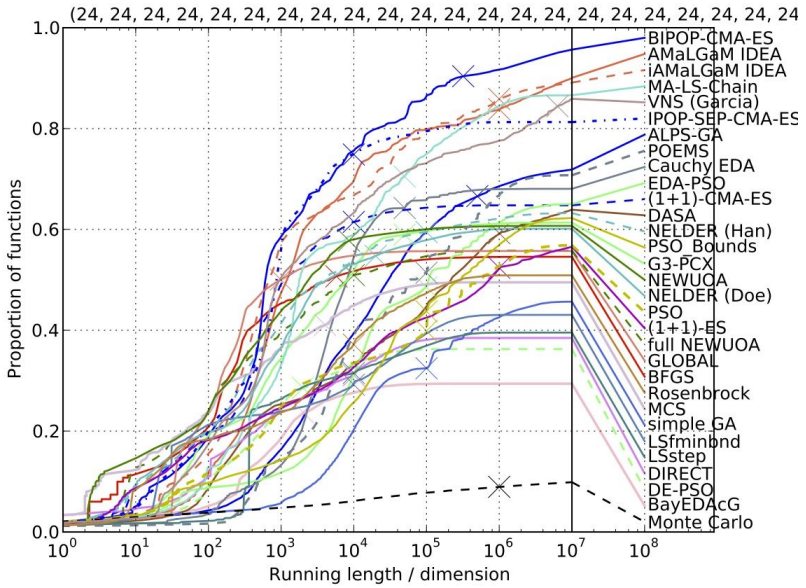


Figure 3.1: Example of ECDF comparing the performances of 31 optimization algorithms on the whole BBOB benchmark for a precision of 10^{-8}.

As a result, some combination of parameter settings is often manually chosen, thanks to some expert knowledge based on very few assumptions, and without any consideration of possible characteristics of the problem, leading to unjustified choices and generalizations. By contrast, in many domains, automated mechanisms more and more tend to outperform humans on complex high-dimensional optimization task with many interdependent decisions. In particular, the automated mechanism for parameter configuration has been demonstrated to be efficient in different fields, e.g. operational research, on SAT problems [HHL11a], in Constraint Programming, and on continuous domains [LS13b; LS13a; SE10b].

3.2.2 Notations and Problem Definition

In its simplest form, the Algorithm Configuration problem can be simply defined as follows: given an algorithm and its parameters, and a set of problem instances (aka training instances), we want to find the parameter values that maximize the performance of the algorithm on the training instances. This section will

concentrate on the first issue – find the best parameter setting minimizing some aggregated performance indicator on the training instances, leaving out the generalization issue for further discussion (see Section 3.2.4). This problem can be seen as an optimization problem in the space of parameter configurations, as described in Figure 3.2.

Let us first introduce the notations, and more formally define the Algorithm Configuration problem as a Black Box Optimization problem. Let $\mathcal{A}$ be an optimization algorithm, with parameters $\theta = (\theta_1, \ldots, \theta_k)$. Each parameter θ_i is defined on domain Θ_i, and let $\Theta \subseteq \Theta_1 \times \ldots, \Theta_k$ denote the domain of all possible parameter configurations. Note that domains can be continuous or discrete, bounded or unbounded in the continuous case, finite or infinite in the discrete case, ordinal (discrete and ordered) or categorical (discrete and unordered). Then $\mathcal{A}(\theta)$ denotes the instantiation of the algorithm $\mathcal{A}$ with the parameter configuration $\theta \in \Theta$.

Let $\{\pi_1, \ldots, \pi_n\}$ be the set of training instances. Running an algorithm $\mathcal{A}$ with parameters θ on some instance π has a cost $c(\theta, \pi)$ (computed based on runtime, solution quality, ..., see Section 3.1), to be minimized (without loss of generality). The algorithm to be configured can be deterministic (e.g. Constraint Programming or deterministic SAT solver,...) or stochastic (e.g. Evolutionary Algorithms, Machine Learning algorithms, ...).

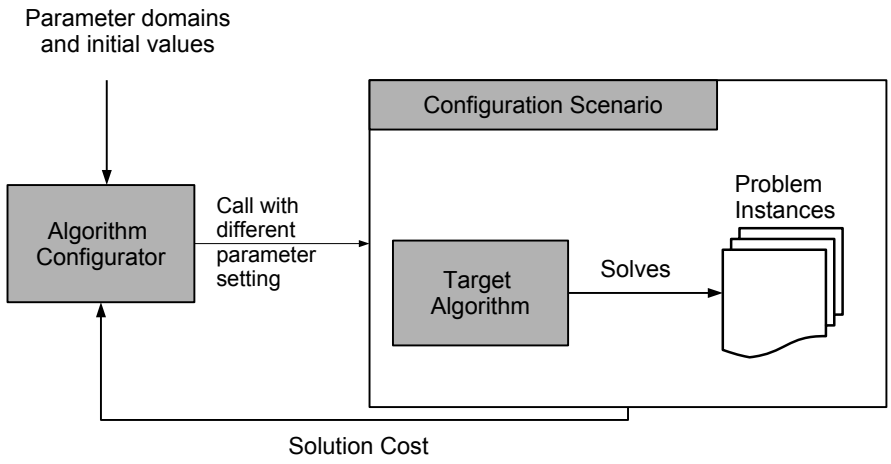


Figure 3.2: Schema of the general Algorithm Configuration procedure.

In the deterministic case, we simply want to find the optimal parameter configuration $\theta^* \in \Theta$ such that

$$\theta^* = ArgMin_{\theta \in \Theta} \sum_{i=1}^{n} c(\theta, \pi_i)$$

By contrast, when the algorithm to configure is stochastic, the function c is replaced by a stochastic process, and the goal is to find the parameter configuration $\theta^* \in \Theta$ that minimizes some statistics $\tau(c(\theta, \pi))$, e.g. its mean or its median, that can only be empirically approximated by running several independent runs of $\mathcal{A}$ with parameters θ on π. The problem then reads

$$\theta^* = ArgMin_{\theta} \sum_{i=1}^{n} \tau(c(\theta, \pi_i)) \tag{3.2}$$

The main drawback of this approach, for a given set of training instances, is its computational cost: in order to compute the statistics τ for one parameter setting θ on one instance π, one needs to actually run algorithm $\mathcal{A}$ with parameters θ on π several times[3] (the more the better for estimating τ). And this already huge cost increases linearly with the number of instances in the training set.

3.2.3 Algorithm Configuration Approaches

3.2.3.1 Search based Methods

The most straightforward approach to the Algorithm Configuration problem as set above is to use some optimization algorithm on the optimization problem defined on the parameter space by Equation (3.2).

A first attempt can be traced back to Grefenstette [Gre86], who proposes to use a Genetic Algorithm for tuning the parameters of a Genetic Algorithm, demonstrating he could improve the results of his GA by tuning the population size, mutation and crossover rates, hence pointing out the need for …parameter tuning for GAs.

In the early 1990's, Gratch and DeJong [GD92] perform a hill-climbing search in the space of parameter configurations, improving the best parameter configuration by iteratively gathering information in the neighborhood of the best parameter configuration at each iteration; they demonstrate the efficiency of their methods on a scheduling problem [GC96]. In addition, Minton [Min93; Min96] pro-

[3] Note that it will depend on the practitioner want to split the overall budget.

poses a method to generate domain-specific LISP programs, by adapting generic heuristics to the problem domains, by using some search methods to evaluate programs by running them on different problem instances of a given distribution.

However, Terashima-Marín, Ross, and Valenzuela-Rendón [TRV99] are notably the first to introduce a genetic algorithm for tuning the parameter of a Constraint Satisfaction algorithm, for solving large-scale exam scheduling problems. In this work, the authors designed and configured an algorithm with seven categorical parameters, finding different optimal parameter configurations for each of twelve problem instances.

Coy et al. [Coy+01] introduced a two stages approach using a fixed training set of problem instances. For each problem instance i, they aim at finding a good parameter configuration by a combination of Experimental Design with a full factorial or fractional factorial [BH61] (see [AW00; Kle05] for an overview of the Experimental Design methods) together with a gradient based method. The second stage is to combine all the parameter configurations by averaging their values. This method of course only applies to continuous parameters, and furthermore assumes that the average of the optimal parameters for different instances will be a good choice in the general case (see Section 3.2.4).

A similar approach implemented in the CALIBRA system [AL06] starts by evaluating each parameter configuration of a full factorial design with two values per parameter, and iteratively directs the search to more promising regions of the parameter space. When a local optimum is found, a coarse grained search is restarted. This method shows some limitation, in particular for numerical parameters, as it only supports integer arithmetics, requiring a discretization of the parameters to the desired precision. It is also limited in the number of parameters it can handle at the same time (up to 15 at most).

In the context of a two-stage search process, ParamILS [Hut+09c] proposed a method based on an iterated local search [LMS10], and an intensification phase to find promising algorithm parameters. The stochastic local search is initialized with the default parameter setting that is perturbed iteratively, and keeping in memory the different parameter settings that have already been tested. The intensification part aims at re-evaluating and comparing most promising parameter setting to find the parameter setting that dominates others; then the dominating parameter setting is used as candidate solution to the stochastic local search. ParamILS has empirically been demonstrated to perform well on several solvers, e.g., SAT or MIP solvers (see [Hoo11] for an overview of the successful applications of ParamILS). Recently, Blot et al. [Blo+16] proposed MO-ParamILS, a multi-objective variant of ParamILS, that aims at finding a parameter setting with respect to more than one performance indicator, which involves finding a

Pareto set of configurations of a given target algorithm that characterizes trade-offs between multiple performance indicators. They empirically demonstrated that it produces good results in several bi-objectives algorithm configuration scenarios compared to a baseline obtained from using the original version of ParamILS, that focus on single-objective algorithm configuration scenarios.

Dealing with global optimization algorithm, Tolson and Shoemaker [TS07] proposed a method that could be understood as a variable neighborhood local search mechanism. The method is closely related to a simulated annealing algorithm, defining a probability for selecting the parameters to be perturbed. Despite promising results, empirically demonstrated when the number of call of the objective function is limited (1000 evaluations), this empirical study is limited to 30 real valued parameters, and only validated on 4 continuous black box problems.

In addition, some works were proposed in the numerical optimization community: Audet and Orban [AO04] proposed the mesh adaptive direct search (MADS) algorithm, designed for numerical parameter configuration. They empirically demonstrate to converge to a local optimum of the cost function (here, the number of function evaluations), for unconstrained optimization problems [GOT03]. In order to reduce the parameter tuning cost when considering a collection of problems, they propose to use the problems from this collection that have a smaller computational complexity to evaluate the performance of the algorithm to be tuned, though sometimes the best parameter setting for the easy problems might result in worse performance on more complex problems from the same collection.

The REVAC approach [NE07] relies on an EDA (Estimation Distribution Algorithm). It is based on information theory to measure parameter relevance, such that instead of estimating the performance of an algorithm for different parameter values or ranges of values, the method estimates the expected performance when parameter values are chosen from a probability distribution maximizing Shannon entropy. This approach has been empirically validated by optimizing the parameter settings of CMA-ES [SE10b] (see also Section 2.3.3).

In a similar context, Ansótegui, Sellmann, and Tierney [AST09] propose GGA (Gender Based Genetic Algorithm), yet another algorithm configuration method based on GAs. They propose to exploit genders in Genetic Algorithm, in order to deal with multiple problem instances, resulting in a robust parameter setting. Thanks to a tree representation of parameters, they deal with different type of parameters (continuous or categorical), and aim at optimizing independently parameters [Tie]. In contrast to [Hut+09b], which focuses on most promising parameter setting on all the problem instances, [AST09] aims at racing parame-

ter configurations over an increasing subset of problem instances. However, the results are limited to small problem instances, without fully exploiting the parallelization of GAs (use of only 8 workers).

Finally, Smit, Eiben, and Szlávik [SES10] propose to use multi-objective evolutionary algorithm for Algorithm Configuration. They empirically demonstrate that multi-objective optimization can be used for finding a robust parameter settings: each objective is the performance on each problem instance. This work is a preliminary work, allowing to get insights of the robustness of GA on different problem instances, and observing parameter setting that improve the performance of a GA on two problem instances (Sphere and Rastrigin function). Based on the same idea, Smit and Eiben [SE11] proposed BONESA, a new method that relies on multi-objective optimization and a model based approach (see next Section), to find a robust parameter setting across multiple problem instances. Nevertheless, the method was empirically investigated on an artificial performance landscape in which they can control different aspect. This results in a tool (available at https://sourceforge.net/projects/tuning/ for analyzing and have a deeper understanding of EA.

3.2.3.2 Model based Optimization

Among optimization methods, model-based approaches build a model (aka response-surface) that tries to predict the value of the objective function over the whole search space based on previously evaluated points. Though also pertaining to search-based methods, they share sufficiently specific characteristics to be presented together, apart from the methods surveyed in a previous Section.

Model-based methods are iterative, combining a response surface model with a criterion for the selection of the next design point. The choice of the response surface model is one of the key factors for estimating the most promising region of the parameter search space. A prominent response surface model, used in the statistics fields [JSW98; Sac+89; SWN03], is a combination of linear models, completed by Gaussian Process (called *kriging* model in the statistics field) on the residual errors of the linear model. This model is also known as the DACE (Design and Analysis of Computer Experiments) model, popularized by [Sac+89].

The selection of the next design point directly depends on the criterion to compare candidate parameter configurations. One popular criterion is the expectation of improvement over the incumbent solution θ (where the expected improvement is computed from the response at θ with respect to the current model), proposed by Mockus, Tiesis, and Zilinskas [MTZ78].

The EGO (Efficient Global Optimization) algorithm proposed by Jones, Schon-

lau, and Welch [JSW98] combines the expected improvement criterion and the DACE model with a branch and bound method, making it a particularly popular framework for deterministic black box optimization with a low budget in the field of statistics.

The EGO algorithm has been extended by three main works [Hua+06b; Hua+06a; WSN00]. The first two mainly deal with noisy optimization problems. SKO [Hua+06b; Hua+06a] extends the EGO framework by adding noise to the DACE model and augmenting the expected improvement criterion. SPO [BLP05; Bar09] computes empirical summary statistics for each design point and fits a noise-free Gaussian process model to the values of these statistics. Then, it increases over time the number of runs based on this statistic and gradually improves the accuracy of the model and determines the most promising parameter values. The third work Williams, Santner, and Notz [WSN00] aims at constructing a Gaussian process model that predict a response value with respect to environmental conditions, like noise, using its model to optimize the marginal performance across these environmental conditions. According to Hutter [Hut09], in the context of algorithm configuration, this approach is particularly efficient to optimize the performance across a set of different problem instances. This method is, in particular, applicable when the practitioners aim at finding a parameter setting with average performances on a set of problem instances, hence questioning the generalization of the parameter setting (see Section 3.2.4).

In spite of promising results, several drawbacks can be pointed out for these approaches:

- They are limited to continuous parameters. But, for instance, in the Evolutionary Computation community, several algorithms have discrete or categorical parameters, e.g. Differential Evolution with the strategy parameter (see Section 2.4 for the details of parameters).
- They are limited to noise free problems, or problems with Gaussian distributed noise.
- they are limited to parameter tuning of a single instance, or the average of multiple instances.
- Last but not least, these methods scale poorly, with a cubic time complexity of the Gaussian process model with the number of sample points.

Main approaches built on top of the EGO framework rely on a Gaussian Process model. But different methods exist to model the response surface of the performance function over the parameter configurations domain, for example some fractional factorial design [RK07]. Also, in Srivastava and Mediratta

[SM05], a decision tree classifier is used to partition the parameter configuration space. But this approach remains limited, only able to deal with around 200 parameter configurations, and based on a simple comparison to the default parameter configuration or the best one.

Bartz-Beielstein and Markon [BM04] empirically investigate the regression methods and their performances with respect to their properties and limitations, and conclude that tree-based regression techniques have comparable results to the DACE model. Regarding tree-based methods, Hutter, Hoos, and Leyton-Brown [HHL11b] proposes the SMBO (Sequential Model-Based Optimization) approach, that relies on a Random Forest regression model. In contrast to EGO based methods, SMBO [HHL11b] is able to deal with noisy optimization, in particular when the algorithm to be tuned is stochastic. In addition, it is particularly efficient for the parameter tuning across multiple problem instances, thanks to some characteristics of each problem instances given by the practitioners and that are used during the tuning process in order to assess the expected improvement of parameter configurations on all problem instances. As Random Forest regression models are trained iteratively with batches of samples, this method is particularly efficient when the number of samples and the number of parameters of each sample is large. Hutter, Hoos, and Leyton-Brown [HHL11b] introduced at the same time SMAC (Sequential Modeling Algorithm Configuration), an implementation of SMBO, that demonstrated promising results in many research topics, such as Machine learning with Auto-Weka [Tho+13], SAT [HHL11b] and in particular continuous black box optimization[HHL13; TF15].

3.2.3.3 Racing Algorithms

Another approach for the algorithm configuration is the adaptation of the racing algorithms introducing the Hoeffding races by Maron and Moore [MM93; MM97] for the selection of machine learning models. Later, in [Bir+02] is proposed a new racing (F-Race) method relying on a non-parametric statistical test to compare the parameter configurations for stochastic local search algorithms, completing the empirical study and the validation of the method by Birattari and Dorigo [BD04].

Given an algorithm $\mathcal{A}$, and a finite set of parameter configurations Θ, F-Race iteratively runs the target algorithm with parameter configurations on a number of parameter configurations θ sampled from Θ according to a distribution $\mathcal{T}$. After each iteration, F-race performs a non-parametric Friedman test (F-test) in order to check if there is any significant difference among the results of parameter configurations, and discard bad parameter configurations. This is iterated until a single parameter configuration is left in the race, or after a cutoff time κ is

reached.

Although F-Race demonstrates good performance [Bir05], this approach is limited to tuning algorithms with an enumerable (and not too large) parameter space, or dealing with parameters with small sensitivity. To this end, a variant was proposed by Balaprakash, Birattari, and Stützle [BBS07], the iterated F-race, or I-Race, dealing with new subsets of parameter configurations at each iteration, competing with the best observed parameter configurations, while still discarding bad parameter configurations. An overview and empirical comparison of the two methods is proposed by Birattari et al. [Bir+10], giving promising results for tuning different stochastic algorithms. In addition, the iterated F-race demonstrated interesting and promising results on the continuous black box optimization, e.g. on CMA-ES [Lóp+11] or with different experimental setting [LS13b].

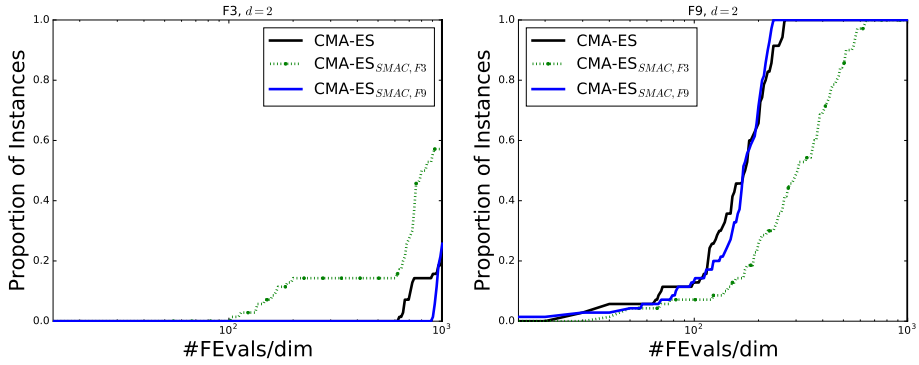
3.2.4 The generalization issue

In the simplest case, Algorithm Configuration aims at finding parameter setting of a algorithm for one problem instance π[4]. This results in a *specialist* parameter setting (in accordance with Smit and Eiben [SE10a]), such that the parameter setting delivers good performances at solving π, but without any claim or indications of its performance on other problem instances, even if they look similar. Indeed, such settings are suitable if the practitioner is only interested in solving π, but at a huge cost when a new problem might be solved. This opens the issue of the robustness of the *specialist* parameter setting. As briefly discussed in Section 3.1, the robustness of a parameter setting is related to the ability to deliver good performance when some changes are observed on the instance, e.g. scaling or rotation.

As an example of *specialist* parameter setting, Figure 3.3 shows the ECDF on test functions F3 (Rastrigin) and F9 (Rosenbrock) of the BBOB test bench. We compare the performance of the default parameter setting of CMA-ES to two *specialist* parameter settings optimized using SMAC [HHL11b], respectively on F3 and F9. We observe that a *specialist* parameter setting generalizes poorly to another function.

In general, practitioners, in particular algorithm designers and experts, are interested in algorithms that perform well on several problem instances. The algorithm configuration process is then performed on some training set $\pi_1, \ldots, \pi_n$, with the aim of delivering a robust or *generalist* parameter setting (again following Smit and Eiben [SE10a]), i.e., a setting that will perform well on the whole class of

[4]This is a very special case of Algorithm Configuration



(a) F3 Rastrigin **(b)** F9 Rosenbrock

Figure 3.3: Examples of Performances of *specialist* parameter setting: a parameter setting tuned on F3 Rastrigin (CMA-ES$_{SMAC,F3}$) and one tuned on F9 Rosenbrock ((CMA-ES$_{SMAC,F9}$) of the BBOB test bench. Both *specialist* parameter setting are compared to the default parameter setting of CMA-ES on F3 (on the left) and F9 (on the right).

which $\pi_1, \ldots, \pi_n$ are representatives. While most Algorithm Configuration tools and methods that are discussed in Section 3.2.3 cannot perform parameter tuning for multiple instances, some works[Coy+01; AST09; GOT03; WSN00; Lóp+11; HHL11b; SES10; SE11] explicitly aim at finding a robust parameter setting given a training set of instances.

Many approaches were proposed to deal with multiple problem instances, from multi-objective optimization [Dré09; Blo+16] to aggregation of performances, as described by Equation (3.2) in Section 3.2.2. These works pointed out relevant issues of finding a robust parameter setting. Another approach is proposed by Williams, Santner, and Notz [WSN00] that aims at maximizing the marginal performances across the training instances, instead of favoring averaged performances.

Gould, Orban, and Toint [GOT03] point out the need to choose the appropriate problem instances that are relevant to a class or type of problems. It aims at finding a robust parameter setting that can be used for simple to more complex problem instances that lie in the defined class.

Figure 3.4 shows typical results of a robust parameter setting found with SMAC [HHL11b], such that the ECDF is aggregated by classes of the BBOB

test bench, and shows the performance of robust parameter setting computed for each classes F6 to F9 and F10 to F14 (each robust parameter setting has been found by brute force here, after trying 1000 different parameter configurations). We observe that robust parameter settings tuned for a given class of problems perform better for this class while their performances are worse on others: they are specialized for their target class.

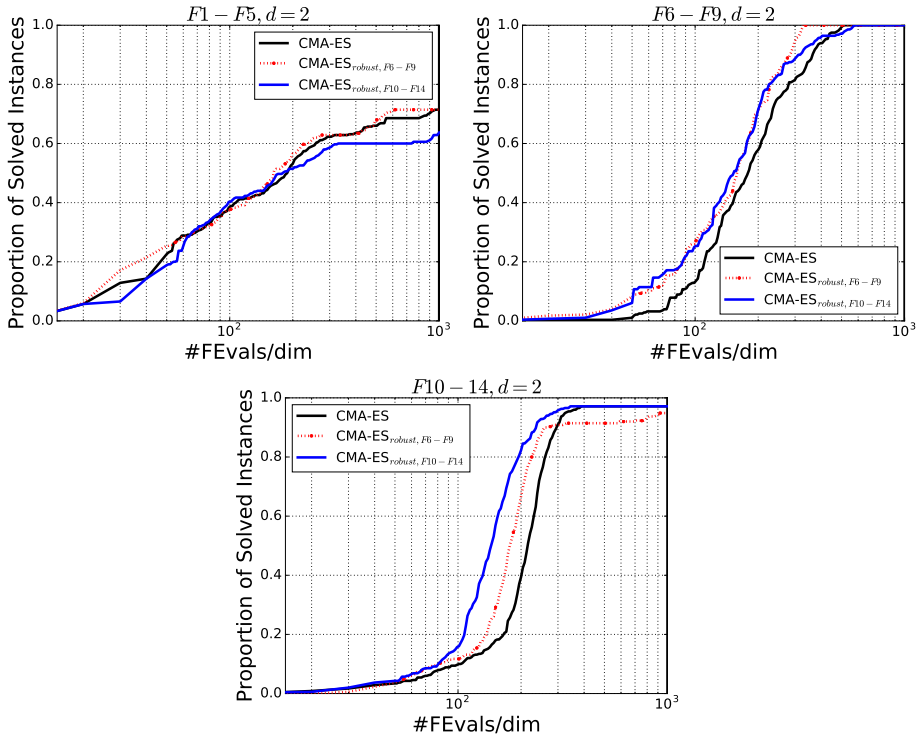


Figure 3.4: Example of Performance of robust parameter settings: CMA-ES$_{robust,F6-F9}$ is a robust parameter setting for the test function F6 to F9 of BBOB, CMA-ES$_{robust,F10-F14}$ is a robust parameter setting for F10 to F14 test functions; they are compared to the default parameter settings. Each plot is an ECDF aggregated by classes of the BBOB test bench for $d = 2$.

One way to overcome these difficulties is to rely on some more precise description of an instance than just a class – or to define very fine-grained classes by using more than only a few descriptors (separable/non-separable or unimodal/-

multimodal – see Section 3.1.1). This is the underlying idea of the Per Instance Algorithm Configuration (PIAC).

3.3 Per Instance Algorithm Configuration

In order to learn a specialist parameter setting for each objective function (see discussion above), one needs to solve a meta-optimization problem in the space of parameter settings for a given algorithm $\mathcal{A}$, incurring a huge computational cost. From this point of view, the PIAC approach can be seen a solving this meta-optimization using a surrogate model to replace the actual evaluation of parameter settings, i.e., runs of an algorithm $\mathcal{A}$ on the target function f: after the surrogate model has been learned, it can be used to derive the optimal parameter setting. However, learning such surrogate for each objective function would still be very costly. Going one step further, the idea is here to learn a surrogate model that would also include the characteristics of the objective function at hand. Assuming the existence of a representation for the objective functions, this is, again, a supervised learning problem.

3.3.1 Methodology

This approach was initially proposed by Leyton-Brown, Nudelman, and Shoham [LNS02], who tackled the combinatorial domain, and empirically analyzed the empirical *hardness* of combinatorial problems with respect to problem features derived from their descriptions. Different works [Hut+06; Xu+08; Hut+14] have later investigated the application of this approach to the SAT domain, where they learned to predict the runtime of parameter configurations of a SAT solver on different problem instances.

PIAC aims at using some characteristics of objective functions, aka *features*, that can be computed without any domain knowledge, to derive an *Empirical Performance Model* for a given algorithm that maps the features and the parameters to the performance of the algorithm. Learning such a model requires a large example base of algorithms performances on known objective functions, but these can be acquired once and forall off-line.

3.3.1.1 Outline

This approach can be described as a two-phase process, summarized in Figure 3.5:

1. The learning/training phase consists in learning a model of performance (e.g. the best quality reached with a given allowed time, the time needed

to reach a given quality, ...) —the runtime in the above-cited works— of a set of parameter configurations of an algorithm on a set of known problem instances described by their features. This model of performance maps pairs of parameter configuration, instance (described by their features) to the performance of the algorithm using this configuration on the instance.

2. The second phase consists in using this model to predict the best parameter setting of the algorithm for a new, unknown instance. Given this new instance, its problem features are computed. For any parameter configuration, the runtime of this configuration on this new problem instance can be predicted by the model of performance, and the configuration that minimizes this predicted runtime is selected.

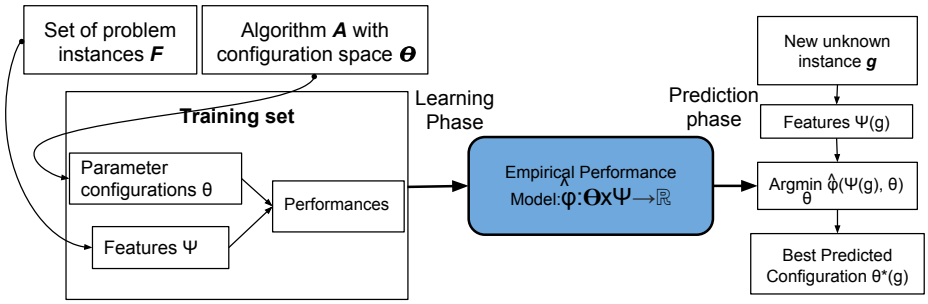


Figure 3.5: schema describing the process of the Algorithm Runtime Prediction for offline parameter setting based on problem features..

3.3.1.2 Problem Definition and Notation

This section introduces more formally the context of the work and the notation. Note that the notation is the same as in Section 3.2.2, recalled here for the sake of completeness, though complemented with the specific notation for PIAC and feature-based Algorithm Configuration.

The context is that of the optimization of functions[5] $f : \Omega \mapsto \mathbb{R}$. Given an algorithm $\mathcal{A}$ together with its control parameters $\theta \in \mathbf{\Theta}$, we assume that the objective function can be described by some *features* $\psi \in \mathbf{\Psi}$.

[5]at this point, the approach is not specific to continuous optimization, and the definition domain Ω of the objective functions can be any space.

The goal of Features-based Algorithm Configuration is to automatically find, for a given objective function f, described by its features ψ^f, the best possible *configuration* of $\mathcal{A}$, i.e., values $\theta_f^* \in \mathbf{\Theta}$ such that running $\mathcal{A}$ with parameters θ_f^* on f leads to optimal performance w.r.t. a given *performance measure* φ.

3.3.1.3 Learning an Empirical Performance Model

The first step aims at fitting a joint model, called the *Empirical Performance Model* (EPM). The EPM $\widehat{\varphi}$ approximates the true performance function φ on $\mathbf{\Psi} \times \mathbf{\Theta}$.

The algorithm $\mathcal{A}$ is run with different parameter configurations $\theta_j \in \mathbf{\Theta}$ on different objective functions f_i, described by their problem features ψ_{f_i}. This allows to compute the exact values $\varphi(\psi_{f_i}, \theta_j)$ for different pairs (i, j). Ideally, all the parameter configurations θ_j may be tried for all f_i, but this is by no means mandatory, though a sparse training set might lead to a loss of generality of the EPM.

The set of all $((\psi_{f_i}, \theta_j), \varphi(\psi_{f_i}, \theta_j))$ can then be considered as a training set for some supervised learning problem, and can be input to any learning method, here any regression method. The result will be the model $\widehat{\varphi}$ for φ.

Note that, though such an approach might be expensive, especially when gathering the training set, this step is performed only once, and hence the computational cost is not a (too) critical issue.

3.3.1.4 Using the Empirical Performance Model

Once the EPM has been built, it can be used for any new unknown objective function g that needs to be optimized with $\mathcal{A}$.

First, the features $\psi(g)$ are computed for instance g. Then, the optimization problem described in Equation (3.3) below is solved: the problem features are set to $\psi(g)$, and the goal is to optimize $\widehat{\varphi}(\psi(g), \theta)$ in the space of parameter configurations $\mathbf{\Theta}$, leading to the empirical optimal parameter configuration $\widehat{\theta(g)^\star}$ of $\mathcal{A}$ for g.

$$\widehat{\theta^\star} = \operatorname{argmin} \widehat{\varphi}(\psi(g), \theta) \tag{3.3}$$

The cost of the optimization problem (3.3) is usually neglected, as it does not involve any call to the objective function. But the computation of problem features $\psi(g)$ remains an important issue, as the quality of these features greatly influences the optimization problem. Hence the computation of problem

features is of utter importance. Both the choice of the problem features and their computation are introduced and discussed in the following Chapter.

3.3.2 State-of-the-art

The underlying idea behind PIAC – for any given algorithm, there exists a relationship between the space of problem instances and the space of parameter settings of this algorithm, that associates to each instance the best setting for a given performance measure – is not new. Knuth [Knu75] introduced a Monte-Carlo approach to estimate the size of a search tree, for estimating the hardness of an instance. Similar approaches, based on the Monte-Carlo approach were proposed by Lobjois and Lemaître [L+98] and Kilby et al. [Kil+06] for selecting the fastest algorithm to traverse the entire search tree.

A portfolio-based algorithm selection approach, using a predictive model of the algorithm runtime, is introduced by Leyton-Brown et al. [Ley+03] and Leyton-Brown, Nudelman, and Shoham [LNS09]. Called *empirical hardness model*, these models predict the hardness of a new unseen problem instance from some computed features. This approach has been successfully applied on SAT problems [Nud+04], and even on randomized algorithms [Hut+06]. Following this approach, Xu, Hoos, and Leyton-Brown [XHL07] introduced the hierarchical hardness model, applying a classifier to predict the type or class of the problem, and training a specific hardness model for each class. These portfolio approaches repeatedly demonstrated successful results, from their first use [Ley+03] to the later extension [Xu+08; XHL07; Xu+07], in particular in SAT competitions.

If the algorithm selection problem on a per instance basis is well investigated, this is not so much the case for PIAC. In the SAT community, Patterson and Kautz [PK01] introduced the Auto-WalkSAT algorithm. This approach, based on easily computable features (e.g. the "invariant ratio"[MSK97]), empirically found parameters that are 10% less than the optimal parameter values for the noise parameter of WalkSAT. It has been reported that the proposed approach found almost optimal settings of the noise parameters for heterogeneous problem classes, whereas it failed on some problem instances where the relationship between invariant ratio and the noise parameter doesn't hold.

Another approach, proposed by Gebruers et al. [Geb+05], uses a case based reasoning method [Kol14] in order to discover the best parameter configuration of a constraint programming algorithm on a per-instance basis. The parameters being optimized include problem modelling, propagation, the variable selection heuristic, and the value selection heuristics. But, they did not exploit the structured parameter configuration space, resulting in a simple algorithm selection problem. Furthermore, Gebruers et al. [Geb+05] does not specify the number of

parameter configurations, nor the features that were used for the experimental settings. He reports that the case based reasoning method performs better than C4.5 (a supervised machine learning algorithm based on decision trees) with default parameter setting to solve the social golfer problem (a well-known constraint programming problem)

In [HH05; Hut+06], a similar approach to the empirical hardness model is used on two SAT solvers. Their approach is based on a linear regression, fitting a joint model of problem features and algorithm parameters (continuous parameters), as described in Section 3.3.1. The authors empirically demonstrated that the predicted parameter setting outperformed the best fixed parameter setting, on a mixed benchmark.

Following the work introduced in [HH05; Hut+06], Hutter et al. [Hut+14] empirically investigated different regression methods (Random Forest, Gaussian Processes, Ridge regression, …) to learn an empirical performance model. This empirical study demonstrated that Random Forest regression [Bre01] (known in the machine learning community for its robustness against overfitting) outperformed other regression methods. Extending this work, the authors empirically demonstrated that this approach can be efficiently used for different domains and/or algorithms, validating the method on SAT, MIP and TSP domains. Hutter et al. [Hut+06], Xu et al. [Xu+08], Xu, Hoos, and Leyton-Brown [XHL10], and Xu et al. [Xu+11] successfully tackled the algorithm selection/configuration problem, e.g. with SATZilla or Hydra, and the results are summarized in [Hut+14].

Regarding the use of problem features within the Algorithm Configuration, Kadioglu et al. [Kad+10] proposed ISAC, that relies first on some clustering methods, and then learns the best parameter configurations for each cluster, using problem features. This method aims at finding a *robust* parameter setting for each automatically determined cluster. The next phase tackles the Per Instance Algorithm Configuration, by using problem features and optimizing the cluster-based model. One advantage is that any optimization algorithm can be used on the model, thus the approach is able to handle any number of parameters, whereas the cluster based approach would suffer from a much more intense curse of dimensionality because of problem features. Another drawback is that the method relies on some cluster based meta-optimization, that might give poor results as the number of cluster increases, hindering their homogeneity with respect to parameter setting. By contrast, the approach used in our work directly takes into account the features of the problem instance at hand, thus avoiding some generalization due to some ill-defined clusters, e.g. clusters that embeds problem instances with too different features.

3.3.3 PIAC for continuous domains

Previous Section surveyed some works on Per Instance Algorithm Configuration for discrete domains. However, the PIAC approach remains under-investigated in the continuous optimization domain, with the notable exceptions of [AMT12; MKH12; Bos+15].

Based on computed features (through local search or randomized samples), Abell, Malitsky, and Tierney [AMT12] proposed to use the already-mentioned ISAC method [Kad+10] to find the best solver for problem instances of the BBOB benchmark (Section 3.1.1). This empirical study is a preliminary experiment to demonstrate the use of continuous black box problem features to predict the best algorithm among 27 different solvers. The results have been empirically validated using a 10-fold cross validation procedure, but the authors make no mention if it was done with replacement, hence limiting in this case the results. In addition, the experimental settings account for different groups of problem features, and it was observed that a limited number of features can be used to find a good solver. But the main drawback of this work is that the empirical study relies on problem features whose computation requires a large number of calls of the objective function: the authors are well aware of the problem, and even discuss the issue of accurately sampling the objective function for a good accuracy of problem features, but conducted their empirical study with the maximal size of 200 hill climbings of 400 function evaluations, 8×10^4 functions evaluations. By contrast, our work aims at taking into consideration problem features that can be computed from one single (and hopefully small) sample set, focusing on the cost of the computation of problem features, in order to cope with black box optimization limitations, i.e. an overall limited budget of evaluation functions.

In [MKH12], the feature based algorithm configuration problem is tackled by performance prediction approach, using an empirical performance model, with a case study on the CMA-ES algorithm. The authors used problem features based on samples of candidate solutions to predict the best parameter setting for CMA-ES (the population size λ, and if the mirrored variant is used[Bro+10]) on each function of the BBOB benchmark[Han+10]. In contrast to [Hut+14], Muñoz, Kirley, and Halgamuge [MKH12] train an empirical performance model by using a multi-layer neural network method (known for its overfitting properties over the training set). It shows promising empirical results, by predicting the best parameter setting to CMA-ES, on the BBOB test bench. However, here again, this study considered problem features that require large samples of candidate solutions, and it should be more deeply investigated when objective functions are expensive and the computational budget is limited.

The work by Bossek et al. [Bos+15] proposes to directly learn a mapping

between problem features and parameter settings using the SMBO approach described in [HHL11b]. The authors consider the work on Expected Profile Optima [Gin+14] to train a model based on the mapping of problem features and parameter settings. It is then optimized as described in the general model (see the Figure 3.5). This approach is validated on CMA-ES, but only the population size λ is tuned, based on a single problem feature, the dimension d of the problem. The main contribution of this work is the use of the Expected Profile Optima, aiming at finding the best parameter setting that maximizes the performance and robustness of the algorithm. Despite promising results, this approach is investigated for only one parameter and one feature; which greatly limits the description of problems properties, as it is discussed in Chapter 4.

CHAPTER 4

Continuous Black Box Problem Description

This thesis has a special interest in the Per Instance Algorithm Configuration that relies on *features* that describe problem properties such that a relationship exists between them and parameter settings and their performances. This Chapter introduces the description of optimization problems, Section 4.2 gives an overview of *fitness landscape analysis*, surveying a set of well-studied properties of optimization problems. These properties usually come with some metrics used to quantify them, and we study these properties from an operational point of view. Next, in Section 4.3, we focus on *features*, which are operational quantities assessing some properties of the landscapes, and we can use to describe continuous optimization problem instances and their difficulties. Finally, we conclude and discuss open questions related to the *features* and their use for Per Instance Algorithm Configuration.

4.1 Motivation

The fitness landscape metaphor is widely used for the description of problems. Wright [Wri32] is usually cited as the originator of this concept in the computational community. On the other hand, physicists have also been using the metaphor of energy landscapes for decades. Although the analogy to real landscape can give an intuitive understanding of the problem properties and hardness, and might give hints about how an algorithm has to operate on the function at hand, this analogy remains imprecise and can be misleading in high dimensions. Nevertheless, Jones [Jon95] and Stadler and Happel [SH99] propose a formal definition of fitness landscape that is today widely accepted by the evolutionary

theory community, and beyond.

Given an objective function f, it is clear that the bare fitness values assigned to an individual are not enough to precisely describe the problem at hand, as they do not relate the points of the search space to one another. Some notion of neighborhoods on the search space must be added. However, because the goal of fitness landscape analysis is to have a better understanding of algorithm performances on related optimization problem instances, in the general case, the characteristics of the target algorithms, and in particular the *variation* operators they use to explore the search space, should be taken into account to define the neighborhood relationships under study. Nevertheless, when it comes to continuous optimization problems, the 'natural' Euclidean topology is generally sufficient to describe and analyze the behavior of most optimization algorithms, and will be the only one considered here.

The goal of fitness landscape analysis is twofold. On the one hand, the analysis is used in order to help getting a deeper understanding of the intrinsic properties of the problem at hand, and possibly whole problem classes [MF98; FRP97]. In particular, fitness landscape analysis can help to identify similarities between problems, and hence to define problem classes in a grounded way. On the other hand, the idea is to derive some recommendations in terms of algorithms choice, or parameter choice for a given algorithm, for different problems, or classes o problems, from measurements of problem difficulties and characteristics.

The following Section will briefly introduce the bases of fitness landscape analysis in the general case, while the following section will detail the specific features that have been proposed for the continuous case.

4.2 Fitness Landscape Analysis

On this section, after giving some background definitions and set our notations, we will describe different properties of fitness landscapes that have been defined in the literature to describe more accurately the difficulties that an optimization algorithm will have to face when trying to solve the corresponding optimization problem.

4.2.1 Notations and Definitions

Given an objective function $\boldsymbol{f}$, defined on some search space $\mathcal{S} \subset \mathbb{R}^d$, A fitness landscape $\mathcal{F}$ is defined by the objective function $\boldsymbol{f}$ and a distance $\boldsymbol{d}$ defined on $\mathcal{S}$ [RR03a]. Whereas, in discrete search spaces, the notion of distance is intimately linked to that of variation operators, as discussed above, $\boldsymbol{d}$ will be

here the standard Euclidean distance defined on $\mathbb{R}^d$. Whenever possible, we will give most definitions in the general case, and instantiate them in the continuous case to discuss the specificities.

In addition to the distance function and the $\boldsymbol{f}$, fitness landscape analysis relies on simple notions that express the relationship between candidate solutions:

- The local σ- neighborhood of a point $x \in \mathcal{S}$, defined in Equation 4.1, is the set of all points of the search space that are at distance less than σ of x.

$$N_\sigma : \mathcal{S} \to 2^{\mathcal{S}},\ x \mapsto \{y | \mathbf{d}(x, y) \leq \sigma\} \tag{4.1}$$

  In the continuous setting, the σ-neighborhood of x is nothing but the open ball of radius σ centered on x.

- The connectedness between points of the search space is defined with respect to that of σ-neighborhood: two points x and y are said to be $\varepsilon-$connected if there exists a *connecting sequence* $x_0 = x, x_1, \ldots, x_k = y$ such that $x_{i+1} \in N_\varepsilon(x_i)$ for all $i \in 0, k-1$. In the continuous domain, two points are said to be connected if they are ε-connected for all $\varepsilon > 0$. The limit of a ε-connecting sequence is a *continuous connecting path* between x and y. An open set is said to be *connected* if all pairs in that set are connected, or, equivalently, iff it cannot be represented as the union of two or more disjoint non-empty open subsets.

- The notion of local optimality is another important concept: Assuming a minimization problem, a candidate solution $x \in \mathcal{S}$ is a local minimum if it has a lower fitness value $\boldsymbol{f}(x)$ than all other candidate solutions in its σ-neighborhood, for some $\sigma > 0$.

$$\text{local optimum}(x) \equiv (\exists \sigma > 0)(\forall y \in N_\sigma(x))\boldsymbol{f}(x) \leq \boldsymbol{f}(y) \tag{4.2}$$

A local optimum is a global optimum (or, by default, an optimum) if its fitness value is lower than that of any other point in $\mathcal{S}$.

4.2.2 Dimensionality

The dimensionality refers to the number of decision variables d in the search space. The dimensionality is one of the main factors of problem difficulty [WCT12], and in general, specific algorithms have to be designed to tackle large-scale problems, i.e. when the number of decision variables of the optimization problem is significantly large ($d \geq 500$), because performances of most algorithms deteriorate

rapidly when the number of variables increases, a phenomenon known as the curse of dimensionality [Ric57].

For very large dimensions, many concepts such as the neighborhood, the proximity, become less meaningful, and our intuition, based on 3D references, can go completely wrong. As the dimension increases, the volume of the space increases exponentially fast, and for instance most datasets become sparse. In general, for any method that requires statistical significance, such sparsity becomes critical, e.g., when trying to have statistically sound and reliable measures or results. In particular, the amount of necessary data to support sound statistical measures grows exponentially with the dimensionality too [Hou+10].

In the continuous case, for instance, specific algorithms assuming the separability of the objective function allows to fight the curse of dimensionality further, decomposing the problem into sub-problems [Liu+01; Omi+14]. However, such methods remain limited to separable problems, and often give very poor results on non-separable problems.

4.2.3 Modality

The term *modality* stems from the statistic community, where a distribution with one *mode* (one most likely value) is called uni-modal, while a multimodal distribution refers to a distribution with several most likely values. In optimization problems, it refers to the number of local optima: a function is uni-modal if it has at most one local optimum (which is hence also a global optimum). It is said to be multimodal if it has at least two local optima (note that all local optima can be global optima too). The empirical study of modality is one key element for a better understanding of the problem difficulty, and is one underlying notion necessary to study properties of objective functions, like plateaus. Furthermore, in the continuous case, multimodality is directly related to non-convexity of the objective function.

A straightforward measure of modality is the number of local optima of the objective function – though, of course, not a practical one. From there on, different measures can be derived, such as the average distance between local optima, as studied in Jones and Forrest [J+95]. In addition, the frequency or the distribution of $\boldsymbol{f}$ values of local optima in the search space $\mathcal{S}$ directly refers to the modality of the fitness landscape.

4.2.4 Basins of Attraction

Given a minimization problem, a basin of attraction is simply defined by the neighborhood around a local optimum that directly leads to it when going downhill, i.e., following a *monotone connecting sequence*, i.e., a connecting sequence (for the distance at hand) $x_1, \ldots, x_n$ such that $f(x_{i+1}) \leq f(x_i)$. Whereas modality is a global property of the problem, the properties of the basins of attraction (e.g., their respective sizes, depths, …) can give local information about the local optimum they contain.

In the general context, different situations can occur, depending on the operators that are chosen to follow the downhill paths and find the nearest local optimum from a given starting point. Following Stadler and Happel [SH99], one can then distinguish between *strong* and *weak* basins of attraction. However, in the continuous case when using the Euclidian distance on the search space, the definition of a basin of attraction is unambiguous: the basin of attraction of a local optimum is the set of all points from which there exists a monotone connecting path to this local optimum. The union of all basins of attraction of all local optima covers the search space. Note that one point can belong to several different basins of attraction.

Assuming everything is known about the objective function, the basin of attraction description allow to define some obvious measures, e.g. the basin extent (the fraction of points in the search space that fall in the basin). Further, some statistics about all basins of attraction and their extents can be derived, and, more importantly, the relation between the basin size and its depth can be studied.

4.2.5 Epistasis

The notion of epistasis is a recurrent notion in the fitness landscape analysis literature, directly inspired from the field of biology. In biology, the epistasis is the impact that one gene has on the effects of another one, i.e. the non-linearity or degree of concerted effect of several genes (e.g. a gene that inhibits or enhances the expression in the phenotype of another gene)[1].

In the context of the fitness landscape analysis, the total lack of epistatis is clearly defined: a function f has no epistasis iff it can be written as $f(x) = \sum_{i \in I} f_i(x_i)$, for some single-variable functions f_i.

The definition of a fully epistatic function, on the other hand, is ambiguous,

[1] with respect to Cordell [Cor02], though, there is still some ambiguity in the definition of epistasis.

and depends on the search space. It is clear for Boolean functions, looking at their Walsh decomposition [KNR01], that explicitly describes all possible interactions of all possible subsets of $[1, d]$. In that context of binary strings, the function f can be decomposed into a superposition of simpler functions

$$f(x) = \sum_j \omega_j \xi_j(x)$$

where the $\xi_j(x)$ should have some meaning that is related to the properties of f, and x a vector of d component denoted x_i that represent an individual. Hence the most well-known set of functions for binary string are the Walsh functions:

$$\psi_j(x) = \prod_{i=1}^{d} (1 - 2x_i)^{j_i}$$

where j, such that for a string of length d, there are 2^d Walsh functions, and the Walsh decomposition of the function f is described as follows:

$$f(x) = \sum_{j=0}^{2^d-1} \omega_j \psi_j(x),$$

with the Walsh weights ω_i that can be found from the values of $f(x)$ by means of the Walsh transform[Bea75].

In the general case, including the continuous case, different measures of epistasis have been proposed, but the most widely used is the *epistasis variance* [Dav90]

$$\eta = \frac{\sum (f - \sum \text{lineareffects})^2}{\sum (f - \overline{f})^2},$$

where $\overline{f}$ is the mean value of f.

Although the epistasis variance has been used to analyze the hardness of a fitness landscape in the Boolean case [Dav90], it has also been shown to demonstrate to be inefficient in several cases [Dav91; RW95]. Naudts and Verschoren [N+99] proposed a more complete definition of the epistasis and compared it to the deceptiveness, i.e. the presence of misleading information which may guide the algorithm away from the global optima. However, according to [RW95], it requires full knowledge of the objective function and is hence not practical at all.

Rochet [Roc97] introduced the graded epistasis , which can be described as a model with at most j interacting genes as

$$f^g = \sum_{i \in G(g)} f_i(x - i)$$

where the index i from the power set of all indexes i and limited to g simultaneous elements through $G(g) = \{i | i \in 2^I, \|i\| \leq g\}$. Hence, these measures consider a more complex combination that include orders of non-linearity, in contrast with the epistasis variance.

While the concept of epistasis appears to be an important notion to understand the problem difficulty, Reeves and Rowe [RR03b] pointed out some difficulties about this approach, e.g. the lack of simple and easy way to practically compute some epistasis-related measure with enough predictive power.

4.2.6 Separability

A function f is separable if the global optimum of f can be approached by independent one-dimensional searches along each coordinate in turn. Exploiting the separability of functions can break the curse of dimensionality, as a line search is supposed to be an easy task (using e.g., repeated dichotomy). The separability is clearly linked to epistasis, though both properties arose from different communities: separability was defined within the numerical optimization community, i.e., for functions of continuous variables. It is clear, however, that a function without any epistasis (see previous Section4.2.5 is fully separable, though there exist functions that can be optimized by successive line searches that are not the sum of d different one-variable functions.

A function f is called non-separable if finding its optimum cannot be achieved by a series of one-dimensional searches, and it is called partially-separable if it is non-separable, but has blocks of coordinates which can be optimized separately. Many real-world problems are partially-separable, because humans tend to view the world as a structured system/object. Structured view of the world favors extensive use of decomposition, which allows to separately solve sub-problems of a large partially-separable problem. The decomposition is a common approach for complex design and optimization problems, such as the design of a new car or airplane. We also often tend to simplify our daily life using decomposition (e.g., we first brush our teeth, then we drive a car to our workplace, then we work). However, some people find that it is better to do these things simultaneously.

The separability is probably the first property that should be checked when analyzing the performance of an optimizer on a problem, but there is no known easy way do to so directly. Some algorithms explicitly or implicitly exploit separability, and, therefore perform surprisingly well on separable problems. The number of function evaluations to reach the optimum or some target objective value then may scale almost linearly with d, giving a hint that the problem might be separable. The separability of a problem should be exploited to improve the search if it is automatically detected.

Unfortunately, many well-known benchmark problems are separable, and many optimizers have become overfitted to this property [Haw04]: they perform well on these benchmarks, but fail on real-world problems when they are non-separable or partially-separable. A simple way to tackle this kind of overfitting is to rotate the coordinate system, leaving f properties unchanged except for a possible separability. The overfitting to separability has misled many designers of algorithms and may be viewed as a drawback of approaches such as real-coded GAs, PSO,... A detailed analysis can be found in [Sal96; Han+08].

4.2.7 Fitness Barrier

Another concept borrowed to physics is that of *fitness barrier* Stadler and Happel [SH99]. In physics, the energy *barrier* is some part of the space with high energy that has to be crossed in order to reach some (low-energy) meta-stable state. It is easily generalized to the fitness landscape as proposed by Stadler [Sta02b].

In the context of fitness landscape, a barrier between two points x and y of the search space is the minimum fitness value that must be climbed (assuming again a minimization problem) on a path between x and y. In following Equation 4.3, $p_{(x,y)}$ is a connected path between x and y.

$$\boldsymbol{B}_f(x,y) = min_p\{\, max_z\{\, \boldsymbol{f}(z) \mid z \in p_{(x,y)}\}\} \tag{4.3}$$

The concept of fitness barriers can be used as yet another measure of separation between different local optima, that can capture another type of difficulty of the landscape, that of moving between optima.

4.2.8 Landscape Walk

Regarding the analogy to *real* landscape, the notion of the *walk* is essential for a better understanding of the fitness landscape structure. Simply put, a walk across the fitness landscape is an exploration of the domain search space by repeated moves of one candidate solution, considering the evolution of the fitness along the way.

The move is in the general case defined by operators of the algorithms under study. In the continuous case, however, two different moves are considered, defined by the probability distribution from which the successor of the current position is chosen. Assume the current position is x. Then the next position can be drawn either uniformly on B_σ, or following the normal distribution $\mathcal{N}(0,\sigma)$. In both cases, σ will be called the step-size of the walk.

In order to explore the fitness landscape, an appropriate strategy of exploration must be defined. Considering a step-size σ, different types of walks can be defined [Lov93; Sta02a].

- *Random walk*: any arbitrary candidate solution is selected in the probabilistic neighborhood defined by the step-size σ
- *Adaptive walk* selects a neighbor whose fitness is better than the fitness of the current point.
- *Reverse adaptive walk* does the opposite of the adaptive walk, i.e., selects a new point only if it is worse than the current solution.
- *Uphill-Downhill walk*: an adaptive walk is performed until the fitness cannot be improved any more, then a reverse adaptive walk is performed, until it cannot be worsened any more.
- *Neutral walk*, proposed by Reidys and Stadler [RS01], only chooses points with equal fitness with the current point, and try to increase the distance from the starting point, something that is particularly appropriate for problem instances with plateaus.

More formally, given all the neighbors N_σ of all candidate solutions x, then an appropriate strategy $\zeta_f : 2^S \rightarrow S$ is defined in order to select the appropriate new neighbor. From a candidate solution x_1, we get a sequence $\{x_i\}_{i=1}^n$ from a random walk of length n over the search space $\zeta(N_\sigma(x_i))$.

More complex walk method can be used in order to maximize the exploration of the search space, like Metropolis random walk [Has70], or some metaheuristics, like Evolution Strategies (with an appropriate step-size σ).

Landscape analyses based on landscape walks aim at exploring the search space, gathering samples in a controlled way. Multiple walks can be performed from different starting points, to increase the statistical significance of the results. While, many fitness landscape metrics — to be discussed later in this Chapter — are only computed from a uniform sample, several metrics can only be computed from the samples gathered during directed landscape walks. For instance, when the metric aims at investigating the modality of the landscape, and the existence and repartition of local optima.

4.2.9 Ruggedness

The most prominent measure to describe the ruggedness of a fitness landscape has been proposed by Weinberger [Wei91] and relates to the *auto-correlation*

and *correlation length* of random walks. These were later generalized by Jones [Jon95]. For instance, a random landscapes has zero correlation length, and rugged landscapes have correlation lengths that decrease as ruggedness increases. The correlation length can be described by $\frac{1}{\ln(\tau)}$ where τ is the auto-correlation.

The most common formulation proposed by Weinberger [Wei90] assume a stochastic process, like a landscape walk, which yields an exponential decay of the autocorrelation function, as defined by Equation 4.4, computed at each step of the walk.

Manderick, Weger, and Spiessens [MWS91] propose a variant of the auto-correlation function, as described in Equation 4.5, which gives a corrected measure of the correlation for different distances, with respect to the variance of the fitness.

Although these measures were successfully applied to describe problem difficulty [MWS91; Cze08; MF98], they only give a local view of the fitness landscape, and remain inaccurate for global hardness estimation.

$$\rho(\sigma) = \frac{\mathrm{E}(f_t \cdot f_{t+\sigma}) - \mathrm{E}(f_i)\mathrm{E}(f_{t+\sigma})}{\mathrm{Var}(\mathrm{f_t})} \tag{4.4}$$

$$\rho(f_t, f_{t+\sigma}) = \frac{\mathrm{Cov}(f_t, f_{t+\sigma})}{\sqrt{\mathrm{Var}(f_t)\mathrm{Var}(f_{t_\sigma})}} \tag{4.5}$$

4.2.10 Neutrality

Introduced by Kimura [Kim+68], in the biological evolution theory, neutrality is the degree to which a landscape contains connected areas of equal fitness. In the discrete space, a "neutral network" is a set of n points of the search space that have the same fitness values. In the continuous space, it is a connected open subset of the search space with identical fitness, though the latter requirement can be relaxed, and only required to be true up to some precision – one then talks about quasi-neutral areas.

An important characteristic of evolution in a neutral area is whether or not the algorithm accepts new candidate solution with same fitness as the former ones (constant evolution [Huy96; SHO02]) - favoring exploration rather than exploitation, and avoiding getting stuck in neutral areas.

Reidys and Stadler [RS01] introduced the first measures of neutrality, using neutral walk, i.e., a variant of random walk restricted to neutral areas, with the aim of continuously increasing the distance to the starting point. Following

a similar scheme, [Bar97; Bar98] investigated the neutrality, by using simple measures, like variance of fitness values, auto-correlation, or average distance between candidate solutions in the population; and shows that the empirical study of neutrality is complementary to the analysis of the ruggedness of fitness landscapes.

In the same direction, the Nei's standard genetic distance [Nei72] is widely investigated in [KOU04; KO06] in order to get more insight of the neutrality of the landscape and how it can be observed during the run of GA.

Katada, Ohkura, and Ueda [KOU04] empirically investigated the Nei's standard genetic distance on a binary coded artificial landscape generator for which they can control the neutrality, and showed that the number of changes in the population increases as the neutrality increases, and decreases when the search space contains fewer regions of equal fitness.

As a validation Katada and Ohkura [KO06] empirically investigate the neutrality on real-world problems, with evolutionary robotics, and observing similar results to [KOU04]. This strongly suggests that the measure is reliable for estimating the neutrality. However, this measure is used during the algorithm run, and thus consider a relatively large number of calls of the objective function, making it impractical in a real-world conditions.

4.2.11 Deception and Fitness Distance Correlation

While ruggedness, local optima analysis and information analysis provide insights into the connectedness and local structure of fitness landscapes, the fitness distance correlation (FDC) was designed to obtain a global view related to the notion of deception.

A problem is said to be *deceptive* if following the optimal descent direction (for minimization problems) does not lead to the global optimum. This notion was first coined for Genetic Algorithms [Hol75; Whi91], and heavily used and discussed there [Gre14; GDH92]. In order to quantify deception, Jones [Jon95] and Jones and Forrest [J+95] proposed a measure termed *Fitness Distance Correlation* (FDC). Assuming that the global optimum is known, and that a random sample of the landscape (i.e., pairs of $(x, f(x))$ for the objective function at hand f) is available, the FDC is basically (as its name says) the correlation between the distance from the optimum and the values of the fitness.

More formally, let x_{min} be the global optimum of f, $\{x_i\}_{i=1}^n, x \in \mathcal{S}$ the sample of points of the search space, with associated fitness values $\{f_i\}_{i=1}^n, x \in \mathcal{S}$ (with $f_i = f(x_i)$) and distance to the optimum $\{d_i\}_{i=1}^n, x \in \mathcal{S}$ (with $d_i = d(x_i, x_{min})$). Let $\overline{f}$ (resp. $\overline{d}$) denote the mean of the (f_i) (resp. (d_i)) and σ_F (resp. σ_D) their

standard deviation. Then FDC is defined by

$$FDC = \frac{\frac{1}{n}\sum i = 1n(f_i - \overline{f})(d_i - \overline{d})}{\sigma_F \sigma_D} \tag{4.6}$$

One of the major limitations of the Fitness Distance Correlation is that it requires the knowledge of the global optimum. As an alternative, when x_{min} is not known, Kallel and Schoenauer [KS96] proposed to use the best-known candidate solution of the available sample, and termed the result *local Fitness Distance Correlation*. Kallel and Schoenauer [KS96] empirically demonstrated that the local FDC gives insights of the modality of regions of the search space, while FDC will give insight of the global structure. Therefore, when the local FDC is computed along the optimization process, it can give insight of the homogeneity of the search space, hence completing the results of the FDC. In that direction, in order to have a more precise 'picture' of the search space, Kallel and Schoenauer [KS96] suggest using additional measures, like the variance of FDC over subsets of the sample, or some clustering analysis methods.

4.2.12 Evolvability

According to [Alt+94; MHR99; WA96], evolvability refers to the ability of a population or individuals to generate fitter offsprings. Thus, evolvability is more related to the *potential* of a fitter fitness than the fitness itself, such that two individuals with equal fitness may have very different evolvabilities [Tur02]. A long-term change cannot be due to straight fitness selection[SHO02], such that it can only be understood through some second order selection mechanism by which the evolution tends to select solutions that have a more evolvable genetic system [Daw03]. Therefore, for evolutionary computation practitioners, evolvability is directly related to the study of the ruggedness (see Section 4.2.9) and the modality (4.2.3) of the landscape [Wei90].

Altenberg [Alt+94] propose to use the *transmission function* T which can be defined as the probability density function of offspring fitnesses from a single parent (h, k) where h is parent representation and k its fitness value. Smith, Husbands, and O'Shea [SHO02] describes measures of evolvability for continuous optimization:

$$E_a = \int_k^\infty T(f : h, k)\ df$$

, such that for low fitness parents maybe have larger E_a than high fitness parents, because of the number successful mutation increase for fitter parents.

However, these measures need to consider an exhaustive sampling, and Smith, Husbands, and O'Shea [SHO02] highlights that such measures may be inefficient with sparse sampling, as they are then prone to misleading variations.

4.2.13 Discussion

Most of the properties we have surveyed above are difficult to define accurately, both because they require a complete knowledge of the objective function and because they can be interfering.

In the black-box scenario, the workaround to fight the lack of theoretical knowledge about the objective function is to use statistics based on some samples of values of the function on a set of points of the search space, computing some features that will reflect some particular aspects of the objective function. However, these features often require a very large amount of samples, and their accuracy grows with the number of samples, as for all statistical approaches. A critical issue in real-world scenarios with expensive objective functions is hence how these metrics behave when only small amount of samples can be used to compute them.

Furthermore, the possible interferences between different properties imply that each of the features might reflect several different aspects of the properties listed in the present section, and it should be clear that the starting point of the description of an objective function will be from now on the set of features (described in net Section) that will be computed from a sample of pairs (x, f(x)) of values taken by the function, whether or not these features accurately reflect some of the properties that have been listed above.

4.3 Problem Features for Continuous Black Box Optimization

In this Section, we will focus on continuous problem description, and survey a number of features that have been proposed in the literature for describing accurately continuous black-box optimization problems, i.e., that can be practically computed in a reasonable time.

4.3.1 Motivation

Previous section has introduced different characteristics of problem landscapes that have been proposed in order to have a better understanding of optimization

problems in general, and of their hardness for different algorithms. These characteristics did not make any specific assumption on the search space, only assuming a distance metric to compare samples.

This section first restricts the study to continuous black-box optimization problems, i.e., problems defined on a subset of $\mathbb{R}^d$ for some dimension d, and for which the only available information can be obtained through the oracle black-box that computes $f(x)$ given x (spending one from the computational budget for the optimization of the problem under scrutiny).

In contrast to the previous Section, rather than defining and quantifying precise characteristics of the fitness landscape of an optimization problem, problem features have been proposed as measurable quantities for the problem at hand (here, in the black-box setting) that do tell something about the problem itself, from sample pairs $(x, f(x))$, as this is all we can get, but in general without a priori ideas of how this relates to solving the optimization problem. The underlying idea is that each of these features will indeed contain some information about the optimization problem, and if we can compute sufficiently many of these features, we should be able, using some Machine Learning techniques, to gather enough information to solve the Algorithm Selection and Algorithm Configuration problems. Of course, the measures on the fitness landscape described in the previous section, when computable in the black-box setting, will also be used as problem features: in spite of their drawbacks in terms of accuracy in predicting the property they were designed for, we know that they nevertheless contain some interesting information related to solving the AS and AC problems.

In the last decades, a large variety of features were proposed in the literature to capture some problem characteristics, and many of these features were initially proposed for the combinatorial or the discrete domain (see [MF00; PA12; RS01; TPC08] for an overview and examples of features for combinatorial and discrete domains). More recent works tackled the fitness landscape analysis of the continuous domain, introducing the term *exploratory landscape analysis* [Mer+11; M+15; LW06], possibly adapting some of the features that had been originally proposed in the combinatorial domain. All of the features re suitable to the black box setting as they rely on a uniformly randomized sample of $(x, f(x))$ pairs, whereas some features use this sample as starting point for landscape walks.

4.3.2 Fitness Distance Correlation

The Fitness Distance Correlation (FDC) [J+95] has been presented in Section 4.2.11, defined by Equation 4.6, which requires that the global optimum be known. However, replacing the global optimum with the best point in the sample [KS96] allows to directly use the resulting *local FDC* as a feature – that can be easily

computed from a given sample set.

4.3.3 Information Analysis

Closely related to the notion of ruggedness, Information analysis proposed in [VFM00], is a measure the difficulty to describe a system based on its information content.

The main idea is to analyze the amount of information that is necessary to describe a random walk: the more information is needed the more difficult is the problem. It is closely related to the Shannon entropy [Sha01], in that it tries to compress the information of a random walk into a limited number of bits, or as a function of the distribution of elements over the states in the system.

This measure relies on the fitness values encountered during a random walk or a fixed sample set on the search space $\{f_t\}_{t=1}^n$, and more specifically it relies on the differences of successive fitness values $\{\Delta f_t\}_{t=1}^n = \{f_t - f_{t+1}\}_{t=1}^n$.

Let ϵ be the *smoothness threshold*, i.e., the tolerance that defines how much a change in the fitness between two consecutive steps of the random walk can still be considered smooth[2], and let the relaxed sign function be defined as

$$sign_\epsilon(x) = \begin{cases} +1 & if\ x > \epsilon \\ 0 & if\ |x| \leq \epsilon \\ -1 & if\ x < -\epsilon \end{cases}$$

Then the sequence of the relaxed signs of the differences in fitness along the random walk $S(\epsilon) = \{sign_\epsilon(\Delta f_t)\}_{t=1}^n$ is used by Vassilev, Fogarty, and Miller [VFM00] to define several features related to the random walk as follows.

- *Information Content* $H(\epsilon)$ aims at capturing the variety of shapes in the landscape by looking at the occurrences of pairs of different symbols in the sequence $S(\epsilon)$. For all pairs (p, q) of different symbols in $\{-1, 0, 1\}$ (there are 6 such pairs), define $P_{[pq]}$ as the relative frequency of this pair in the $S(\epsilon)$ sequence: $P_{[pq]} = \frac{n_{pq}}{n}$ (where n_{pq} is the number of times the two consecutive symbols pq appear). Then the information content is defined as the Shannon entropy of these pairs:

$$H(\epsilon) = -\sum_{p \neq q} P_{[pq]} \log_6 P_{[pq]} \tag{4.7}$$

[2] in contrast with the step-size σ of the random walk as described in Section 4.2.8

- The *Partial Information Content* is a measure that aim at capturing the number of slope changes in the landscape. It is achieved by creating a new sequence $S'(\epsilon)$ from $S(\epsilon)$ by removing all zero symbols and replacing all subsequences of identical symbols by a single symbol. Let μ be the length of this new sequence $S'(\epsilon)$. The Partial Information Content is defined as

$$M(\epsilon) = \frac{\mu}{n} \tag{4.8}$$

- The *Information Stability* $\epsilon^\star$ is the highest fitness difference between neighbors in a series. It can be defined as the minimal ϵ for which the landscape appears totally flat, i.e., for which $S(\epsilon) = \{0, \ldots, 0\}_{t=1}^{n}$
- The *Density Basin Information* $H(\epsilon)$ aims at analyzing the smooth areas (i.e., the areas where the fitness function behaves homogeneously), by computing the entropy of the number of consecutive symbols that are identical:

$$H(\epsilon) = -\sum_{p \in \{-1,0,1\}} P_{[pp]} \log_3 P_{[pp]} \tag{4.9}$$

By varying the parameter ϵ, the information content set of measures allow to view the landscape, through a random walk, at a different level of granularity. These measures provide another viewpoint on the ruggedness of the landscape, by analyzing the distribution of rugged and smooth parts of the landscape.

4.3.4 Distribution of Fitness values

The use of the distribution of the fitness value $p(y)$, was first proposed in [REA96], and refers to the probability density function of the fitness value y. In [REA96], the authors suggest that this probability density function can be used to approximate the global optimum. In addition, it has been pointed out that the distribution can be used as a classification tool, supposing that a problem with a high decay might be hard to solve.

Regarding a probability distribution, naturally, the kurtosis, skewness and the number of peaks of the distribution are three important properties of a distribution that can be used as a metric to identify problem characteristics. This metric was proposed in [Mer+11], to characterize the problem landscape for automated algorithm selection or algorithm configuration.

- The kurtosis is a measure of the sharpness of a probability distribution. In the view of fitness landscape, the kurtosis refers to the ruggedness, neutrality or smoothness of the landscape. For example, given a fitness landscape

exhibiting a region of neutrality, the kurtosis value of the probability distribution is high.

- The skewness is a measure of the symmetry of the probability distribution. The combination with the kurtosis gives a better insight on the landscape and the neutrality of the problem.
- The number of peaks is a relevant measure of the probability distribution. Given the probability density function computed from fitness values $\mathcal{Y}$ and a acceptance threshold, the probability of all ordered values $y \in \mathcal{Y}$ are computed; a y value is a peak if its probability is greater than the probability of its neighbors in the ordered set, and greater than the threshold, then the number of peaks is the total number of found peaks.

  According to [Mer+11] and with respect to the fitness landscape analysis (Section 4.2.3), this measure is relevant to observe the multimodality and plateaus of a landscape.

4.3.5 Length Scale

The *Length Scale* metric, proposed in [MG12], measures the ratio of changes in the fitness values with respect to a step two between candidate solutions in the search space. For x_1 and x_2 two points of the search space, the length scale is defined by

$$LS(x_1, x_2) = \frac{|f(x_1) - f(x_2)|}{\|x_1 - x_2\|}$$

The length scale can be viewed as an approximation of the derivative of the fitness function when x_1 and x_2 are close to one another. In any case, the length scale greatly depends on the step between x_1 and x_2.

Whereas the length scale is a local metric, the distribution of the length scale on the fitness landscape is yet another global metric. It describes the probability of observing different values of length scales for the fitness landscape. Morgan and Gallagher [MG12] empirically investigated the length scale in order to describe the test functions of the BBOB test bench. They empirically demonstrated that it can be used to discriminate the functions with different properties, whereas it requires an enormous sample size, that they generate from a random Levy walk [SWK87] of 10^5 samples: this is clearly not applicable in real-world conditions.

4.3.6 Dispersion Metric

Lunacek and Whitley [LW06] initially proposed the *Dispersion* metric in order to analyze the average distance of best candidate solutions generated by CMA-ES

[HO01a]. In [LW06], the Dispersion metric is used to empirically study the fitness landscape properties and difficulties.

The dispersion metric is computed from a uniformly sampled set of points of the search space. These samples are first ranked. The dispersion is then defined, following equation 4.10, as the average pairwise distance within the top q^{th} percentile of the sample.

$$disp_q = \frac{1}{(qn)(qn-1)} \sum_{i=1}^{qn-1} \sum_{j=i+1}^{qn} \|x_i - x_j\| \tag{4.10}$$

A low dispersion measure, intuitively suggests that the top-ranked sample is restricted to a better region of the search space, which supposes the uni-modality of the function. By contrast, a high dispersion measure suggests that the top-ranked candidate solutions are lifted in different restricted good regions of the search space, supposing different regions with local optima. Then, the choice of the threshold is essential to have a better overview of the search space, such that different thresholds can be applied to identify high multimodality in the same region of the search space.

The dispersion metric remains a compressed variable to express the hardness of the problem, with respect to a particular threshold of the candidate solution. The authors propose to compare the dispersion at different thresholds and have summaries of statistics in order to get a better insight of the proximity of better regions. This is especially discussed when the dispersion is *low*, that can exhibit either uni-modality or a high multimodality in a close region of the search space.

It is relatively important to note that the dispersion metric supposes a uniform sample and avoiding to favor a specific region of the search space. Because it aims at quantifying the proximity of better regions of the search space, the choice of a sample that favors the exploration of the search space is a necessary condition for an appropriate analysis.

The dispersion features can give a good insight of the hardness of a problem in terms of a compressed information of the global topology, e.g. ruggedness, deceptiveness, or neutrality, of the fitness landscape.

4.3.7 Convexity Metric

The convexity is an essential property of an optimization problem, which plays an important role in the choice of the appropriate algorithm. A real-valued function f defined on an interval is said to be *convex*, if the line segment between any two

point on the graph of the function lies above or on the graph in a vector space (e.g. Euclidean space).

Mersmann et al. [Mer+11] proposed to estimate the probability of convexity from a uniformly randomized sample X^d of candidate solution in the search space of dimension d, and additional samples during the computation of these measures. Simply put, it is achieved by randomly selecting two random points a b, from the initial sample X^d and generate a new random point c between a and b and compare its fitness value to the fitness value of a and b.

The convexity metric can be summarized into three different measurements, in such way that one is estimating the convexity p_{conv}, a second one is estimating the linearity, $p_{lin} = 1 - p_{conv}$. The probability of convexity is computed by averaging the number of trials where the computed difference of the new sample x is lower than a pre-defined negative threshold, while the probability of linearity is computed by considering all absolute differences that are smaller than the absolute value of the pre-defined threshold.

4.3.8 Meta-model

Given a limited sample of candidate solutions X^d, well-known methods in statistics propose to use meta-models to study the property of an unknown problem. Considering the continuous domain, regression methods approximate the true objective function from a sample of candidate solutions.

To this end, the meta-model metrics proposed in [Mer+11] considers different meta-modeling methods from the statistical field, in order to compute a linear, a quadratic regression and a mixture of Gaussian models for the initial sample $\mathcal{X}^d$ and $\mathcal{Y}$. For each meta model, the coefficient of determination R^2 is computed as one measurement of the accuracy of the models correlation. It aims at identifying the relationship between the variables of the objective function. It is computed from a linear model approximating the objective function, following the Equation 4.11 where r_{xy} is the vector of cross correlation between the predictor variables on the set $(x_i, f(x_i))$ and R_{xx} is the matrix of the inter-correlation between predictor variables. In addition, the variable significance [SM07] estimates the amount of information that a subset of variables provides for the criterion variable $\mathcal{Y}$.

$$R^2 = r_{xy}^T R_{xx}^{-1} r_{xy} \tag{4.11}$$

These measures directly reflect the difficulty of the problem as it approximates

the problem structure with an analytical function. Then a simple unimodal function might be well approximated by a quadratic model or a linear function by a linear model. By contrast, a more complex problem with variable scaling or a multimodal function will not allow a good fit of regression models.

4.3.9 Curvature

The study of the convexity and the curvature is essential for a better understanding of the problem and greatly help to choose the appropriate algorithm or parameter configuration of an algorithm. The curvature features consider the numerical approximation of the gradient in different regions of the search space.

Given an initial sample of the search space, a sub sample is randomly selected X^{dr} in which the numerical approximation of the gradient is computed with respect to the Richardson's extrapolation method [RG27]. Then, the resulting features consider the basic statistics of the respective derivatives, and the relatives Euclidean norm of the latter, as well as the relation of the maximum and minimum of the respective partial derivatives. Also, similar statistics are applied to the condition number of the numerical approximation of the Hessian [LP89].

4.3.10 Level Set

The level set is proposed by Mersmann et al. [Mer+11], and is especially relevant when a problem is multimodal.

Given a threshold on the objective function values, the initial sample $\mathcal{X}^d$ is divided into two classes. One possibility is to use the lower and upper quartiles of the distribution of the fitness values, which will result in non-equally sized classes, or the median. From this modified sample, different classification techniques such as LDA (Linear Discriminant Analysis), QDA (Quadratic Discriminant Analysis) or MDA (Mixture Discriminant Analysis) are trained in order to predict the position of a testing set with respect to the threshold. Then it is used in order to predict if a fitness value falls below or exceeds the given threshold (see [Mer+11] for the different threshold used for their empirical validation). The features are summary statistics(mean, median,...) of the misclassification errors of the different classifier.

According to results in [Mer+11], the level set features are particularly efficient to detect the multimodality of the problem that should result in several unconnected sub-level set for different thresholds.

4.3.11 Local Search

The study of the local optima of an optimization problem is essential for a better understanding of the difficulty and the properties of the problem as described in Section 4.2.

The study of local optima considers a local search algorithm (e.g., Nelder-Mead). From an initial sample $\mathcal{X}^d$, different local searches are run from different starting points defined in a random sub sample $\mathcal{X}^{d\prime}$. The resulting solutions of the different runs are clustered, with the main goal to identify the possible local optima of the objective function. In addition, the basin sizes of problems are approximated by the number of local searches [Mer+11; MKH12; AMT12]. Also, the different statistics (means, variance,median of fitness of local optima, and their pairwise distances), gathered during the different runs are used as problem features, and give a hint of the problem landscape characteristics.

4.3.12 Fitness Cloud and Negative Slope Coefficient

Considering an evolutionary algorithm or any metaheuristic that favors the exploration of the search space, the fitness cloud [VCC03] can be defined by the graph of fitness values of the parents compared to the fitness values of their offspring that have been observed during a search.

The study of the fitness cloud can then be used to visualize the search space, and also characterize the set of local optima, as studied by Collard, Verel, and Clergue [CVC07]. Considering the fitness cloud, if the fitness value of parents is higher than the offspring's fitness value, then the fitness value is improved. From the fitness cloud, it is easy to have a better understanding of the problem landscape difficulty. Indeed, a large region under the threshold $f(parents) = f(offsprings)$, correspond to an easy landscape such that most parents mutate as improving offspring, supposing a fast convergence.

Vanneschi et al. [Van+04] introduced the negative slope coefficient as a problem feature of the fitness landscape analysis, based on the fitness cloud. To proceed, a partitioning of the x axis (referred to the fitness values of the parents) in m segments. For each M_i segment, the average fitness value of offspring N_i, where $1 \leq i \leq m$, is computed. Then, the slope k_i between two segments is computed, as described in the Equation 4.12, where $1 \leq i \leq m-1$. Then the negative slope coefficient is given by the Equation 4.13

$$k_i = \frac{N_{i+1} - N_i}{M_{i+1} - M_i} \tag{4.12}$$

$$NSC = \sum_{i=1}^{m-1} min(k_i, 0) \tag{4.13}$$

The negative slope coefficient can be summarized as the averaging of the negative slope in the partition of the fitness cloud. A negative slope corresponds to an improvement of the fitness values of parents that results in worsen fitness values of their offspring, hence it is a efficient measure of the deceptiveness or evolvability of the landscape.

Despite promising results [VCC03; Van+04], these measures require a large sample of candidate solutions collected from a search algorithm. Its use is thus limited to some empirical analysis of problem properties without any claim for efficiency as a feature.

4.3.13 Cell Mapping

Initially proposed by Hsu [Hsu13], cell mapping aims at analyzing the global behavior of nonlinear dynamic systems, by considering a discretization of the continuous search space domain into hypercubes: the cells. Thus, the continuous search space $\mathbb{R}$ of the problem is reduced to the cell space.

A more general approach called Generalized Cell Mapping aims at investigating the global characteristics of the problem landscape by considering a transition function between cells, which leads to an absorbing Markov chains [KS76].

More formally, we assume that the problem is bounded by a box constraint defined by lb and ub, which constitutes a new cell domain $Q = \{(x_1, \ldots, \boldsymbol{x}_n)^T \in R^n : lb_i \leq x_i \leq ub_i, i = 1, \ldots n\}$. Also, the domain search space in N_i section on size $h_i = (lb_i - ub_i)/N_i$, such that we get finite subdivisions of the domain search space that are called *cells*. Kerschke et al. [Ker+14] discuss different methods for the selection of a representative points in each cell, e.g. the closest point to the center of the cell or the average point in the cell.

Different features are proposed by Kerschke et al. [Ker+14] to characterize the global properties of the landscape from the cell mapping. The proposed features aim at describing the landscape with summary statistics of the absorbing and transient cell, in the sense of Markov chains: such that absorbing cells are cells that, once entered, cannot be left, like a local optima, and transient cell is a cell that is not absorbing such that it has a probability one to leave the cell.

The resulting features efficiently help to describe the global properties of the problem landscape, in particular funnel structures [Loc05; Ker+15]. However,

these metrics are only investigated for $d = 2$ and easily extended for $d = 3$, whereas most challenges for continuous black box problems are when $d > 3$.

4.4 Which Features in the Expensive Black-Box Scenario?

Having defined a feature space $\mathbf{\Psi}$, $\mathbb{R}$-vector space of dimension p, the black-box context implies that the features of any function $f : \mathbb{R}^d \mapsto \mathbb{R}$ should be computable from samples of f[3], i.e. a set $\mathcal{X}$ of n pairs $(x_i, f(x_i))_{i=1,\ldots,n}$ (the set of values $\{f(x_i)|i = 1, \ldots, n\}$ is denoted $\mathcal{Y}$). We will assume here that these samples are specifically gathered for the purpose of computing the features, and setting the algorithm hyper-parameters before running the algorithm on f: hence, their cost (number of function calls, i.e. here, n) should be included in the total cost of the optimization.

While several the features presented above only rely on a fixed sample, some of them use this fix sample as starting position for some random walks that are needed to compute them. In any case, the accuracy of the computed values (w.r.t. the exact values, computed over the whole search space) highly depends on the sample size, hence on the budget that is allocated to this part of the whole optimization process. It is, hence, utterly important to study the complexity of the feature computation.

First, we observe that some of these features, like the length scale (Section 4.3.5) or the Fitness Distance Correlation (Sections 4.2.11 and 4.3.2), often require large sample size ($\geq 10^4 \times d$) according to existing literature [MG12; MS11].

By contrast [LW06; Mer+11; MKH12; M+15] empirically identified some features that could be used with sample sizes that are admissible in the expensive black box optimization scenario.

Let us now look at all the features introduced earlier in this Chapter from the point of view of their computational cost. We can regroup the proposed features into two main groups:

- *Cheap* features that only require a fixed initial sample of the objective function, such that the features values are computed once for all. This group is composed of 51 different features classified into 5 sets: 16 **Dispersion** [LW06] (see Section 4.3.6), 3 **y-Distribution** [Mer+11] (see Sec-

[3] d, the dimension of the search space, can be considered as the only external feature

tion 4.3.4), 5 **Information Content** [M+15] (see Section 4.3.3), 9 **Meta-Model** [Mer+11](see Section 4.3.8) and 18 **Levelset** features [Mer+11](see Section 4.3.10).

- *Expensive* features that are computed using a fixed sample set and some additional samples evaluated on the fly during their computations, such that the exact size of additional samples can be determined a priori (the order of magnitudes are about $10^3 \times d$), making their use impossible in practice. This group is composed of 25 different features classified into 3 sets: 4 **Convexity** [Mer+11] (see Section 4.3.7, 14 **Curvature** [Mer+11] (see Section 4.3.9) and 7 **Local Search** features [Mer+11] (see Section 4.3.11).

All these features (*cheap* and *expensive*) were successfully used for the Algorithm Selection problem [Mer+11; M+15], though these works all considered large sample sizes. Recent works [Bis+12; Ker+16] used only *cheap* features with small samples [Ker+16], such that the Section 5.1 investigates the accuracy of these *cheap* features with decreasing sample size and proposes a new methodology to overcome the loss of accuracy of sub-sampled features.

Finally, this thesis aims at investigating the computation of features, with respect to their computation and their selection. Since the size of the sample greatly influences the accuracy of the features, our work aims at using these features into the Per Instance Algorithm Configuration to describe the test functions. Despite several features described in this Chapter, only few of them are sufficient to characterize the properties of black box problems when dealing with an operational context, thus this remains an open research question that is tackled in this thesis. On the one hand, in the ideal case, very large samples can be used to compute the features: both groups can be used to characterize accurately the landscape properties of a given optimization problem, but not practically, to actually solve this problem. On the other hand, we need to minimize the number of calls to the objective function and avoid any additional costs, thus only cheaply computable features can be used.

Part III

Contributions

CHAPTER 5

Toward Per Instance Algorithm Configuration

In this Chapter, we discuss issues related to the computation of problem features in continuous domains introduced in the previous Chapter, and present the general experimental methodology we will use in the remaining of the thesis. In Section 5.1, we start by studying the problem features in the ideal case, i.e., when they are computed from a very large sample of objective function values, then address the issue of a small computation budget, proposing to use surrogate models of the objective function to reduce the computational cost of the features without impacting their efficiency for PIAC. Finally, Section 5.2 introduces the general methodology for running experiments and assessing the results of our approach to Per Instance Algorithm Configuration in the continuous black box domain.

5.1 Low-budget Computation of Problem Features

In this Section, we investigate the computation of problem features, in order to assess their accuracy and efficiency in a low-budget setting. The long-term goal is to use these features for Algorithm Selection or Per Instance Algorithm Configuration.

5.1.1 Problem Statement

As discussed in Section 4.3, different sets of features have been proposed in the literature to characterize continuous black box problems, requiring different sets of samples for their estimation. Mersmann et al. [Mer+11] proposed six different sets of features: level set, meta-model, y-distribution, convexity, curvature, local search features (see Section 4.3 for details) , and empirically demonstrated that they can indeed be used for some learning task, though there is still room for improvement. Following the same experimental procedure, Muñoz, Kirley, and Halgamuge [M+15] empirically demonstrated that the information content features (see Section 4.3.3) can also be used to characterize continuous problems.

However, despite these promising results, the required sample set used to compute these features remains too large (around $2000 \times d$) when only a small budget of function evaluation calls is available (say, around $500 \times d$). Therefore, a first research question is to study how badly the estimations of the features behave when the size of the sample set used to compute them is decreased.

On the other hand, a prominent approach to handle expensive objective functions in optimization has already been proposed in the numerical engineering community, relying on *surrogate models*, i.e., regression models of the objective function built upon sample points gathered during the run of the algorithm, and used in the optimization algorithm in lieu of the actual objective function.

Building on these ideas, we will proceed in three steps here. The first step is to choose the problem features that can be used in an operational context, i.e., whose computation only depends on a fixed sample set of objective function values, hence upper bounding the number of calls to the objective function (see Section 4.4 for the description of the selected features). Next, we empirically study the accuracy of features when smaller sample sets are used, which we will call *sub-sampled features*. Finally, the approach using surrogate models will be investigated: the features of the surrogate model can be easily computed since the cost of evaluating a surrogate model is negligible compared to the original objective function. However, the question of the accuracy of the estimation of the features of the actual objective function will be discussed, as the ultimate goal is not to estimate accurately the features, but to efficiently configure some optimization algorithm. In any case, both properties will be experimentally assessed here using the test functions from the BBOB test bench [Han+10] (see Section 3.1.1).

5.1.2 Features for Continuous Optimization

As discussed in Section 4.4, the features introduced for continuous optimization problems can be divided into two classes w.r.t. their computational cost:

- *Cheap* features are composed of 51 different features classified into 5 sets – 3 $y-$*Distribution* , 18 *Levelset*, 9 *Meta-Model*, 16 *Dispersion* and 5 *Information Content* features – that are computed from a fixed sample set.
- *Expensive* features are composed of 25 features classified into 3 sets – 14 *Curvature*, 4 *Convexity*, 7 *Local Search* features – that are computed using a fixed sample set **plus** some additional samples computed on the fly during their evaluation. The size of the additional sample set cannot be predicted a priori, making the use of these features impossible in practice.

Because, in real-world conditions we aim at using the smaller sample size, then using *Expensive* features with a fixed budget for their additional cost. Therefore, in the following of this work, only the **cheap features** will be considered.

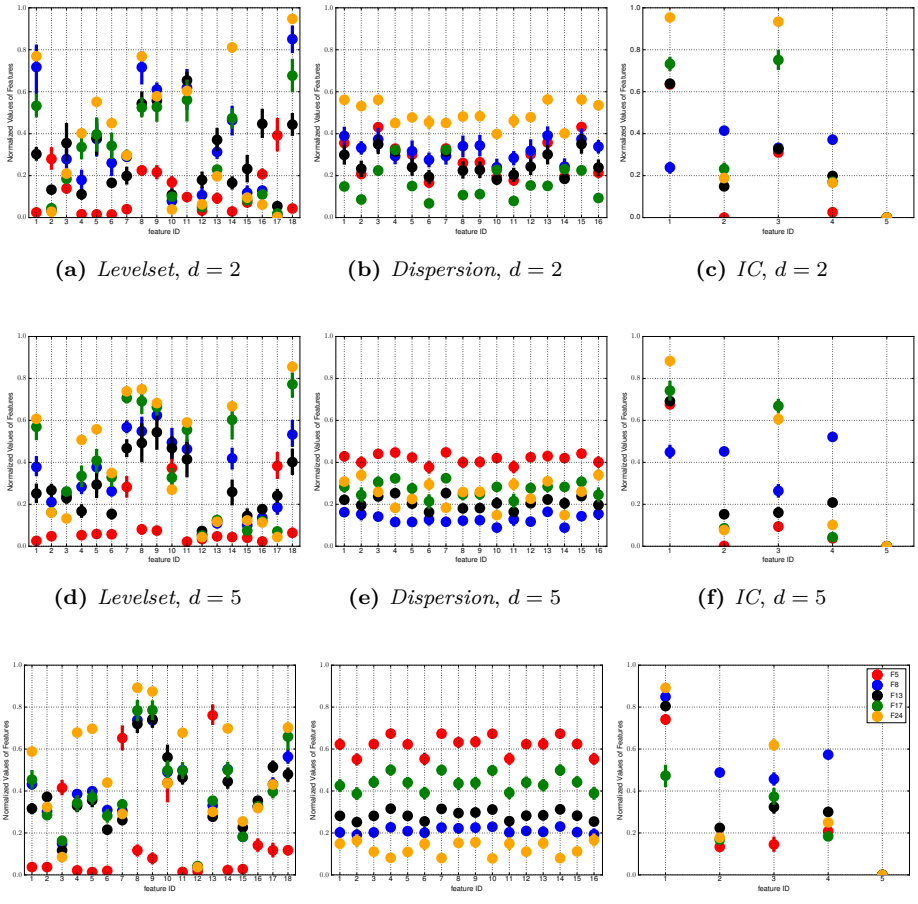
5.1.3 Features Computation: the "Ideal" Case

Muñoz, Kirley, and Halgamuge [M+15] and Mersmann et al. [Mer+11] argue that the minimal sample size to compute the aforementioned features is of size $s \geq 500 \times d$, and their results seem to demonstrate that this sample size is probably sufficient to appropriately estimate the features. However, for a better understanding of the computation of the features, and for the validation of their claim, we will here compute the values of the features using a very large sample of size $2000 \times d$, and denote the resulting values as 'exact values' in the following.

We have chosen five test functions (randomly chosen one function of each BBOB test classes), namely F5, F8, F13, F17, and F24 of the BBOB testbench, i.e., one of each manually defined class (see Section 3.1.1), and we compute the 'exact values' of all the features class described in Section 5.1.2 for each of these test functions.

Figure 5.1 shows the mean and standard deviation of the exact feature values (i.e., computed from a sample set of size $s = 2000 \times d$) of 3 different features classes, over 15 independent sample sets uniformly drawn over the search space. For each feature of each class, the values are first normalized across the whole BBOB set of test functions. Each subplot displays the mean and standard deviation (x-axis), and all 5 functions cited above (colors).

In some cases, the variation of the feature values from one function to the next is clearly visible, for example in the Levelset class for any dimension. However, no



(a) *Levelset*, $d = 2$ **(b)** *Dispersion*, $d = 2$ **(c)** *IC*, $d = 2$

(d) *Levelset*, $d = 5$ **(e)** *Dispersion*, $d = 5$ **(f)** *IC*, $d = 5$

(g) *Levelset*, $d = 20$ **(h)** *Dispersion*, $d = 20$ **(i)** *IC*, $d = 20$

Figure 5.1: Examples of mean values (y-axis) with standard deviation (bar) from 15 independent samples, for different features (x-axis), for different classes of features (from left to the right, *Levelset*, *Dispersion*, *Information Content*) and for different dimensions (from top to bottom, $d = 2, 5, 20$). .

single feature class emerges, that would be sufficient to discriminate the different classes.

We will now use T-SNE (T-distributed Stochastic Neighbor Embedding) to display on a 2-dimensional plot all the functions from the BBOB testbench represented by their features. T-SNE [MH08] is a well-known dimensionality reduction method used to visualize high-dimensional data, that converts similarity metrics between initial data points into joint probabilities so as to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. In other words, points that are close in the high-dimensional space remain close, while distances are not preserved for faraway points.

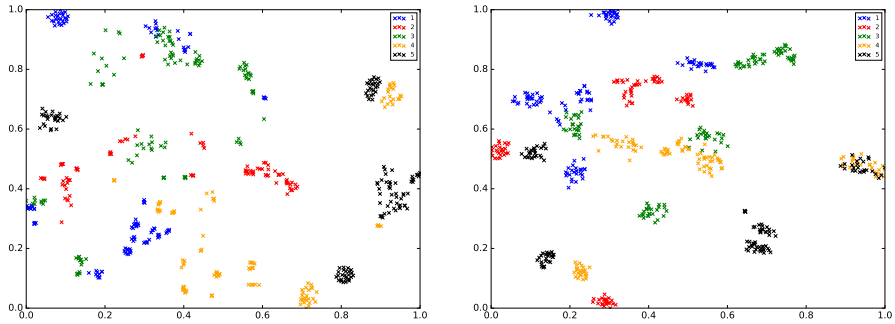
Figure 5.2 shows the results of the T-SNE (with two components), displaying in different colors the 5 classes of BBOB (24 test function with 15 independent variants — from for each function), originally described by the 51 exact features (cheap features have been computed for this experiment). We can observe that all the classes are indeed well separated. We observe natural clusters composed of the classes 4 and 5 that are well separated in lower dimensions. But for higher dimensions the results are unclear: classes 1, 2 and 3 are well separated while classes 4 and 5 tend to be closer. Regarding the results for class 1 (separable functions) , we observe a dispersion of the test functions in the T-SNE component space, whereas classes 2 and 3 tend to be very close to each other.

Classes 4 and 5 are multimodal test functions, which explains the natural clusters observed in lower dimensions. These observations are in accordance with the Figure 5.1, that shows a large standard deviation of the mean for some features. The observations are in accordance with the BBOB classification: in particular, the unimodal test functions (classes 1, 2 and 3) and the multimodal test functions (classes 4, 5) are well separated.

5.1.4 Accuracy vs Efficiency

Given a costly objective function, when a limited budget is available, we aim at computing features from as small as possible samples of objective function values. This raises the issue of the choice of the minimal sample size that can be used for an accurate approximation of the features.

The experiments described in this Chapter will aim at assessing the quality of the features obtained on small sample sets. A natural approach to measure this quality is, of course, to compare them with the 'exact values' that can be computed using a large sample set (see Section 5.1.3 above) on a representative set of functions (e.g., the BBOB test functions). A first measure will hence be



(a) $d = 2$ **(b)** $d = 20$

Figure 5.2: Visualization of the problem functions of the BBOB test bench with respect to their problem features, using a T-SNE dimensionality reduction tool. T-SNE is applied with the euclidean distance between the 51 cheap features as similarity metric, in order to observe how the the BBOB classes (**class 1**, **class 2**, **class 3**, **class 4**, **class 5**) are spatially partitioned in the 2D space by T-SNE. .

the L^2 norm of the error vector, the difference between the approximated and the 'exact' values.

However, our ultimate goal is to embed these features as decision variables in some learning process, e.g. Algorithm Selection or Algorithm Configuration. But before doing so, in accordance with [Mer+11; Bis+12; M+15], we will empirically investigate the efficiency of these features in simple learning process that is less prone to noise: a classification problem which aim at retrieving the known classes of manually defined test functions defined on the BBOB testbench [Han+10] (see their description in Section 3.1.1).

5.1.5 Surrogate Modeling

Coming from another field, numerical engineers have tackled this problem using *Response Surface Methods* for many decades now: after few iterations of any optimization algorithm, many points of the search space have already been evaluated, and this sample set can be used to build a *surrogate model* of the objective function, that can in turn be used in the optimization algorithm as a proxy for the actual objective, being less costly to compute (see e.g., [JCS00; Jin05] for surveys in the engineering domain and in Evolutionary Computation domain, respectively).

The most critical issue to be addressed when using surrogate modeling techniques is the choice of the function space where to look for a model. Several approaches have been proposed in the literature to solve such regression problem, from Neural Networks to Gaussian Processes (aka Kriging) [Mat63] (see, e.g. [JMY05; MS05; Sim+01; Kle09], for an overview on kriging as surrogate model) to Support Vector Machines [VV95; V+97], with or without kernel based methods [AW99; CS00], and Regression Random Forest[Bre01].

Regarding the evolutionary computation community, surrogate models, have also been considered to reduce the overall number of objective function calls [Jin11; LSS13], or reduce the domain search space to the more discriminant variables. In particular, some recent works [BPH15] empirically study the best choice of surrogate model to embed into an optimization algorithm. Indeed, a surrogate model used within an optimization process should be accurate, and should take into account the local characteristics of the problem [KHK06]. The surrogate model can be trained *a priori* of the optimization process, then a method that is able to capture the global topology of the problem may be more interesting to locate a basin of attraction or a plateau. Or the surrogate model can be tightly embedded within the optimization process, such that the surrogate model is recomputed to improve its accuracy, or a new surrogate model is recomputed to infer the best new points to consider for the next iteration of the algorithm.

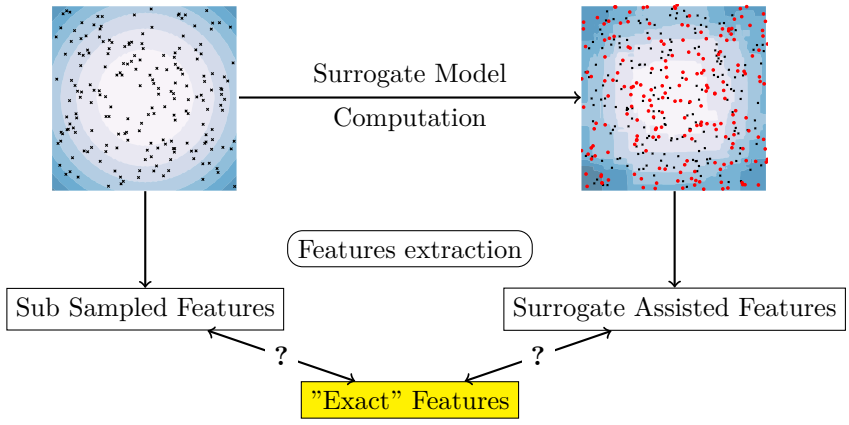


Figure 5.3: Schema of the features computations (sub-sampled and surrogate-assisted features) description .

When it comes to the computation of sub-sampled features, it is clear that the goal is to retrieve the global properties of the landscape, using some uniform sample of the whole search space. This leads to the following process.

The surrogate model is trained from an initial sample $\mathcal{X}^d$ and then used as a *proxi* to artificially increase the number of function evaluation calls to hopefully improve the estimations of the problem features. Figure 5.3 gives an overview of the proposed methodology: the features are computed from the initial sample $\mathcal{X}^d$ augmented with the additional samples $\mathcal{X}'^d$ of values of the surrogate model.

The rest of this Section will experimentally investigate the choice of the type of surrogate model, and assess the efficiency of the proposed approach. Indeed, before investigating the choice of surrogate model, in a first phase, we will empirically investigate the features computation from smaller samples of the true objective function, so that we can compare the effect of surrogate models in a second phase.

5.1.6 Experimental Setup

5.1.6.1 Experimental Setting

All following experiments are based on the BBOB test functions (Section 3.1.1) with dimension $d \in \{2, 3, 5, 10, 20\}$.

Sample Sets
The quality of the features values computed over a sample set greatly depend on its size (object of study of these experiments) and the sampling method. Following [Mer+11; M+15; LW06], we use a uniform sampling, as we aim at capturing information about global properties of the search space, without favoring any particular region.

All sample sets are hence drawn uniformly on $[-5, 5]^d$ for a BBOB function in dimension d. In the remaining of this empirical study, all samples set sizes are normalized w.r.t. the dimension d of the definition domain of the objective function. For the sake of brevity, we will only mention the ratio between the sample set size and the dimension: "a sample of size k" will actually mean "a sample of size $k \times d$". In all experiments, $k \in \{30, 50, 100, 500, 1000, 2000\}$ (the largest value 2000 was used in Section 5.1.3).

Features
As discussed in Sections 4.4 and 5.1.2, only the *cheap* features will be used in this work, namely: 3 Distribution features, 9 Meta-Model features, also 16 Dispersion features, and 5 Information Contents features. All considered features are computed using the R package kindly made publicly available by Pascal Kerschke at http://github.com/flacco.

Surrogate Models
As discussed in Section 5.1.5, several approaches to surrogate modeling will be used and compared: Gaussian Processes, Random Forests, Support Vector Machines with polynomial or RBF kernel, denoted respectively GP, RF, SVM_P and SVM_{RBF} (see Appendix B.1 for more details on each methods). But learning a surrogate model requires some hyper-parameters to be tuned: a grid search is performed in the hyper-parameter space (4 parameters for GP, 5 for the other models) , using a 5-folds cross-validation procedure — randomly re-sample (Bootstrap) the sample set, using 80% for training the surrogate and the remaining 20% for testing— for 300 iterations, optimizing the approximation accuracy on the test set. All surrogate modelling procedures are implemented using the python scikit-learn library[1].

5.1.6.2 Experimental Protocol and notations

For a given objective function $\mathcal{F}$ (one trial of one instance of one $d-$dimensional function from BBOB), the basic experiment goes as follows: one samples set of a given size is drawn from the definition domain of $\mathcal{F}$. Following the features description in Figure 5.3, a uniformly randomized sample set is used to compute the sub-sampled features on the true objective function. Next, with the sample sample set, a surrogate model is computed by using one of the chosen modeling techniques. The sample set is then completed with more samples, using the surrogate model in lieu of the original function. Surrogate assisted features are then computed using this extended sample set.

An immediate validation of such approximated feature values can be made by comparing them to the "exact" values: a proxy for these values will be the features computed with the largest initial sample set, of size 2000 (see Section 5.1.3). However, the global validation (see below) requires to compute such approximated features for all BBOB functions, and for several different samples set sizes.

Let $\mathcal{X}^s, s = 1, \ldots, S$ be some sample sets from the domain of $\mathcal{F}$[2]. Features $\Phi(\mathcal{F}^s)$ (vector of $\mathbb{R}^F$ if F is the number of features) are computed for $\mathcal{F}$ from sample set $(\mathcal{X}^s, \mathcal{Y}^s)$ (with $y_i = \mathcal{F}(x_i)$ for all $(x_i, y_i) \in (\mathcal{X}^s, \mathcal{Y}^s)$, see Section 4.4). Let us denote by $\Phi(\mathcal{F}^*)$ the features computed from the largest sample set (of size 2000).

Each sample set $\mathcal{X}^s$ is also used to learn some surrogate models $\widehat{\mathcal{F}_t^s}$ using different surrogate modeling techniques $t = T_1, \ldots, T_T$. For each (s, t), the set of

[1] http://scikit-learn.org/

[2] By abuse of notation, s will denote both the sample set and the (normalized) size of the sample set.

features $\Phi(\widehat{\mathcal{F}_t^{s,s'}})$ is computed for $\widehat{\mathcal{F}_t^s}$, after completing the sample set $\mathcal{X}^s$ with s' new points using $\widehat{\mathcal{F}_t^s}$, resulting in sample set $(\widehat{\mathcal{X}_t^{s,s'}}, \widehat{\mathcal{Y}_t^{s,s'}})$ of size $s + s'$ (i.e., $\mathcal{X}^s \subset \widehat{\mathcal{X}_t^{s,s'}}$ and, for all $(x_i, y_i) \in (\widehat{\mathcal{X}_t^{s,s'}}, \widehat{\mathcal{Y}_t^{s,s'}})$, $y_i = \mathcal{F}(x_i)$ if $x_i \in \mathcal{X}^s$ and $y_i = \widehat{\mathcal{F}_t^s}(x_i)$ otherwise). All approximate features $\Phi(\mathcal{F}^s)$ and $\Phi(\widehat{\mathcal{F}_t^{s,s'}})$ can then be compared to the "exact" features $\Phi(\mathcal{F}^*)$, and their accuracy assessed using the L^2 norm in $\mathbb{R}^F$ of the errors on the feature values ($Err(\mathcal{F}^s) = ||\Phi(\mathcal{F}^s) - \Phi(\mathcal{F}^*)||_2$, $Err(\widehat{\mathcal{F}_t^s}) = ||\Phi(\widehat{\mathcal{F}_t^{s,s'}}) - \Phi(\mathcal{F}^*)||_2$).

However, as discussed in Section 5.1.4, another comparison is needed between the approximate features and the "exact" values, that relates to the ability of the approximate features to correctly classify the BBOB classes. Such validation requires the computation of all approximate features with same sizes of sample sets for all instances of functions of BBOB testbench.

Classification Efficiency

We will measure the efficiency of a set of approximated features as a whole by using them as input for learning a classifier in order to discriminate the five BBOB classes. This is done using a 5-fold cross-validation procedure, repeated 100 times. Let us denote $Cl(\mathcal{F})$ the class (in 1..5) a given function $\mathcal{F}$ belongs to. For a given sample set size s, the example set for the classification task consists of $(\Phi(\mathcal{F}^s), Cl(\mathcal{F}))$ pairs when dealing with features computed on $\mathcal{F}$ and $(\Phi(\widehat{\mathcal{F}_t^{s,s'}}), Cl(\mathcal{F}))$ when dealing with surrogate model t built on $\mathcal{F}$ ($t \in \{GP, RF, SVM_P, SVM_{RBF}\}$). Such example set is made of 5 trials $\times$ 5 instances $\times$ 24 functions $\times$ 5 dimensions. Out of these 3000 examples, 80% are randomly chosen *without replacement and avoid any overlapping the training and test set*, equally distributed in the 5 classes, to build the training set, on which a Random Forest classifier is trained (with default hyper-parameters from scikit-learn).

The accuracy of the resulting classifier should then be assessed on the remaining 20% of the global example set. However, different scenarii are possible in real-world situations. A first scenario is when the training phase is done on cheap functions, for which it is possible to compute the features with large enough sample sets, and the unknown functions on which to perform algorithm selection/configuration (the test phase) are all expensive. An "orthogonal" scenario is when the functions available for training are also expensive. In the latter case, only approximate features will be available for training, either computed on small samples, or computed using a surrogate of the functions used for training.

Two situations similar to the ones described above will be experimented with here, involving different sample sizes for the training of the classifier and its test.

When no surrogate model is involved, an experiment studying the efficiency of the approximate features as a basis for classification (Section 5.1.4) is defined with only two parameters: the size s_{train} of the sample set used to learn the features for the training set, and the size s_{test} of the sample set used to learn the features for the test set. The classification accuracy of the resulting classifier will be denoted $Eff(s_{train}, s_{test})$.

But when analyzing the efficiency of surrogate assisted feature computation, an experiment is defined with 5 parameters: the type T of surrogate model (in $\{GP, RF\ , SVM_P, SVM_{RBF}\}$) , and, for both the training features and the test features, the sizes of the original sample sets used to learn the surrogate models (respectively s^{org}_{train} and s^{org}_{test}), and the additional number of points added to these original sample sets using the surrogate models (respectively s^{surr}_{train} and s^{surr}_{test}). The classification accuracy of the resulting classifier will be denoted $\widehat{Eff}(T, s^{org}_{train}, s^{surr}_{train}; s^{org}_{test}, s^{surr}_{test})$. Note that if one of the s^{surr}_{train} or s^{surr}_{test} is 0, only the true values of $\mathcal{F}$ are used in the corresponding step. In particular $\widehat{Eff}(T, s^{org}_{train}, 0; s^{org}_{test}, 0) = Eff(s^{org}_{train}, s^{org}_{test})$ (the surrogate model is never used).

5.1.7 Experimental Results

Two series of experiments are performed for this empirical study. The first one (Section 5.1.7.1) does not involve any surrogate model, and aims at studying how the features diverge from their "exact" baseline values when the size of the sample decreases. The goal of the second series (Section 5.1.7.2) is to check whether using a surrogate model built on the same available small sample set to complement it, can help to cope with such divergence. In both series, the divergence with the baseline values will be assessed by the accuracy of the approximated values (individual comparison for a given feature and a given function, and their aggregation in the L^2 error, and by the efficiency of the whole set of approximate features, using them to discriminate 5 BBOB classes.

5.1.7.1 Sub-Sampled Features

Accuracy of Sub-Sampled Features
Five test functions (F5, F8, F13, F17, F23) are used here to assess the effect of subsampling on the feature values. Figure 5.4 shows some typical feature behaviors on those 5 functions: whereas plot (a) display a smooth behavior, where feature values stabilize for $s \geq 100$, both other plots show that even $s = 2000$ might still be somehow too small for the multimodal functions F17 and F23. However, most features on most functions exhibited a smooth behavior, and were stable for $s \geq 500$, justifying the decision to take $\Phi(\mathcal{F}^{2000})$ as true features. It is

also clear from Figure 5.4 that small sample sizes (e.g., 30) will provide a poor approximation of the feature values, and might not allow to discriminate among different functions.

The dimension of the objective function is a relevant information that can be considered itself as a feature of the problem. Given an objective function, we assess the effect of sub-sampling on the features values for different dimension d. Figure 5.5 shows typical feature behaviors for one function with increasing dimensions, related to an increasing difficulty. The feature values are stable until $s = 100$, with an increasing variance while the sample size decreases. However, the feature values are worsened when $s \leq 100$, especially when the dimension increase. Then, small sample sizes will provide a poor approximation of the features values.

Efficiency of Sub-Sampled Features

Figure 5.6 displays the efficiency of the approximated features to discriminate among BBOB classes (see Section 3.1.1). Each line corresponds to a sample size s_{train} used to train the classifier, and each point corresponds to a different sample size s_{test} used to compute the features of the test instance to be classified.

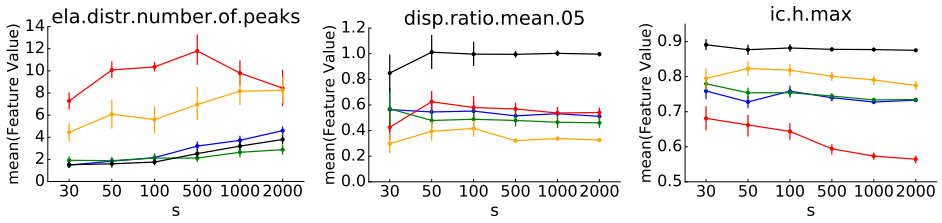
Two conclusions can be drawn from this Figure. First, there is no reason to use a larger test sample size s_{test} than the size s_{train} that was used to train the classifier, as the efficiency does not increase after s_{test} has reached s_{train}. Second, if you know that only a limited budget will be available at test time (i.e., all new instances will be very expensive), then you should train the classifier with a small budget too: for a given s_{test}, the best efficiency is obtained by the classifier trained with $s_{train} = s_{test}$.

A possible explanation, to be investigated deeper in further work, is that sub-sampling does not only increase the variance of the feature values, but it also induces some bias that might be also tracked by using the same sample size for training than for testing.

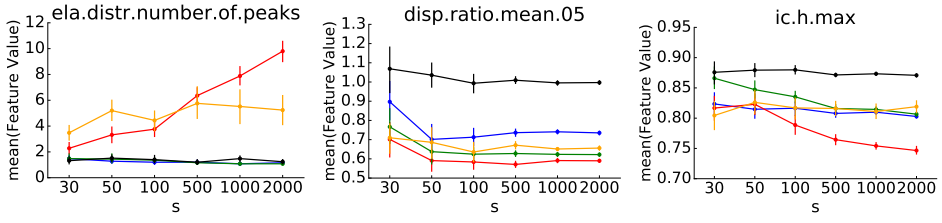
5.1.7.2 Surrogate-Assisted Features

Accuracy of Surrogate-Assisted Features

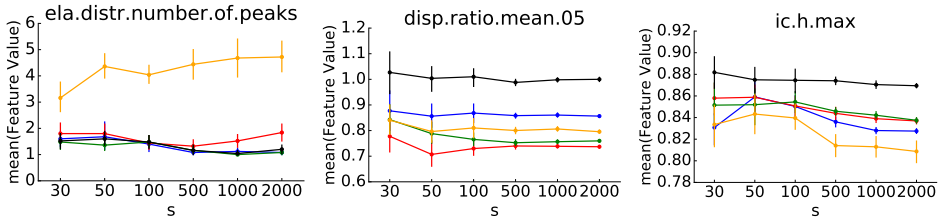
We use the 5 test functions F1, F8, F13, F17, F24 (one per class of the BBOB testbench) to empirically assess the accuracy of the features computed from the initial sample $\mathcal{X}^d$ and a sample set $\mathcal{X}'^d$ evaluated on the surrogate. Figure 5.7 displays feature values for 3 different features (columns) and for three different surrogate models (rows): Gaussian Process (Figure 5.7(b)), Random Forest (Figure 5.7(b)), SVR (Figure 5.7(d)). The effects of small sample sizes are here rather similar to those on the function alone as displayed in Figure 5.7(a):



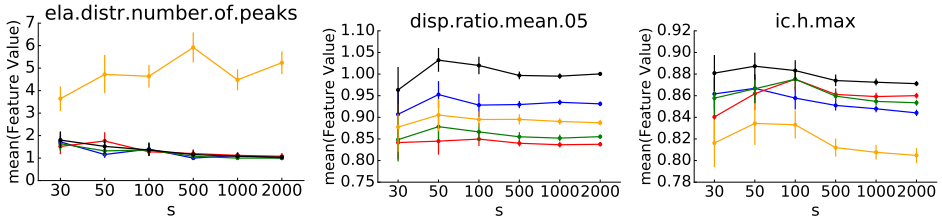
(a) $d = 2$



(b) $d = 5$

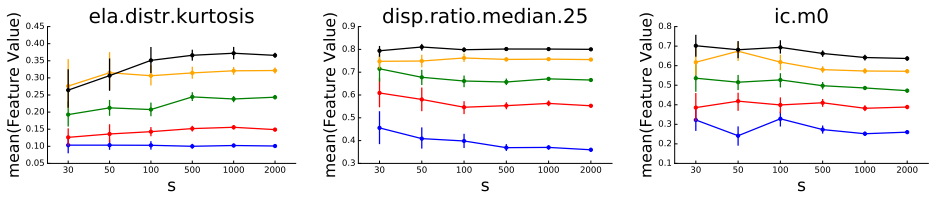


(c) $d = 10$

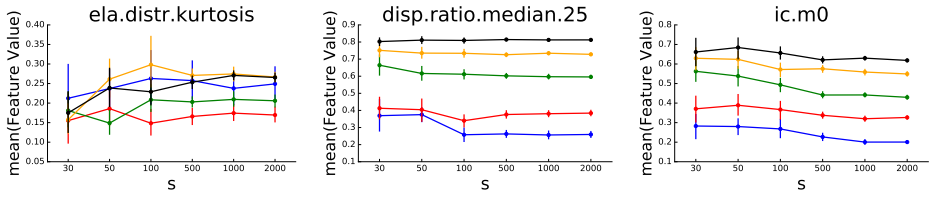


(d) $d = 20$

Figure 5.4: Examples of effect of sub-sampling (x-axis) on feature values for five different test functions (**F5**, **F8**, **F13**, **F17**, **F24**) in dimension $d \in 2, 5, 10, 20$.



(a) F5



(b) F13

Figure 5.5: Example of effect of sub-sampling (x-axis) on feature values for different dimensions ($d = 2$, $d = 3$, $d = 5$, $d = 10$, $d = 20$) for the F13 test function.

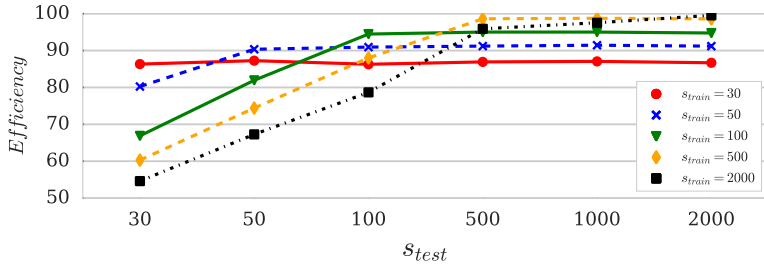
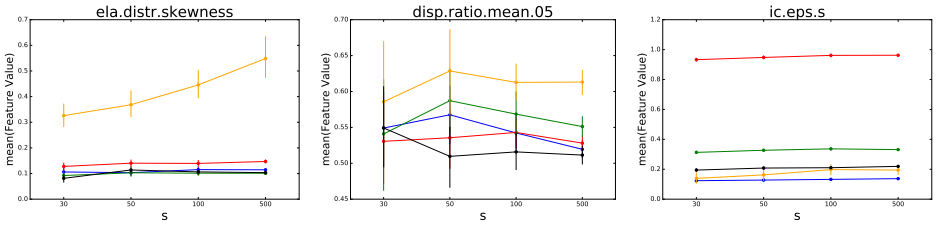


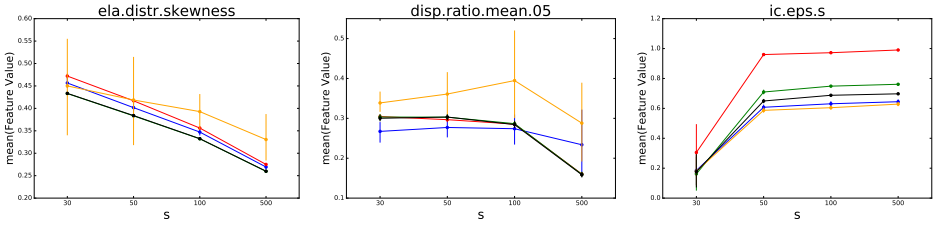
Figure 5.6: $Eff(s_{train}, s_{test})$ vs s_{test} for different values of k_{train}.

the standard deviation increases and mean value differs from the baseline.

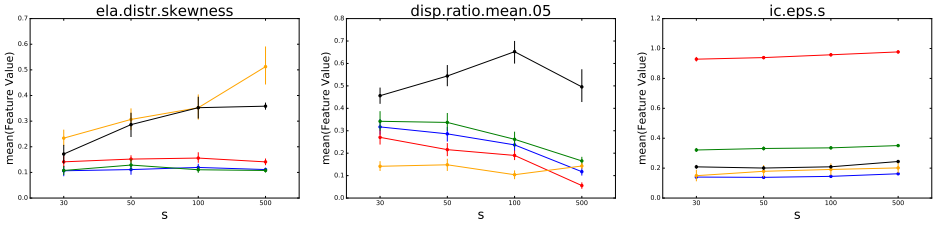
But we observe that for a given feature, the different surrogate models exhibit different behaviors, except for the Random Forest model, for which the behavior is similar to that of the sub-sampled features.



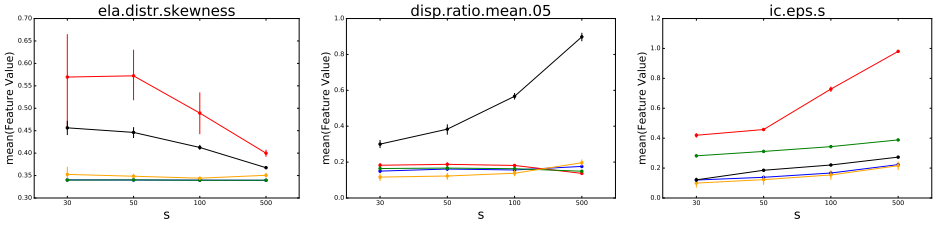
(a) Objective Function



(b) Objective Function + Gaussian Process

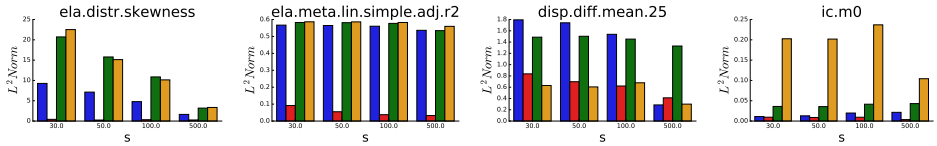


(c) Objective Function + Random Forest

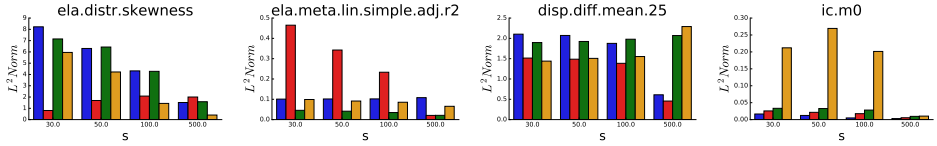


(d) Objective Function + SVR with polynomial kernel

Figure 5.7: Values of 3 features vs initial sample size $k \times d$, for the 5 test functions (**F1**, **F8**, **F13**, **F17**, **F24**) ($d = 20$) , using a surrogate model (GP for (b) , RF for (c) and SVR for (d)) to add $s' = 2000$ points to the sample set..



(a) F8 $d = 20$



(b) F24 $d = 20$

Figure 5.8: Typical results for the L^2-norm to the true value of 4 features on F8, F24 ($d = 10$) , for different values of s (x-axis). For each k, $Err(k^\star, k)$, $Err(\widehat{\mathcal{F}^{s,2000}_{GP}})$, $Err(\widehat{\mathcal{F}^{s,2000}_{SVM_P}})$, $Err(\widehat{\mathcal{F}^{s,2000}_{SVM_{RBF}}})$ and $Err(\widehat{\mathcal{F}^{s,2000}_{RF}})$ are plotted..

An alternative point of view and a comparison with the approximated features directly computed using the initial small sample with exact objective values is given in Figure 5.8. Here, the Random Forest surrogate model is used to add 2000 points to the initial sample set. It is clear (and results on other features confirm this trend) that Random Forests give a much smaller error than Support Vector Machines (with both polynomial and RBF kernel), and even more so with Gaussian Processes, as it is observed in Figure 5.8. More interestingly, using the Random Forest surrogate model results ,in most cases, in more accurate approximate features than computing their values only on the few available exact values (see the $s = 30$ histograms on Figure 5.8).

Efficiency of Surrogate-Assisted Features

Let us now look at the efficiency of the approximated features to discriminate among BBOB classes. Figure 5.9 displays $Eff(s, s)$ (the upper hull of the plots on adjacent Figure 5.6) as well as the different $\widehat{Eff}(T, s, *; s, *)$, for $T \in [GP, SVM_P$ and SVM_{RBF}, $RF]$. It is again obvious here that the Random Forest model outperforms all others, which is consistent with previous results (as well as with several other results not presented here). Hence, from now on, **only Random Forests surrogate models** will be considered. But another observation that can be made here is that there is no gain to be expected by using the surrogate model

during the learning phase too, as both plots for $Eff(s,s)$ and $\widehat{Eff}(RF,s,*;s,*)$ are almost identical.

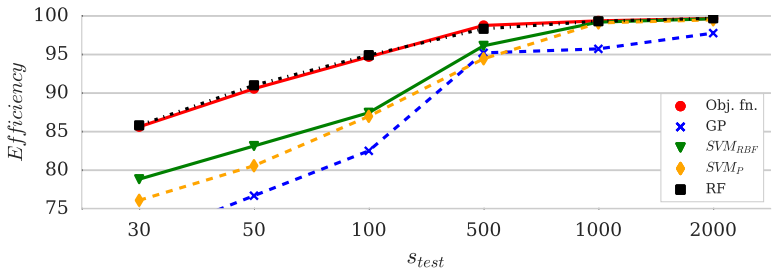


Figure 5.9: $Eff(s,s)$ (black circles) and $\widehat{Eff}(T,s,*;s,*)$ for different T..

A final experiment will try to answer the main question that motivated this work: can the use of a surrogate model improve the efficiency of the approximated features in the context of expensive objective functions. Figure 5.10 displays 4 plots, corresponding to different values of s_{train}. On each plot, the black continuous line is $Eff(s_{train},s_{test})$, i.e., the corresponding line of Figure 5.6, and the dotted grey line shows $\widehat{Eff}(RF,s_{train},0;s_{test},*)$, i.e., the efficiency obtained when using the RF surrogate model to augment the sample set with 2000 new samples. And indeed, there is some advantage in using the surrogate model during the test phase, the more so for small training budgets. Furthermore, this advantage of using the surrogate is statistically significant, as witnessed in Table 5.1 where the same data are given together with the standard deviations. Figures in bold are statistically better than the corresponding non-bold figures according to a Wilcoxon signed rank test with 95% confidence.

5.1.8 Conclusion and Perspective

In this Section, we proposed a methodology to compute features, [Mer+11; M+15; LW06], by using surrogate models to cope with expensive objective functions: to-date methods to compute such features rely on large sample sets of evaluated points, which are not practically available when dealing with expensive real-world problems. Approximated features has been empirically studied by measuring their accuracy, related to the error on their values when compared to values computed on very large sample sets, and their efficiency, ability to train a classifier

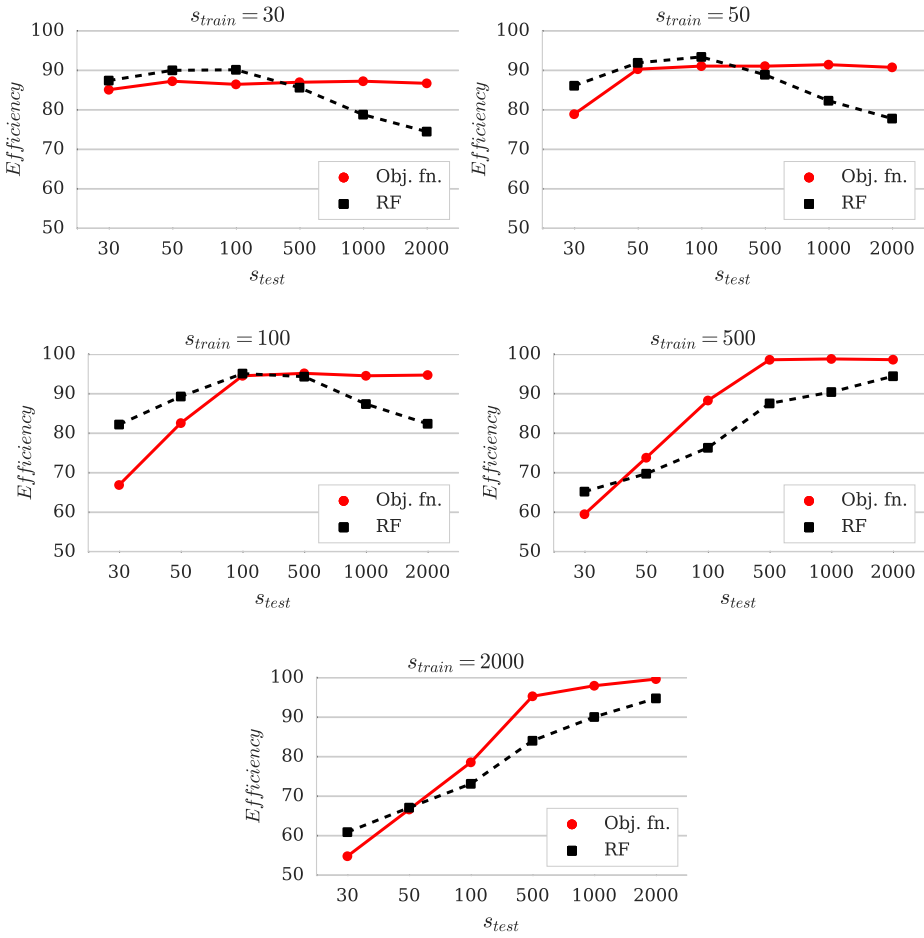


Figure 5.10: Comparison, for different values of s_{train}, of the efficiency of sub-sampled features $Err(s_{train}, s_{test}$ (continuous black line) with that of surrogate assisted features $\widehat{Eff}(RF, s_{train}, 0; s_{test}, *)$ (grey dotted line)..

that can correctly discriminates the five classes of the test functions in the BBOB testbench [Han+10].

We first investigated the computation of features with a large sample size of

$s_{train} = 30$			$s_{train} = 100$			$s_{train} = 500$			$s_{train} = 2000$		
s_{test}	Obj. fn.	RF	s_{test}	Obj. fn.	RF	s_{test}	Obj. fn.	RF	s_{test}	Obj. fn.	RF
30	85.1±1.6	**87.4±1.5**	30	66.9±3.7	**82.2±1.5**	30	59.5±3.7	**65.2±3.1**	30	54.8±3.5	**60.9±3.1**
50	87.3±1.4	**90.0±1.3**	50	82.6±3.0	**89.3±1.2**	50	**73.8±2.1**	69.8±3.1	50	66.6±3.0	**67.1±3.3**
100	86.5±1.5	**90.2±1.2**	100	94.6±0.9	**95.1±1.0**	100	**88.3±2.0**	76.3±2.8	100	**78.6±3.6**	73.1±3.0
500	**87.0±1.6**	85.6±2.0	500	**95.2±1.1**	94.3±1.0	500	**98.6±0.5**	87.6±1.8	500	**95.3±1.1**	84.0±2.4

Table 5.1: Mean and Standard deviation of the efficiency of sub-sampled features $Eff(s_{train}, s_{test}$ (columns Obj. Fn.) and surrogate assisted features $\widehat{Eff}(RF, s_{train}, 0; s_{test}, *)$ (columns RF). Statistically significantly better results (Wilcoxon signed test with 95% confidence) are in bold. .

$2000 \times d$ beyond which the computed features exhibit stable values with small variance accross sample sets. Taken separately, we empirically observed that most of these features are efficient to differentiate the BBOB function, hence concluding that this sample size can be used as a the 'ground truth' for additional experiments.

Then we investigate the loss of accuracy of the features due to sub-sampling. It was followed by the study of the efficiency of sub-sampled features, which led to the conclusion that if the sample budget is going to be small at test time, it is preferred to allocate a small budget for the training too.

Next, we empirically investigated the surrogate-assisted features, and demonstrated that in the context of this work, only Random Forest surrogate models gave satisfactory results. Surprisingly, Gaussian Processes, well known when it comes to expensive black box optimization, gave the worst results. The most interesting observation is that when only small budgets are available at test time, using a surrogate model to augment the sample leads to better classification results than using only the sub-sampled features. This somehow validates the proposed methodology.

Beyond these promising results, the link between these features and Algorithm Configuration remains to be established and investigated for small budgets. This will be done in Chapter 6.

5.2 General Experimental Protocol

In this Section, we introduce a general methodology and an experimental protocol to investigate and empirically study the Per Instance Algorithm Configuration for the continuous black box optimization.

5.2.1 Motivation

The Per instance Algorithm Configuration has been widely investigated for combinatorial and discrete domains [Hut+06; Xu+07; Hut+14], as it has been introduced in Section 3.3. But this topic remain under investigated for the continuous domain [Bos+15; MKH12; AMT12].

While several works (see [Hut+06; Xu+07; Hut+14; Kad+10; MKH12] among many others) use different experimental protocols to empirically study the performance of PIAC; there is no general experimental protocol, as far as we are aware, that aims at investigating all components of the PIAC, for any algorithm of the continuous domain.

As discussed in Section 3.3, PIAC requires a large set of problem examples to be used for the learning process of the empirical performance model (EPM). But, in order to be used in some real-world conditions when no prior knowledge on the problem can be made, the set of problem examples should exhibit several properties in order to cover a wide range of problem classes and types, starting with different dimensions, and different fitness landscape properties (such as the ones introduced in Section 4).

There exist only few test benches in the continuous domain that exhibit several different properties: like the BBOB [Han+10] and CEC [Sug+05; Lia+06b] discussed in Section 3.1. However, in addition to these benchmarks, several test problems have been used in the literature [HS80; Shi87; AD05], but most of the test functions used herein have similar properties or only exist in small dimensions, hence limiting the investigation of PIAC in larger dimensions.

The remaining of this Chapter will detail an experimental protocol that generalizes the methodology that was proposed by Leyton-Brown, Nudelman, and Shoham [LNS02] (see Section 3.3.1), and that describes a general mechanism of the PIAC in the continuous domain. Then, we propose a general setting to investigate the different component of this methodology from the training set to the learning methods, in particular including the black box continuous context. Thus, we aim at finding the best uses of PIAC such that it is assessed w.r.t. two validation procedures that involves different testbenchs.

5.2.2 Training Data and Setting

Following the discussion in Section 3.3 (see also Figure 3.5), three important aspects of the training phase must be carefully defined:

- The set of problem instances, that must exhibit various problem properties. All problem instances are described by their features, computed from a

fixed sample of objective function values.

- The set of parameter configurations, that can be either numerical (continuous or discrete) or categorical. An appropriate choice of parameter configurations among the parameter domains must be done: from uniform sampling to biased sampling toward the best known parameter configurations, possibly based on a discretization of the search space.
- The performance measure should as much as possible reflect the future experimental or real-world conditions (as discussed in Section 3.1). On the one hand, the practitioner aim at finding a parameter setting that maximize the accuracy of the solution in a given limited budget. On the other hand, the practitioner aim at minimizing the overall optimization cost (expressed in terms of the number of function evaluation calls) in order to reach a given quality.

5.2.2.1 Problem Instances

In contrast to the *robust* parameter setting, that is representative for a set of problem instances, with similar properties. PIAC aims at predicting the best parameter setting for a given problem. To this goal, the set of problem instance that is used in the training set must amount several problem properties and difficulties, like different dimensions or different classes of problems.

As discussed in Section 4.2.2, the dimension of a problem is itself a feature. Also, the complexity of optimization problems increase drastically with the dimension.

On the one hand, our ideal goal is to obtain an EPM that is valid for all dimensions (taking the dimension as a feature). On the other hand, the dimension is always available as domain knowledge, and takes, in real-world applications, only a small number of possible values. Hence two different approaches will be investigated in the following:

- The EPM_s is learned on the entries of the training data for a given dimension.
- The EPM_g is learned with all entries of the training data for all the dimensions, and the dimension d is then used as an additional feature in feature vector ψ.

In the training data, problems are described by their features. Regarding the results in Section 5.1, PIAC must be investigated w.r.t the feature computation and the initial design $(\mathcal{X}, \mathcal{Y})$:

- $\boldsymbol{\psi}^{\star}$ includes all aforementioned *cheap* features, with an initial sample of size $k = 2000 \times d$.

- $\boldsymbol{\psi}^{\bullet}_{\mathbf{k}}$ (where $k \times d$ is the size of the initial sample) only considers *cheap* features.

5.2.2.2 Parameter Configurations

The choice of the set of parameter configurations is an open research question as it highly depends on the target algorithm and experimental conditions — e.g. the performance measure or allowed budget, and the behavior of the algorithm.

Muñoz, Kirley, and Halgamuge [MKH12] tackled the PIAC and proposed to use a small set of parameter configurations (eight different parameter configurations) for their empirical validation. By contrast, our general methodology aims at exploring the parameter space and finding the best strategy for constructing an efficient set of parameter configuration.

To this end, we propose to investigate different types of parameter configuration designs:

- A popular approach is to use *grid search* (after some discretization of the continuous parameters). However, this approach typically results in an enormous set of parameter configurations ($\approx$ 8000 for Differential Evolution in Chapter 6).

- In order to reduce the size of the parameter configuration set, we can choose a subset of the whole grid, e.g., only a ratio τ of the most promising parameter configurations for each problem instance, thus avoiding all parameter configurations that have poor performance on all problems.

- Going even further in that direction, we can run some instance-specific Algorithm Configuration method, e.g., SMAC [HHL11b] or IRACE [Bir+10] (see Section 3.2.4), and retain those parameter configurations that are close to the best one selected for each problem instance.

5.2.2.3 Performance Measure

PIAC relies on a performance measure that must reflect the future usages of practitioners, who might want to maximize the convergence speed to some a priori precision, or maximize the probability to reach a given quality within a given time. Therefore, we propose to investigate the two following cases:

- First, given a large budget of function evaluation calls ($10^4 \times d$), we aim at minimizing the Expected Runtime (see the Section 3.1.1).
- Second, given a limited budget, we aim at approaching at best the optimal solution, i.e., the difference between the optimal value and the best known value, given by $\Delta_f = |f^\star - f_{best}|$

In both cases, the results will be averaged over 15 independent runs.

5.2.3 Learning Phase

Hutter et al. [Hut+14] investigated different regression methods used in Statistical Machine Learning in order to learn the Empirical Performance Model, from Ridge regression to Random Forests, and empirically demonstrated that Random Forests is the preferred method for learning the EPM. Based also on some preliminary experiments (not shown here), we have chosen Random Forest as the learning method here – using the Scikit-learn library implementation. The same preliminary experiments showed that using 10 trees and a maximal depth of 200 is a very robust setting, that will be used in all that follows.

However, while different regression methods have been widely investigated [Xu+07; Hut+14; MKH12; Bos+15], our ultimate goal is to find the best parameter configuration for a new and unseen problem, meaning that we only need to be able to **rank** the different parameter configurations. Therefore, other learning approaches, like ranking methods or collaborative filtering [MS13], can also be used as alternatives to regression methods. Thus, it seems worthy to investigate ranking methods, and we did so with the three main families of ranking methods that can be found in the literature [Liu09; Liu11]: pointwise, pairwise and list-wise approaches (see Appendix B.2 for more details of each approaches).

5.2.4 Testing Phase

The experimental protocol and settings cover several aspects that can influence the predictive power of the EPM, so that we could gain a more in-depth understanding of PIAC in the continuous domain. Muñoz, Kirley, and Halgamuge [MKH12], Abell, Malitsky, and Tierney [AMT12], and Bossek et al. [Bos+15] use a cross-validation procedure to assess the performance, but their experiments remain limited to BBOB test functions. Here, we propose two validation procedures: a cross-validation procedure within BBOB, analog to a "leave-one-function-out" cross-validation, in order to get some insight of the most promising setting within the BBOB framework; and a validation on a new test bench, such that no prior assumptions are made on the functions, as "real-world" conditions.

5.2.4.1 Cross Validation Procedure

In order to avoid any bias, the first series of experiments will use a leave-one-out procedure at the level of the test function: all examples in the training set (all BBOB functions) that are related to one problem instance (i.e., in all dimensions in the case of all-dimension learning) are removed from the training set before learning an EPM; then, considering this left-out function as 'unknown', predict the best parameter configuration from that EPM (see Section 3.3.1.3).

5.2.4.2 Validation on a new Test Bench: ExtBench

The second validation step aims at assessing the performance of the EPM on test functions that are not from the BBOB testbench. An EPM is learned on the whole BBOB testbench, and is used to predict the best parameter setting on functions that do not belong to BBOB. Therefore, we introduce a set of test functions taken from the literature [HS80; Shi87; AD05] (see http://al-roomi.org/benchmarks/unconstrained/n-dimensions and http://www.sfu.ca/~ssurjano/calibrat.html for details about their properties and their optimum values).

From these numerous functions available in the literature, we selected a set of 21, described in Table 5.2, trying to diversify their properties:

- The test functions are defined in some bounded domain, and must exist for different dimensions, as we aim at investigating the scalability of the proposed method from low to medium dimensions (≥ 32)
- The global optimum of the test functions must be known, as this is necessary for the proposed performance measures (see Section 5.2.2.3)
- The test functions have different fitness landscape properties (see Section 4.2). Hence, for each function, Table 5.2 shows the their properties.

Similarly to BBOB [Han+10], several independent runs must be done on different variants of all test functions in order to properly assess the performance of the algorithms and parameter configurations. For each independent run, the test function is rotated in the d-dimensional space with respect to the Modified Gram Schmidt orthogonalization procedure [Lon81].

5.3 Discussion

In this Chapter, we have paved the way to the experimental analysis and validation of our PIAC approach, to be presented in next Chapters.

Table 5.2: Description of n-dimensional Numerical Optimization problems with their definitions, lower and upper bounds lb and ub). In the columns properties M,U, S, P, refers to multimodal, uni-modal, separable, with plateaus. For all test function the optimal value $f^\star = 0$, except for Needle eye and Happy Cat for which $f^\star = 1$.

Function Name	**Function Definition**	$[lb, ub]^d$	Properties
Alpine1	$\sum_{i=1}^{d} \lvert x_i \sin(x_i) + 0.1 x_i \rvert$	$[-10, 10]$	MS
Ackley	$-20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + \mathrm{e}$	$[-5, 5]$	M
Rastrigin	$10d\sum_{i=1}^{d}[x_i^2 - 10cos(2\pi x_i)]$	$[-5, 5]$	S
Ridge	$\sum_{i=1}^{d}(\sum_{j=1}^{i} x_j)^2$	$[-64, 64]$	U
Rosenbrock Saddle	$\sum_{i=1}^{n-1}(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2)$	$[-64, 64]$	P
Schaffer f7	$\left[\frac{1}{d-1}\sqrt{s_i}\cdot(\sin(50.0 s_i^{\frac{1}{5}}) + 1)\right]^2$ $s_i = \sqrt{x_i^2 + x_{i+1}^2}$	$[-100, 100]$	M
Schwefel01	$\left(\sum_{i=1}^{d} x_i^2\right)^{\alpha}$, with $\alpha = \sqrt{\pi}$	$[-100, 100]$	U
Schwefel04	$\sum_{i=1}^{d}[(x_i - 1)^2 + (x_1 - x_i^2)^2]$	$[0, 10]$	U
Schwefel20	$\sum_{i=1}^{d} \lvert x_i \rvert$	$[-100, 100]$	US
Schwefel21	$max_{1 \le i \le d} \lvert x_i \rvert$	$[-100, 100]$	US
Schwefel22	$\sum_{i=1}^{d} \lvert x_i \rvert + \prod_{i=1}^{d} \lvert x_i \rvert$	$[-100, 100]$	US
W Wavy	$1 + \frac{1}{d}\sum_{i=1}^{d} cos(k x_i) e^{-\frac{z_i^2}{2}}$	$[-\pi, \pi]$	M
Schaffer f6	$0.5 + \frac{\sin^2\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{\left[1 + 0.001\left(x_1^2 + x_2^2\right)\right]^2}$	$[-50, 50]$	M
Zakharov	$\sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{d} 0.5 i x_i\right)^4$	$[-5, 10]$	U
Double Sum	$\sum_{i=1}^{d}\left(\sum_{j=1}^{i} x_j\right)^2$	$[-100, 100]$	U
Modified Double Sum	$\sum_{i=1}^{d}\left(\sum_{j=1}^{i}(x_j - j)^2\right)$	$[-10.24, 10.24]$	U
Lunacek Two Sphere	$\{\sum_{i=1}^{d}(x_i - \mu_1)^2\}, \{b\cdot d + s\cdot\sum_{i=1}^{d}(x_i - \mu_2)^2\}) + 10\sum_{i=1}^{d}(1 - cos 2\pi(x_i - \mu_1)$ with $\mu_1 = 2.5$, $\mu_2 = -\sqrt{\frac{\mu_1^2 - b}{s}}$, $b = 1$, $s = 1 - \frac{1}{2\sqrt{b+20} - 8.2}$	$[-100, 100]$	M
Griewank	$1 + \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d}\cos(\frac{x_i}{\sqrt{i}})$	$[-512, 512]$	MP
Needle Eye	$\begin{cases} 1 & \text{if } \lvert x_i \rvert < eye \ \forall i \\ \sum_{i=1}^{d}(100 + \lvert x_i \rvert) & \text{if } \lvert x_i \rvert > eye \\ 0 & \text{otherwise} \end{cases}$	$[-10, 10]$	M
Salomon	$1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{d} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{d} x_i^2}$	$[-100, 100]$	0
Happy Cat	$[(\lVert x \rVert^2 - d)^2]^{\alpha} + \frac{1}{d}\left(\frac{1}{2}\lVert x \rVert^2 + \sum_{i=1}^{d} x_i\right) + \frac{1}{2}$, with $\alpha = \frac{1}{8}$	$[-10, 10]$	U

Recent works [M+15; Mer+11; LW06] investigated the features for continuous black box optimization problems. However, their results remain limited to experimental conditions where a large budget of function evaluations is available for the computation of these features. Therefore, with the goal to use these features into real-world conditions with expensive objective functions, we investigated the computation of the features with low budgets, and proposed to use surrogate models to reduce the computational cost of features, while maintaining their accuracy and efficiency. Random Forest was found to be the best choice of the surrogate modelling in this context.

Moving toward tackling the Per Instance Algorithm Configuration, in Section 5.2, we introduced a general experimental protocol for experimentally investigating and assessing the performance of PIAC approaches, for different target algorithms, and in a way that can be generalized to different domains.

Despite a certain appeal to investigate the PIAC only on the best optimization algorithms, like CMA-ES, we now propose, as a first step, to investigate the PIAC on a simpler algorithm, with no sophisticated parameter adaptation that might be able to recover some poor initial parameter values found by our PIAC approach: the next Chapter investigates the PIAC with the original Differential Evolution as the target algorithm.

CHAPTER 6

Per Instance Algorithm Configuration

This Chapter is an empirical study of the Per Instance algorithm Configuration (see Section 3.3), tackling the continuous black box domain.

In Section 6.2, we introduce the specific case study when Differential Evolution is used as the target algorithm for PIAC, such that in Section 6.3 is introduced the experimental setting as advocated in Section 5.2. This is followed by a series of results based on different settings: first when PIAC is computed in a *cheap* case, in which 'exact' values of features can be computed; next in an expensive case when a low budget to compute features is available, such that we can investigate the several components of the training set as described in the experimental settings. Finally, in Section 6.5 we discuss the results, and conclude in Section 6.6 with some hints of further works and recommendations to learn and use an empirical performance model when the overall optimization budget is limited.

6.1 Motivation

As detailed in Section 5.2, PIAC relies on different experimental conditions, that open several research questions. These experimental conditions include different components that must be investigated before PIAC can be used in practice.

Muñoz, Kirley, and Halgamuge [MKH12] empirically demonstrated that an

empirical performance model can be learned from features of continuous black box problems. Despite promising results on CMA-ES, their experimental setting is limited: only one sample size is considered for computing features, and only 8 different parameter settings are used, and only BBOB functions are used for the cross-validation procedure. Also, it does not take into account real-world limitations of continuous black box problems, i.e., a small budget in terms of function evaluations.

This Chapter proposes an empirical study of the different key aspects of PIAC that were described in Section 5.2, and aims at empirically validating the settings of PIAC in some actual running conditions, i.e., including the cost of the feature computation in the optimization budget. The target algorithm will be Differential Evolution.

6.2 Differential Evolution: Case Study

6.2.1 Differential Evolution

Differential Evolution (DE) [SP97][1] is a well known continuous optimization algorithm that encountered many successes in the last decade (see Section 2.4). It is particularly known for its simplicity, at least in the original version. However, this original version is also known for its high sensitivity to its parameter setting, that can lead to poor performances on some optimization problems known to be difficult. This is assessed in Figure 6.1, showing the performance (Expected Run Time) for parameter settings of DE in a discretized parameter space, for two different functions with dimension $d = 2$. These Figures give a global view of the performances of DE w.r.t two of its parameters: some good parameter settings for one problem lead to a poor performance on another problem of the same test bench, which is in accordance with Section 3.2.4.

Because of this sensitivity to parameter setting, DE is a good candidate as a target algorithm for PIAC, as it makes it easier to see big differences in the results if a bad parameter setting is predicted.

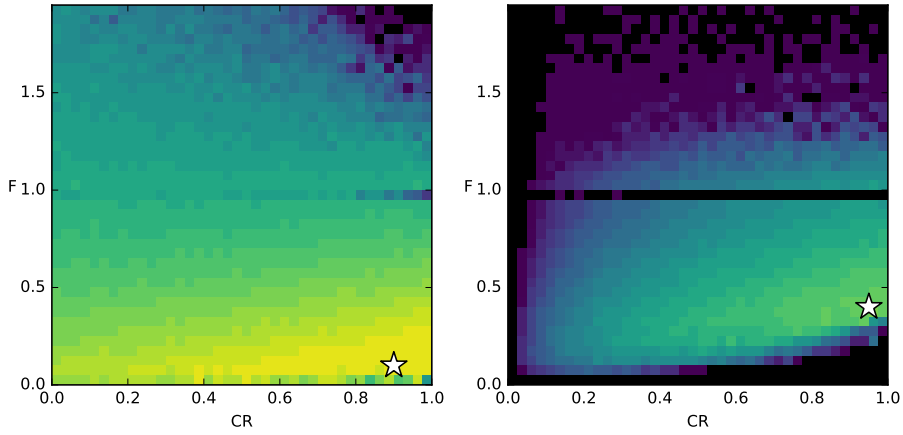
DE evolves a population of individuals, and generates new individuals from the current population by adding to each individual in turn a difference vector between two other individuals, recombining the result with another individual from the population. The original version of DE has only four static parameters (see Section 2.4):

[1] see also http://www1.icsi.berkeley.edu/~storn/code.html

1. the population size $\mathbf{NP} \in \mathbb{N}$
2. the strategy $\mathcal{S} \in \{\text{best1bin}, \text{randtobest1bin}, \text{best2bin}, \text{rand2bin}, \text{rand1bin}\}$ controls how to choose the endpoints of the difference vector;
3. the scaling factor $\mathbf{F} \in [0, 2]$ controls the intensity of the difference vector;
4. the crossover rate $\mathbf{CR} \in [0, 1]$.

The population size **NP** is kept to the default value $15 \times d$ recommended by the authors.

The authors Storn and Price [SP97] also give default values for the other parameters: $\mathcal{S} = best1bin$, $\mathbf{F} = 0.8$, and $\mathbf{CR} = 0.9$. This setting is considered for this work as a baseline for comparison, and named θ^d in all further experiments.



(a) $f =$ F1 Sphere function **(b)** $f =$ F10 Ellipsoid function

Figure 6.1: Example of the performance (Expected Run Time from 15 trial runs) colormap for Differential Evolution parameter settings for parameters **F** and **CR**, over a discretized grid, resulting in 8000 parameter configurations, on test functions sphere and ellipsoid in dimension $d = 2$. Darker colors indicate worst performances, whereas lighter colors (e.g. yellow) indicate best performances. Black colors refers to an infinite ERT, meaning that the DE never reached the target value $f_{target} = 10^{-6}$ (see Section 3.1 for more explanations) ; and ☆ mark the optimum parameter values.

6.2.2 Test Bench

As discussed in Section 5.2, two experimental validations are conducted: a cross-validation procedure on the BBOB test bench only; and a validation on the new test bench described in Section 5.2.4.2 to better assess the generalization efficiency.

For both experiments, the empirical performance model is learned on the BBOB test bench (see Section 3.1.1 for details of the different manually defined classification of test functions). However, because of the poor performance of Differential Evolution on multi-modal test functions and high dimensions [FR10], the experiment are limited here to the 14 test functions F1-F14 of the BBOB test bench, and for dimensions $d \in 2, 3, 5, 10$.

As advocated in the original framework of the BBOB test bench, for each function, 15 independent runs are actually performed on a variant of the function, in order to avoid possible algorithm biases.

Regarding the series of experiments involving the new test bench ExtBench (see Section 5.2.4.2), all test functions are used, and in accordance to the poor performance of DE, only in dimensions $d \in \{2, 4, 5, 8, 10, 12\}$.

6.3 Experimental Setting

Following the general experimental protocol (see Section 5.2), this Section describes the experimental settings related to empirically study the different aspects of the PIAC when learning an empirical performance model.

6.3.1 Learning an Empirical Performance Model

6.3.1.1 Training Dataset

Features for PIAC
In Section 5.1, we investigated the computation of feature w.r.t. the size of the sample set. From these preliminary experiments we empirically demonstrated that using a low budget of evaluation function calls can affect the efficiency of the features, whereas using some surrogate models as *a proxy* maintains their efficiency for the classification of optimization problems.

We propose to investigate the features computation in the context of the PIAC on DE as follows:

- Only the *cheap* features (see Section 4.4) will be used, and their 'exact'

values will be computed on a large sample size ($2000 \times d$). This setting will be used as the baseline for comparisons.

- Experiments will investigate sub-sampled features (see Section 5.1.7.1) and different sample sizes will be used, of size $k \times d$, with $k \in \{30, 50, 100, 500\}$.

- Last, experiments will also investigate the surrogate-assisted features (see Section 5.1.7.2). In accordance with the results observed in Section 5.1.7.2, surrogate models are built using Random Forests with the default hyper-parameters from scikit-learn [Ped+11].

Parameter Configurations
In all experiments on DE, a 40-steps discretization is used for $\mathbf{F} \in [0, 2[$ and $\mathbf{CR} \in [0, 1]$, resulting in $5 \times 40 \times 40$ different parameter configurations considering the 5 possible strategies. As a reminder, the population size NP is fixed with $15 \times d$.

For each of the 14 functions of the test bench, and for each dimension $d \in \{2, 3, 5, 10\}$, each one of its 15 variants is optimized with these 8000 DE configurations, and the ERT is computed, and features are computed for each variant. The initial dataset is hence made of the results of $14 \times 15 \times 8000$ runs of DE, i.e, **448,000** entries per dimension, and **6,720,000** entries for all the dimensions. All runs have a maximum budget of $10^4 \times d$ function evaluations.

As explained in Section 5.2.2.2, different sets of parameter configurations are studied, first with all parameter configuration of the discretized parameter space. Next we study subsets of parameter settings that only contains the 1% ($\Theta_{1\%}$) and 10% ($\Theta_{10\%}$) best parameter setti ngs (i.e. after 15 independent runs, all runs reached the target value) of each problem instances and dimensions: ≈ 100 and ≈ 360 different parameter settings[2] respectively for ($\Theta_{1\%}$) and ($\Theta_{10\%}$). Finally, we study $\Theta_\star$ the subset of best parameter settings of each problem instance and dimension of the training set, that results in 56 different[2] parameter settings.

Dimensionality
As discussed in Section 5.2.2.1, we suppose that the dimension d play a key role for PIAC and for the scalability of PIAC. As the performance of Differential Evolution worsens when the dimension increases [FR10], the generalization of the EPM to different dimensions will be investigated in the following way:

- The EPM_s is learned with the entries of the dataset for only one given dimension.

[2] Note that duplicate parameter setting are removed

- The EPM_g is learned with all the entries of the dataset, for all the dimensions, and the dimension d is then be used as an additional feature in the feature vector ψ

6.3.1.2 Learning Methods

The following experiments aim at investigating the choice of the method for learning the empirical performance model. Following the general experimental protocol defined in Section 5.2, the EPM is first learned with a Random Forest regression, with the default hyper-parameters from scikit-learn [Ped+11].

Next, rank based methods (see Appendix B.2) are investigated, such that for each problem instance and dimension, the rank (found w.r.t. the ERT metric) of parameter configurations is used in lieu of their actual performance (ERT), and three different methods are studied: Logistic Ordinal regression [McC80; Wat86], RankSVM [HGO99; Joa02], and Listnet [Cao+07] – all with the default hyper-parameters as defined in their original papers.

6.3.2 Cross-Validation on BBOB

In order to avoid possible biases, all experiments are based on a leave-one-out procedure: one of the 14 functions in the test bench is completely removed from the dataset (all dimensions and all variants). An EPM is learned, and the left-out function, considered as 'unknown', is tested. The ERT (Expected Run Time, see Section 3.1.1) is used to compare the performances. At this point, we are only interested in the values of predicted parameters, and their performances when they are applied to DE. Hence, the cost of features (the sample size) is not included in the performance of predicted parameter settings.

6.3.3 Validation on ExtBench

The validation on ExtBench is performed only on a generalized empirical performance model EPM_g. Once the EPM is learned for some features setting (sample size and computation method), and a set of parameter settings. Then for each test function and dimension of the new validation test bench (see Section 5.2.4.2), with the dimensions and test functions defined in Section 6.2.2. The overall budget is limited to $10^4 \times d$ function evaluation calls, including now both the features cost and the optimization cost of the problem instance.

Empirical Cumulative Distribution Functions –ECDF (see Section 3.1.1 for details)– plots are used to present all the results.

6.3.4 Baseline Comparison

As the goal of PIAC is to find a parameter setting that improve the performances of DE, all experiments will be compared to the default parameter setting, described in Section 6.2.1. As discussed in Section 3.3, in contrast to a *robust* parameter setting that is found for a set of problem instances, the PIAC aims at finding a good parameter setting for one problem instance based on its features. Therefore, as a baseline comparison, we propose to also compute the *robust* parameter setting, for each dimension $d \in 2, 3, 5, 10$, using SMAC [HHL11b] (as described in Section 3.2.4), run on the 14 test functions of BBOB.

Last, we aim at empirically studying the PIAC when only a small budget is available to compute features, in order to cope with some realistic real-world situations. Therefore, one baseline comparison will be the PIAC with 'exact' features requiring too many additional function evaluations to be used when dealing with expensive objective functions.

6.4 Results

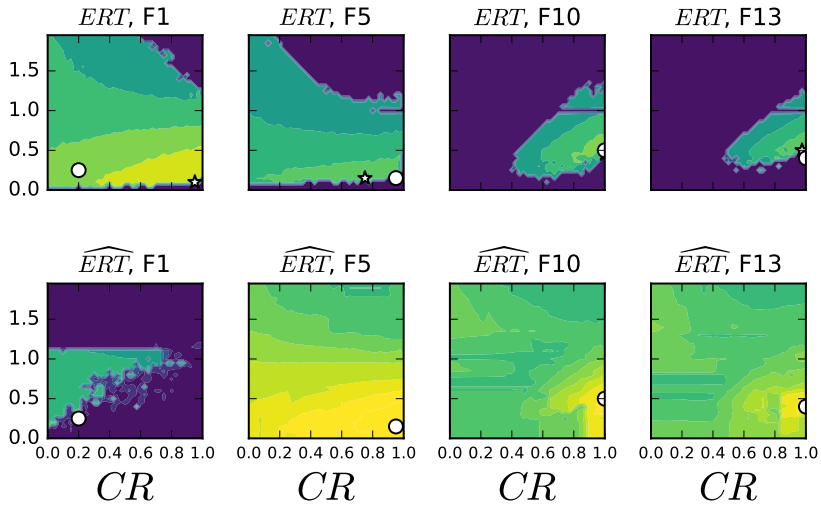
6.4.1 PIAC: Ideal Case

6.4.1.1 Predicted Performance Map

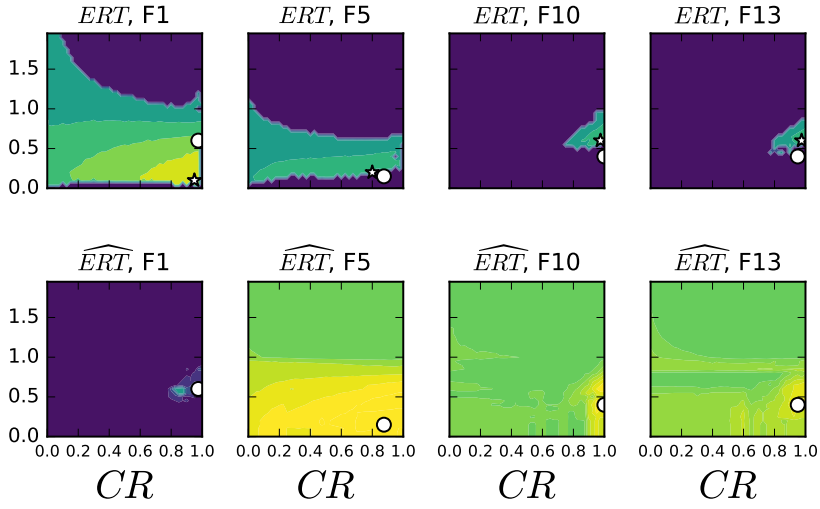
Regarding a cross-validation procedure, once an EPM is learned, for each instance of the removed test function, a grid search is performed to find the optimum of the empirical performance model. Figure 6.2 displays an excerpt of the grid search of the discretized parameter space of DE, for different test functions for $d = 5$ and $d = 10$.

We can observe similarities between the true and predicted performance maps, in particular in regions of good parameter settings. In most cases, for most test functions, we observe that good parameter settings are predicted with the EPM, that approaches the true best parameter setting.

However, for a few test functions like F1 Sphere or F5 Linear Slope, the predicted performances map greatly differs from the true performances map, hence misleading the location of best parameter settings. Regarding F5, the only linear function of the test bench, considering that the EPM is learned without any linear function in the training set, the global best predicted parameter setting is rather good. But unfortunately, in some other cases, like F1 for $d = 5$ or F13 for $d = 10$, the small region of the parameter space where the EPM differs from the true ERT is the region that contains the optimal configuration, hence the predicted parameter setting has a very poor performance.



(a) $d = 5$



(b) $d = 10$

Figure 6.2: Examples of comparisons between the true ERT (top) and the predicted ERT of the EPM (bottom) for 4 functions. The EPM, has been learned on all the dimensions, with $\psi^{\star}_{2k}$ and the set of discretized parameter setting. Each subplot shows ERT and $\widehat{\text{ERT}}$ colormaps (without interpolation). ★'s indicate the true optimal parameter configurations, and ∘'s the best predicted parameter configurations..

6.4.1.2 Empirical Optimal Parameter Setting at Work

After the cross validation, for each test functions and dimension, Figure 6.3 displays the performance of empirical optimal parameter settings found within a grid search into the parameter configuration set (here, all the parameter configurations after a discretization of DE parameter space). It compares the ERT of the empirical optimal parameter setting predicted from EPM_s or EPM_g, with those of the *robust* parameter setting found from all the test functions, and of the default parameter setting θ^d. Finally, the *specific* parameter setting (tuned for each test function) θ^L, found with SMAC [HHL11b], is also used as the ideal case for comparison. Without any surprise the specific configuration outperforms all parameter configurations.

The good news is that the default parameter setting recommended by DE authors is the clear loser: all EPM-based approaches seem to be able to improve on the default setting. The results in dimensions 2,3 and 5 bring several other good news: most proposed approaches of instantiation of the EPM also outperform the *robust* configuration. However, the EPM demonstrates poor performance on some functions, e.g. F3, F4, and F13. On the opposite, in some few cases, the performances of some parameter configurations predicted by the EPM approach the best parameter configurations, like for F8, F9 or F10.

Focusing on the generalization of the EPM, as expected, learning only from the single target dimension gives better results than learning all aggregated dimensions: the EPM_g rarely outperforms EPM_s, except for $d = 10$.

On the other hand, in several cases, EPM_s clearly outperform EPM_g. But when d increases, both fail on some functions, i.e. they never reach the target f_{target} within the allocated budget. In some cases, it can be very hard to achieve a good performance prediction, as witnessed with functions F3 and F4, the only multimodal functions of the test bench (though being separable, and hence belonging to the first BBOB class) or F5, the only linear test function from the test bench.

6.4.2 EPM with Sub-Sampled Features

The following experiments investigate the learning phase of the PIAC with respect to the features computed from samples of different sizes, hence investigate the minimal sample size required to compute features while maintaining good predictions of the optimal setting. Despite the fact that there is no budget limitation to compute features in the training set, we also empirically investigate the quality of the features used in the training set by using sub-sampling there, too.

Figure 6.4 shows the results of cross-validations for different sample sizes

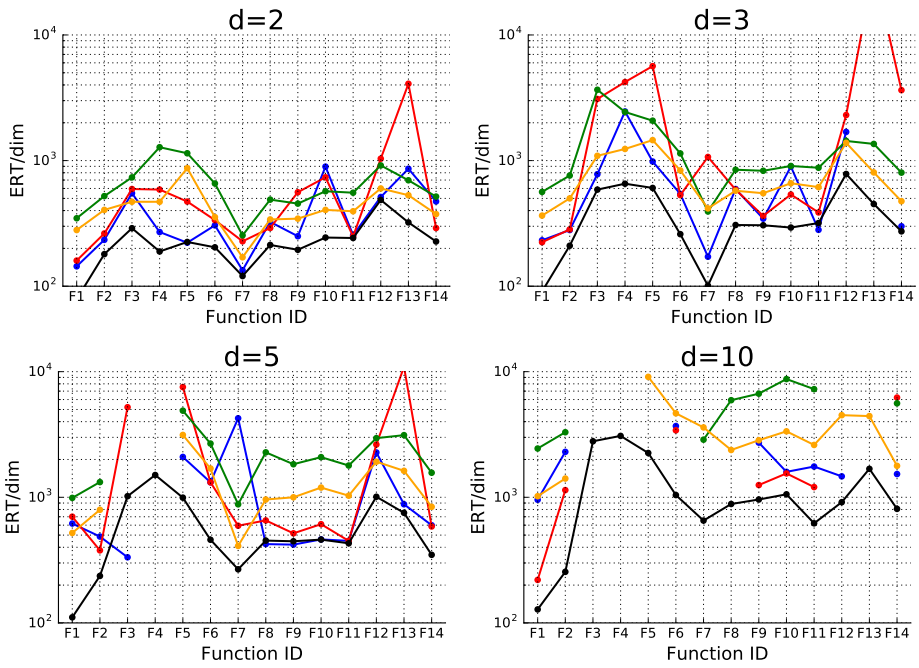


Figure 6.3: ERT of different Empirical Optimal Configurations: predicted settings **EPM**$_s$ and **EPM**$_g$ are compared to the *robust* (θ^r), the default ($\boldsymbol{\theta^d}$), and the specific ($\boldsymbol{\theta^L}$) parameter settings. Missing ERT values indicate that parameter settings never reach the target value..

used to compute features in the training set (s_{train}) and in the test set (s_{test}). It shows the results the performance (expressed as $\log_{10}$(ERT/dim)) for different instantiation of the EPM, and compares them to those of the robust parameter setting θ^r and of the EPM computed in an with the 'exact' values of features EPM $^{\star}_{2k}$.

In general, we observe that sub-sampled features help to find good parameter configurations, as for EPM$^{\star}_{2k}$. Paradoxically, for some problem functions, it even outperforms EPM$^{\star}_{2k}$ when the latter never finds the target, like for F3 or F4. Regarding the comparison with the robust setting θ^r, EPMs with sub sampling outperforms it in several cases, but θ^r is more stable in particular when $d = 10$.

Regarding the generalization of the EPM, the results are unclear, such that

the results vary with the settings, i.e. the sampling sizes s_{train} and s_{test}. The performances of EPM_g and EPM_s are similar when $d < 5$, but rather mixed when d increases: this is confirmed in Figure 6.5 that shows the performances averaged per dimension, though a slight advantage appears for EPM_s when the sample size is very small $s_{train} = s_{test} = 30$.

Focusing on small samples sizes, we observe in Figure 6.4 that when $s_{test} \leq 100 \times d$, EPM almost always outperforms θ^d and θ^r. Also, we observe that the performances of EPM_s approach the performances of $\text{EPM}^{\star}_{2k}$ without outperforming it.

In Figure 6.5, the ERT, aggregated per dimension is displayed for each pair of sample sizes for features in the training (s_{train}) and the test set (s_{test}), across the 14 test functions. We observe that the performance varies greatly with respect to the sample size. Surprisingly, we observe that low budget to compute features can greatly help to approach the $\text{EPM}^{\star}_{2k}$, hence outperforming θ^r. In addition, we observe that it is preferable to use similar sample sizes for s_{train} and s_{test}.

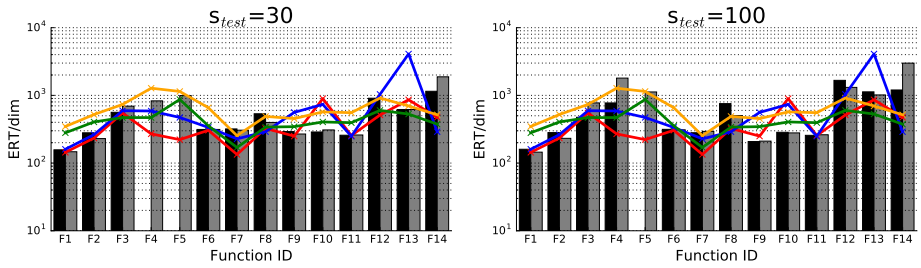
6.4.3 EPM with surrogate assisted features

The following experiments empirically study the computation of surrogate-assisted features (see Section 5.1) *in lieu* of sub-sampled features in order to cope with the low budget constraint. Figure 6.6 shows typical results of performances of the EPM with surrogate assisted features, and compares them to the robust setting and the EPM with 'exact' features. It shows typical results, in $d = 5$ and $d = 10$, comparing the ERT for different instantiations of the EPM with surrogate-assisted problem features computed with respect to increasing sample sizes.

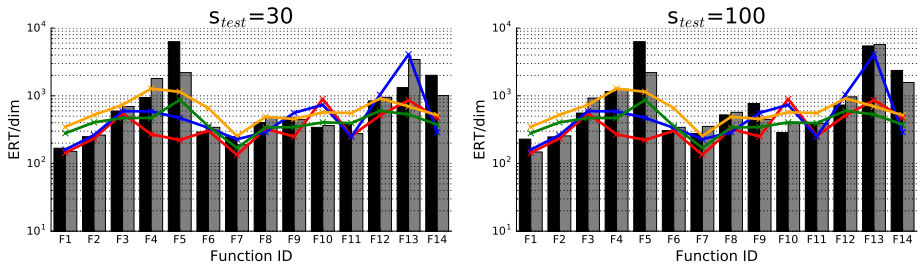
Similarly to EPM with sub-sampled features, EPM with surrogate-assisted features outperforms the robust setting θ^r when d increases, and outperforms $\text{EPM}^{\star}_{2k}$ in several cases, in particular when $d = 10$. The results are similar to those presented in Section 6.4.2. In particular, the performance remains poor on some test functions, for example for the F4 and F5 test functions. Regarding the generalization efficiency of the EPM, the results are unclear: EPM_s and EPM_g tend to have similar performances, even though EPM_s slightly outperforms EPM_g on some test functions, in particular on F5.

Figure 6.6 shows the averaged performances on the 14 test functions, for different sample sizes for the training set (s_{train}) and the test set (s_{test}) to compute the surrogate-assisted features, and compares them to sub-sampled features.

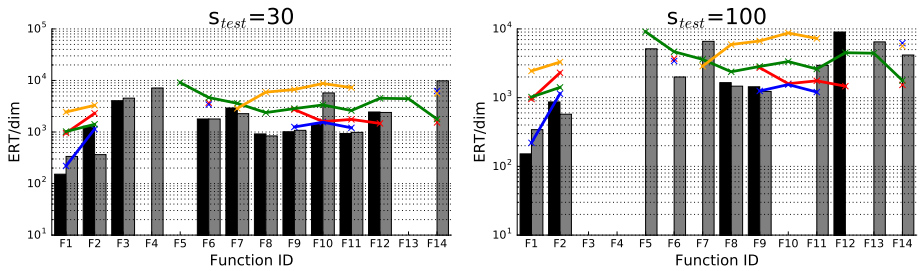
Similarly to the results in Section 6.4.2, using the same budget of function evaluations for the computation of the features in the training and the test set



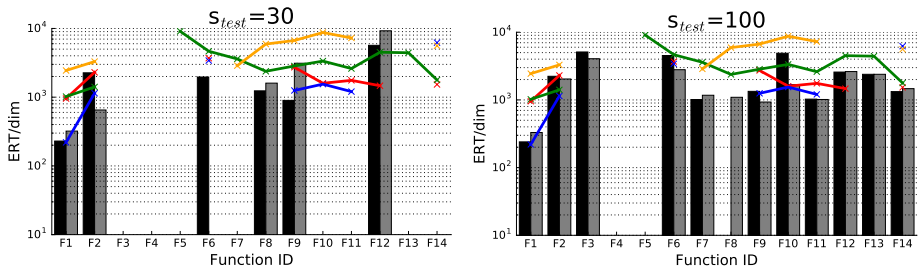
(a) $s_{train} = 30$, $d = 2$



(b) $s_{train} = 100$, $d = 2$

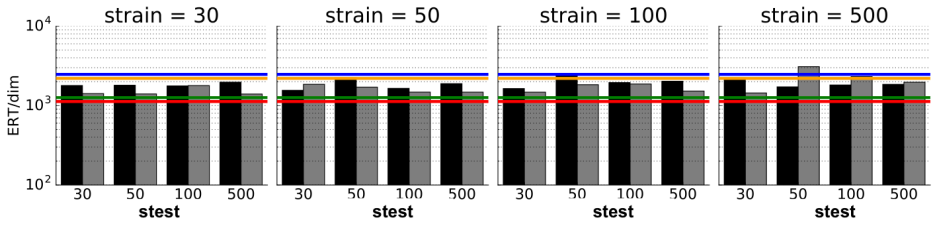


(c) $s_{train} = 30$, $d = 10$

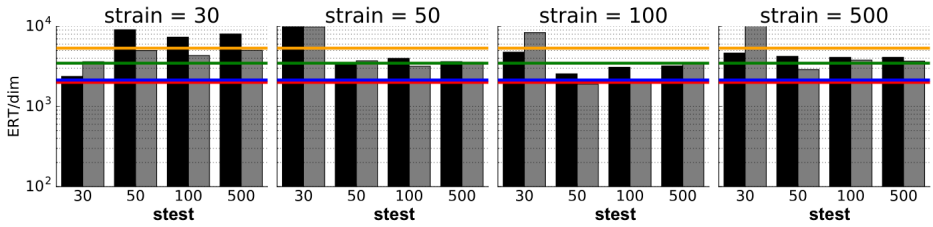


(d) $s_{train} = 100$, $d = 10$

Figure 6.4: Comparison of ERT for different instantiations of the EPM and different values of s_{train} and s_{test}. Six parameter setting methods are compared: $\mathbf{EPM}_s$ and $\mathbf{EPM}_g$, $\mathbf{EPM}^{\star}_{2k,s}$ and $\mathbf{EPM}^{\star}_{2k,g}$ (learned with 'exact' features), and the robust $\boldsymbol{\theta}^r$ and the default $\boldsymbol{\theta}^d$.



(a) $d = 5$



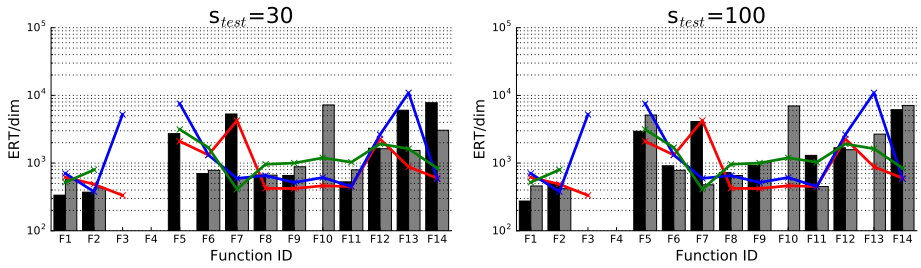
(b) $d = 10$

Figure 6.5: Typical results of ERT averaged on all test functions (F1 to F14) , for dimension d=5 and d=10, and expressed in $\log_{10}$ for different s_{train} and s_{test},in order to compare: $\mathbf{EPM}_s$ and $\mathbf{EPM}_g$, to the robust $\boldsymbol{\theta}^r$ and the default θ^d parameter settings, and the $\mathbf{EPM}^\star_{2k,s}$and $\mathbf{EPM}^\star_{2k,g}$, computed with 'exact' features.

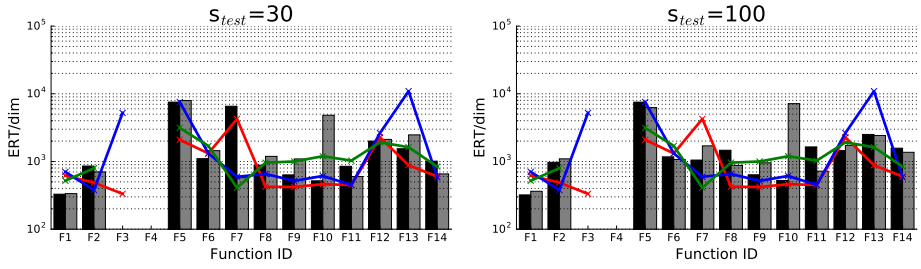
leads to the best results. Despite good performance of the EPM with surrogate-assisted features, we observe that in small dimensions, only small s_{train} and s_{test} have better performances than the robust setting and get similar results than $\text{EPM}^\star_{2k}$. However, regarding the dimension, we observe that, whereas the robust setting is the clear winner when $d < 5$, EPM_g should be strongly preferred when d increases.

Figure 6.7, shows typical results of performance and a comparison between sub-sampled features and surrogate assisted features for EPM_g and EPM_s. The goal here is to directly compare sub-sampled and surrogate-assisted features. We observe that when $d < 5$ the robust parameter setting outperform the PIAC. By contrast when $d \geq 5$, both sub-sampled and surrogate assisted features give the best results: PIAC should strongly be preferred in these contexts.

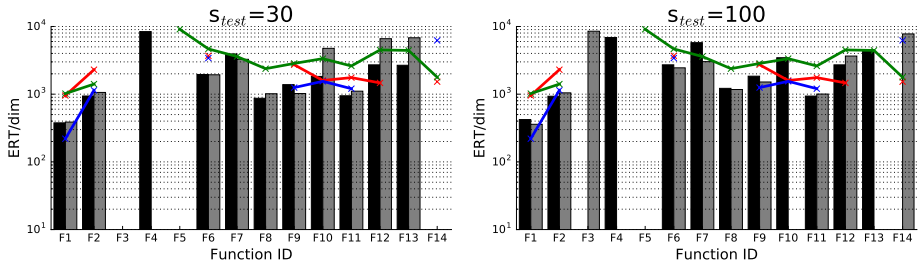
Regarding the features computation methods, it does not seem worthy to use surrogate-assisted features when d is small, in particular for EPM_s. But when



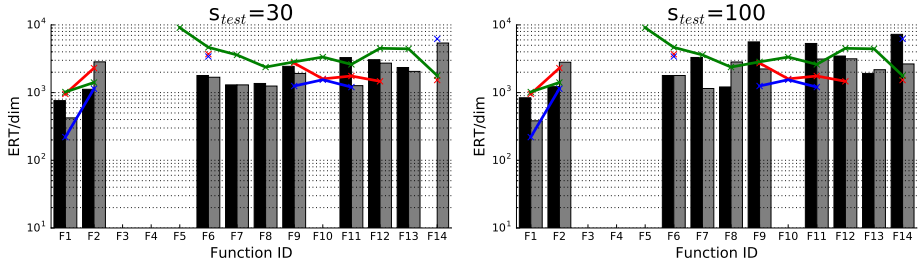
(a) $s_{train} = 30$, $d = 5$



(b) $s_{train} = 100$, $d = 5$



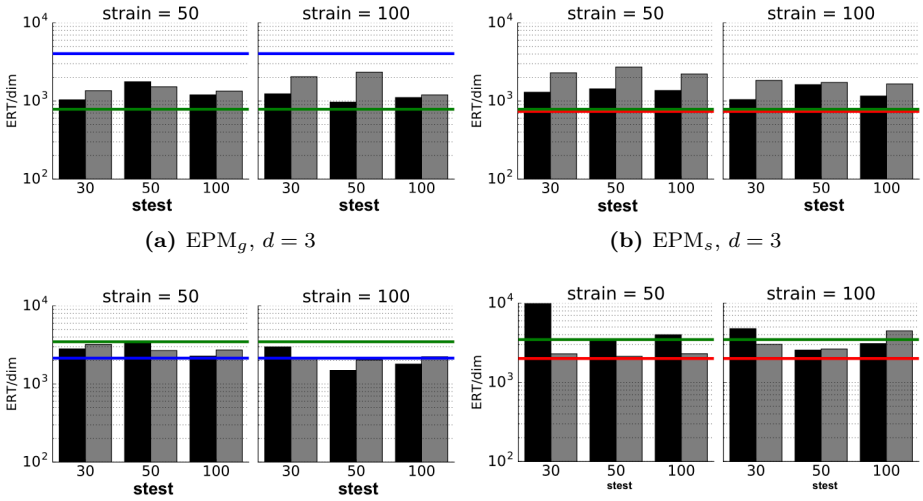
(c) $s_{train} = 30$, $d = 10$



(d) $s_{train} = 30$, $d = 10$

Figure 6.6: Comparison of ERT for different settings of the EPM with surrogate assisted features and different values of s_{train} and s_{test}. Five parameter setting method are compared: $\mathbf{EPM}_s$ and $\mathbf{EPM}_g$ to the results of the $\mathbf{EPM}^{\star}_{2k,\ s}$ and $\mathbf{EPM}^{\star}_{2k,\ g}$ learned with the 'exact' features, and the robust parameter setting $\boldsymbol{\theta}^r$.

d increases, the surrogate-assisted features start to outperform the sub-sampled features. Regarding EPM_g, however, sub-sampled features perform globally better.



(a) EPM_g, $d = 3$ **(b)** EPM_s, $d = 3$

(c) EPM_g, $d = 10$ **(d)** EPM_s, $d = 10$

Figure 6.7: Example of Averaged performance ERT_g across all test bench functions, comparing the sub-sampled EPM (ψ) (in black) and the surrogate-assisted EPM ($\widehat{\psi}$) (in grey) for different settings of EPM: EPM_g and EPM_s, with different size of s_{train} and s_{test}. The results are compared to θ^r and $\mathbf{EPM}^{\star}_{2k}$.

6.4.4 Elite Learning of the EPM

All experiments up to now used the complete set of parameter configurations (8000 per dimension and per problem, see Section 5.2.2.2) as training set for learning the EPM. The goal of the following experiments is to focus the training set for the EPM on the best configurations only, with the rationale that we are only looking for the best configurations for the unknown instances, and do not really need a good approximation of the EPM in regions of low performance parameter configurations.

Three different sets of parameter configurations are used in these experiments: the best parameter configurations, the best 1% or the best 10% of all parameter configurations, respectively denoted by $\Theta_\star$, $\Theta_{1\%}$ and $\Theta_{10\%}$. For each set, the per-

formance of each parameter configuration is computed on all problem instances, hence, it aims at avoiding misleading learning of the EPM with incomplete performance measures. They are compared to the EPM learned with all parameter configurations Θ_{all} (as in all previous results). While the set of parameter configurations is investigated, the sub sampled and surrogate assisted features will be compared only to the results of the generalized EPM (EPM_g).

Figure 6.8 shows typical performances aggregated over all test functions, for different sample sizes to compute features, respectively with sub-sampled features (ψ) and surrogate-assisted features($\widehat{\psi}$). They are compared to the performance of the EPM learned with 'exact' values of features, and the robust parameter setting.

Globally, we observe that EPMs with a smaller set of parameter settings are worse than the robust setting in most situations. Although, they approach the performance of EPM with Θ_{all} when d increases, they hardly reach the same level of performance. Regarding, $\text{EPM}^{\star}_{2k}$, we observe that a smaller set of parameter settings have the same behavior of EPM with Θ_{all}: they are only better than $\text{EPM}^{\star}_{2k}$ when $d < 10$.

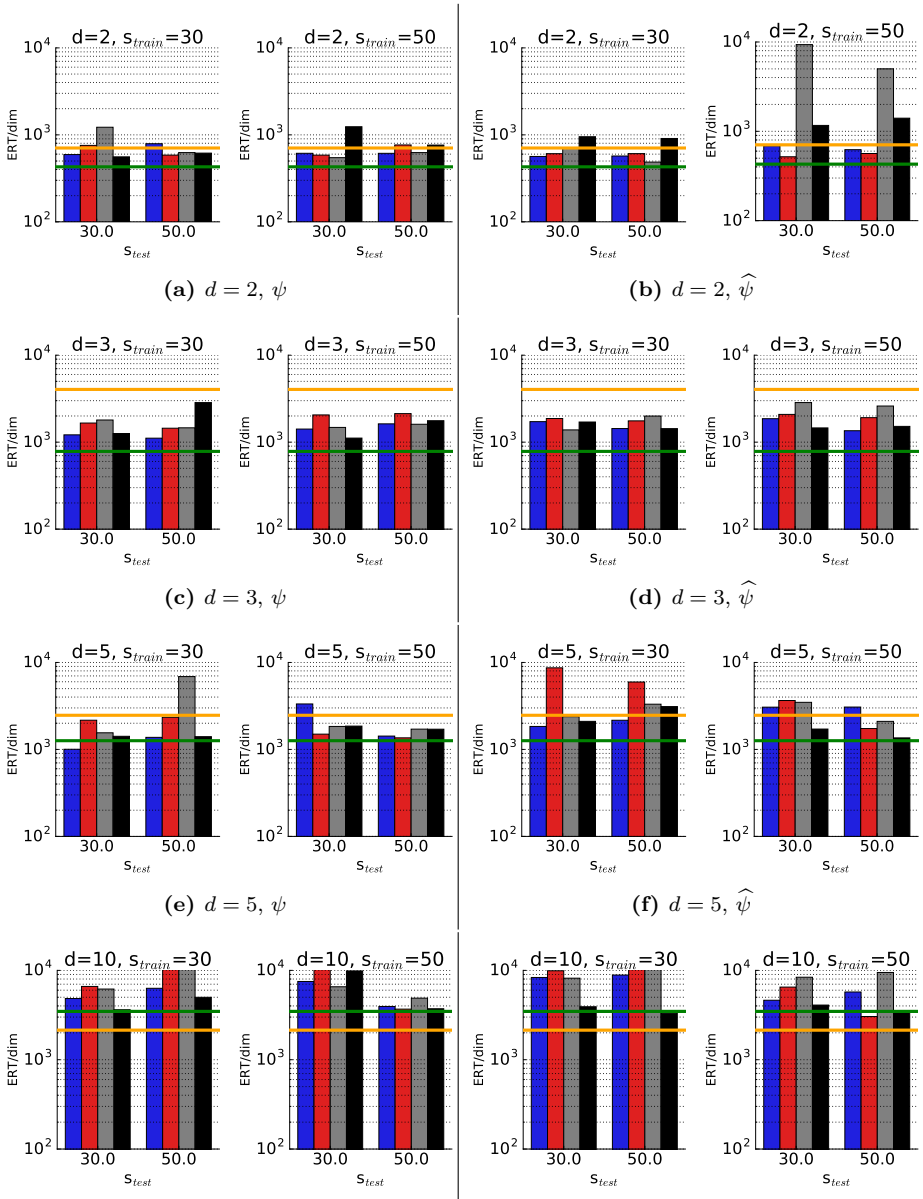
Despite heterogeneous performances over the different samples sizes and feature computation methods, $\Theta_{\star}$ tends to outperform all other subsets of parameter settings, in particular when $s_{train} = s_{test} = 30 \times d$ though rarely outperforming Θ_{all}. Regarding $\Theta_{1\%}$ and $\Theta_{10\%}$, the results are unclear, the performance varies greatly when surrogate-assisted features are used.

Regarding the computation of features, the results are similar to Section 6.4.2 and Section 6.4.3, such that the best performance is observed when $s_{train} = s_{test}$. Regarding the computation of the features, the situation is also unclear: only $\Theta_{\star}$ with sub-sampled features seems beneficial, approaching Θ_{all}.

6.4.5 Choice of the learning method

The following figures will only display typical results of the performance aggregated per dimensions, as they are representative of the results in almost all other contexts.

Figure 6.9 shows the results of EPM with the different ranking methods and compares them to those obtained with Random Forests. They are also compared to the robust parameter setting, the EPM with all Θ_{all} (see Section 6.4.2 and Section 6.4.3), and $\text{EPM}^{\star}_{2k}$. The learning methods are empirically studied by varying the sample size of both sub-sampled features and surrogate-assisted features, but also for different sets of parameter configurations: $\Theta_{1\%}$ and $\Theta_{\star}$.



(a) $d = 2$, ψ **(b)** $d = 2$, $\widehat{\psi}$

(c) $d = 3$, ψ **(d)** $d = 3$, $\widehat{\psi}$

(e) $d = 5$, ψ **(f)** $d = 5$, $\widehat{\psi}$

(g) $d = 10$, ψ **(h)** $d = 10$, $\widehat{\psi}$

Figure 6.8: Excerpt of performances (expressed in ERT) aggregated over all test functions, EPMs with several sets of parameter configurations: $\boldsymbol{\Theta_\star}$, $\boldsymbol{\Theta_{1\%}}$ and $\Theta_{10\%}$ are compared to the EPM with all parameter configurations ($\boldsymbol{\Theta_{all}}$) and the **robust** parameter setting (the – line) and $\mathbf{EPM}^\star_{2k}$ (the – line) .

As could be expected, we observe that RankSVM is the clear winner when subsets of parameter settings are used: in particular when $\Theta_\star$ is used, SVMs outperform all methods, including the robust parameter setting and EPM using Random Forests and Θ_{all}. On the other hand, the situation is different for Logistic Ordinal Regression and ListNet, which are the worst methods for all settings.

Regarding the baseline comparison, RankSVM and Random Forests approach the performance of $\text{EPM}^\star_{2k}$, and the robust parameter setting. But this remains limited to sub-sampled features: using surrogate-assisted features does not seem beneficial, in particular when d increases.

Despite rather poor performances in the cross-validation procedure, ranking methods remain promising methods that deserve at least to be investigated and validated on a new test bench, such as all other aspects of the learning of an empirical performance model. This is the goal of the next Section.

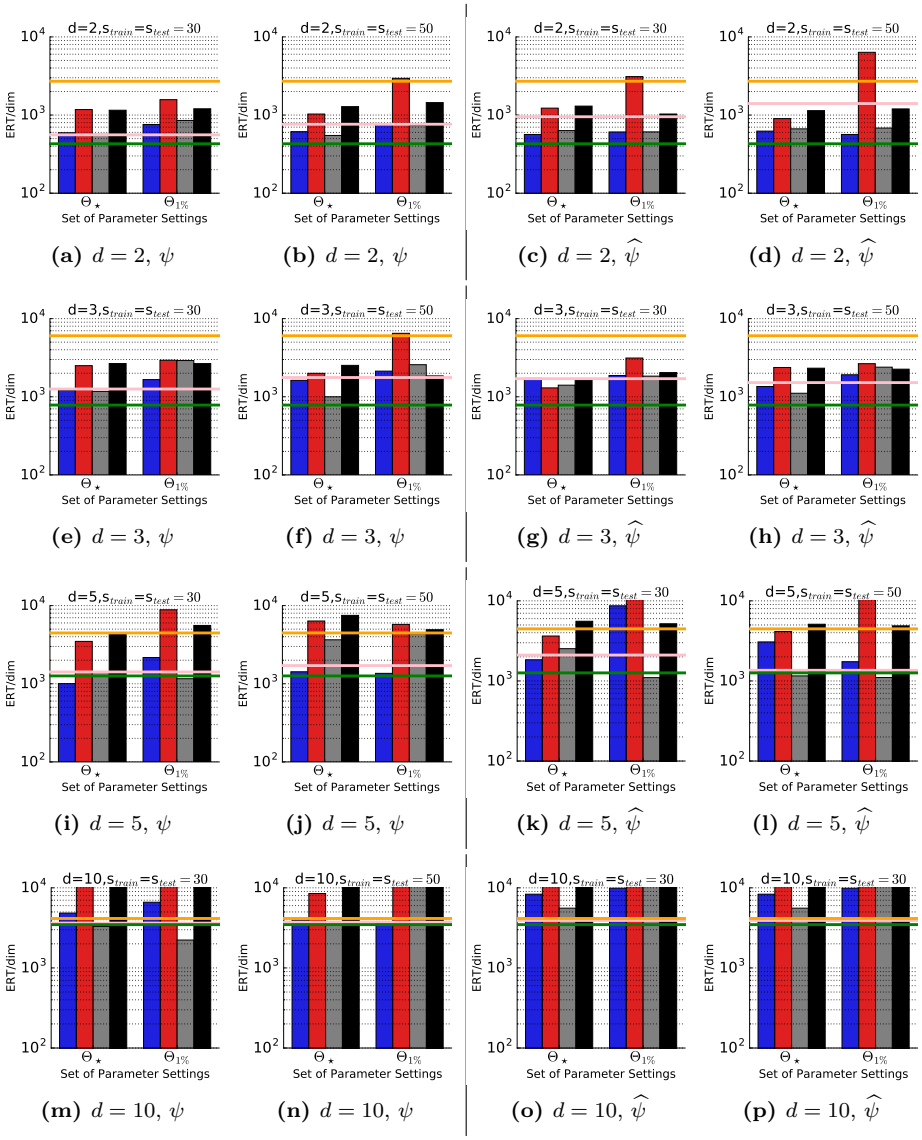
6.4.6 Validation on ExtBench

Previous experiments have empirically demonstrated some promising results for the performance of PIAC for continuous black box problems. However, these experiments are based on a cross-validation procedure on the widely used BBOB benchmark, hence there is always a risk or overfiting this testbench. Therefore, we propose in this Section to empirically assess the results of PIAC trained on BBOB (as presented in all previous Sections of this Chapter) on the new test bench introduced in Section 5.2.

Here, the overall budget of evaluation function calls for the optimization is limited to $10^4 \times d$, and the EPM is learned *a priori*, once for all, using the 14 test functions of BBOB as training set, with dimensions $d \in \{2, 3, 5, 10\}$ (see Section 6.2.2). In addition, the sample size in the training set is $s_{train} = 50 \times d$ such that before the optimization process the features are computed from a sample size of same size for the prediction of the most appropriate parameter settings.

Figure 6.10, shows the EDCF (see Section 3.1.1), aggregated per dimension, allowing us to compare the performance of PIAC with respect to the set of parameter configurations ($\Theta_\star$ or $\Theta_{1\%}$) with surrogate-assisted features $\widehat{\psi}$ and sub-sampled features ψ. Figure 6.10 shows the results on dimensions that are not used in the training set, in order to investigate the generalization capacity of the proposed approach.

Globally, DE with tuned parameters clearly outperforms DE with default parameters θ^d. Surprisingly, surrogate-assisted features do not improve the per-



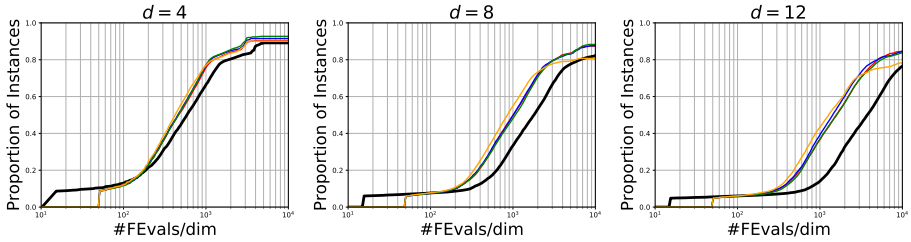
(a) $d = 2$, ψ **(b)** $d = 2$, ψ **(c)** $d = 2$, $\widehat{\psi}$ **(d)** $d = 2$, $\widehat{\psi}$

(e) $d = 3$, ψ **(f)** $d = 3$, ψ **(g)** $d = 3$, $\widehat{\psi}$ **(h)** $d = 3$, $\widehat{\psi}$

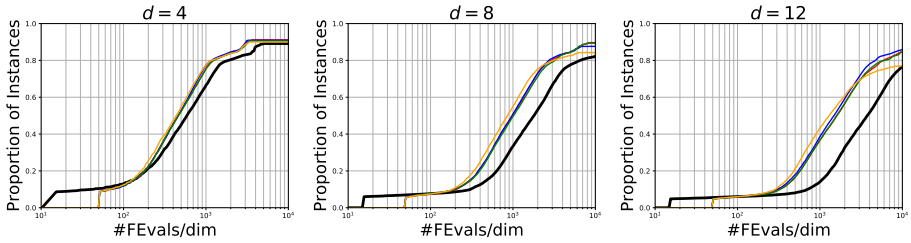
(i) $d = 5$, ψ **(j)** $d = 5$, ψ **(k)** $d = 5$, $\widehat{\psi}$ **(l)** $d = 5$, $\widehat{\psi}$

(m) $d = 10$, ψ **(n)** $d = 10$, ψ **(o)** $d = 10$, $\widehat{\psi}$ **(p)** $d = 10$, $\widehat{\psi}$

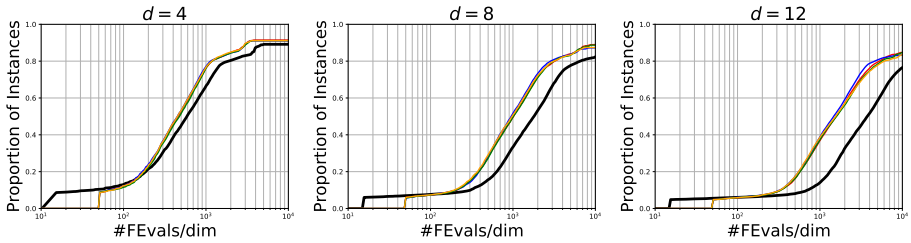
Figure 6.9: Excerpt of performances (expressed in ERT) aggregated over all test functions, EPMs with different learning methods: Random Forest as a baseline comparison, **Logistic Ordinal Regression**, **RankSVM** and **ListNet**. They are compared to the EPM with all parameter settings (the – line, see Section 6.4.2 and Section6.4.3),the robust parameter setting (in green) and with $\text{EPM}^{\star}_{2k}$ (in yellow). The set of features are also compared.



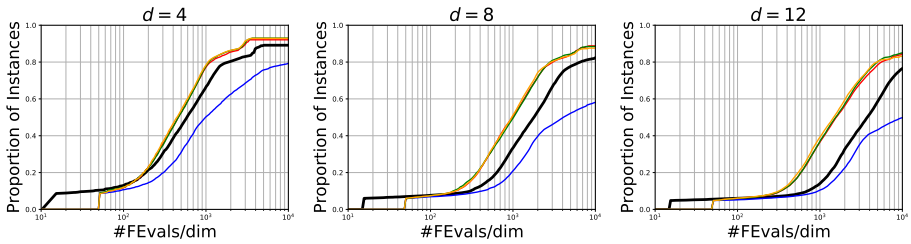
(a) EPM with $\Theta_\star, \psi$



(b) EPM with $\Theta_\star, \widehat{\psi}$



(c) EPM with $\Theta_{1\%}, \psi$



(d) EPM with $\Theta_{1\%}, \widehat{\psi}$

Figure 6.10: Excerpt of ECDF for each dimension, comparing ListNet **LN**, RankSVM **RS**, Logistic Ordinal Regression **LOR**, Random Forests **RF** to $\boldsymbol{\theta^d}$. Here sample size for features are $s_{train} = s_{test} = 50 \times d$.

formances, and even deteriorate them, when using the set of parameter configurations $\Theta_{1\%}$.

Comparing the different learning methods, Random Forest is slightly better than other methods and with homogeneous results on all experimental conditions. Except RankSVM, other learning methods have performances near those of Random Forests. However, the improvement in terms of proportion of solved instances is still small ($\leq$ 10%), but the notable improvement is observed on the convergence speed: DE with PIAC has a faster convergence speed than the default parameter setting of DE. This convergence becomes more visible when d increases.

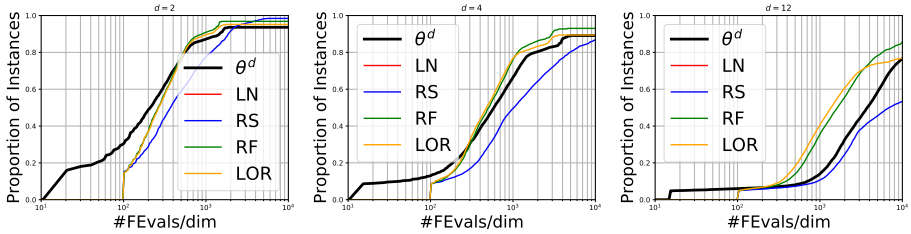
For comparison purposes regarding the sample size, Figure 6.11 shows the ECDF for aggregated dimensions, when the sample size is $100 \times d$. The performance is similar to the experiments with the sample size is $50 \times d$ for the Random Forest, Logistic Ordinal Regression, and ListNet. But for RankSVM, the results are unclear, when the sample size to compute features is increased, using $\Theta_\star$ provides the worst performance, while using $\Theta_{1\%}$.

When the sample size is $100 \times d$, DE with EPM is still better than the default parameter setting, except for $d \leq 5$, where the situation is not clear: the feature computation takes a large part of the budget, but thanks to a faster convergence, DE with EPM solves an higher proportion of problem instances. Therefore, in order to maximize the performance of Differential Evolution it is preferred to use PIAC with an empirical performance model learned with sub-features computed from a sample of size $50 \times d$, using only the set of parameter configurations $\Theta_\star$, as no visible improvement are observed when the sample is larger or when surrogate-assisted features are used.

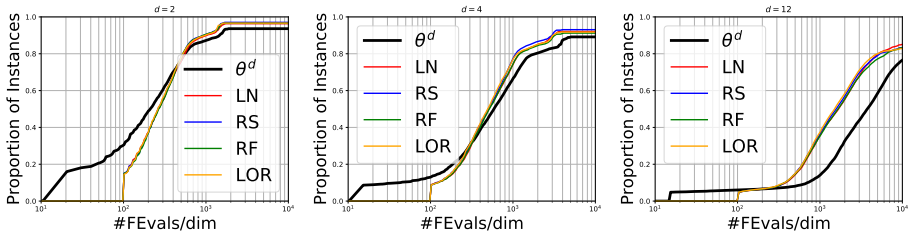
6.5 Discussion

First we investigated if an empirical performance model (EPM) can be learned for continuous black box problems, by using features introduced in Section 5.1, and with a sufficiently large budget of samples to have accurate features. We empirically demonstrated that an EPM can be learned and be used to predict a parameter setting that outperforms the default or a robust parameter setting. Hence, we observed that a good accuracy over the predicted performance map is not required to reach the ultimate goal of PIAC. Hence, the only important property of the EPM is to be able to robustly identify good-performing regions of the parameter space.

Therefore, the next experiments empirically compute the performance of EPM



(a) EPM with $\Theta_\star, \psi$



(b) EPM with $\Theta_{1\%}, \psi$

Figure 6.11: Excerpt of ECDF for each dimension, comparing ListNet **LN**, RankSVM **RS**, Logistic Ordinal Regression **LOR**, Random Forest **RF** to $\boldsymbol{\theta^d}$. Here sample size for features are $s_{train} = s_{test} = 100 \times d$.

with only sub-sampled features. We empirically demonstrated that smaller sample sets ($\leq 100 \times d$) can maintain the efficiency of the EPM.. In accordance to the results in Section 5.1, we empirically demonstrated that it was preferable to use similar sample sizes to compute features both in the training set and in the test set. Hence, if practitioners have a limited budget, like $50 \times d$, it is preferred to learn an EPM with features computed with the same sample size.

In Section 5.1, we introduced surrogate-assisted features, and empirically demonstrated that they maintain the accuracy and efficiency of features. In that direction, we empirically studied them w.r.t. sub-sampled features. Despite promising results for some classification tasks (see Section 5.1), PIAC with surrogate-assisted features has shown limited performances: they slightly outperform sub-sampled features on marginal situations.

As the training set required to learn the EPM is also composed of a set of parameter configurations, this was the natural direction for further experiments.

Initial experiments in Section 6.4.2 and Section 6.4.3 used a large set of parameter configurations, composed of more than 8000 parameter configurations defined by the discretization of the parameter space of DE (as defined in Section 6.2.1). We proposed to use subsets based on most promising parameter configurations, such that for each problem and dimension, we selected three sets of parameter configurations composed alternatively of only the best parameter setting, 1% and 10% of the parameter settings. Promising results are observed in particular when the set of parameter configurations uses only the best parameter setting on each problem of the training set. Nevertheless, these results show that there is still a room of improvement and that a smaller set of parameter configurations can be used, which is particularly necessary when the learning method is computationally costly, so that we can reduce the overall size of the training dataset.

Next, we proposed to investigate other learning methods that rely on the ranking of parameter configurations in lieu of their respective performances. We used three methods: RankSVM, ListNet and a Logistic Ordinal Regression that belong to three groups of ranking methods. We empirically demonstrated that these methods can be used to learn the EPM, such that in the case of RankSVM it can outperform the Random Forest Regression, or the robust setting.

All these experiments were based on a cross-validation procedure. We empirically demonstrated that PIAC can be used in some real-world conditions, like with limited budgets. Therefore, for assessment purposes we proposed to use a new test bench. We empirically demonstrated that PIAC clearly outperforms the default parameter setting, also when the dimension of problems was not considered into the training dataset to learn the EPM. Hence, we empirically demonstrated that the method can generalize a parameter tuning to different dimensions of similar order. However, in order to obtain the best performance, it is recommended to use very small samples set like $50 \times d$, otherwise limited or no improvement might be observed, because of the cost of the features. Despite promising results for surrogate-assisted features, the validation procedure assesses that it is not worthy in most cases, and can even deteriorate the performance of the PIAC, in particular for an EPM based on RankSVM.

6.6 Conclusion and Perspectives on DE Test Case

In this Chapter, we empirically studied the PIAC method for continuous domains following the experimental protocol defined in Section 5.2 and using Differential Evolution as a test case. We empirically demonstrated that an empirical performance model can be learned based on problem features and a set of parameter configurations, and can be used on in some close-to-real-world experimental con-

ditions, i.e., where problems are unknown and might be different from those used for training, and when the budget of function evaluations is limited.

This empirical study, composed of several experiments, opens new research opportunities for the Per Instance Algorithm Configuration, and its use for real-world problems, and for other algorithms. Here, the empirical study was limited to Differential Evolution [SP97], since the algorithm is simple. Furthermore, we used the original version of DE, in which the parameters are fixed at initialization time – thus avoiding possible biases due to some adaptive mechanism. However, because the performance of Differential Evolution is in any case rather poor on several types of problems, like multimodal problems or high dimensional problems, the study was limited to a subset of the BBOB test bench. Therefore, it is mandatory now to investigate PIAC on other optimizers that can be used in more complex situations – using the results of this Chapter as guidelines.

Regarding the experimental settings, we observe that the choice of the learning model has an important role in the efficiency of PIAC, such that Random Forest is the most appropriate method to learn the empirical performance model. Regarding the sample size, we observe that a low budget to compute features is sufficient to find better parameter setting, hence a sample of size $50 \times d$ is recommended. Finally, about the set of parameter setting, although it is recommended to have a discretization of the parameter space Θ_{all}, but using the set of best parameter setting $\Theta_{\star}$ significantly reduce the size of the training set and approaches the results of Θ_{all}.

CHAPTER 7

Tuning CMA-ES in a Limited Budget Context

This Chapter is an empirical study of the Per Instance Algorithm Configuration using CMA-ES as the target algorithm; the goal is to tune three of CMA-ES hyper-parameters, with special care given to the computational cost of the features and the overall cost of the optimization process. Therefore, we propose to investigate PIAC when only a limited budget is available, in order to cope with black box optimization competitions and real-world conditions.

Section 7.2 briefly describe the CMA-ES case study, and the test benches that will be used for assessing the performance of PIAC. Next, Section 7.3 introduces the experimental protocol and settings: from the choice of the features and the different methods used to compute them. Section 7.4 describes the results of the experiments that are discussed in Section 7.5. A partial conclusion and some perspectives are given in Section 7.6.

7.1 Motivation

The Previous Chapter empirically studied PIAC for Differential Evolution. DE was chosen for its simplicity, and its known sensitivity to parameter setting. In particular, no online parameter control mechanism was used, that could have hidden the effects of the offline mechanism under test. For these reasons, DE was an ideal optimizer for PIAC for assessing the efficiency and finding the limits of the proposed parameter tuning.

But, ultimately our goal is to use PIAC for solving real-world problems, which requires a more efficient optimizer than vanilla DE. Hence it seemed natural to investigate PIAC on CMA-ES – considered today as the state-of-the-art of derivative free algorithms for the numerical black box optimization in small dimensions.

Muñoz, Kirley, and Halgamuge [MKH12] and Bossek et al. [Bos+15] already investigated PIAC for the Algorithm Configuration of CMA-ES. But these works did not consider several key aspects of PIAC, relevant to the choice of the set of possible parameter configurations or the computation of features: Bossek et al. [Bos+15] propose to use the dimension of the problem as the only feature, whereas it is clear that the dimension can not be used alone to amount properties of a problem or its difficulties (see Chapter 4); Muñoz, Kirley, and Halgamuge [MKH12] is using a larger set of features, but these features are computed from a very large sample set, requiring a very large number of function evaluations ($\approx 10^3 \times d$), something that is not of practical use when dealing with expensive real-world objective functions; finally, both works only consider the BBOB testbench, with the risk of overfitting this widely used test set.

In this Chapter, we propose to empirically investigate the use of PIAC for CMA-ES, within real-world conditions, i.e. limited budget, and using, as in Chapter 6 the ExtBench testbench that contains functions that are not in the BBOB test set (see Section 5.2.4.2).

7.2 The Case Study

7.2.1 CMA-ES Parameters

The Covariance Matrix Adaptation-Evolution Strategy (CMA-ES) has been described in detail in Section 2.3. This Section will briefly recall its main features, and introduces the parameters that will be the object of PIAC here.

CMA-ES [HO01b], and some of its variants (e.g., the BI-POP-CMA-ES [Han09b]) is considered today as one of the state-of-the-art optimizer for continuous black box optimization, in particular considering the results of the successive BBOB workshops, held during the GECCO conferences [Han+09a; Han+10; AHS12].

CMA-ES evolves a multi-variate Gaussian distribution, adapting online its covariance matrix and some scaling parameter called the step-size. It is often considered to be quasi-parameter-free, thanks to its invariance properties [Han+11a], which allowed to determine default values for its main visible parameters (except the population size) on a small set of functions. Due to the invariance properties, the quality of this default setting extends to the whole equivalence classes of the

functions in this small function set.

However, as discussed in Chapter 3 (see in particular Section 3.2.3), there is still room for improvement, in particular in specific situations (e.g., expensive functions [SE10b; And+15]). Indeed, several works [LS13b; LS13a; LLS12; Los+14] already empirically investigated the search of new parameter settings for CMA-ES in some specific experimental conditions.

In Section 3.2.4, we briefly demonstrated that both a *specific* parameter setting for one problem instance, and a *generalist* –aka *robust*– parameter setting from a set of problem instances, can be found for CMA-ES. But we observed that it remains difficult to generalize such parameters settings. PIAC promises to overcome such limitations, by adapting the parameter setting to the problem instance at hand.

Along a similar direction, Loshchilov et al. [Los+14] proposed an on-line adaptation of three parameters of the adaptation mechanism of CMA-ES, the learning rates c_1, c_μ and c_c, and empirically demonstrated to outperform the static version of CMA-ES on some of the test functions of the BBOB benchmark (Section 3.1.1). This was one of the motivations here to use PIAC for tuning these parameters for the BI-POP-CMA-ES.

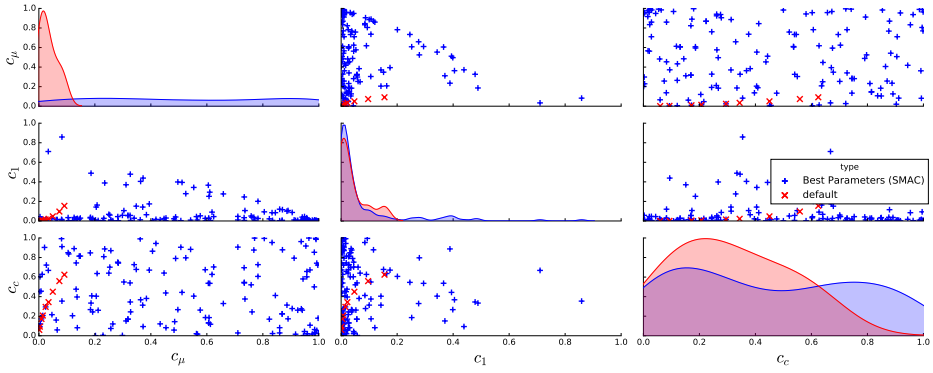


Figure 7.1: Comparison of the default (×) and best (+) parameter setting for c_1, c_μ and c_c parameters of CMA-ES, for each function from BBOB and each dimension $d \in \{2, 4, 5, 8, 10, 16, 20, 32, 40, 64\}$. Two axes projections are displayed out-diagonal, while density distributions for the corresponding parameter are shown on the diagonal.

A further hint that these parameters might be good candidates for per in-

stance configuration is given in Figure 7.1. For each function of the BBOB testbench (Section 3.1.1) and each dimension, a single-instance Algorithm Configuration is run for the configuration of c_1, c_μ and c_c, using the SMAC framework[1][HHL11b] optimizing the accuracy of the solution found within a fixed (small) budget of $10^3 \times d$ function evaluations. All resulting optimal parameters are displayed in Figure 7.1 (blue '+') together with the default values (red 'x'), demonstrating that the specialist tuning can be quite different from the overall robust setting.

7.2.2 The testbenches

According to the general experimental protocol introduced in Section 5.2, two sets of test functions are used in this work: the BBOB test bench is used as a training set for learning the EPM, while ExtBench, an original set of 21 analytically defined functions gathered from several sources of optimization literature, is used as a test set to validate the approach. For the sake of completeness, both testbenches will rapidly be introduced here, together with the details of their use here.

7.2.2.1 BBOB for learning an EPM

The Black Box Optimization Benchmark (BBOB) [Han+10] is made of 24 functions analytically defined on $[-5, 5]^d$, with known global optima. They have been manually classified in five classes of problems with respect to their global properties (e.g. separability, multi-modality, …) and have been used in many of the works cited in this PhD thesis as a test bench. The EPM is learned from a training set made of these 24 test functions in different dimensions d, namely here $d \in \{2, 4, 5, 8, 10, 16, 20, 32, 40, 64\}$.

As advocated in the original framework, each of the 15 independent runs that are run on each function actually uses a variant of the original function, obtained by a random translation of the optimum and, for the non-separable functions, a random rotation of the coordinate system.

7.2.2.2 The Validation Testbench

The validation testbench ExtBench, described in detail in Section 5.2.4.2, is made of functions that are completely independent from the BBOB testbench, in contrast with most previous works on parameter tuning for CMA-ES [MKH12;

[1] http://www.cs.ubc.ca/labs/beta/Projects/SMAC/

Bos+15]: in these works, a cross-validation over the BBOB test functions was used to assess the performance of the proposed approaches.

We propose with ExtBench, introduced in Section 5.2.4.2, to use 21 analytically defined test functions that are not in BBOB. In agreement with the BBOB methodology, 15 independent runs for each function are run on variants of the original function, as described above. Furthermore, the validation runs are performed for dimensions $d \in \{2, 4, 8, 10, 16, 20, 32, 40, 50, 66, 100\}$, i.e. additional dimensions are used, that had not been used to learn the EPM during the training phase. The goal is to study the generalization ability of PIAC to new problems and new dimensions.

7.3 Experimental Setting

7.3.1 The Features

In accordance with the results in Chapter 6 and Chapter 5, only *cheap* features are used in the following experiments, resulting in a feature space $\mathbf{\Psi}$ of size 51.

In Chapter 6, we observed that a low budget of evaluation function calls can be used to nevertheless provide efficient performance of PIAC. Furthermore, as said above, the goal is to experiment PIAC in real-world-like situations, i.e., with expensive objective functions. Therefore, the following empirical study only focuses on low budget contexts: the sample size n used to compute the features of all instances is set to $k \times d$ for some $k \in \mathbb{N}$.

Several values of k are investigated: $k \in \{10, 30, 50\}$. For each k, two different approaches to the $k \times d$ budget for feature computations are also investigated: the features can be computed directly from $\mathcal{S}$ (denoted $\boldsymbol{\psi}_k$), or, following Section 5.1.5, a surrogate model $\widehat{f}$ of f can be built from $\mathcal{S}$ and used as a proxy to compute "surrogate-assisted features", denoted by $\widehat{\boldsymbol{\psi}}_{\boldsymbol{k}}$, using as many samples from the surrogate model as necessary to compute the features. In the latter case, as in Section 6.3, a Random Forest regressor is used to learn $\widehat{f}$ (using the Scikit-Learn library implementation with 20 trees and maximal a depth of 500).

7.3.2 The Empirical Performance Model

In accordance with Chapter 3, the empirical performance model is learned with a Random Forest regressor with the default meta-parameters from scikit-learn, i.e., 10 trees and a maximal depth of 200.

7.3.3 The Experimental Protocol

The experimental protocol follows the one described in Section 5.2. First, for each of the 24 test functions from the BBOB testbench, and for each dimension $d \in \{2, 4, 5, 8, 10, 16, 20, 32, 40, 64\}$, 15 independent samples of the objective functions are generated. Their respective features are computed. On the other hand, SMAC is used to compute the best specialist parameter setting of CMA-ES[2]: the performances of these specialist settings are considered as the upper bound of the performance. From this dataset, the EPM is learned once and forall.

The validation of the EPM is done using the 21 functions from ExtBench and the 11 dimensions $d \in \{2, 4, 8, 10, 16, 20, 32, 40, 50, 66, 100\}$, running 15 independent runs for each function and dimension. Their features $\boldsymbol{\psi}_k$ and $\widehat{\boldsymbol{\psi}}_{\boldsymbol{k}}$ are computed using $k \times d$ samples, uniformly drawn within the given bounds of the function, and their empirical optimal *specialist* parameter settings are computed accordingly (Section 3.3). The versions of CMA-ES using these empirical optimal parameters are denoted by EPM-CMA-ES-$\boldsymbol{\psi}_k$ when sub-sampled features are used and by EPM-CMA-ES-$\widehat{\boldsymbol{\psi}}_{\boldsymbol{k}}$ when surrogate-assisted features are used.

For all optimization runs, including the computation of features, the maximum evaluation budget is set to $10^3 \times d$, and the target precision to be reached is $\Delta_f = 10^{-6}$.

Finally, because the BI-POP-CMA-ES[3] is used in all experiments with the default restart parameters (number of restarts=15, $\sigma_0 = 0.3$), several restarts sometimes take place during a run, and it could be the case that this premature stopping occurred because of a poor parameter setting: we propose to recompute the features using a new sample of size $k \times d$ that is added to the original one. This variant is denoted by CMA-ES-θ_{new}.

7.3.4 Performance measure

The Empirical Cumulative Distribution Functions (ECDF) plots are used in the next Section to present all the results (see Section 3.1.1 for explanations of the plots).

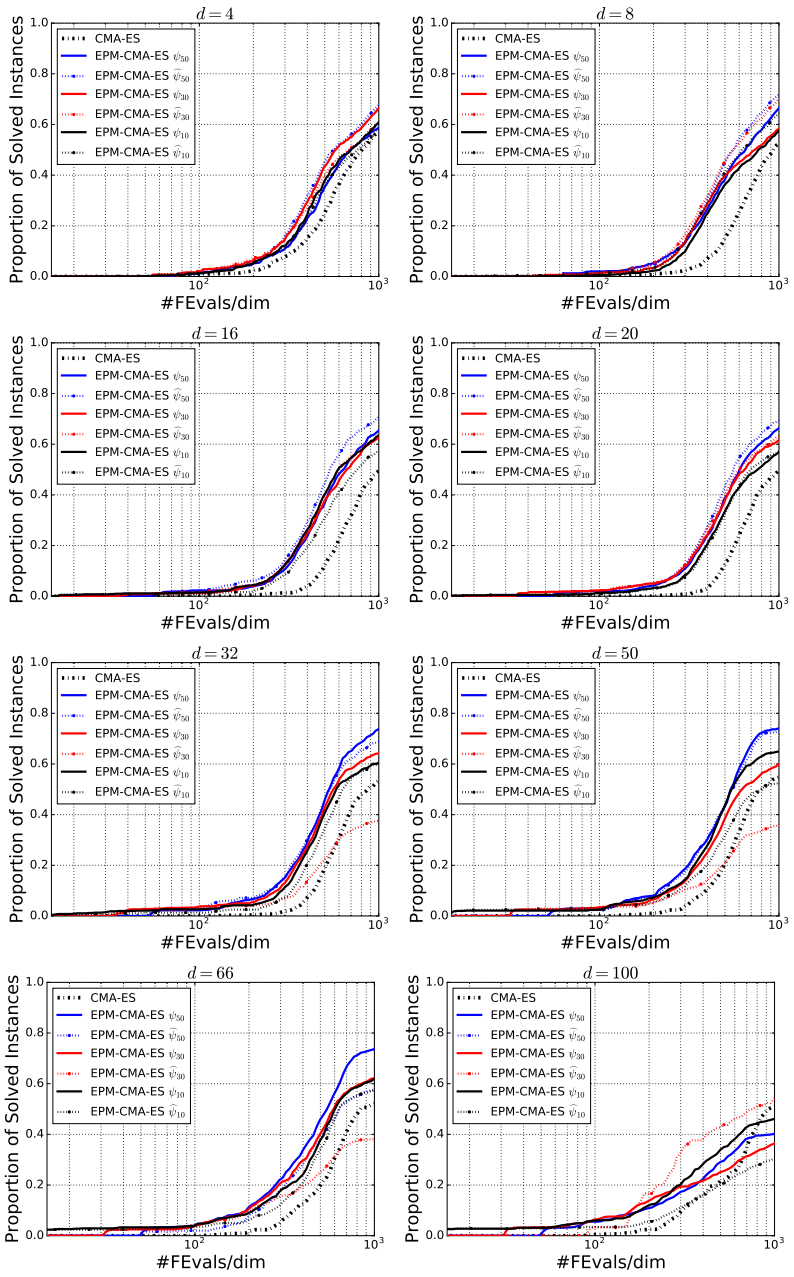


Figure 7.2: ECDF comparing EPM-CMA-ES with $\boldsymbol{\psi_k}$ or $\widehat{\boldsymbol{\psi_k}}$ ($k \in \{10, 30, 50\}$), and default CMA-ES.

7.4 Results

Figure 7.2 compares the ECDF, aggregated per dimension, of several sample sizes k and computation methods $\boldsymbol{\psi}$ and $\widehat{\boldsymbol{\psi}}$ to those of the default CMA-ES.

First, regarding the computation of features $\boldsymbol{\psi}$, we observe a clear preference of a larger sample size $k > 30$. As could be expected, when k decreases, the performance of EPM-CMA-ES decreases accordingly, and the largest value $k = 50$ is the best setting to compute features, but, in accordance with the results in Chapter 6 $k = 50$ is sufficient to find best parameter settings when only a limited budget is available. Similarly, the quality of the empirical performance model tends to greatly improve when k increases. The results are more visible for $d > 8$, and some improvement of almost 20% over default CMA-ES is observed with $\boldsymbol{\psi_{50}}$.

Regarding the use of surrogate assisted features $\widehat{\boldsymbol{\psi}}$, the results are unclear with respect to sample sizes and dimensions. When $k = 50$, the performances of $\widehat{\boldsymbol{\psi_{50}}}$ and $\boldsymbol{\psi_{50}}$ are roughly similar, whereas when d is small, surrogate assisted features tend to slightly outperform the directly computed features. On the other hand, and rather surprisingly, the surrogate assisted features are clearly worse when $d > 8$, in contrast to the results for small k.

Figure 7.3 displays typical ECDF comparing the simple $\boldsymbol{\psi_{50}}$ approach with the $\boldsymbol{\psi_{50}} - \theta_{new}$ restart strategy (and the baseline default CMA-ES). The results are similar for all the dimensions. While we can observe a faster convergence on some dimensions for EPM-CMA-ES without θ_{new}, no strategy significantly and repeatedly outperforms the other.

Figure 7.4 displays some typical results of EPM-CMA-ES and default CMA-ES when run beyond the $10^3 \times d$ budget that was used for training: the domination of EPM-CMA-ES remains clear, even though the surrogate assisted features tend to stay at the same level of performance.

Figure 7.5 and Figure 7.6 display some typical results of EPM-CMA-ES, for some test functions, and dimensions $d = 4$ and $d = 50$ respectively. The domination of EPM-CMA-ES is clear for most of the test functions, except for Needle Eye and Happycat, for which all methods fail. A specific case is that of Rosenbrock Saddle, for which all methods fail if we only consider the limited budget of $10^3 \times d$, whereas EPM-CMA-ES is the clear winner when the budget is extended.

Surprisingly, we observe for some test functions, like Ackley or Wavy, that

[2] SMAC is used with budget of 1000 meta-runs

[3] using the Python version available at URL https://www.lri.fr/~hansen/cmaes_inmatlab.html

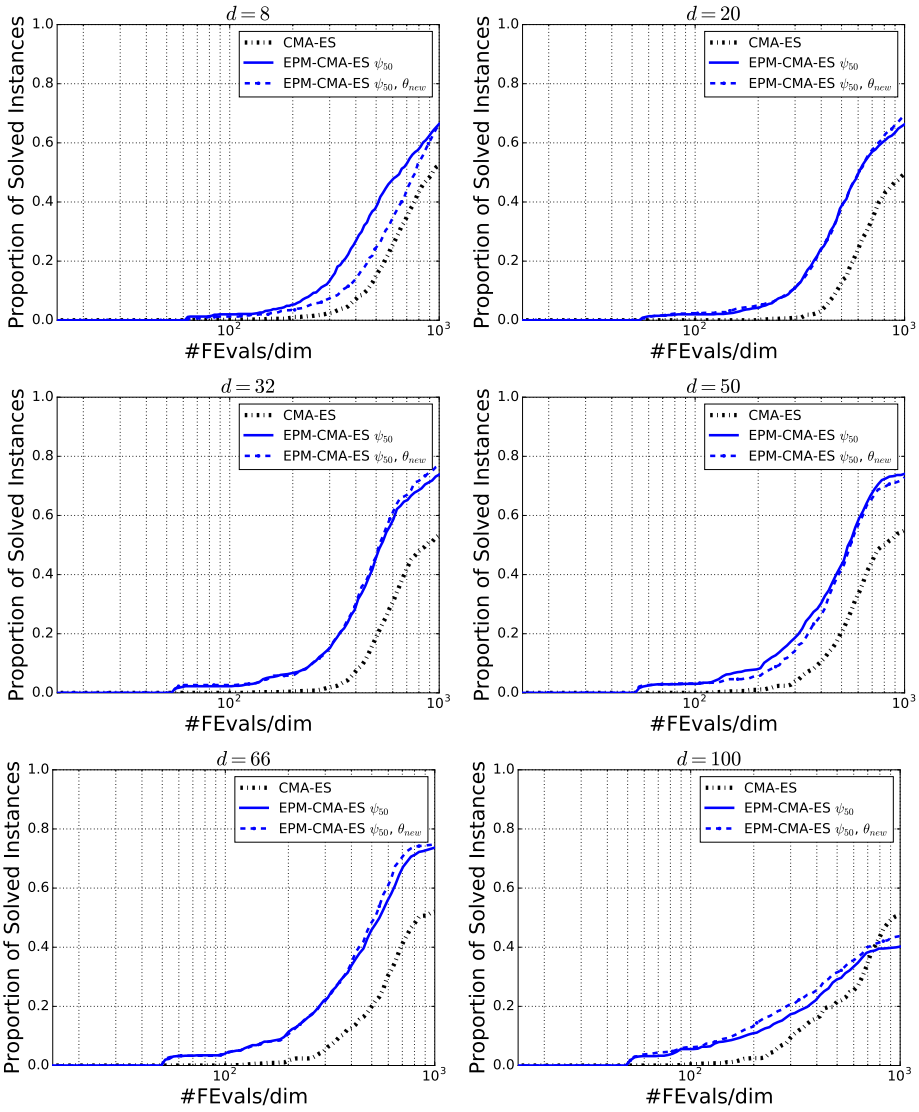


Figure 7.3: ECDF comparing ψ_{50} without and with the alternative restart strategy (θ_{new}), and the default CMA-ES..

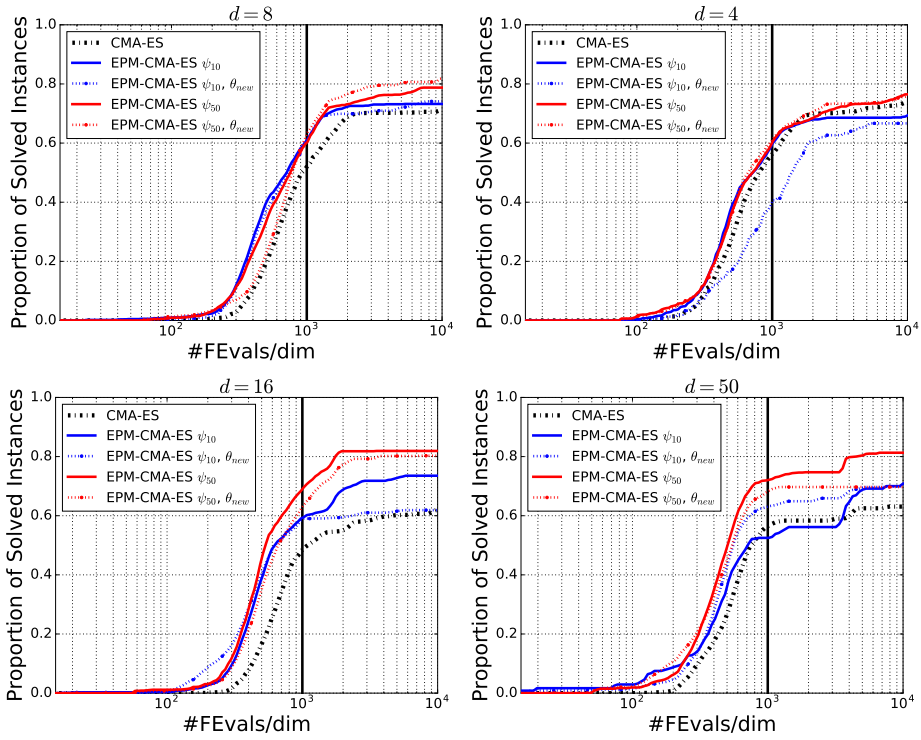


Figure 7.4: Typical Results of ECDF of EPM-CMA-ES compared to CMA-ES beyond the $10^3 \times d$ initial budget limit .

EPM-CMA-ES shows a better performance in higher dimensions, whereas in lower dimensions the default CMA-ES is preferred. However, when d increases, the situation changes, and CMA-ES repeatedly fails, like for Wavy, for which it succeeds in no run at all.

Regarding the restart strategy, in most situations, the new restart strategy does not improve the performance of EPM-CMA-ES, or even worse, prevents from finding better solutions. Only few examples, like Schaffer6, show situations for which the new strategy $-\theta_{new}$ can be beneficial.

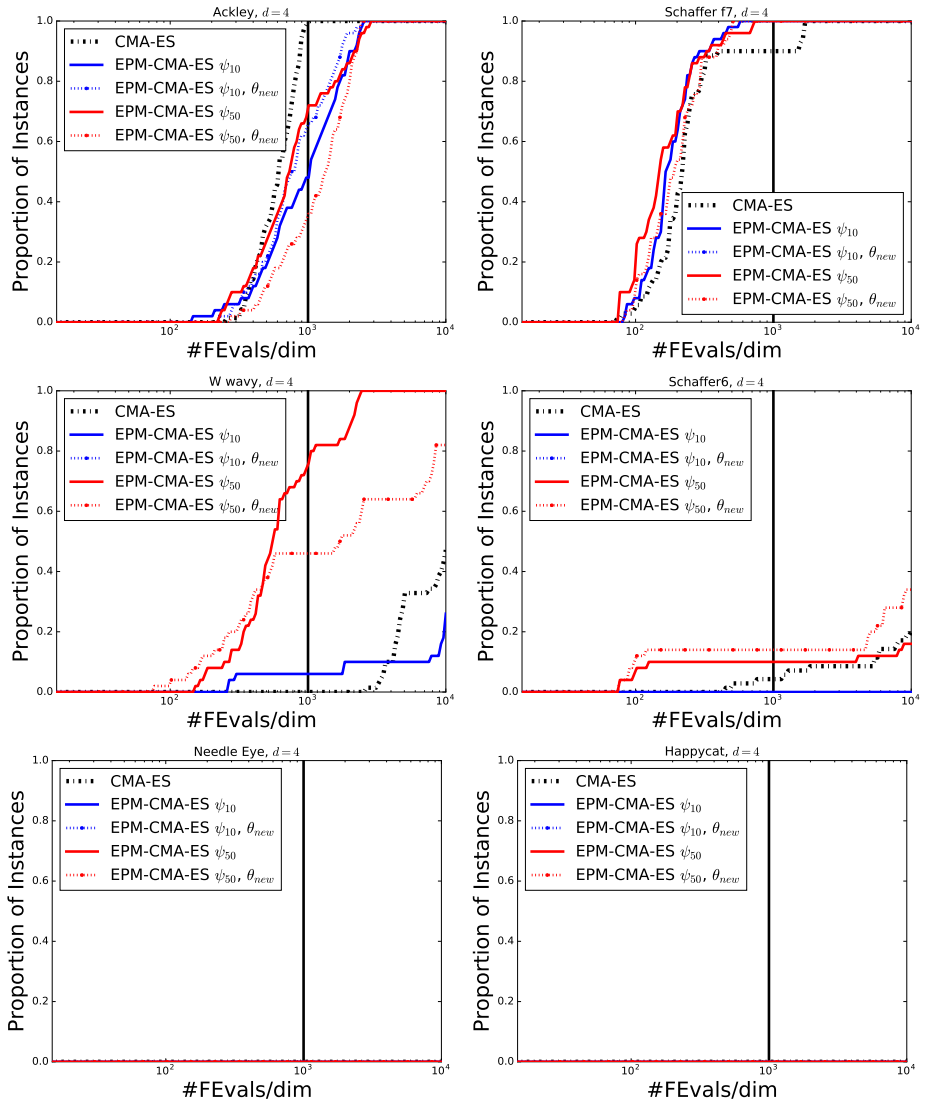


Figure 7.5: Typical Results of ECDF of EPM-CMA-ES, for an excerpt of test functions with $d = 4$, compared to CMA-ES beyond the $10^3 \times d$ initial budget limit .

7.5 Discussion

In accordance with Chapter 6, the results presented here first show that the PIAC approach of learning an EPM from a large sample of results of runs of different parameter settings on different instances can be successful, outperforming the default setting of CMA-ES, an algorithm which is known for its robustness parameter-wise.

We empirically demonstrated in Chapter 6, that the computation of features is the core element of the empirical performance model and its predictive power. We had empirically shown that the accuracy and the efficiency of the features critically depends on their computational costs.

Based on these observations, the present study on CMA-Es focused on low budgets to compute features. From the results, it is clear that reducing the computational cost of the feature indeed reduces the efficiency of the approach (e.g., for $k < 50$), even more so when the dimension of problems increases. For very low sample sizes ($10 \times d$), the improvement over the default CMA-ES becomes insignificant, making the PIAC methodology useless in such context.

Regarding the surrogate-assisted features, as already witnessed in previous experiments with DE, the efficiency of this approach for PIAC is not as clear as for the task of retrieving the BBOB classification (see Section 5.1). On the one hand, surrogate assisted features seem beneficial when the dimension is low ($d < 16$), or when the initial sample size is large enough to correctly learn the surrogate model, but $50 \times d$ might still be a little small, in particular in large dimensions. As a matter of fact, surrogate assisted features are completely misleading when the dimension increases, and in particular with even lower sample sizes (10 or $30 \times d$). The final conclusion at the light of these experiments seems to be that directly computing the features from the initial sample, however small, remains the best option. However, this opens a path for research around the improvement of feature computation from small samples, hence suggesting to explore other methods to approximate an objective function, …or to come up with new features that are less sensitive to the sample size.

Even if EPM-CMA-ES is globally better than CMA-ES, some of the test functions must be distinguished from the others. In particular, when run on the Needle Eye, or the Happycat test functions, all variants fail to come even close to the optimum. These test functions were intentionally included with the hope that the PIAC based tuning would allow the algorithm to more quickly reach the surrounding of the needle optimum, and hence have more iterations to eventually find it, but that did not happen. Indeed, Needle Eye has a plateau shape, on which CMA-ES is known to perform poorly, and BBOB is known to

have a few test functions with plateaus. However, the EPM was not able to learn more successful setting for plateaus, and additional experiments should be made including more examples of test functions with plateaus.

After detailed observations, we proposed and investigated an alternative strategy after a stopping criterion of BIPOP-CMA-ES is met, such that at each restart the features are recomputed and the corresponding parameter setting is applied to CMA-ES. Because of their poor performances, surrogate assisted features have not been tested together with this alternative strategy. However, while few improvements can be observed on some functions and dimensions, the additional cost of re-computing the features tend to hinder the performance too much, as can be observed on $d = 8$ for instance. Also, in such experimental conditions where the maximum number of function evaluations is rather small ($10^3 \times d$), the effect of such an alternative strategy might not be visible. Nevertheless, additional experiments with a larger budget ($10^4 \times d$ as in Figure 7.4), did not exhibit any improvement, with small plateaus representing restart with no improvements.

Finally, we empirically investigated the PIAC methodology such that an empirical performance model is learned from a large set of test functions with various properties and dimensions (up to 64). The experimental setup aimed at empirically investigating EPM-CMA-ES for test functions with dimensions considered in the learning phase or dimensions in the same range, or close to them, e.g. $d = 50$ or $d = 66$. We observed that the empirical performance model was able to generalize to dimensions close to those of the learning set. By contrast, when test functions and dimensions are very different, e.g. $d = 100$ in Figure 7.3, the results of EPM-CMA-ES are worse than those of the default CMA-ES, strongly suggesting that the predicted parameter setting is poor. Such results question the generalization power of EPM-CMA-ES to larger dimensions, hence requiring that the EPM be learned on all dimensions that are likely to be considered during the later decision phase.

Then, when a new problem instance g is processed, the features $\psi(g)$ are computed and the empirical best setting for g is found by solving the auxiliary optimization problem $ArgMin_\theta \widehat{\varphi}(\psi(g), \theta)$. At the moment, for the three parameters involved in our experiments, this is done by a brute grid search. However, one advantage of the PIAC approach over cluster-based methods such as ISAC [Kad+10; AMT12] is that any optimization algorithm could be used on the EPM thus being able to handle any (reasonable) number of distinct parameters, whereas the cluster approach would suffer from a much more intense curse of dimensionality. Another drawback of the cluster-based approach is that it relies on some class-based meta-optimization (see Section 3.3) that will give poor results as the size of the clusters increases, hindering their homogeneity with respect to the parameter setting. On the other hand, the PIAC approach will always take

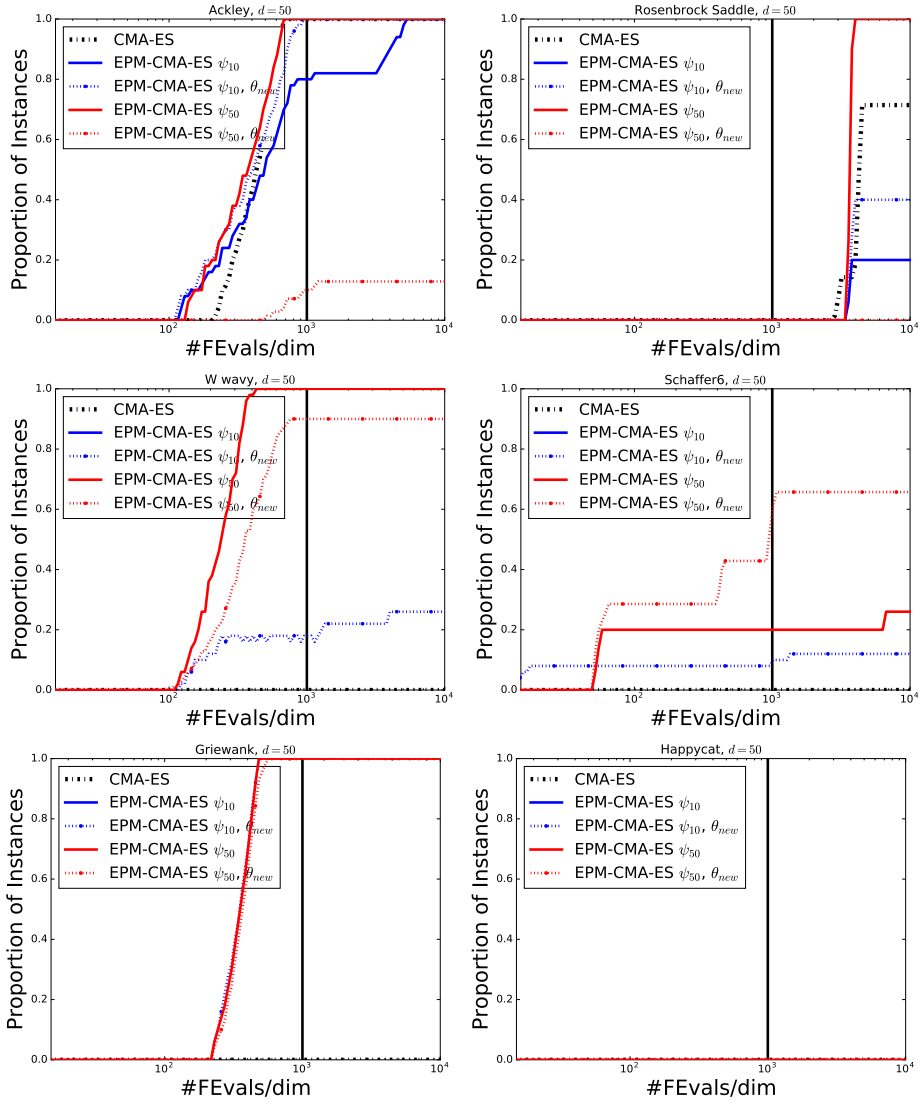


Figure 7.6: Typical Results of ECDF of EPM-CMA-ES, for an excerpt of test functions with $d = 50$, compared to CMA-ES beyond the $10^3 \times d$ initial budget limit .

into account the specific features of the instance at hand.

7.6 Conclusion and Further Works

This Chapter empirically validated the Per Instance Algorithm Configuration methodology on BIPOP-CMA-ES, one of the state-of-the-art optimizer for continuous black box optimization. We empirically investigated the computation and the use of an Empirical Performance Model (EPM) with a small overall budget of $10^3 \times d$, de facto implying some low upper bound on the computation cost of the feature themselves. As the extended investigation done on Differential Evolution, this study relied on the general experimental protocol introduced in Section 5.2: the EPM was trained on the famous BBOB benchmark, but was tested on independent test functions from the optimization literature. As a result, in accordance with Chapter 6, given a same overall budget, the PIAC methodology greatly helped to find a parameter setting that outperforms the default parameter setting, hence EPM-CMA-ES was consistently outperforming the default CMA-ES, when the EPM is computed with sub-sampled features.

We proposed an incremental strategy that recomputes the instance features at each restart of CMA-ES using some additional samples, but it did not show any significant improvement, raising the issue of dynamic feature (re)computation. Additional samples are de facto gathered by the optimization algorithm, and could be used to compute the features more accurately as the fitness landscape changes. Therefore it suggests that a new research direction should focus on the on-line parameter control.

This empirical study confirms that surrogate-assisted features are not beneficial to improve the performance of the EPM, and can even deteriorate its performances. However, it raises new research questions on other methods to compute features from small samples, hence other approaches for learning the surrogate $\widehat{f}$ in the surrogate assisted feature approach should be investigated.

Finally, a promising research path is to extend the use of the Empirical Performance Model to the full Algorithm Selection and Configuration problem, as in [Xu+11; XHL10]: the EPM is used to predict the best algorithm with the best parameter setting for a new and unseen problem.

Part IV

General Conclusion

CHAPTER 8

Toward Online Per Instance Parameter Tuning

8.1 Rationale

The main limitation of the parameter tuning approach –regardless of the method– lies in the concept of the Evolutionary Algorithm concept itself, as an EA is intrinsically dynamic, i.e. the current view of the search domain evolves while the search progresses. Using fixed parameters that do not change during the search process contradicts one's intuition: it is intuitively obvious that different values of parameters might be optimal at different stages of the evolutionary process. As an example, for an Evolution Strategy with a mutation step size parameter (see Section 2.3.1), large values will favor a good exploration of the search space at the beginning of the process, whereas small mutation step sizes will be needed in the late generations to help a fine-tuning of the candidate solutions. Therefore, a fixed parameter values may lead to sub-optimal performances. Eiben et al. [Eib+07] gives an overview of parameter control mechanisms that are the core of well-known algorithms based on Evolution Strategies — like CMA-ES [HO01b]– or adaptive variants of Differential Evolution — like JADE [ZS09] or jDE [Bre+06].

This thesis investigated the PIAC when only low budget to compute features is available, and empirically demonstrated that a parameter setting that outperforms the default one can be found by using instance-specific parameters, gathered from an Empirical Performance Model based on features that are computed from a sample of values of the objective function. However, the computation of these features, even with very small sample set (10 to $50 \times d$ samples), still eats a large part of the total optimization budget, e.g., for CMA-ES with a standard

population size of $\lambda = 4 + \lfloor 3 \ln n \rfloor$.

As discussed in Section 3.3, PIAC relies on features that describes the global problem properties, and might ignore some local properties that are visible and true on a small region of the search space. From the point of view of the optimizer considered as a dynamic process, the properties encountered at different times of the search might differ greatly. A natural approach would hence be to embed PIAC and the learned performance model into a parameter control mechanism that would recompute the features of the objective function based on the recent evaluations performed, and predict the best parameter values along the search.

This Chapter presents Proof-Of-Concept results toward online algorithm configuration based on a dynamic re-computation of features as the optimization proceeds, using the points that are evaluated by the optimizer in the normal curse of its run.

8.2 Differential Evolution Case Study

In Chapter 6, we empirically demonstrated that PIAC can be used to predict parameter settings that outperform both the default parameter setting of DE and a robust parameter setting. However, more recent variants of DE that embed a parameter control mechanism [QHS09; Bre+06; Zha+09; Bre+09; WCZ11; ZS09] have shown results that consistently outperform the original version of DE.

The following empirical study aims at investigating a parameter control mechanism for DE that embeds an empirical performance model to predict values of the three parameters of DE: F, CR and $strategy$, as empirically studied in Chapter 6.

8.2.1 Experimental Settings

The following experiments are based on the general experimental protocol defined in Section 5.2, with a specific focus on the validation with ExtBench.

8.2.1.1 Test Bench

Here again, as described in Chapter 6, two sets of test functions are used: the BBOB test bench [Han+10] is used to learn the empirical performance model once for all, and ExtBench, the set of 21 analytically defined test function described in Section 5.2.4.2, is used as a test set, to validate the results. Each test function will be tested with $d \in \{2, 4, 8, 10\}$. The following experiments are achieved with

a limited evaluation budget $10^4 \times d$, such that the target precision to reach is $\Delta_f = 10^{-6}$.

8.2.1.2 Empirical Performance Model

Following the experimental setting detailed in Section 6.3, the empirical performance model is learned on the first 14 test functions of the BBOB test bench, for the dimensions $d \in \{2, 3, 5, 10\}$. Fifteen independent uniform samples of size $15 \times d$ are generated to compute the features (only *cheap* features are used –see Section 4.4). Also, according to the observation in Chapter 6, we only use subsampled features as surrogate-assisted features have shown limited performance. Finally, the set of optimal parameter settings $\Theta_\star$ is used.

8.2.1.3 Parameter Control Mechanism

The adaptive variant of Differential Evolution that embeds an empirical performance model is denoted by φDE. At each iteration, the current population is used to compute features and predict the parameter setting for the next iteration.

In the following experiment, φDE is compared to four variants of DE:

- The original version of DE [SP97] with the default parameter setting
- The JADE adaptive variant with default parameter values as defined in [Zha+09], a well-known adaptive version that uses an external archive to keep track of the most promising individuals
- jDE (see the description in Section 2.4.5) with the default parameter setting as advocated in [BZM06]
- EPM-DE– a Differential Evolution that uses an EPM at initialization to tune the initial parameter values of DE, such that features are computed from a sample of $50 \times d$, the set of optimal parameter setting $\Theta_\star$ is used and the EPM is learned with a Random Forest regression, as defined in Chapter 6.

8.2.2 Results

Figure 8.1 displays the ECDF of all five algorithms, aggregated per dimensions, for $d \in \{2, 4, 8, 10\}$.

The good news is that φDE outperforms all other DE variants for small budgets: it solves more function up to budgets of $2.10^3\times$ to $4.10^3\times$ evaluations. Only DE is better in this context - and only for dimensions 8 and 10.

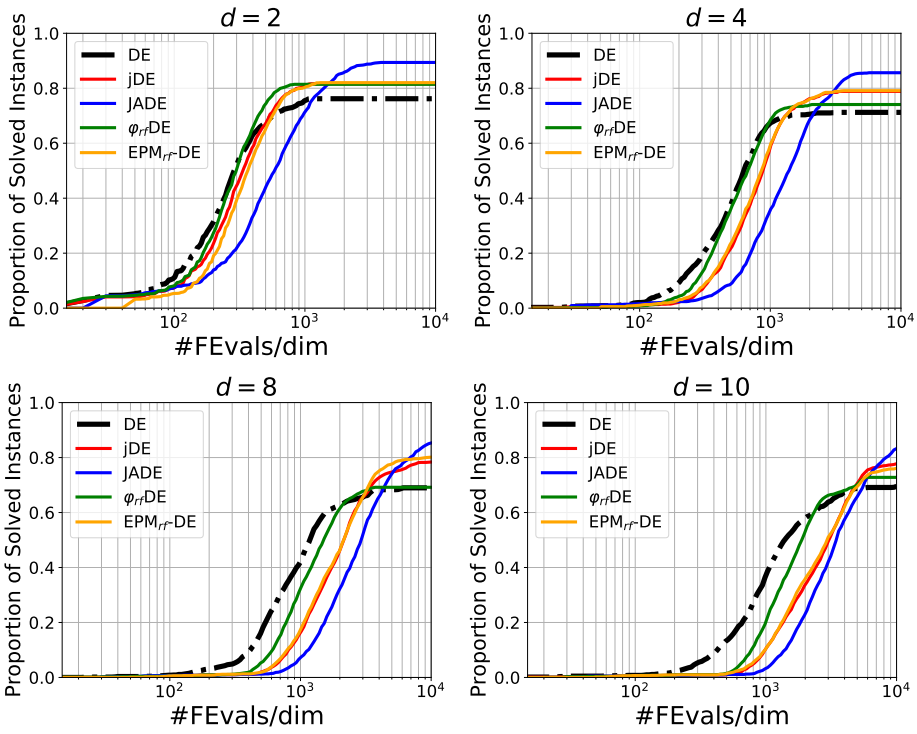


Figure 8.1: ECDF comparing φDE to four variants of DE: the original version, JADE, jDE, and the static PIAC version using the same Empirical Performance Model than φDE. .

Interestingly, both φDE and DE are outperformed for larger budgets, by the other DE variants. A very peculiar behavior is that of JADE, that performs very poorly for small budgets, but catches up and reaches the best performances for $10^4\times$evaluations, that maximal budget for these experiments. A possible reason for that is that JADE needs to fill in the archive before actually taking advantage of it - but then has a similar slope, and goes even higher than the other algorithms. The other DE variants are intermediate between φDE and DE on the one hand, and JADE on the other hand. In particular, the static PIAC performs much worse than φDE for low budgets, but can solve more functions for budgets up to $10^4 \times d$ evaluations - except in dimension 2.

8.3 CMA-ES Case Study

Chapter 7 empirically assessed the performance of PIAC with CMA-ES as the target algorithm, and it empirically demonstrated that using sample of very low budget, like $10 \times d$ can be used to predict a parameter setting that outperforms the default setting of CMA-ES. The natural direction is to avoid any additional call of the objective function and only rely on the population generated by CMA-ES.

Therefore, this Section aims at investigating the same online parameter control mechanism than in the previous section, embedding an EPM to tune the values of the c_1, c_μ and c_c parameters during the run.

8.3.1 Experimental Setting

This empirical study follows the same experimental protocol as described in Section 8.2, using the same test bench of functions. Following to Section 7.3, the EPM is learned by using a Random Forest regression, such that the features are computed from samples of size $50 \times d$, in accordance with the results in Chapter 7, and with the set of optimal parameter settings $\Theta_\star$.

This new variant of CMA-ES is denoted by ϕCMA-ES. At each iteration, the current population is used as samples to compute features and predict the values of c_1, c_μ and c_c for the next iteration. In accordance with Chapter7, ϕCMA-ES is based on the BIPOP-CMAES variant [LSS12a] with $\lambda = 100$, $\sigma_0 = 0.3$. The empirical performance model is learned once for all experiments.

ϕCMA-ES is compared to three variants of CMA-ES (all using $\lambda = 100$, $\sigma_0 = 0.3$ and 9 restarts):

- The default parameter setting of BIPOP-CMA-ES.
- The Self-CMA-ES variant [Los+14] (described in Section 2.3.3).
- the EPM-CMAES variant, using the static PIAC empirically studied in Section 7, i.e., setting the parameters at the beginning of the run. The features for this static version are computed from a sample size of $50 \times d$ and the EPM is a Random Forest regression.

These algorithms are empirically tested on the test functions proposed in Section 5.2.4.2 with a limited evaluation budget $10^3 \times d$, such that the target precision to reach is 10^{-6}. Dimensions $d \in \{2, 4, 10, 20\}$ are investigated, with 15 trials for each test functions.

8.3.2 Results

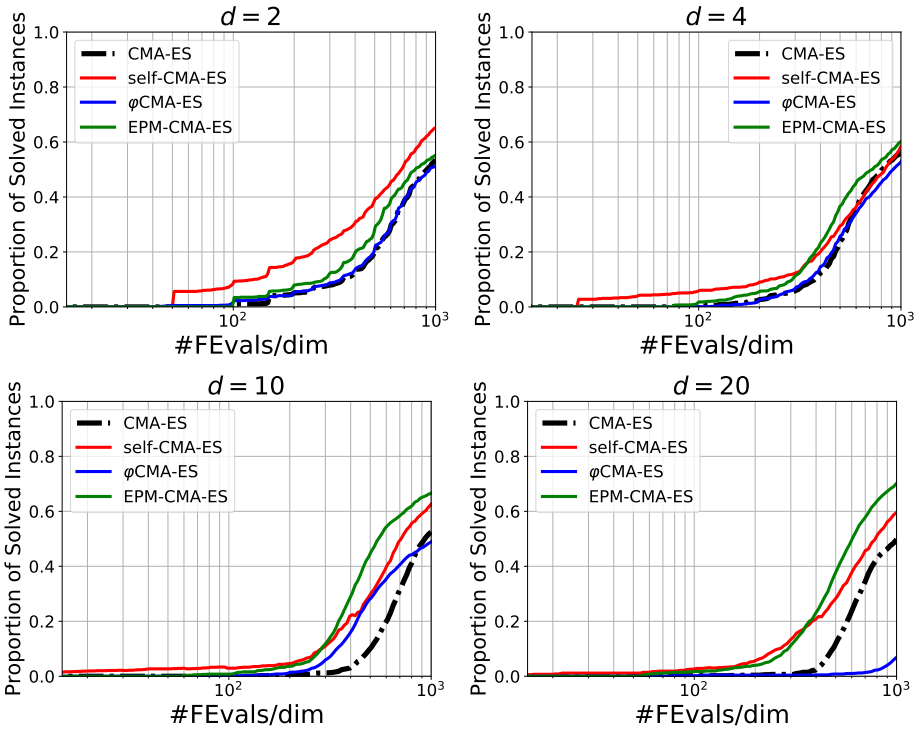


Figure 8.2: ECDF comparing φCMA-ES to the original version of CMA-ES, EPM-CMA-ES, and self-CMA-ES .

Figure 8.2 displays the ECDF of all four algorithms, aggregated per dimensions.

First of all, none of the algorithms there obtains very good results with this limited budget, as only 60% of the functions are solved in the maximum budget of $10^3 \times d$.

Interestingly, the static PIAC variant of CMA-ES obtains the best results – except in dimension 2, and in all dimensions for very low budgets, $100 \times d$ to $300 \times d$, where the self-CMA-ES performs best. Unfortunately, the EPM-based online tuning φCMA-ES obtains rather poor results, almost identical to those of CMA-ES for small dimensions, and terrible for $d = 20$.

However, because the training phase of the EPM was done using a maximal

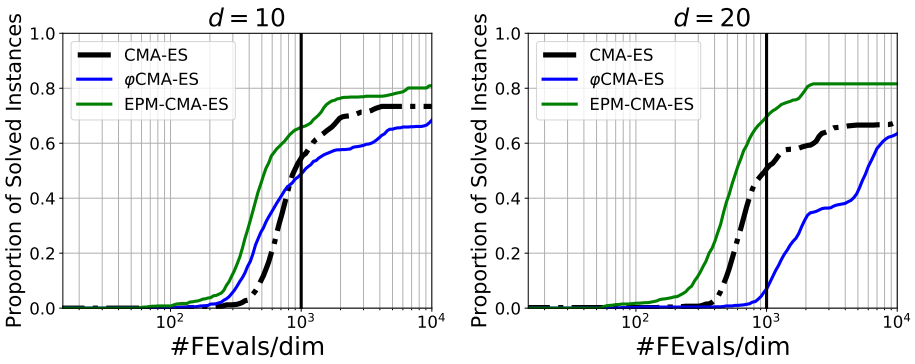


Figure 8.3: ECDF comparing φCMA-ES to the original version of CMA-ES, and EPM-CMA-ES, with maximum budget of function evaluation is $10^4 \times d$.

budget per run of $10^3 \times d$ evaluations, it is interesting to check what happens if the EPM is used for larger budgets. Figure 8.3 compares the ECDF, averaged per dimensions, of the same algorithms when a maximum budget of $10^4 \times d$ is used. We observe that when the budget is larger, the performance of φCMA-ES remain worse than that of the default parameter setting of CMA-ES, even if it catches up on the default CMA-ES for $d = 20$. Surprisingly, the performance of CMA-ES does not seem to increase much either – but these results are in accordance with Chapter 7, where we observe that CMA-ES fails in many test functions, like *Happy-Cat* or *Needle Eye*.

Figure 8.4 shows typical run with median performance of CMA-ES and φCMA-ES for an excerpt , and displays the values of c_1, c_μ and c_c. Regarding φCMA-ES we observe two main behaviors: on the one hand small changes in the behavior can be observed, in particular when test functions are solved without restart; on the other hand, we observe abrupt changes in the behavior, particularly after a restart or when the test function is multimodal, like *Happy-Cat* for which c_c and c_μ greatly vary after $10^3 \times d$.

8.4 Discussion

This Chapter has proposed an original online parameter control mechanism based on an Empirical Performance Model: the training is the same as in the static PIAC described in length in Section 3.3, but when running the algorithm on an unknown test function, the features are recomputed every generation, using

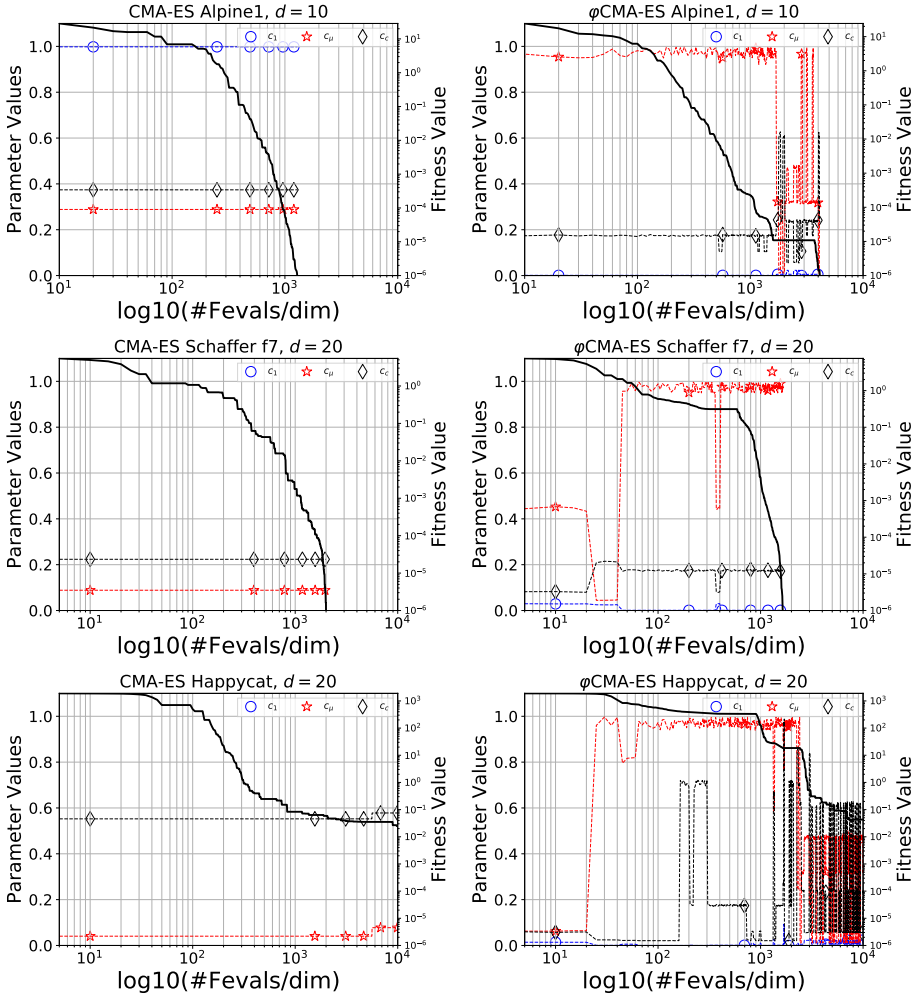


Figure 8.4: Example of a run of median performance of CMA-ES (left) and φCMA-ES (right). The parameter values and the best f_{min} values are displayed..

the most recently visited points of the search space, and new parameters are computed from the EPM for these new values of the features. The idea is that the fitness landscape does change along the run, and the optimal parameters hence might change too. We then presented very preliminary investigations of this new online parameter control mechanism for DE and CMA-ES.

The results are twofold: promising for DE, at least for small budgets, and disappointing for CMA-ES, even more so in large dimension ($d = 20$). In particular, whereas the online version performs better than the static one for DE, the reverse is true in the CMA-ES case, where the online version performs worse than the default setting. In any case, considering these limited experiments, the static PIAC is to be preferred to the online setting when the target algorithm is CMA-ES.

However, many paths remain unexplored regarding the proposed inline tuning approach. In the above experiments, all features are recomputed at every iteration using only the most recently visited points of the search space. This results in sometimes sudden variations of the controlled parameters, that can only harm the smoothness of the search. Other possibilities include adding the recent points to the archive of samples (and possibly forgetting the oldest points from the archive), or performing a sliding average rather than a sudden replacement of the continuous parameters under control.

More experiments should also be made with varying sample size for the computation of the Empirical Performance Model. Furthermore, the EPM itself could be recomputed or refined using all the points visited by the search process. In any case, this possibility opens at the moment more research paths than it answers questions. As a first hint, Mascia et al. [Mas+14] empirically investigated the parameter adaptation of a Tabu Search for solving Quadratic Assignement Problems, and proposed a self-adaptive version of Tabu Search that relies on the problem instance features.

CHAPTER 9

Conclusion

9.1 Summary of the Contributions

In this thesis, Per Instance Algorithm Configuration has been explored in the context of continuous black box optimization. The thesis followed the strategy introduced in Chapter 1 and investigated the different components that are necessary to the PIAC (see Section 3.3), from those in the training set to those during the optimization process.

In a first part, we introduced the background of this thesis with an overview of evolutionary computation for continuous black box optimization (see Section 2), and Algorithm Configuration and Per Instance Algorithm Configuration (see Section 3), followed by the description of problem features that characterize optimization problems required for PIAC, in Section 4.

In Chapter 5, we first empirically studied the computation of features in the continuous black box domain. Initial experiments considered the 'exact' values of features, i.e., that are computed from a large sample set of function values, until the variance of the empirical value of the feature becomes very small, and we empirically demonstrated that they can be used to retrieve the BBOB classification (see Section 3.1.1). From there on, in order to be able to use these features into real-world conditions with expensive objective functions, we empirically investigated the values taken by these features when computed with a low budget. First, when the features are directly computed from a small random sample set; then we proposed to first compute a surrogate model of the objective function, and to use it to artificially add samples to the original sample set. We empirically demonstrated that sub-sampled and surrogated-assisted features maintain their accuracy and efficiency w.r.t. the 'exact' values.

Because there is no consensus on a benchmark or experimental protocol to empirically study the PIAC, we proposed a general experimental protocol to study properly the different components of the PIAC. This protocol relies on both a cross-validation procedure on the BBOB testbench, and a validation on a new testbench.

In Chapter 6, we applied the general experimental protocol to empirically study the PIAC with Differential Evolution as the target algorithm. Based on the cross-validation procedure, a preliminary experiment aimed at investigating PIAC when 'exact' features can be computed, and these results were used as a baseline for comparison for the remaining experiments. It was compared to the default parameter setting of DE, as well as to a robust parameter setting learned on each problem separately (see Section 3.2.4). These experiments demonstrated that an empirical performance model (EPM) can be learned from the relationship between features and parameter configurations. Despite a poor approximation of the EPM, leading to a misleading predicted performance map of some test functions, good parameter settings could be found in general, that outperform the robust parameter setting.

Tackling costly objective functions, we empirically investigate PIAC w.r.t. the computation of features. For different sample sizes, we compared the sub-sampled features and the surrogate-assisted features with the 'exact' features and their parameter settings to the robust parameter setting. We empirically demonstrated that the performances are maintained even when only low budget for feature computation was available. However, we observed questionable performances for PIAC with surrogate-assisted features: whereas they still outperform the robust parameter setting, sub-sampled features repeatedly tend to outperform the surrogate-assisted features for most dimensions and test functions.

Next, in order to explore the different components of PIAC, we focused the experiments on the set of parameter configurations that is required to learn the EPM. We empirically demonstrated that using a smaller set of parameter configurations maintains the performance PIAC, except when using surrogate-assisted features, for which the performance of PIAC decreases dramatically in that context. Last, we empirically investigated the choice of the method to learn the EPM. Analogous to what was reported in [Hut+14], we empirically found out that Random Forest regression [Bre01] was performing best: this is the method that was used in all further experimental studies.

The exact value of the performance is only an indicator to compare the parameter settings, as the quantity of interest is in fact the ranking among the parameter settings. This is why we also investigated rank-based approaches in lieu of Random Forest. We compared three group of rank based methods [Liu09; Liu11]: pointwise methods with Logistic Ordinal Regression [McC80; Wat86], rankwise methods with RankSVM [HGO99; Joa02], and listwise with ListNet [Cao+07]. With some surprise, it turned out that Random Forest is the best method to learn the EPM, even if it is closely followed by RankSVM.

Following the general experimental protocol (see Section 5.2), we empirically validated PIAC on ExtBench, our new test bench, and we compared different settings of PIAC to the default parameter setting of DE. We assessed the performance of PIAC and empirically demonstrated that Random Forest regression is also the best learning method with sub-sampled features. Given a fixed evaluation budget, we empirically demonstrated that using only the set of optimal parameter configurations to learn the EPM is the best choice, in particular when the sample size to compute features is rather small ($\leq 50 \times d$). In particular, it strictly outperforms the default parameter setting.

While PIAC has show promising results on Differential Evolution, it was necessary to assess PIAC on a state-of-the art optimizer. Along this line, Chapter 7 proposes to assess the performance of PIAC applied to CMA-ES, a praised derivative free numerical black box optimizer. We proposed to use PIAC in order to tune three hyper-parameters of the adaptation mechanism of the Covariance matrix in CMA-ES — the learning rates c_1, c_μ and c_c — in experimental conditions close to real-world context: no prior knowledge of test functions, and a limited budget of function evaluations. Based on the results obtained when PIAC was applied to DE, we proposed a set of experiments that investigated the learning of the EPM, by restricting the experimental protocol to the validation on a new test bench. For different sample sizes, we compare the use of sub-sampled and surrogate-assisted feature. Also, we investigated the embedding of PIAC into the restart strategy [LSS12a], proposing that, at each restart, a new sample set is added to the original one – but no significant improvement was observed.

In this thesis, we empirically demonstrated that PIAC can be used with different algorithms, and that the proposed approach can outperform both the default parameter setting and a robust parameter setting learned on the full BBOB testbench. We observed that PIAC become useless when the dimensions of the functions of the training set are very different to those of the functions in the operational context where the learned EPM is later used, hence emphasizing the need to have a very large set of different problem instances to learn the EPM. However, when the dimensions of the unknown functions have been used for some functions in the training set, we recommend to use the PIAC approach we proposed: even in the context of a low budget to compute features ($30 \times d$), it can outperform the default CMA-ES, and the improvement increases with the dimension of the problem.

To summarize, this thesis empirically demonstrated that PIAC can be used in real-world conditions, first by studying the behavior of PIAC with respect to all its components, by validating our methodology on the state of the art algorithm, CMA-ES, demonstrating promising results compared to the default parameter

setting.

9.2 Perspectives

We would like now to mention several directions of future research which look the most promising in the light of our results. Several such directions of research were highlighted along this work, in particular when pointing out different limitations of the proposed methodology for continuous black box optimization.

At the core of the proposed approach are the features, the most relevant part on which PIAC relies. Though departing from the strict context of this work, it remains to investigate the "expensive" features, as defined in Section 4.4. Even though their computation require additional samples, there could be an interesting trade-off between their cost and the benefits for PIAC.

Along the same lines, new problem features should be investigated, that are as robust to sub-sampling as possible, e.g. by investigating the use of information geometry for black box optimization [Oll+11; MP14; Ben15]. In particular, they could require few additional samples, by contrast with the expensive features mentioned above.

Another related issue is the way the samples are gathered. Only uniform sampling has been used, whereas some random walks, biased by the objective function, might bring more beneficial information for the construction of the Empirical Performance Model.

Also, in the context of expensive objective functions for which only very small samples can be afforded, we have proposed *surrogate assisted features*, in which a surrogate is first built from the the available examples, and then then used to add artificial new samples that is less costly. However, we observed that such approach did not perform as well as expected. Another direction still relying on surrogate modeling of the objective function would be to investigate other surrogate models that are traditionally used in engineering like kriging models (aka Gaussian Processes) [JCS00; Jin05]. Though rapidly suffering from the curse of dimensionality, they could be efficient in small to medium dimensions (e.g., $d < 40$).

Of course, other target algorithms should be addressed by the proposed PIAC approach. First, as was done for CMA-ES with the parameters of the Covariance Matrix Adaptation, we should consider the different adaptive variants of Differential Evolution (briefly surveyed in Section 2.4.5), and try to automatically tune the hyper-parameters of their adaptation mechanism. Other algorithms and their most recent variants for continuous Black-Box optimization are natural

candidates for algorithm configuration, from EGO [JSW98], based on Gaussian Processes, to PSO, ACO, and other swarm intelligence algorithm.

This naturally leads to consider another level of automatic optimization algorithm, that would first address the Algorithm Selection problem, considering that the chosen algorithm will then be subject to Algorithm Configuration, as proposed by Xu, Hoos, and Leyton-Brown [XHL10] and Xu et al. [Xu+11]. Note that a first step in that direction has been done with the participation to the GECCO 2017 Black-Box Competition, see Appendix A, though only AS or AC has been done there. Interesting issues regard the Empirical Performance Models to be used in this context, one per target algorithm, with heterogeneous hyper-parameters to tune.

An initial proposal for an online parameter control mechanism based on PIAC has been presented in Section 8.1, hence opening several research paths. First, in this preliminary work, the features are computed only from the most recent population, hence favoring errors on their estimation because of the size of the samples and the sampling method. An alternative would consider an archive of the population, possibly sliding over time, in order to ensure both a sufficiently good estimation, thanks to a larger sample size, and a sufficiently dynamic process able to take into account the changes in the objective function landscape as the search is progressing.

Similarly, regarding parameter control itself, rather than directly setting the parameters to the new values predicted by the PIAC approach, some resulting in abrupt changes in the parameter values, some specific mechanism could be used for smoothing the trajectory of parameter values, e.g., by using a weighted sliding average, or an archive of the parameters or a threshold preventing too abrupt changes. In that direction, an empirical analysis of the parameter adaptation of the parameters w.r.t. local features should be done, as studied in [Mas+14].

Furthermore, during online control, in order to cope with local structures, and at the same time cope with very small samples (given by the population of the algorithm), new problem features should be investigated. A first hint on these features would be to use information gathered during the run of the algorithm. For example, for CMA-ES, the path of the step-size, and the Covariance matrix should be exploited as they give information on local properties of search space. Hence, a set of features that are based on them should be investigated.

Part V

Appendices

APPENDIX A

GECCO 2017 Black-Box Competition

This section is copied from a description of the algorithm provided to the competition, and is supposed to be more or less self-contained. Hence some redundancies here, as an appendix of this PhD dissertation.

A.1 Global Description

In the context of the GECCO 2017 Black-Box Competition (bbcomp.ini.rub.de), we proposed an approach that relies on Instance-based Algorithm Selection and Algorithm Configuration, where traditional AS/AC are unusable. Traditional AS/AC approach can be viewed as meta-optimization problems, respectively, in the algorithms space or in a given algorithm parameter space. The meta-objective function is a performance measure of an algorithm or a parameter setting of the algorithm. Nonetheless, such approaches can only be empirically assessed at the cost of running algorithms/parameter configurations on a training set of problem instances, hence resulting in an immensely costly meta-optimization task. Furthermore, a critical issue is that of the generalization of the learned parameters to other instances outside the training set.

Alternatively, recent approaches proposed to rely on a description of the objective function by some *features*, that can be given for free or computed with a rather small sample of objective function values. Then, given a large set of optimization problems, and algorithms/parameter configurations, the goal is to learn a mapping from the features space (that describes problem examples) and

the algorithm/parameter settings space to the performance of the algorithms or the parameter settings. When an unknown objective function is to be optimized, its set of features is computed, and the best algorithm of parameter setting is obtained by solving a simple and costless optimization problem: the cost of the choice of the algorithm or the parameter setting is only that of the features.

The problem instances of BBComp are pure black box: only remote calls of the objective function are possible, and strictly limited to the defined budget. Therefore we need to be able to compute problem features with very low budget, and have problem features that efficient even when using only such very small budget.

Preliminary results on the Per Instance Algorithm Configuration with CMA-ES or Differential Evolution as target algorithm were obtained when the over optimization budget was respectively $10^3 \times d$ and $10^3 \times d$. However, within BBComp, when $d \leq 10$, the overall optimization budget is smaller: we had to adapt the approach that was proposed in this dissertation. Our general approach still relies on problem features, computed from a uniformly randomized sample for each new problem instance. But, two situations are distinguished depending on the dimension d of the problem: when $d < 10$ a Per Instance Algorithm Selection is performed, otherwise the Per Instance Algorithm Configuration described and analyzed in this dissertation is used. In both cases, Empirical Performances Models are trained, that rely on the computation of problem features for the prediction of the 'best' algorithm or the 'best' parameter setting respectively.

A.2 Experimental Settings

A.2.1 BBComp

The BBcomp competition is composed of 1000 problem instances including different dimensions $d \in \{2, 4, 5, 8, 10, 16, 20, 32, 40, 64\}$ and with a function evaluation budget of $100 \times d^2$.

A.2.1.1 BBOB for learning an EPM

All Empirical Performance Models are trained on the full BBOB testbench, as described also in Section 3.3.

The Black Box Optimization Benchmark (BBOB) [Han+10] is made of 24 functions analytically defined on $[-5, 5]^d$, with known global optima. They have been manually classified in five classes of problems with respect to their global properties (e.g. separability, multimodality, ...) and have been used in many of

the works cited in this PhD thesis as a test bench. The EPM is learned from a training set made of these 24 test functions in different dimensions d, namely here $d \in \{2, 4, 5, 8, 10, 16, 20, 32, 40, 64\}$.

As advocated in the original framework, each of the 15 independent runs that are run on each function actually uses a variant of the original function, obtained by a random translation of the optimum and, for the non-separable functions, a random rotation of the coordinate system.

A.2.2 The Features

In accordance with the results in Chapter 6 and Chapter 5, only *cheap* features are used in the following experiments, resulting in a feature space $\mathbf{\Psi}$ of size 51.

In Chapter 6, we observed that a low budget of evaluation function calls can be used to nevertheless provide efficient performance of PIAC. Furthermore, as said above, the goal is to experiment PIAC in real-world-like situations, i.e., with expensive objective functions.

A.3 Experimental Protocol

A.3.1 Per Instance Algorithm Selection

Five different optimizers are considered in this phase:

- 1+1 CMA-ES;
- CMA-ES with the default parameter setting (including $\lambda = 4 + 3\ln(n)$);
- A restarted Nelder-Mead [NM65; Han09c];
- L-BFGS [LN89];
- Differential Evolution with default parameter setting from Storn and Price [SP97]

Similarly to the PIAC methodology introduced in Section 3.3, an empirical performance model is learned, hence mapping the features space of BBOB and the algorithm to their performances on the BBOB testbench. Similarly to Chapter 6 and Chapter 7, the performance measure is the Expected Run Time at precision 10^{-6}.

The Per Instance Algorithm Selection is only performed when $d < 10$ such that for each new problem instance, the feature are computed from a uniformly randomized sample of size $10 \times d$. Then, the 'best' algorithm is the one that minimizes the predicted Expected Run Time.

A.3.1.1 Per Instance Algorithm Configuration

The Per Instance Algorithm Configuration is the same method that has been proposed in this phD thesis – see Chapter 7: the EPM-CMA-ES is embedding an empirical performance model trained on the BBOB test bench.

The EPM-CMA-ES is automatically selected when $d \geq 10$, and for each new problem instance the features are computed from a uniform sample of size $50 \times d$, in accordance with the results in Chapter 7

A.3.2 Hybrid Algorithm

Given the dimension of the problem instance, a budget of $0.1 \times Budget$ is allocated for a final local search. With the remaining $0.9 \times Budget$, Per Instance Algorithm Selection (when $d < 10$) or Per Instance Algorithm Configuration (when $d \geq 10$) is used: A uniform sample of $50 \times d$ is used to compute the features, and the chosen algorithm or parameter setting for CMA- ES us run for the remaining of the $0.9 \times Budget$. A final local search with the Nelder-Mead algorithm [Han09b] is performed for the last $0.1 \times Budget$.

A.4 Results

The results of the competition are visible on BBComp web site. A first table gives both the total paper score and the sum of rank obtained by each competitor.

The ranking and score are computed as follows. *The aggregation of performance over all problems in a track is analog to the formula one scoring system. In this analogy each participant corresponds to a driver and each benchmark problem of the competition track corresponds to a race track. For each problem participants are sorted w.r.t. performance and the top scorers receive points depending on their rank. The sum of these points over all problems is the overall score. Participants are ranked w.r.t. this overall score.For some (rather easy) problems we expect multiple participants to achieve near-optimal and hence extremely close results. In this situation purely numerical differences–possibly even depending on programming language and compiler–can decide upon the lion's share of the points (distributed among the top ranks). Hence the ranking procedure is*

slightly modified as follows: Let f denote the best overall performance achieved for a problem. Based on this value a numerical imprecision range is defined as* $\epsilon = max(10 - 14 \times f^*, 10^{-20})$. *All algorithms with performance values better than* $f^* + \epsilon$ *share the first rank, while all worse algorithms are ranked according to their exact performance. Alternative assessment procedures will be considered as well, however, their results will not affect the selection of the winners.* Based on this rule, our approach has obtained respectively 1068.28 (score) and 5257 (rank), while the second best algorithm received 857.317 and 5506.

Then, several Empirical Cumulative Distribution Function (ECDF) plots are displayed, representing the proportion of instances (y-axis) for which the corresponding precision (x-axis) has been reached in the allocated budget.

For instance, Figure A.1 is such ECDF plot for problem instances when $d < 10$, corresponding, in our hybrid algorithm, to using the Per Instance Algorithm Selection to automatically select the 'best' algorithm from the problem features. We observe that our approach is ranked first when $d = 5$ and second when $d = 8$, whereas for $d = 2$ or $d = 4$, it performs rather poorly.

Similarly Figure A.2 shows the ECDF for problem instances with $d \geq 10$, i.e., when our hybrid algorithm applied the Per Instance Algorithm Configuration with CMA-ES, denoted EPM-CMA-ES as in Chapter 7. We observe that EPM-CMA-ES is ranked first when $d \in \{10, 16, 20, 32, 40\}$. However, we observe poor performances when $d = 64$.

Figure A.3 shows the aggregated ECDF over all dimensions. Despite poor performances on some dimensions, our approach is the best when all dimensions are aggregated.

A.5 Discussion

We proposed an approach based on the computation problem features for the Per Instance Algorithm Selection (PIAS) and the Per Instance Algorithm Configuration (PIAC). This hybrid algorithm was ranked first out of 17 entries. Indeed, excellent results are observed when the EPM-CMA-ES is used, even though we observe poor performances when $d = 64$. Note however that for this dimension, only 50% of the test functions could effectively be optimized to the target precision, hence explaining the poor performances. On the other extreme, for $d = 2$ or 4, PIAS is performs poorly, suggesting that this approach is inadequate to small dimensions, where the overall optimization budget is very small. These results open a new research path, on the goal to automatically find the best algorithm **and** its optimal parameter setting for a given optimization problem. A first re-

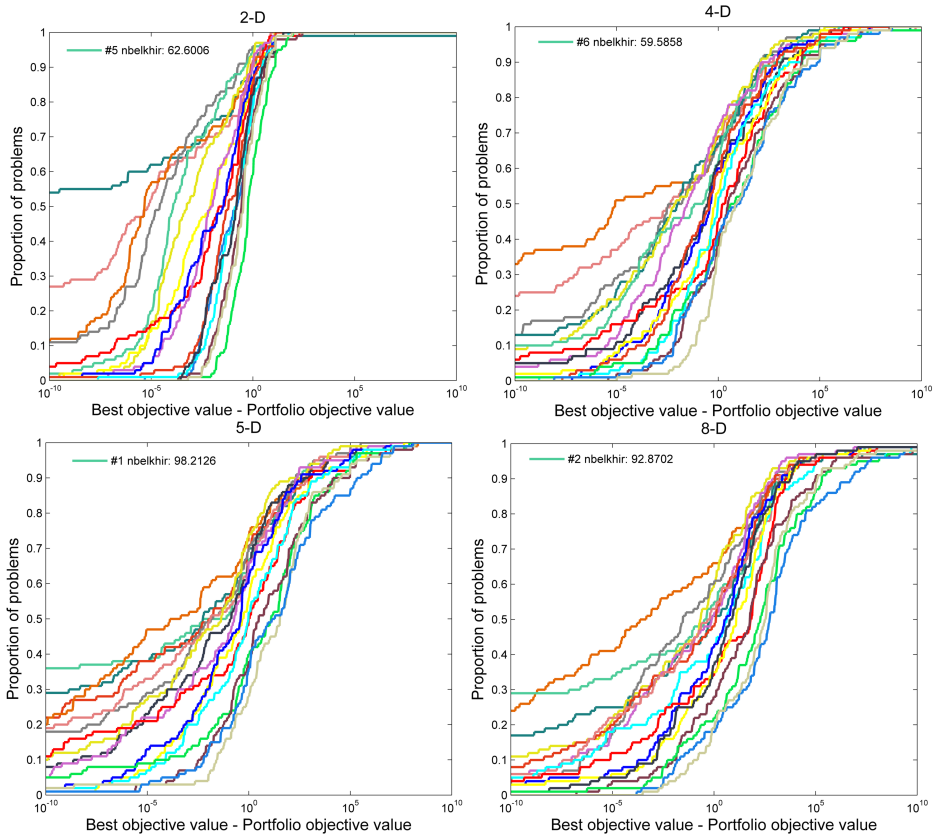


Figure A.1: Empirical Cumulative Distribution Function of the proportion of solved problem instances w.r.t. the numerical precision of the best fitness value, for dimensions $d \in \{2, 4, 5, 8$: for which Per Instance Algorithm Selection is used.

search path would be to investigate the different candidate algorithms to use in the training set. Next, regarding the experimental conditions: further works may have to focus on a specific performance measure for real-world conditions where a fixed budget is available $100 \times d^2$. Last, many explorative work remains to be done for PIAS. In particular, the general experimental protocol proposed in Section 5.2 for PIAC could be used as a starting point toward an adapted protocol for PIAS.

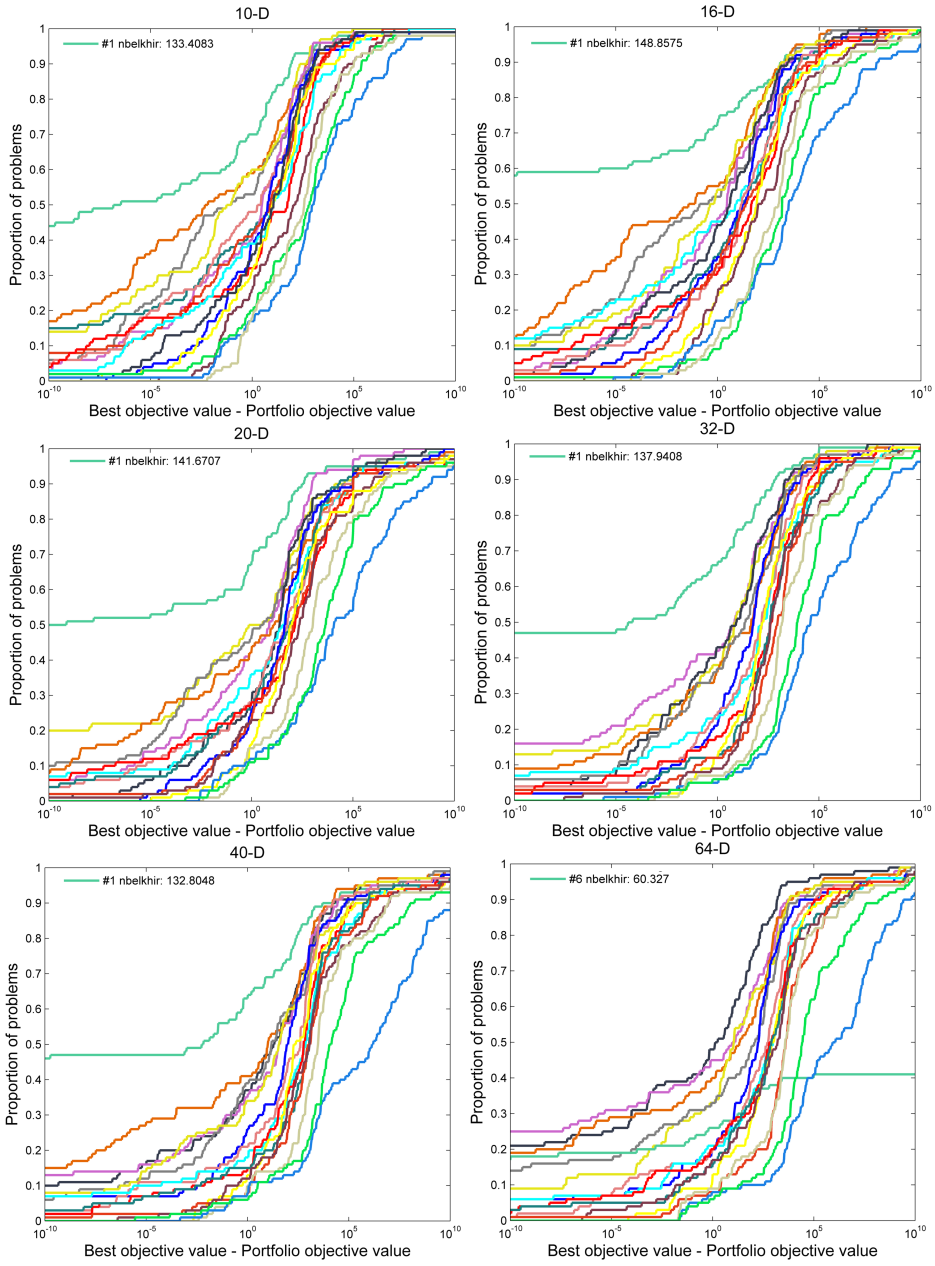


Figure A.2: Empirical Cumulative Distribution Function of the proportion of solved problem instances w.r.t. the numerical precision of the best fitness value, for dimensions $d \in \{10, 16, 20, 32, 40, 64$: for which Per Instance Algorithm Configuration with EMP-CMA-ES is used.

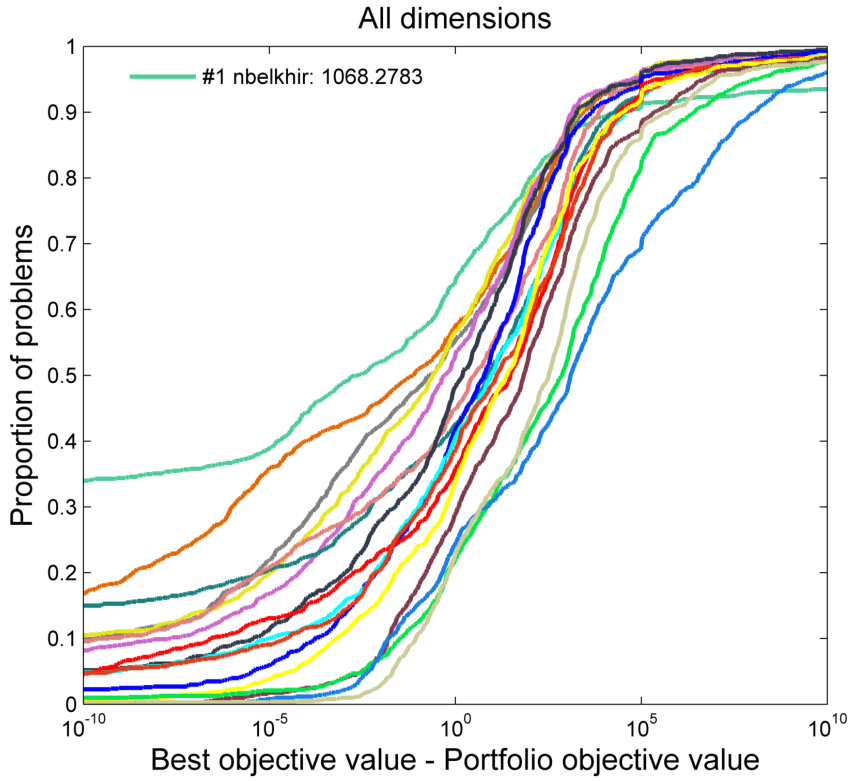


Figure A.3: Proportion of solved problem instances vs numerical precision of the best fitness value, for all the aggregated dimensions..

APPENDIX B

Machine Learning Tools

B.1 Regression Model

Regression methods comes from the Statistical field and Machine Learning for estimating the relationships among variables. It exists a wide variety of techniques for modeling and analyzing the relationships between dependent variables with respect to one or more independent variables. Regression methods are used in statistical analysis to have an insight on the independent variables, and focus on the relationship between values of dependent variables.

Regression methods are widely used for forecasting and prediction. However, the performance of regression methods depends on the form of the data and how it relates to the regression approach. Usually the real relationship between variables are generally unknown, while most regression methods rely to some extents on the assumptions on this relationship.

Here, we focus on the most popular techniques based on regression for prediction, forecasting but also surrogate modeling (see [JCS00; Jin05] for an overview of their use in engineering or evolution computation).

B.1.1 Gaussian Processes (GP)

Also known as Kriging, Gaussian processes regression, originates from the geostatistics community by Danie Krige in the 1950's and described by Matheron [Mat63]. Kriging is a well-known method for surrogate modeling, demonstrating good results for the global approximation of numerical optimization problems [JMY05; MS05; Sim+01], where kriging is used as a global rather than local surrogate model. Indeed, Kleijnen [Kle09] suggests that Kriging is more suited for larger problems than those involving low-order polynomial models.

Kriging relies on a collection of random variables, based on a Gaussian distribution [Ras04]. Then, the vector of function values, or dependent variables can be viewed as a sample of multivariate Gaussian distributions with joint probability $p(y_l|X_l)$ where X_l is the sample of point directly associated to the function values y_l, and where the predictive distribution of a new point x_{l+1} is determined from $l+1$ dimensional joint Gaussian distributions for the outputs of the l point.

B.1.2 Support Vector Machine (SVM)

Support Vector machines are popular Machine Learning methods, first proposed by Boser, Guyon, and Vapnik [BGV92], related to supervised learning methods used for classification and regression. SVM is a generalized methods based on machine learning theory by maximizing the predictive accuracy, while avoiding overfitting to the data. Initially, SVM is defined as systems which consider the hypothesis of linearity, but in a higher dimensional variable space.

The foundations of SVM have been developed by Vladimir and Vapnik [VV95], and gained a wide popularity due to some promising results [Gun+98; SS04]. It uses the Structural Risk Minimization [Sha+98; VV98] that minimize an upper bound on the expected risk, while Empirical Risk Minimization minimize the error on the training data.

While the method was initially proposed for classification tasks, it was quickly formulated for regression tasks [V+97]. This is successfully applied to regression tasks with the introduction of an alternative loss function [CS00; Smo+96], in order to include a distance metric.

Originally, SVM aim at finding the fitting of the linear hyperplanes between data in a high dimension variable space. But, when the data are not linear, it results in a bad fitting of the hyperplane. Then the use of kernels is proposed by Cristianini and Shawe-Taylor [CS00] and Amari and Wu [AW99] in order to non-linearly map the data to the high-dimensional space, resulting in a linear separability. Kernel functions aim at enabling operations to be performed in the input space rather than the possibly high dimensional feature space, with the goal to perform a mapping of the attributed of the input space to the feature space. Different Kernel functions are proposed by Amari and Wu [AW99], in order to deal with non-linear spaces: most notably, the polynomial kernel well known for non-linear modeling, the Gaussian radial basis relying on a Gaussian form, or the exponential radial basis function that produces piecewise linear solutions (that can be used when discontinuity is acceptable).

B.1.3 Random Forest

Random Forest, proposed by Breiman [Bre01] is a well know ensemble learning method for classification and regression, by constructing a set of decision trees adding some randomness by bootstrapping samples in each trees. Random Forest is a well-know for its robustness against overfitting, hence improving decision trees.

Given n the number of trees, n bootstrap samples are drawn from the original data, and for each bootstrap sample, an unprunned tree is grown, following the modification: at each node, rather than choosing the best split among all predictors, randomly sample m of the predictors and choose the best split from among those variables.

The basic novelty of Random Forest, is based on the use of an out-of-bag error as an estimate of the generalization error (by predicting data from samples not in the bootstrap samples), and especially the measure of variable importance. In the Random Forest, the variable importance is estimated by computing how much prediction error increases by considering a permutation between the bootstrap sample and out-of-bag samples for the variable to be measured while other variables are kept unchanged, then the importance measure is computed by averaging the difference of the out-of-bag error on all trees. Each leaf of trees contains a distribution of the continuous output variable, the predicted value is the mean of this distribution.

B.2 Rank based Methods

Rank Based methods also called *Learning to Rank* or Machine Learning Ranking, are popular application of machine learning with the main goal to learn a model based on the ranking of examples. The training dataset contains several lists of examples with a partial order, that are given by an ordinal or numerical score. The ranking model purpose is to find a ranking of new and unseen example. As an example Ranking problem is a central part, of Information Retrieval [SM86; B+99; Sin01] (e.g. sentiment analysis, collaborative filtering , or online advertisement [Joa02] …).

Many machine learning application can be viewed as a ranking problem,e.g. medical imaging problem [Ped+12] or in Black Box optimization (as a surrogate model [LSS12b; LSS13]), and demonstrated promising results when a ranking that preserve ordering is required. The PIAC methodology purpose is to find the best parameter configuration among other configurations, making ranking method worth investigating and compared to classical regression methods.

In the context of this thesis, the PIAC training dataset is composed of parameter configurations evaluated with respect to a a numerical performance measure (ERT), thus a partial ordering is possible.

Many machine learning methods are relevant when dealing with *learning to rank* problems. In [Liu09; Liu11] gives an extensive overview of machine learning methods directly related to *learning to rank* problems. Three main approaches that model the *learning to rank* process in different ways are introduced:

- Pointwise approaches consider an input space of variable vector of each example(e.g. $\psi \times \theta$), such that the output space is assumed to contains a scoring value (numerical or ordinal). Then, the ranking can be modeled as a regression, classification or ordinal regression, such that the corresponding loss function is used. While some of these methods, are efficient, and can be widely applied on ranking problem [CG05a; CK05; CG05b; CGD92; CZ06; Fuh89; Gey94; Kra+00; LWB07; SL02], it has some limitations (e.g. no preservation of ordering).

- Pairwise approach consider a pairing of comparable examples of the training dataset, such that for each transformed example, the pairwise preference (taking a value $\{-1, 1\}$) is associated. A loss function must be defined when dealing with such approaches. However for some algorithms, the ranking is modeled as a pairwise classification, thus the corresponding classification loss function is used. Different algorithm belongs to the pairwise approach [Bur+05; SS98; Joa02; HGO99; Qin+07], including specific loss function for some of them.

- Listwise approach, consider a similar input and output space as pointwise approaches. In contrast to pointwise approach, a specific loss function must be minimized with respect to the ordering, hence it can naturally consider the positions of examples in the ranked list. The literature proposes different algorithm to tackle this approach, by proposing new methodologies or new loss functions [QL07; Cao+07; Qin+06; Qin+08; Tay+08; Xia+08; Yue+07]

B.2.1 Logistic Ordinal Regression

The Logistic Ordinal Regression, known as proportional odd model or ordered logit model, is a generalized linear model specially tailored for the case of predicting ordinal variables. This method was first introduced in [McC80; Wat86] as an extension of the logistic regression to ordinal variables.

Given $X \in \mathbb{R}^{n \times p}$ as input data and $y \in \mathbb{R}^n$, that is assumed to be a non-decreasing vector. As the probability posterior used in logistic regression models, a cumulative probability

$$P(y \leq j|X) = \phi(\theta_j - w^T X_i = \frac{1}{1 + \exp(w^T X_i - \theta_j)}$$

is modeled, where w and θ are vectors to be estimated from the data and ϕ is the logistic function defined as

$$\phi(t) = \frac{}{1}(1 + \exp(-t))$$

By contrast to multi-class logistic regression, a constraint is added such that hyperplanes that separates each classes are parallel for all classes, such that w is common to all classes.

The vector θ is used to predict the class for which X_i belongs to, such that if there exists K different classes, θ is a non-decreasing vector of size $K - 1$. Then, the class j is assigned if the prediction $w^T X$ lies in the interval $[\theta_{j-1}, \theta_j[$ such that for external classes $\theta_0 = -\infty$ and $\theta_K = +\infty$. We aim at seeking the vector w such that Xw produces a set of values that are well separated into the different classes. The model is estimation can be viewed as an optimization problem, that minimize the loss function of the model, as the minus log likelihood

$$\mathcal{L}(w, \theta) = -\sum_{i=1}^{n} \log(\phi(\theta_{y_i} - w^T X_i) - \phi(\theta_{y_i - 1} - w^T X_i))$$

such that the gradient ∇ can easily computed using the formula $\log(\phi(t))' = (1 - \phi(t))$

B.2.2 RankSVM

RankSVM is a pairwise *learning to rank* proposed by [HGO99; Joa02], relies on the SVM framework for the task of pairwise classification. In contrast of ordinal

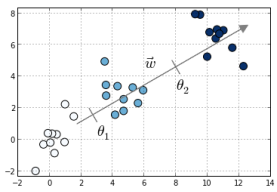


Figure B.1: Example of Ordinal Regression separating hyperplanes.

regression methods, the output space is in the form of $y \in \{-1, 1\}$.

Given $X \in \mathbb{R}^{n \times p}$, all comparable examples $x_i \in X$ are paired $(x_u^{(i)}, x_v^{(i)})$ and the corresponding ground truth label $y_{u,v}^{(i)}$ is computed as follows

$$y_{u,v}^{(i)} = \begin{cases} 1, & \text{if } x_u^{(i)} \geq x_v^{(i)}, \\ -1, & \text{otherwise.} \end{cases}$$

for all paired examples. The objective function of RankSVM is detailed in Equation (B.3) where the linear function $f(x) = w^T x$ is used

$$\min \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \sum_{v: y_{u,v}^{(i)}=1} \xi_{u,v}^{(i)} \tag{B.1}$$

$$\text{with } w^T(x_u^{(i)} - x_v^{(i)}) \geq 1 - \xi_{u,v}^{(i)}\ ,\ if y_{u,v}^{(i)} = 1, \tag{B.2}$$

$$\xi_{u,v}^{(i)} \geq 0,\ i = 1, \cdots, n. \tag{B.3}$$

The RankSVM formulation is identical to the SVM where the margin $\frac{1}{2}\|w\|^2$ aim at controlling the complexity of the model w. the main difference with SVM lies in the pairwise constraint between two paired examples. Also a Hinge loss function is defined on paired examples. Thus, RankSVM inherits the properties of SVM, i.e. the marginalization maximization, thus having good generalization properties. Indeed, the kernel trick (see Section B.1 for more details) can be applied to handle non-linear problems.

B.2.3 ListNet

ListNet is a list-wise *learning to rank* method that uses a loss function that measures the inconsistency between the output of the ranking model and the ground truth permutation π_y directly related to the scoring function of examples.

The main contribution of the ListNet, lie in the definition of a listwise ranking loss function that is initially proposed in [Cao+07], which is based on the probability distribution on permutations. Such probability distribution has been extensively studied in the field of probability theory such that different well know model has been proposed to represent the probability distribution of permutations, e.g. the Luce model [Luc59; Pla75] and the Mallows model [Mal57]. As a permutation has strong correspondence with ranked list, these model may be applied to ranking problems. Therefore, ListNet is an application of the use of the Luce model to ranking model. Given the scoring function $f(s = \{s_j\}_{j=1}^{m}$ where

$s_j = f(x_j)$ that output the relevance scores of examples in $X \in \mathbb{R}^{n \times p}$, such that a probability for each possible permutation π of examples is defined with respect to the Luce model, based on a chain rule as follows:

$$P(\pi|\mathbf{s}) = \prod_{j=1}^{m} \frac{\varphi(s_{\pi(j)})}{\sum_{u=j}^{m} \varphi(s_{\pi(u)})}$$

where $\pi(j)$ denotes one example at the j-th position in the permutation π, φ is transformation function that can be linear or non-linear. Then the probability distribution is a product of each items denoted by a conditional probability $\frac{\varphi(s_{\pi(j)})}{\sum_{u=j}^{m} \varphi(s_{\pi(u)})}$.

ListNet defines a probability distribution based on the Luce model and a scoring function f. Then it defines a new probability distribution $P_y(\pi)$ from the true label. The K-L divergence is used between these two probability distribution and used a listwise ranking loss, the K-L divergence loss defined as follows

$$L(f; x, \pi_y) = D(P(\pi|\varphi(f(w, x)))\|P_y(\pi))$$

. In [Cao+07], a neural network is employed in the ListNet for the learning procedure, with a stochastic descent gradient that minimizes the K-L divergence loss.

Bibliography

[AD05] Ernesto P Adorio and U Diliman. "Mvf-multivariate test functions library in c for unconstrained global optimization". In: *Quezon City, Metro Manila, Philippines* (2005) (cit. on pp. 100, 104).

[AH05] Anne Auger and Nikolaus Hansen. "A Restart CMA Evolution Strategy with Increasing Population Size". In: *CEC'05*. Vol. 2. IEEE. 2005, pp. 1769–1776 (cit. on p. 23).

[AHS12] Anne Auger, Nikolaus Hansen, and Marc Schoenauer. *Benchmarking of continuous black box optimization algorithms*. 2012 (cit. on p. 132).

[AL06] Belarmino Adenso-Diaz and Manuel Laguna. "Fine-tuning of algorithms using fractional experimental designs and local search". In: *Operations Research* 54.1 (2006), pp. 99–114 (cit. on p. 40).

[Alt+94] Lee Altenberg et al. "The evolution of evolvability in genetic programming". In: *Advances in genetic programming* 3 (1994), pp. 47–74 (cit. on p. 66).

[AMT12] Tinus Abell, Yuri Malitsky, and Kevin Tierney. *Fitness landscape based features for exploiting black-box optimization problem structure*. Tech. rep. TR-2012-163. IT University of Copenhagen, 2012 (cit. on pp. 53, 75, 100, 103, 143).

[And+15] M. Andersson et al. "Parameter tuned CMA-ES on the CEC'15 expensive problems". In: *Proc. CEC'15)*. 2015, pp. 1950–1957 (cit. on p. 133).

[AO04] Charles Audet and Dominique Orban. "Finding optimal algorithmic parameters using a mesh adaptive direct search". In: *Cahiers du GERAD G-2004-xx* (2004) (cit. on p. 41).

[AST09] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. "A gender-based genetic algorithm for the automatic configuration of algorithms". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2009, pp. 142–157 (cit. on pp. 41, 46).

[AT10] Anne Auger and Olivier Teytaud. "Continuous Lunches are Free plus the Design of Optimal Optimization Algorithms". In: *Algorithmica* 57.1 (2010), pp. 121–146. DOI: 10.1007/s00453-008-9244-5. URL: https://hal.inria.fr/inria-00369788 (cit. on p. 4).

[Aug+09] Anne Auger et al. "Experimental Comparisons of Derivative Free Optimization Algorithms". In: *8th International Symposium on Experimental Algorithms*. Ed. by Jan Vahrenhold. LNCS. Springer Verlag, 2009. URL: https://hal.inria.fr/inria-00397334 (cit. on p. 29).

[AW00] Mark J Anderson and Patrick J Whitcomb. *Design of experiments*. Wiley Online Library, 2000 (cit. on p. 40).

[AW99] Shun-ichi Amari and Si Wu. "Improving support vector machine classifiers by modifying kernel functions". In: *Neural Networks* 12.6 (1999), pp. 783–789 (cit. on pp. 87, 176).

[B+99] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*. Vol. 463. ACM press New York, 1999 (cit. on p. 177).

[Bar09] Thomas Bartz-Beielstein. "Sequential parameter optimization". In: *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2009 (cit. on p. 43).

[Bar97] Lionel Barnett. *TANGLED WEBS – Evolutionary Dynamics on Fitness Landscapes with Neutrality*. MSc dissertation, School of Cognitive Sciences, University of East Sussex. 1997 (cit. on p. 65).

[Bar98] Lionel Barnett. "Ruggedness and neutrality: The NKP family of fitness landscapes". In: *Artificial Life VI: Proceedings of the sixth international conference on Artificial life*. 1998, pp. 18–27 (cit. on p. 65).

[BB07] Markus F Brameier and Wolfgang Banzhaf. *Linear genetic programming*. Springer Science & Business Media, 2007 (cit. on p. 16).

[BBS07] Prasanna Balaprakash, Mauro Birattari, and Thomas Stützle. "Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement". In: *International Workshop on Hybrid Metaheuristics*. Springer. 2007, pp. 108–122 (cit. on p. 45).

[BD04] Mauro Birattari and Marco Dorigo. “The problem of tuning metaheuristics as seen from a machine learning perspective”. In: (2004) (cit. on p. 44).

[BD69] George EP Box and Norman Richard Draper. *Evolutionary operation: a method for increasing industrial productivity*. JSTOR, 1969 (cit. on p. 13).

[Bea75] Kenneth George Beauchamp. *Walsh functions and their applications*. Vol. 3. Academic press, 1975 (cit. on p. 60).

[Ben15] Jérémy Bensadon. “Black-box optimization using geodesics in statistical manifolds”. In: *Entropy* 17.1 (2015), pp. 304–345 (cit. on p. 162).

[BFK13] Thomas Bäck, Christophe Foussette, and Peter Krause. *Contemporary Evolution Strategies*. Natural Computing Series. Springer, 2013 (cit. on p. 19).

[BGV92] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. “A training algorithm for optimal margin classifiers”. In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM. 1992, pp. 144–152 (cit. on p. 176).

[BH61] George EP Box and J Stuart Hunter. “The 2 k—p fractional factorial designs”. In: *Technometrics* 3.3 (1961), pp. 311–351 (cit. on p. 40).

[Bir+02] Mauro Birattari et al. “A Racing Algorithm for Configuring Metaheuristics.” In: *GECCO*. Vol. 2. 2002, pp. 11–18 (cit. on pp. 7, 44).

[Bir+10] Mauro Birattari et al. “F-Race and iterated F-Race: An overview”. In: *Experimental methods for the analysis of optimization algorithms*. Springer, 2010, pp. 311–336 (cit. on pp. 45, 102).

[Bir05] Mauro Birattari. *The Problem of Tuning Metaheuristics*. Dissertations in Artificial Intelligence 292. {IOS Press}, 2005 (cit. on p. 45).

[Bis+12] Bernd Bischl et al. “Algorithm selection based on exploratory landscape analysis and cost-sensitive learning”. In: *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM. 2012, pp. 313–320 (cit. on pp. 6, 78, 86).

[Blo+16] Aymeric Blot et al. “MO-ParamILS: A Multi-objective Automatic Algorithm Configuration Framework”. In: *International Conference on Learning and Intelligent Optimization*. Springer. 2016, pp. 32–47 (cit. on pp. 40, 46).

[BLP05] Thomas Bartz-Beielstein, Christian WG Lasarczyk, and Mike Preuß. “Sequential Parameter Optimization”. In: *CEC’05*. Vol. 1. IEEE. 2005, pp. 773–780 (cit. on p. 43).

[BM04] Thomas Bartz-Beielstein and Sandor Markon. “Tuning search algorithms for real-world applications: A regression tree based approach”. In: *Evolutionary Computation, 2004. CEC2004. Congress on*. Vol. 1. IEEE. 2004, pp. 1111–1118 (cit. on p. 44).

[BM14] Mohammad Reza Bonyadi and Zbigniew Michalewicz. “A locally convergent rotationally invariant particle swarm optimization algorithm”. In: *Swarm Intelligence* 8.3 (2014), pp. 159–198 (cit. on p. 19).

[BM16] Mohammad Reza Bonyadi and Zbigniew Michalewicz. “Particle swarm optimization for single objective continuous space problems: a review”. In: *Evolutionary computation* (2016) (cit. on p. 19).

[Bos+15] Jakob Bossek et al. “Learning Feature-Parameter Mappings for Parameter Tuning via the Profile Expected Improvement”. In: *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*. ACM. 2015, pp. 1319–1326 (cit. on pp. 53, 100, 103, 132, 135).

[Box57] George EP Box. “Evolutionary operation: A method for increasing industrial productivity”. In: *Applied Statistics* (1957), pp. 81–101 (cit. on p. 13).

[BPH15] Lukáš Bajer, Zbyněk Pitra, and Martin Holeňa. “Benchmarking gaussian processes and random forests surrogate models on the BBOB noiseless testbed”. In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 1143–1150 (cit. on p. 87).

[Bre+06] Janez Brest et al. “Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems”. In: *IEEE transactions on evolutionary computation* 10.6 (2006), pp. 646–657 (cit. on pp. 17, 29, 149, 150).

[Bre+09] Janez Brest et al. “Dynamic optimization using Self-Adaptive Differential Evolution.” In: *IEEE congress on evolutionary computation*. 2009, pp. 415–422 (cit. on pp. 17, 30, 150).

[Bre01] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32 (cit. on pp. 52, 87, 160, 177).

[Bre62] Hans J Bremermann. “Optimization through evolution and recombination”. In: *Self-organizing systems* 93 (1962), p. 106 (cit. on p. 13).

[Bro+10] Dimo Brockhoff et al. “Mirrored sampling and sequential selection for evolution strategies”. In: *Parallel Problem Solving from Nature, PPSN XI* (2010), pp. 11–21 (cit. on p. 53).

[BRS65] HJ Bremermann, M Rogson, and S Salaff. “Search by evolution”. In: *Biophysics and Cybernetic Systems* (1965), pp. 157–167 (cit. on p. 13).

[BS02] Hans-Georg Beyer and Hans-Paul Schwefel. “Evolution strategies–A comprehensive introduction”. In: *Natural computing* 1.1 (2002), pp. 3–52 (cit. on p. 21).

[Bur+05] Chris Burges et al. “Learning to rank using gradient descent”. In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 89–96 (cit. on p. 178).

[BZM06] Janez Brest, Viljem Zumer, and Mirjam Sepesy Maucec. “Self-adaptive differential evolution algorithm in constrained real-parameter optimization”. In: *2006 IEEE International Conference on Evolutionary Computation*. IEEE. 2006, pp. 215–222 (cit. on pp. 30, 151).

[Cag+00] Stefano Cagnoni et al. *Real-World Applications of Evolutionary Computing: EvoWorkshops 2000: EvoIASP, EvoSCONDI, EvoTel, EvoSTIM, EvoRob, and EvoFlight, Edinburgh, Scotland, UK, April 17, 2000 Proceedings*. Springer Science & Business Media, 2000 (cit. on p. 3).

[Cao+07] Zhe Cao et al. “Learning to rank: from pairwise approach to listwise approach”. In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 129–136 (cit. on pp. 112, 160, 178, 180, 181).

[CCW06] ZG Cheng, DZ Chen, and XH Wu. “Continuous ant colony optimization system based on normal distribution model of pheromone”. In: *Systems Engineering and Electronics* 28.3 (2006), pp. 458–462 (cit. on p. 18).

[CG05a] Wei Chu and Zoubin Ghahramani. “Gaussian processes for ordinal regression”. In: *Journal of Machine Learning Research* 6.Jul (2005), pp. 1019–1041 (cit. on p. 178).

[CG05b] Wei Chu and Zoubin Ghahramani. “Preference learning with Gaussian processes”. In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 137–144 (cit. on p. 178).

[CGD92] William S Cooper, Fredric C Gey, and Daniel P Dabney. “Probabilistic retrieval based on staged logistic regression”. In: *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 1992, pp. 198–210 (cit. on p. 178).

[Che+05] Ling Chen et al. “A method for solving optimization problem in continuous space using improved ant colony algorithm”. In: *Data Mining and Knowledge Management*. Springer, 2005, pp. 61–70 (cit. on p. 18).

[CK05] Wei Chu and S Sathiya Keerthi. “New approaches to support vector ordinal regression”. In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 145–152 (cit. on p. 178).

[CM11] Stephen Chen and James Montgomery. “A simple strategy to maintain diversity and reduce crowding in particle swarm optimization”. In: *Australasian Joint Conference on Artificial Intelligence*. Springer. 2011, pp. 281–290 (cit. on p. 19).

[Cor02] Heather J Cordell. “Epistasis: what it means, what it doesn’t mean, and statistical methods to detect it in humans”. In: *Human molecular genetics* 11.20 (2002), pp. 2463–2468 (cit. on p. 59).

[Coy+01] Steven P Coy et al. “Using experimental design to find effective parameter settings for heuristics”. In: *Journal of Heuristics* 7.1 (2001), pp. 77–97 (cit. on pp. 40, 46).

[Cra85] Nichael Lynn Cramer. “A representation for the adaptive generation of simple sequential programs”. In: *Proceedings of the First International Conference on Genetic Algorithms*. 1985, pp. 183–187 (cit. on p. 16).

[CS00] Nello Cristianini and John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000 (cit. on pp. 87, 176).

[CST17] Santanu Chakraborty, Ramesh Kumar Sharma, and Pushpa Tewari. “APPLICATION OF SOFT COMPUTING TECHNIQUES OVER HARD COMPUTING TECHNIQUES: A SURVEY”. In: *International Journal of Indestructible Mathematics & Computing* 1.1 (2017), pp. 08–17 (cit. on p. 3).

[CVC07] Philippe Collard, Sébastien Verel, and Manuel Clergue. “Local search heuristics: Fitness cloud versus fitness landscape”. In: *arXiv preprint arXiv:0709.4010* (2007) (cit. on p. 75).

[CZ06] David Cossock and Tong Zhang. “Subset ranking using regression”. In: *International Conference on Computational Learning Theory*. Springer. 2006, pp. 605–619 (cit. on p. 178).

[Cze08] Zbigniew J Czech. “Statistical measures of a fitness landscape for the vehicle routing problem”. In: *2008 IEEE International Symposium on Parallel and Distributed Processing*. 2008 (cit. on p. 64).

[Dav90] Yuval Davidor. “Epistasis variance: Suitability of a representation to genetic algorithms”. In: *Complex Systems* 4.4 (1990), pp. 369–383 (cit. on p. 60).

[Dav91] Yuval Davidor. “Epistasis variance: A viewpoint on GA-hardness”. In: *Foundations of genetic algorithms* 1 (1991), pp. 23–35 (cit. on p. 60).

[Daw03] Richard Dawkins. “The evolution of evolvability”. In: *On growth, form and computers* (2003), pp. 239–255 (cit. on p. 66).

[DB01] Kalyanmoy Deb and Hans-Georg Beyer. “Self-adaptive genetic algorithms with simulated binary crossover”. In: *Evolutionary Computation* 9.2 (2001), pp. 197–221 (cit. on p. 17).

[DB99] Kalyanmoy Deb and Hans-Georg Beyer. “Self-Adaptation in Real-Parameter Genetic Algorithms with Simulated Binary Crossover”. In: *Proc. Genetic and Evolutionary Computation Conference (GECCO 1999)*. Ed. by Wolfgang Banzhaf et al. Morgan Kaufmann, 1999, pp. 172–179 (cit. on p. 29).

[De 06] Kenneth A De Jong. *Evolutionary computation: a unified approach*. MIT press, 2006 (cit. on p. 16).

[De 07] Kenneth A De Jong. “Parameter setting in EAs: a 30 year perspective”. In: *Parameter setting in evolutionary algorithms*. Springer, 2007, pp. 1–18 (cit. on p. 16).

[DG97] Marco Dorigo and Luca Maria Gambardella. “Ant colonies for the travelling salesman problem”. In: *biosystems* 43.2 (1997), pp. 73–81 (cit. on p. 18).

[Dor92] Marco Dorigo. “Optimization, learning and natural algorithms”. In: *Ph. D. Thesis, Politecnico di Milano, Italy* (1992) (cit. on p. 18).

[Dré+06] Johann Dréo et al. *Metaheuristics for hard optimization: methods and case studies*. Springer Science & Business Media, 2006 (cit. on p. 13).

[Dré09] Johann Dréo. “Using performance fronts for parameter setting of stochastic metaheuristics”. In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. ACM. 2009, pp. 2197–2200 (cit. on p. 46).

[DS02] Johann Dréo and Patrick Siarry. “A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions”. In: *International Workshop on Ant Algorithms*. Springer. 2002, pp. 216–221 (cit. on p. 18).

[DS03] Marco Dorigo and Thomas Stützle. "The ant colony optimization metaheuristic: Algorithms, applications, and advances". In: *Handbook of metaheuristics*. Springer, 2003, pp. 250–285 (cit. on p. 18).

[DS04] Johann Dréo and Patrick Siarry. "Continuous interacting ant colony algorithm based on dense heterarchy". In: *Future Generation Computer Systems* 20.5 (2004), pp. 841–856 (cit. on p. 18).

[DS09] Marco Dorigo and Thomas Stützle. "Ant colony optimization: overview and recent advances". In: *Techreport, IRIDIA, Universite Libre de Bruxelles* (2009) (cit. on p. 18).

[DS11] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. "Differential evolution: a survey of the state-of-the-art". In: *IEEE transactions on evolutionary computation* 15.1 (2011), pp. 4–31 (cit. on p. 29).

[Dua+07] Hai-Bin Duan et al. "Improved ant colony algorithm for solving continuous space optimization problems". In: *Journal of System Simulation* 19.5 (2007), pp. 974–977 (cit. on p. 18).

[Eib+07] A.E. Eiben et al. "Parameter Control in Evolutionary Algorithms". In: *Parameter Setting in Evolutionary Algorithms*. Ed. by F.J. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz. Springer, 2007, pp. 19–46 (cit. on p. 149).

[Fia+10] Álvaro Fialho et al. "Comparison-based adaptive strategy selection with bandits in differential evolution". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2010, pp. 194–203 (cit. on p. 17).

[Fog62] Lawrence J Fogel. "Autonomous automata". In: *Industrial Research* 4.2 (1962), pp. 14–19 (cit. on p. 16).

[Fog66] Lawrence J Fogel. *Artificial Intelligence Through Simulated Evolution.[By] Lawrence J. Fogel... Alvin J. Owens... Michael J. Walsh*. John Wiley & Sons, 1966 (cit. on pp. 13, 16).

[Fog95] D. B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 1995 (cit. on p. 14).

[FR10] Álvaro Fialho and Raymond Ros. *Adaptive Strategy Selection within Differential Evolution on the BBOB-2010 Noiseless Benchmark*. Research Report RR-7259. April 2010. URL: https://hal.inria.fr/inria-00476160 (cit. on pp. 110, 111).

[Fri58] Richard M Friedberg. "A learning machine: Part I". In: *IBM Journal of Research and Development* 2.1 (1958), pp. 2–13 (cit. on p. 13).

[Fri59] RM Friedberg. “Dunham, B. u. JH North: A learning machine.(Part II)”. In: *IBM-J. Res & Dev* 3 (1959) (cit. on p. 13).

[FRP97] C Fonlupt, D Robilliard, and P Preux. “Fitness landscape and the behavior of heuristics”. In: *Evolution Artificielle*. Vol. 97. Citeseer. 1997 (cit. on p. 56).

[Fuh89] Norbert Fuhr. “Optimum polynomial retrieval functions based on the probability ranking principle”. In: *ACM Transactions on Information Systems (TOIS)* 7.3 (1989), pp. 183–204 (cit. on p. 178).

[GC96] Jonathan Gratch and Steve Chien. “Adaptive problem-solving for large-scale scheduling problems: A case study”. In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 365–396 (cit. on p. 39).

[GD92] Jonathan Gratch and Gerald DeJong. “COMPOSER: A Probabilistic Solution to the Utility Problem in Speed-up Learning.” In: (1992) (cit. on p. 39).

[GDH92] David E Goldberg, Kalyanmoy Deb, and Jeffrey Horn. “Massive multimodality, deception, and genetic algorithms”. In: *Urbana* 51 (1992), p. 61801 (cit. on p. 65).

[Geb+05] Cormac Gebruers et al. “Using CBR to select solution strategies in constraint programming”. In: *International Conference on Case-Based Reasoning*. Springer. 2005, pp. 222–236 (cit. on p. 51).

[Gey94] Fredric C Gey. “Inferring probability of relevance using the method of logistic regression”. In: *SIGIR’94*. Springer. 1994, pp. 222–231 (cit. on p. 178).

[Gin+14] David Ginsbourger et al. “Bayesian adaptive reconstruction of profile optima and optimizers”. In: *SIAM/ASA Journal on Uncertainty Quantification* 2.1 (2014), pp. 490–510 (cit. on p. 54).

[GMK02] Roger Gämperle, Sibylle D Müller, and Petros Koumoutsakos. “A parameter study for differential evolution”. In: *Advances in intelligent systems, fuzzy systems, evolutionary computation* 10 (2002), pp. 293–298 (cit. on p. 29).

[GOT03] Nicholas IM Gould, Dominique Orban, and Philippe L Toint. “CUTEr and SifDec: A constrained and unconstrained testing environment, revisited”. In: *ACM Transactions on Mathematical Software (TOMS)* 29.4 (2003), pp. 373–394 (cit. on pp. 41, 46).

[Gre14] John J Grefenstette. “Deception Considered Harmful sk”. In: *Foundations of Genetic Algorithms 1993 (FOGA 2)* 2 (2014), p. 75 (cit. on p. 65).

[Gre86] J. J. Grefenstette. "Optimization of Control Parameters for Genetic Algorithms". In: *IEEE Trans. on Systems, Man and Cybernetics* SMC-16 (1986) (cit. on p. 39).

[Gun+98] Steve R Gunn et al. "Support vector machines for classification and regression". In: *ISIS technical report* 14 (1998) (cit. on p. 176).

[HA96] N Hansen and Ostermeier A. "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation". In: *ICEC96*. IEEE Press, 1996, pp. 312–317 (cit. on p. 21).

[Han+08] Nikolaus Hansen et al. "PSO facing non-separable and ill-conditioned problems". PhD thesis. INRIA, 2008 (cit. on p. 62).

[Han+09a] Nikolaus Hansen et al. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions*. Tech. rep. RR-6829. INRIA, 2009 (cit. on p. 132).

[Han+09b] N. Hansen et al. "A Method for Handling Uncertainty in Evolutionary Optimization with an Application to Feedback Control of Combustion". In: *IEEE Transactions on Evolutionary Computation* 13.1 (2009), pp. 180–197 (cit. on p. 21).

[Han+10] Nikolaus Hansen et al. *Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup*. Tech. rep. RR-7215. INRIA, 2010 (cit. on pp. 4, 24, 26, 34–36, 53, 82, 86, 98, 100, 104, 132, 134, 150, 168).

[Han+11a] Nikolaus Hansen et al. "Impacts of Invariance in Search: When CMA-ES and PSO Face Ill-Conditioned and Non-Separable Problems". In: *Applied Soft Computing* 11 (2011), pp. 5755–5769. DOI: 10.1016/j.asoc.2011.03.001 (cit. on pp. 21, 22, 132).

[Han+11b] Nikolaus Hansen et al. "Impacts of invariance in search: When CMA-ES and PSO face ill-conditioned and non-separable problems". In: *Applied Soft Computing* 11.8 (2011), pp. 5755–5769 (cit. on p. 19).

[Han08] Nikolaus Hansen. "Adaptive encoding: How to render search coordinate system invariant". In: *International Conference on Parallel Problem Solving from Nature*. LNCS 5199, Springer Verlag, 2008, pp. 205–214 (cit. on p. 22).

[Han09a] N Hansen. *References to CMA-ES applications*. 2009 (cit. on p. 4).

[Han09b] Nikolaus Hansen. "Benchmarking a BI-population CMA-ES on the BBOB-2009 Function Testbed". In: *GECCO Companion*. Ed. by Franz Rothlauf. ACM. 2009, pp. 2389–2396 (cit. on pp. 21, 132, 170).

[Han09c] Nikolaus Hansen. "Benchmarking the Nelder-Mead downhill simplex algorithm with many local restarts". In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. ACM. 2009, pp. 2403–2408 (cit. on p. 169).

[Has70] W Keith Hastings. "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1 (1970), pp. 97–109 (cit. on p. 63).

[Haw04] Douglas M. Hawkins. "The Problem of Overfitting". In: *Journal of Chemical Information and Computer Sciences* 44.1 (2004). PMID: 14741005, pp. 1–12. DOI: 10.1021/ci0342472 (cit. on p. 62).

[HGO99] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. "Large margin rank boundaries for ordinal regression". In: *Advances in neural information processing systems* (1999), pp. 115–132 (cit. on pp. 6, 112, 160, 178, 179).

[HH05] Frank Hutter and Youssef Hamadi. "Parameter adjustment based on performance prediction: Towards an instance-aware problem solver". In: *In: Technical Report: MSR-TR-2005125, Microsoft Research*. 2005 (cit. on p. 52).

[HHL11a] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Learning and Intelligent Optimization*. Ed. by Carlos A. Coello Coello. LNCS 6683, Springer Verlag, 2011, pp. 507–523 (cit. on p. 37).

[HHL11b] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "Sequential model-based optimization for general algorithm configuration". In: *Proc. LION 5*. Springer, 2011, pp. 507–523 (cit. on pp. 7, 44–46, 54, 102, 113, 115, 134).

[HHL13] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. "An evaluation of sequential model-based optimization for expensive blackbox functions". In: *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. ACM. 2013, pp. 1209–1216 (cit. on p. 44).

[HMK03] N. Hansen, S. Müller, and P. Koumoutsakos. "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)". In: *Evolution Computation* 11.1 (2003), pp. 1–18 (cit. on pp. 21, 22, 24).

[HO01a] Nikolaus Hansen and Andreas Ostermeier. "Completely Derandomized Self-Adaptation in Evolution Strategies". In: *Evolutionary Computation* 9.2 (June 2001), pp. 159–195 (cit. on pp. 21, 72).

[HO01b] Nikolaus Hansen and Andreas Ostermeier. "Completely derandomized self-adaptation in evolution strategies". In: *Evolutionary computation* 9.2 (2001), pp. 159–195 (cit. on pp. 6, 16, 132, 149).

[Hol67] John Henry Holland. "Nonlinear environments permitting efficient adaptation". In: (1967) (cit. on pp. 13, 15).

[Hol75] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975 (cit. on p. 65).

[Hoo11] Holger H Hoos. "Automated algorithm configuration and parameter tuning". In: *Autonomous search*. Springer, 2011, pp. 37–71 (cit. on p. 40).

[Hoo12] Holger H Hoos. "Programming by optimization". In: *Comm. of the ACM* 55.2 (2012), pp. 70–80 (cit. on p. 4).

[Hou+10] Michael E Houle et al. "Can shared-neighbor distances defeat the curse of dimensionality?" In: *International Conference on Scientific and Statistical Database Management*. Springer. 2010, pp. 482–500 (cit. on p. 58).

[HS04] Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004 (cit. on p. 13).

[HS80] Willi Hock and Klaus Schittkowski. "Test examples for nonlinear programming codes". In: *Journal of Optimization Theory and Applications* 30.1 (1980), pp. 127–129 (cit. on pp. 100, 104).

[Hsu13] Chieh Su Hsu. *Cell-to-cell mapping: a method of global analysis for nonlinear systems*. Vol. 64. Springer Science & Business Media, 2013 (cit. on p. 76).

[Hua+06a] Deng Huang et al. "Global optimization of stochastic black-box systems via sequential kriging meta-models". In: *Journal of global optimization* 34.3 (2006), pp. 441–466 (cit. on p. 43).

[Hua+06b] Deng Huang et al. "Sequential kriging optimization using multiple-fidelity evaluations". In: *Structural and Multidisciplinary Optimization* 32.5 (2006), pp. 369–382 (cit. on p. 43).

[Hut+06] Frank Hutter et al. "Performance prediction and automated tuning of randomized and parametric algorithms". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2006, pp. 213–228 (cit. on pp. 5, 48, 51, 52, 100).

[Hut+07] Frank Hutter et al. “Boosting verification by automatic tuning of decision procedures”. In: *Formal Methods in Computer Aided Design, 2007. FMCAD'07*. IEEE. 2007, pp. 27–34 (cit. on p. 36).

[Hut+09a] Frank Hutter et al. “An experimental investigation of model-based parameter optimisation: SPO and beyond”. In: *GECCO'09*. Ed. by Franz Rothlauf. ACM. 2009, pp. 271–278 (cit. on p. 22).

[Hut+09b] Frank Hutter et al. “ParamILS: An Automatic Algorithm Configuration Framework”. In: *JAIR* 36 (2009), pp. 267–306 (cit. on p. 41).

[Hut+09c] Frank Hutter et al. “ParamILS: an Automatic Algorithm Configuration Framework”. In: *JAIR* 36.1 (2009), pp. 267–306 (cit. on p. 40).

[Hut+09d] Frank Hutter et al. “ParamILS: an automatic algorithm configuration framework”. In: *Journal of Artificial Intelligence Research* 36.1 (2009), pp. 267–306 (cit. on p. 7).

[Hut+14] Frank Hutter et al. “Algorithm runtime prediction: Methods & evaluation”. In: *Artificial Intelligence* 206 (2014), pp. 79–111 (cit. on pp. 48, 52, 53, 100, 103, 160).

[Hut09] Frank Hutter. “Automated configuration of algorithms for solving hard computational problems”. PhD thesis. University of British Columbia, 2009 (cit. on pp. 5, 43).

[Huy96] Martijn A Huynen. “Exploring phenotype space through neutral evolution”. In: *Journal of molecular evolution* 43.3 (1996), pp. 165–169 (cit. on p. 64).

[J+95] Terry Jones, Stephanie Forrest, et al. “Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms.” In: *ICGA*. Vol. 95. 1995, pp. 184–192 (cit. on pp. 58, 65, 68).

[JCS00] Ruichen Jin, Wei Chen, and Timothy W. Simpson. “Comparative Studies Of Metamodeling Techniques Under Multiple Modeling Criteria”. In: *Structural and Multidisciplinary Optimization* 23 (2000), pp. 1–13 (cit. on pp. 86, 162, 175).

[Jin05] Yaochu Jin. “A Comprehensive Survey of Fitness Approximation in Evolutionary Computation”. In: *Soft Computing* 9.1 (2005), pp. 3–12 (cit. on pp. 86, 162, 175).

[Jin11] Yaochu Jin. “Surrogate-assisted evolutionary computation: Recent advances and future challenges”. In: *Swarm and Evolutionary Computation* 1.2 (2011), pp. 61–70 (cit. on p. 87).

[JM91] Cezary Z Janikow and Zbigniew Michalewicz. “An experimental comparison of binary and floating point representations in genetic algorithms.” In: *ICGA*. 1991, pp. 31–36 (cit. on p. 15).

[JMY05] Shinkyu Jeong, Mitsuhiro Murayama, and Kazuomi Yamamoto. "Efficient optimization design method using kriging model". In: *Journal of aircraft* 42.2 (2005), pp. 413–420 (cit. on pp. 87, 175).

[Joa02] Thorsten Joachims. "Optimizing search engines using clickthrough data". In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2002, pp. 133–142 (cit. on pp. 6, 112, 160, 177–179).

[Jon95] Terry Jones. "Evolutionary algorithms, fitness landscapes and search". PhD thesis. Citeseer, 1995 (cit. on pp. 55, 64, 65).

[JSW98] Donald R Jones, Matthias Schonlau, and William J Welch. "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13.4 (1998), pp. 455–492 (cit. on pp. 42, 163).

[Kad+10] Serdar Kadioglu et al. "ISAC - Instance-Specific Algorithm Configuration". In: *ECAI*. Vol. 215. 2010, pp. 751–756 (cit. on pp. 52, 53, 100, 143).

[KE95] J. Kennedy and R. Eberhart. "Particle swarm optimization". In: *Neural Networks, 1995. Proceedings., IEEE International Conference on*. Vol. 4. Perth, WA, Australia: IEEE, November 1995, 1942–1948 vol.4. ISBN: 0-7803-2768-3. DOI: 10.1109/icnn.1995.488968. URL: http://dx.doi.org/10.1109/icnn.1995.488968 (cit. on p. 19).

[Ker+14] Pascal Kerschke et al. "Cell mapping techniques for exploratory landscape analysis". In: *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*. Springer, 2014, pp. 115–131 (cit. on p. 76).

[Ker+15] Pascal Kerschke et al. "Detecting funnel structures by means of exploratory landscape analysis". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM. 2015, pp. 265–272 (cit. on p. 76).

[Ker+16] Pascal Kerschke et al. "Low-budget exploratory landscape analysis on multiple peaks models". In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM. 2016, pp. 229–236 (cit. on p. 78).

[KHK06] Stefan Kern, Nikolaus Hansen, and Petros Koumoutsakos. "Local meta-models for optimization using evolution strategies". In: *Parallel Problem Solving from Nature-PPSN IX*. Springer, 2006, pp. 939–948 (cit. on p. 87).

[Kil+06] Philip Kilby et al. "Estimating search tree size". In: *Proc. of the 21st National Conf. of Artificial Intelligence, AAAI, Menlo Park*. 2006 (cit. on p. 51).

[Kim+68] Motoo Kimura et al. “Evolutionary rate at the molecular level”. In: *Nature* 217.5129 (1968), pp. 624–626 (cit. on p. 64).

[Kle05] Jack PC Kleijnen. “An overview of the design and analysis of simulation experiments for sensitivity analysis”. In: *European Journal of Operational Research* 164.2 (2005), pp. 287–300 (cit. on p. 40).

[Kle09] Jack PC Kleijnen. “Kriging metamodeling in simulation: a review”. In: *European Journal of Operational Research* 192.3 (2009), pp. 707–716 (cit. on pp. 87, 175).

[KNR01] Leila Kallel, Bart Naudts, and Colin R Reeves. “Properties of fitness functions and search landscapes”. In: *Theoretical aspects of evolutionary computing*. Springer, 2001, pp. 175–206 (cit. on p. 60).

[Knu75] Donald E Knuth. “Estimating the efficiency of backtrack programs”. In: *Mathematics of computation* 29.129 (1975), pp. 122–136 (cit. on p. 51).

[KO06] Yoshiaki Katada and Kazuhiro Ohkura. “Estimating the degree of neutrality in fitness landscapes by the nei’s standard genetic distance-an application to evolutionary robotics”. In: *2006 IEEE International Conference on Evolutionary Computation*. IEEE. 2006, pp. 483–490 (cit. on p. 65).

[Kol14] Janet Kolodner. *Case-based reasoning*. Morgan Kaufmann, 2014 (cit. on p. 51).

[KOU04] Yoshiaki Katada, Kazuhiro Ohkura, and Kanji Ueda. “The Nei’s standard genetic distance in artificial evolution”. In: *Evolutionary Computation, 2004. CEC2004. Congress on*. Vol. 2. IEEE. 2004, pp. 1233–1239 (cit. on p. 65).

[Koz+96] John R Koza et al. *Genetic Programming 1996: proceedings of the first annual conference*. Mit Press, 1996 (cit. on p. 17).

[Koz90] John R Koza. “Concept formation and decision tree induction using the genetic programming paradigm”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 1990, pp. 124–128 (cit. on p. 16).

[Koz92] John R Koza. *Genetic programming: on the programming of computers by means of natural selection*. Vol. 1. MIT press, 1992 (cit. on p. 16).

[Kra+00] Stefan Kramer et al. “Prediction of ordinal classes using regression trees”. In: *International Symposium on Methodologies for Intelligent Systems*. Springer. 2000, pp. 426–434 (cit. on p. 178).

[KS76] John G Kemeny and James Laurie Snell. “Finite Markov Chains, Undergraduate Texts in Mathematics”. In: (1976) (cit. on p. 76).

[KS96] Leila Kallel and Marc Schoenauer. *Fitness distance correlation for variable length representations*. Tech. rep. Technical Report 363, CMAP, Ecole Polytechnique, 1996 (cit. on pp. 66, 68).

[L+00] Jouni Lampinen, Ivan Zelinka, et al. “On stagnation of the differential evolution algorithm”. In: *Proceedings of MENDEL*. 2000, pp. 76–83 (cit. on p. 29).

[L+98] Lionel Lobjois, Michel Lemaître, et al. “Branch and bound algorithm selection by performance prediction”. In: *AAAI/IAAI*. 1998, pp. 353–358 (cit. on p. 51).

[Ley+03] Kevin Leyton-Brown et al. “A portfolio approach to algorithm selection”. In: *IJCAI*. Vol. 1543. 2003, p. 2003 (cit. on p. 51).

[Lia+06a] Jing J Liang et al. “Comprehensive learning particle swarm optimizer for global optimization of multimodal functions”. In: *IEEE transactions on evolutionary computation* 10.3 (2006), pp. 281–295 (cit. on p. 19).

[Lia+06b] JJ Liang et al. “Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization”. In: *Journal of Applied Mechanics* 41.8 (2006) (cit. on pp. 30, 100).

[Liu+01] Yong Liu et al. “Scaling up fast evolutionary programming with cooperative coevolution”. In: *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*. Vol. 2. IEEE. 2001, pp. 1101–1108 (cit. on p. 58).

[Liu+02] Junhong Liu et al. “Adaptive parameter control of differential evolution”. In: *Proc. of Mendel*. 2002, pp. 19–26 (cit. on p. 17).

[Liu09] Tie-Yan Liu. “Learning to rank for information retrieval”. In: *Foundations and Trends in Information Retrieval* 3.3 (2009), pp. 225–331 (cit. on pp. 103, 160, 178).

[Liu11] Tie-Yan Liu. *Learning to rank for information retrieval*. Springer Science & Business Media, 2011 (cit. on pp. 103, 160, 178).

[LL02] Pedro Larranaga and Jose A Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*. Vol. 2. Springer Science & Business Media, 2002 (cit. on p. 17).

[LL05] Junhong Liu and Jouni Lampinen. “A fuzzy adaptive differential evolution algorithm”. In: *Soft Computing* 9.6 (2005), pp. 448–462 (cit. on p. 17).

[LLS12] Manuel López-Ibáñez, Tianjun Liao, and Thomas Stützle. "On the anytime behavior of IPOP-CMA-ES". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2012, pp. 357–366 (cit. on pp. 23, 133).

[LMS10] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. "Iterated local search: Framework and applications". In: *Handbook of metaheuristics*. Springer, 2010, pp. 363–397 (cit. on p. 40).

[LN89] Dong C Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical programming* 45.1 (1989), pp. 503–528 (cit. on p. 169).

[LNS02] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. "Learning the empirical hardness of optimization problems: The case of combinatorial auctions". In: *Principles and Practice of Constraint Programming-CP 2002*. Springer. 2002, pp. 556–572 (cit. on pp. 48, 100).

[LNS09] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. "Empirical hardness models: Methodology and a case study on combinatorial auctions". In: *Journal of the ACM (JACM)* 56.4 (2009), p. 22 (cit. on p. 51).

[Loc05] Marco Locatelli. "On the multilevel structure of global optimization problems". In: *Computational Optimization and Applications* 30.1 (2005), pp. 5–22 (cit. on p. 76).

[Lon81] James W Longley. "Modified gram-schmidt process vs. classical gram-schmidt: Modified gram-schmidt process vs. classical gram-schmidt". In: *Communications in Statistics-Simulation and Computation* 10.5 (1981), pp. 517–527 (cit. on p. 104).

[Lóp+11] Manuel López-Ibánez et al. *The irace package, iterated race for automatic algorithm configuration*. Tech. rep. Citeseer, 2011 (cit. on pp. 7, 45, 46).

[Los+14] Ilya Loshchilov et al. "Maximum Likelihood-based Online Adaptation of Hyper-parameters in CMA-ES". In: *PPSN XIII*. Ed. by Thomas Bartz-Beielstein et al. LNCS 8672, Springer Verlag, 2014, pp. 70–79 (cit. on pp. 23, 24, 133, 153).

[Lov93] László Lovász. "Random walks on graphs". In: *Combinatorics, Paul erdos is eighty* 2 (1993), pp. 1–46 (cit. on p. 63).

[LP89] George R Lindfield and John ET Penny. *Microcomputers in numerical analysis*. Halsted Press, 1989 (cit. on p. 74).

[LS13a] Tianjun Liao and Thomas Stützle. "Bounding the Population Size of IPOP-CMA-ES on the Noiseless BBOB Testbed". In: GECCO '13 Companion (2013), pp. 1161–1168. DOI: 10.1145/2464576.2482694. URL: http://doi.acm.org/10.1145/2464576.2482694 (cit. on pp. 23, 31, 37, 133).

[LS13b] Tianjun Liao and Thomas Stützle. "Testing the impact of parameter tuning on a variant of IPOP-CMA-ES with a bounded maximum population size on the noiseless BBOB testbed". In: *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. ACM. 2013, pp. 1169–1176 (cit. on pp. 23, 31, 37, 45, 133).

[LSS12a] Ilya Loshchilov, Marc Schoenauer, and Michele Sebag. "Alternative Restart Strategies for CMA-ES". In: *PPSN XII*. LNCS 7491, Springer Verlag, 2012, pp. 296–305 (cit. on pp. 153, 161).

[LSS12b] Ilya Loshchilov, Marc Schoenauer, and Michele Sebag. "Self-Adaptive Surrogate-Assisted Covariance Matrix Adaptation Evolution Strategy". In: *GECCO'12*. ACM. 2012, pp. 321–328 (cit. on p. 177).

[LSS13] Ilya Loshchilov, Marc Schoenauer, and Michèle Sebag. "Intensive Surrogate Model Exploitation in Self-Adaptive Surrogate-Assisted CMA-ES (SAACM-ES)". In: *GECCO'13*. ACM. 2013, pp. 439–446 (cit. on pp. 87, 177).

[Luc59] Duncan R Luce. "Individual Choice Behavior". In: (1959) (cit. on p. 180).

[LW06] Monte Lunacek and Darrell Whitley. "The dispersion metric and the CMA evolution strategy". In: *Proc. 8th GECCO*. ACM. 2006, pp. 477–484 (cit. on pp. 6, 68, 71, 72, 77, 88, 97, 106).

[LWB07] Ping Li, Qiang Wu, and Christopher J Burges. "Mcrank: Learning to rank using multiple classification and gradient boosting". In: *Advances in neural information processing systems*. 2007, pp. 897–904 (cit. on p. 178).

[M+15] Mario Muñoz, Michael Kirley, Saman K Halgamuge, et al. "Exploratory landscape analysis of continuous space optimization problems using information content". In: *Evolutionary Computation, IEEE Transactions on* 19.1 (2015), pp. 74–87 (cit. on pp. 6, 68, 77, 78, 82, 83, 86, 88, 97, 106).

[Mal57] Colin L Mallows. "Non-null ranking models. I". In: *Biometrika* 44.1/2 (1957), pp. 114–130 (cit. on p. 180).

[Mas+14] Franco Mascia et al. “An analysis of parameter adaptation in reactive tabu search”. In: *International Transactions in Operational Research* 21.1 (2014), pp. 127–152 (cit. on pp. 157, 163).

[Mat63] Georges Matheron. “Principles of geostatistics”. In: *Economic geology* 58.8 (1963), pp. 1246–1266 (cit. on pp. 87, 175).

[McC80] Peter McCullagh. “Regression models for ordinal data”. In: *Journal of the royal statistical society. Series B (Methodological)* (1980), pp. 109–142 (cit. on pp. 112, 160, 178).

[Mer+11] Olaf Mersmann et al. “Exploratory landscape analysis”. In: *Proc. 13th GECCO*. ACM. 2011, pp. 829–836 (cit. on pp. 6, 68, 70, 71, 73–75, 77, 78, 82, 83, 86, 88, 97, 106).

[MF00] Peter Merz and Bernd Freisleben. “Fitness landscape analysis and memetic algorithms for the quadratic assignment problem”. In: *IEEE transactions on evolutionary computation* 4.4 (2000), pp. 337–352 (cit. on p. 68).

[MF98] Peter Merz and Bernd Freisleben. “Memetic algorithms and the fitness landscape of the graph bi-partitioning problem”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 1998, pp. 765–774 (cit. on pp. 56, 64).

[MG12] Rachael Morgan and Marcus Gallagher. “Length scale for characterising continuous optimization problems”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2012, pp. 407–416 (cit. on pp. 6, 71, 77).

[MH08] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605 (cit. on p. 85).

[MHR99] Paul Marrow, Martlesham Heath, and Ipswich Ip Re. “Evolvability: Evolution, computation, biology”. In: *Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program (GECCO-99 Workshop on Evolvability)*. 1999, pp. 30–33 (cit. on p. 66).

[Min93] Steven Minton. “An analytic learning system for specializing heuristics”. In: *IJCAI*. Vol. 93. 1993, pp. 922–929 (cit. on p. 39).

[Min96] Steven Minton. “Automatically configuring constraint satisfaction programs: A case study”. In: *Constraints* 1.1-2 (1996), pp. 7–43 (cit. on p. 39).

[MJ91] Zbigniew Michalewicz and Cezary Z Janikow. "Genetic algorithms for numerical optimization". In: *Statistics and Computing* 1.2 (1991), pp. 75–91 (cit. on p. 15).

[MKH12] Mario A Muñoz, Michael Kirley, and Saman K Halgamuge. "A meta-learning prediction model of algorithm performance for continuous optimization problems". In: *PPSN XII*. Springer, 2012, pp. 226–235 (cit. on pp. 6, 53, 75, 77, 100, 102, 103, 107, 132, 134).

[MM93] Oded Maron and Andrew W Moore. "Hoeffding races: Accelerating model selection search for classification and function approximation". In: *Robotics Institute* (1993), p. 263 (cit. on p. 44).

[MM97] Oded Maron and Andrew W Moore. "The racing algorithm: Model selection for lazy learners". In: *Lazy learning*. Springer, 1997, pp. 193–225 (cit. on p. 44).

[MP14] Luigi Malagò and Giovanni Pistone. "Optimization via Information Geometry". In: *Topics in Statistical Simulation*. Springer, 2014, pp. 343–351 (cit. on p. 162).

[MS05] Jay D Martin and Timothy W Simpson. "Use of kriging models to approximate deterministic computer models". In: *AIAA journal* 43.4 (2005), pp. 853–863 (cit. on pp. 87, 175).

[MS11] Christian L Müller and Ivo F Sbalzarini. "Global characterization of the CEC 2005 fitness landscapes using fitness-distance analysis". In: *European Conference on the Applications of Evolutionary Computation*. Springer. 2011, pp. 294–303 (cit. on p. 77).

[MS13] M Mısır and M Sebag. *Algorithm Selection as a Collaborative Filtering Problem*. Tech. rep. INRIA-Saclay, 2013. URL: http://hal.inria.fr/hal-00922840 (cit. on p. 103).

[MSK97] David McAllester, Bart Selman, and Henry Kautz. "Evidence for invariants in local search". In: *AAAI/IAAI*. 1997, pp. 321–326 (cit. on p. 51).

[MT00] Julian F Miller and Peter Thomson. "Cartesian genetic programming". In: *European Conference on Genetic Programming*. Springer. 2000, pp. 121–132 (cit. on p. 16).

[MTZ78] J Mockus, V Tiesis, and A Zilinskas. "Toward Global Optimization, volume 2, chapter Bayesian Methods for Seeking the Extremum". In: (1978) (cit. on p. 42).

[Müh97] Heinz Mühlenbein. "The Equation for Response to Selection and Its Use for Prediction". In: *Evolutionary Computation* 5.3 (1997), pp. 303–346 (cit. on p. 18).

[MWS91] Bernard Manderick, Mark de Weger, and Piet Spiessens. "The genetic algorithm and the structure of the fitness landscape". In: *Proceedings of the fourth international conference on genetic algorithms*. 1991, pp. 143–150 (cit. on p. 64).

[N+99] Bart Naudts, Alain Verschoren, et al. "Epistasis and deceptivity". In: *Simon Stevin-Bulletin of the Belgian Mathematical Society* 6 (1999), pp. 147–154 (cit. on p. 60).

[NE07] Volker Nannen and Agoston E Eiben. "Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters." In: *IJCAI*. Vol. 7. 2007, pp. 6–12 (cit. on pp. 7, 41).

[Nei72] Masatoshi Nei. "Genetic distance between populations". In: *American naturalist* (1972), pp. 283–292 (cit. on p. 65).

[NM65] John A Nelder and Roger Mead. "A simplex method for function minimization". In: *The computer journal* 7.4 (1965), pp. 308–313 (cit. on p. 169).

[Nud+04] Eugene Nudelman et al. "Understanding random SAT: Beyond the clauses-to-variables ratio". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2004, pp. 438–452 (cit. on pp. 5, 51).

[OGH94] Andreas Ostermeier, Andreas Gawelczyk, and Nikolaus Hansen. "A derandomized approach to self-adaptation of evolution strategies". In: *Evolutionary Computation* 2.4 (1994), pp. 369–380 (cit. on p. 21).

[Oll+11] Yann Ollivier et al. "Information-geometric optimization algorithms: A unifying picture via invariance principles". In: *arXiv preprint arXiv:1106.3708* (2011) (cit. on p. 162).

[Omi+14] Mohammad Nabi Omidvar et al. "Cooperative co-evolution with differential grouping for large scale optimization". In: *IEEE Transactions on Evolutionary Computation* 18.3 (2014), pp. 378–393 (cit. on p. 58).

[PA12] Erik Pitzer and Michael Affenzeller. "A comprehensive survey on fitness landscape analysis". In: *Recent Advances in Intelligent Engineering Systems*. Springer, 2012, pp. 161–191 (cit. on p. 68).

[Pad12] Nikhil Padhye. "Evolutionary Approaches for Real World Applications in 21st Century". In: *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*. GECCO '12. Philadelphia, Pennsylvania, USA: ACM, 2012, pp. 43–48. ISBN: 978-1-4503-1178-6. DOI: 10.1145/2330784.2330792. URL: http://doi.acm.org/10.1145/2330784.2330792 (cit. on p. 3).

[Ped+11] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 111, 112).

[Ped+12] Fabian Pedregosa et al. “Learning to rank from medical imaging data”. In: *International Workshop on Machine Learning in Medical Imaging*. Springer. 2012, pp. 234–241 (cit. on p. 177).

[PK01] Donald J Patterson and Henry Kautz. “Auto-walksat: a self-tuning implementation of walksat”. In: *Electronic Notes in Discrete Mathematics* 9 (2001), pp. 360–368 (cit. on p. 51).

[Pla75] Robin L Plackett. “The analysis of permutations”. In: *Applied Statistics* (1975), pp. 193–202 (cit. on p. 180).

[Pri97] Kenneth V Price. “Differential Evolution vs. the Functions of the 2nd ICEO”. In: *Evolutionary Computation, 1997.,IEEE International conference on Ev*. IEEE. 1997, pp. 153–157 (cit. on pp. 6, 29).

[PSL06] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006 (cit. on p. 29).

[QHS09] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. “Differential evolution algorithm with strategy adaptation for global numerical optimization”. In: *IEEE transactions on Evolutionary Computation* 13.2 (2009), pp. 398–417 (cit. on p. 150).

[Qin+06] Tao Qin et al. *Learning to search web pages with query-level loss functions*. Tech. rep. Citeseer, 2006 (cit. on p. 178).

[Qin+07] Tao Qin et al. “Ranking with multiple hyperplanes”. In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2007, pp. 279–286 (cit. on p. 178).

[Qin+08] Tao Qin et al. “Query-level loss functions for information retrieval”. In: *Information Processing & Management* 44.2 (2008), pp. 838–855 (cit. on p. 178).

[QL07] C Quoc and Viet Le. “Learning to rank with nonsmooth cost functions”. In: *Proceedings of the Advances in Neural Information Processing Systems* 19 (2007), pp. 193–200 (cit. on p. 178).

[Ras04] Carl Edward Rasmussen. “Gaussian processes in machine learning”. In: *Advanced lectures on machine learning*. Springer, 2004, pp. 63–71 (cit. on p. 176).

[REA96] Helge Rosé, Werner Ebeling, and Torsten Asselmeyer. "The density of states—a measure of the difficulty of optimisation problems". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 1996, pp. 208–217 (cit. on p. 70).

[Rec65] Ingo Rechenberg. "Cybernetic solution path of an experimental problem". In: (1965) (cit. on p. 14).

[Rec73] Ingo Rechenberg. "Evolution Strategy: Optimization of Technical systems by means of biological evolution". In: *Fromman-Holzboog, Stuttgart* 104 (1973) (cit. on pp. 15, 16, 19, 20).

[Rec78] Ingo Rechenberg. "Evolutionsstrategien". In: *Simulationsmethoden in der Medizin und Biologie*. Springer, 1978, pp. 83–114 (cit. on p. 20).

[RG27] Lewis F. Richardson and J. Arthur Gaunt. "The Deferred Approach to the Limit. Part I. Single Lattice. Part II. Interpenetrating Lattices". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 226.636-646 (1927), pp. 299–361. ISSN: 0264-3952. DOI: 10.1098/rsta.1927.0008. eprint: http://rsta.royalsocietypublishing.org/content/226/636-646/299.full.pdf. URL: http://rsta.royalsocietypublishing.org/content/226/636-646/299 (cit. on p. 74).

[Ric57] Bellman Richard. "Dynamic programming". In: *Princeton University Press* 89 (1957), p. 92 (cit. on p. 58).

[RK07] Enda Ridge and Daniel Kudenko. "Tuning the performance of the MMAS heuristic". In: *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*. Springer, 2007, pp. 46–60 (cit. on p. 43).

[Roc97] Sophie Rochet. "Epistasis in genetic algorithms revisited". In: *Information Sciences* 102.1 (1997), pp. 133–155 (cit. on p. 60).

[RR03a] Colin R.. Reeves and Jonathan E.. Rowe. *Genetic Algorithms: Principles and Perspectives*. Kluwer Academic Publishers, 2003 (cit. on p. 56).

[RR03b] CR Reeves and JE Rowe. "Genetic algorithms: principles and perspectives—a guide to GA theory. Operaations Research". In: *Computer Science Interfaces Series. Kluwer Academic Publishers, Boston, MA, USA* (2003) (cit. on p. 61).

[RS01] Christian M Reidys and Peter F Stadler. "Neutrality in fitness landscapes". In: *Applied Mathematics and Computation* 117.2 (2001), pp. 321–350 (cit. on pp. 63, 64, 68).

[RW95] Colin R Reeves and Christine C Wright. “Epistasis in Genetic Algorithms: An Experimental Design Perspective.” In: *ICGA*. 1995, pp. 217–224 (cit. on p. 60).

[Sac+89] Jerome Sacks et al. “Design and analysis of computer experiments”. In: *Statistical science* (1989), pp. 409–423 (cit. on p. 42).

[Sal96] Ralf Salomon. “Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms”. In: *BioSystems* 39.3 (1996), pp. 263–278 (cit. on p. 62).

[Sat59a] FE Satterthwaite. “Random balance experimentation”. In: *Technometrics* 1.2 (1959), pp. 111–137 (cit. on p. 13).

[Sat59b] FE Satterthwaite. “REVOP or random evolutionary operation”. In: *Statistical Engineering Institute, Boston Univ. Rept. 10/10* 59 (1959) (cit. on p. 13).

[Sch65] Hans-Paul Schwefel. “Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik”. In: *Master’s thesis, Technical University of Berlin* (1965) (cit. on pp. 14, 15).

[Sch68] HP Schwefel. *Projekt MHD-Staustrahlrohr: Experimentelle Optimierung einer Zweiphasendüse, Teil I*. Tech. rep. Technischer Bericht 11.034/68, 35, AEG Forschungsinstitut, Berlin, Germany, 1968 (cit. on p. 19).

[Sch81] Hans-Paul Schwefel. *Numerical optimization of computer models*. John Wiley & Sons, Inc., 1981 (cit. on pp. 15, 21).

[SD08] Krzysztof Socha and Marco Dorigo. “Ant colony optimization for continuous domains”. In: *European journal of operational research* 185.3 (2008), pp. 1155–1173 (cit. on p. 18).

[SD98] Michèle Sebag and Antoine Ducoulombier. “Extending Population-Based Incremental Learning to Continuous Search Spaces”. In: *Proc. Parallel Problem Solving from Nature - PPSN V*. Ed. by A. E. Eiben et al. Vol. 1498. Lecture Notes in Computer Science. Springer, 1998, pp. 418–427 (cit. on p. 18).

[SE10a] Selmar K Smit and AE Eiben. “Parameter tuning of evolutionary algorithms: Generalist vs. specialist”. In: *European Conference on the Applications of Evolutionary Computation*. Springer. 2010, pp. 542–551 (cit. on p. 45).

[SE10b] S.K. Smit and A.E. Eiben. “Beating the ”World champion” Evolutionary Algorithm via REVAC Tuning”. In: *Proc. IEEE Congress on Evolutionary Computation*. July 2010, pp. 1–8. DOI: 10.1109/CEC.2010.5586026 (cit. on pp. 22, 31, 37, 41, 133).

[SE11] S Smit and AE Eiben. "Multi-problem parameter tuning using bonesa". In: *Artificial Evolution*. 2011, pp. 222–233 (cit. on pp. 42, 46).

[SES10] Selmar K Smit, A Endre Eiben, and Zoltán Szlávik. "An MOEA-based Method to Tune EA Parameters on Multiple Objective Functions." In: *IJCCI (ICEC)*. 2010, pp. 261–268 (cit. on pp. 42, 46).

[SH99] Peter F Stadler and Robert Happel. "Random field models for fitness landscapes". In: *Journal of Mathematical Biology* 38.5 (1999), pp. 435–478 (cit. on pp. 55, 59, 62).

[Sha+98] John Shawe-Taylor et al. "Structural risk minimization over data-dependent hierarchies". In: *IEEE transactions on Information Theory* 44.5 (1998), pp. 1926–1940 (cit. on p. 176).

[Sha01] Claude Elwood Shannon. "A mathematical theory of communication". In: *ACM SIGMOBILE Mobile Computing and Communications Review* 5.1 (2001), pp. 3–55 (cit. on p. 69).

[SHH62] WGRFR Spendley, George R Hext, and Francis R Himsworth. "Sequential application of simplex designs in optimisation and evolutionary operation". In: *Technometrics* 4.4 (1962), pp. 441–461 (cit. on p. 13).

[Shi87] K Shittowski. "More test examples for nonlinear programming codes". In: *More test examples for nonlinear programming codes* (1987) (cit. on pp. 100, 104).

[SHO02] Tom Smith, Phil Husbands, and Michael O'Shea. "Fitness landscapes and evolvability". In: *Evolutionary computation* 10.1 (2002), pp. 1–34 (cit. on pp. 64, 66, 67).

[Sim+01] Timothy W Simpson et al. "Kriging models for global approximation in simulation-based multidisciplinary design optimization". In: *AIAA journal* 39.12 (2001), pp. 2233–2241 (cit. on pp. 87, 175).

[Sin01] Amit Singhal. "Modern information retrieval: A brief overview". In: *IEEE Data Eng. Bull.* 24.4 (2001), pp. 35–43 (cit. on p. 177).

[SL02] Amnon Shashua and Anat Levin. "Ranking with large margin principle: Two approaches". In: *Advances in neural information processing systems*. 2002, pp. 937–944 (cit. on p. 178).

[SM05] Biplav Srivastava and Anupam Mediratta. "Domain-dependent parameter selection of search-based algorithms compatible with user performance criteria". In: *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*. Vol. 20. 3. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. 2005, p. 1386 (cit. on p. 43).

[SM07] Dong-Il Seo and Byung-Ro Moon. "An information-theoretic analysis on the interactions of variables in combinatorial optimization problems". In: *Evolutionary computation* 15.2 (2007), pp. 169–198 (cit. on p. 73).

[SM86] Gerard Salton and Michael J McGill. "Introduction to modern information retrieval". In: (1986) (cit. on p. 177).

[Smo+96] Alex J Smola et al. "Regression estimation with support vector learning machines". PhD thesis. Master's thesis, Technische Universität München, 1996 (cit. on p. 176).

[Sör13] Kenneth Sörensen. "Metaheuristics–the metaphor exposed". In: *International Transactions in Operational Research* 22.1 (2013), pp. 3–18 (cit. on p. 13).

[SP97] Rainer Storn and Kenneth Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". In: *Journal of global optimization* 11.4 (1997), pp. 341–359 (cit. on pp. 17, 25, 27, 28, 30, 108, 109, 130, 151, 169).

[SR96] Patrick D Surry and Nicholas J Radcliffe. "Real Representations." In: *FOGA*. 1996, pp. 343–363 (cit. on p. 15).

[SS04] Alex J Smola and Bernhard Schölkopf. "A tutorial on support vector regression". In: *Statistics and computing* 14.3 (2004), pp. 199–222 (cit. on p. 176).

[SS98] William W Cohen Robert E Schapire and Yoram Singer. "Learning to order things". In: *Advances in Neural Information Processing Systems* 10 (1998), p. 451 (cit. on p. 178).

[Sta02a] Peter F Stadler. "Fitness landscapes". In: *Biological evolution and statistical physics*. Springer, 2002, pp. 183–204 (cit. on p. 63).

[Sta02b] PF Stadler. *Biological Evolution and Statistical Physics*. 2002 (cit. on p. 62).

[Sug+05] Ponnuthurai N Suganthan et al. "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization". In: *KanGAL report* 2005005 (2005), p. 2005 (cit. on pp. 4, 100).

[SWK87] MF Shlesinger, BJ West, and Joseph Klafter. "Lévy dynamics of enhanced diffusion: Application to turbulence". In: *Physical Review Letters* 58.11 (1987), p. 1100 (cit. on p. 71).

[SWN03] TJ Santner, BJ Williams, and WI Notz. "The Design and Analysis of Computer Experiments Springer-Verlag". In: *New York. 283pp* (2003) (cit. on p. 42).

[Tal09] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. Vol. 74. John Wiley & Sons, 2009 (cit. on p. 13).

[Tay+08] Michael Taylor et al. "Softrank: optimizing non-smooth rank metrics". In: *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM. 2008, pp. 77–86 (cit. on p. 178).

[Tey10] Fabien Teytaud. "A new selection ratio for large population sizes". In: *EvoApplications*. Ed. by Cecilia Di Chio et al. LNCS 5024, Springer Verlag, 2010, pp. 452–460 (cit. on p. 22).

[TF15] Ryoji Tanabe and Alex Fukunaga. "Tuning differential evolution for cheap, medium, and expensive computational budgets". In: *2015 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2015, pp. 2018–2025 (cit. on p. 44).

[Tho+13] Chris Thornton et al. "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2013, pp. 847–855 (cit. on p. 44).

[Tie] Kevin Tierney. "GGA: A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms". In: () (cit. on p. 41).

[TPC08] Jorge Tavares, Francisco B Pereira, and Ernesto Costa. "Multidimensional knapsack problem: A fitness landscape analysis". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38.3 (2008), pp. 604–616 (cit. on p. 68).

[TR04] Ashutosh Tiwari and Rajkumar Roy. "Challenges in real world optimisation using evolutionary computing". In: (2004) (cit. on p. 3).

[TRV99] Hugo Terashima-Marín, Peter Ross, and Manuel Valenzuela-Rendón. "Evolution of constraint satisfaction strategies in examination timetabling". In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*. Morgan Kaufmann Publishers Inc. 1999, pp. 635–642 (cit. on p. 40).

[TS07] Bryan A Tolson and Christine A Shoemaker. "Dynamically dimensioned search algorithm for computationally efficient watershed model calibration". In: *Water Resources Research* 43.1 (2007) (cit. on p. 41).

[Tur02] Peter D Turney. "Increasing evolvability considered as a large-scale trend in evolution". In: *arXiv preprint cs/0212042* (2002) (cit. on p. 66).

[V+97] Vladimir Vapnik, Steven E Golowich, Alex Smola, et al. "Support vector method for function approximation, regression estimation, and signal processing". In: *Advances in neural information processing systems* (1997), pp. 281–287 (cit. on pp. 87, 176).

[Van+04] Leonardo Vanneschi et al. "Fitness clouds and problem hardness in genetic programming". In: *Genetic and Evolutionary Computation Conference*. Springer. 2004, pp. 690–701 (cit. on pp. 75, 76).

[VCC03] Sébastien Verel, Philippe Collard, and Manuel Clergue. "Where are bottlenecks in nk fitness landscapes?" In: *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*. Vol. 1. IEEE. 2003, pp. 273–280 (cit. on pp. 75, 76).

[VFM00] Vesselin K Vassilev, Terence C Fogarty, and Julian F Miller. "Information characteristics and the structure of landscapes". In: *Evolutionary computation* 8.1 (2000), pp. 31–60 (cit. on p. 69).

[VV95] Vapnik N Vladimir and V Vapnik. *The nature of statistical learning theory*. 1995 (cit. on pp. 87, 176).

[VV98] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*. Vol. 1. Wiley New York, 1998 (cit. on p. 176).

[WA96] Gunter P Wagner and Lee Altenberg. "Perspective: complex adaptations and the evolution of evolvability". In: *Evolution* (1996), pp. 967–976 (cit. on p. 66).

[Wat86] GS Watson. "Generalized Linear Models (P. Mccullagh and JA Nelder)". In: *SIAM Review* 28.1 (1986), pp. 128–130 (cit. on pp. 112, 160, 178).

[WCT12] Thomas Weise, Raymond Chiong, and Ke Tang. "Evolutionary optimization: Pitfalls and booby traps". In: *Journal of Computer Science and Technology* 27.5 (2012), pp. 907–936 (cit. on p. 57).

[WCZ11] Yong Wang, Zixing Cai, and Qingfu Zhang. "Differential evolution with composite trial vector generation strategies and control parameters". In: *IEEE Transactions on Evolutionary Computation* 15.1 (2011), pp. 55–66 (cit. on p. 150).

[Wei90] Edward Weinberger. "Correlated and uncorrelated fitness landscapes and how to tell the difference". In: *Biological cybernetics* 63.5 (1990), pp. 325–336 (cit. on pp. 64, 66).

[Wei91] Edward D Weinberger. "Local properties of Kauffman's N-k model: A tunably rugged energy landscape". In: *Physical Review A* 44.10 (1991), p. 6399 (cit. on p. 63).

[Whi91] Ld Whitley. “Fundamental Principles of Deception in Genetic Search”. In: *Foundations of Genetic Algorithms* (1991), pp. 221–241 (cit. on p. 65).

[WKG07] Daniel N Wilke, Schalk Kok, and Albert A Groenwold. “Comparison of linear and classical velocity update rules in particle swarm optimization: notes on diversity”. In: *International Journal for Numerical Methods in Engineering* 70.8 (2007), pp. 962–984 (cit. on p. 19).

[WM97] D.H. Wolpert and W.G. Macready. “No free lunch theorems for optimization”. In: *Evolutionary Computation, IEEE Transactions on* 1.1 (1997), pp. 67–82. ISSN: 1089-778X. DOI: 10.1109/4235.585893 (cit. on p. 4).

[Wri+91] Alden H Wright et al. “Genetic algorithms for real parameter optimization”. In: *Foundations of genetic algorithms* 1 (1991), pp. 205–218 (cit. on p. 15).

[Wri32] Sewall Wright. *The roles of mutation, inbreeding, crossbreeding, and selection in evolution*. Vol. 1. na, 1932 (cit. on p. 55).

[WSN00] Brian J Williams, Thomas J Santner, and William I Notz. “Sequential design of computer experiments to minimize integrated response functions”. In: *Statistica Sinica* (2000), pp. 1133–1152 (cit. on pp. 43, 46).

[WW03] Lei Wang and Qi-di Wu. “Ant system algorithm in continuous space optimization”. In: *Control and Decision* 18.1 (2003), pp. 45–48 (cit. on p. 18).

[XHL07] Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. “Hierarchical hardness models for SAT”. In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2007, pp. 696–711 (cit. on p. 51).

[XHL10] Lin Xu, Holger Hoos, and Kevin Leyton-Brown. “Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection.” In: *AAAI*. Vol. 10. 2010, pp. 210–216 (cit. on pp. 5, 52, 145, 163).

[Xia+08] Fen Xia et al. “Listwise approach to learning to rank: theory and algorithm”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1192–1199 (cit. on pp. 6, 178).

[Xu+07] Lin Xu et al. “SATzilla-07: the design and analysis of an algorithm portfolio for SAT”. In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2007, pp. 712–727 (cit. on pp. 51, 100, 103).

[Xu+08] Lin Xu et al. “SATzilla: portfolio-based algorithm selection for SAT”. In: *Journal of Artificial Intelligence Research* (2008), pp. 565–606 (cit. on pp. 5, 48, 51, 52).

[Xu+11] Lin Xu et al. “Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming”. In: *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*. 2011, pp. 16–30 (cit. on pp. 5, 52, 145, 163).

[Yan+03] Yong Yang et al. “Ant colony algorithm for continuous space optimization”. In: *Control and Decision* 18.5 (2003), pp. 573–576 (cit. on p. 18).

[Yue+07] Yisong Yue et al. “A support vector method for optimizing average precision”. In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2007, pp. 271–278 (cit. on p. 178).

[Zah02] Daniela Zaharie. “Critical values for the control parameters of differential evolution algorithms”. In: *Proceedings of MENDEL*. Vol. 2. 2002, p. 6267 (cit. on p. 29).

[Zah03] Daniela Zaharie. “Control of population diversity and adaptation in differential evolution algorithms”. In: *Proc. of MENDEL*. Vol. 9. 2003, pp. 41–46 (cit. on p. 29).

[Zah07] Daniela Zaharie. “A comparative analysis of crossover variants in differential evolution”. In: *Proceedings of IMCSIT* 2007 (2007), pp. 171–181 (cit. on p. 28).

[ZD02] Mark Zlochin and Marco Dorigo. “Model-based search for combinatorial optimization: A comparative study”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2002, pp. 651–661 (cit. on p. 18).

[Zha+09] Zhi-Hui Zhan et al. “Adaptive particle swarm optimization”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.6 (2009), pp. 1362–1381 (cit. on pp. 19, 150, 151).

[Zlo+04] Mark Zlochin et al. “Model-based search for combinatorial optimization: A critical survey”. In: *Annals of Operations Research* 131.1-4 (2004), pp. 373–395 (cit. on p. 18).

[ZS09] Jingqiao Zhang and Arthur C Sanderson. “JADE: adaptive differential evolution with optional external archive”. In: *IEEE transactions on evolutionary computation* 13.5 (2009), pp. 945–958 (cit. on pp. 17, 29, 149, 150).

université PARIS-SACLAY | ÉCOLE DOCTORALE Sciences et technologies de l'information et de la communication (STIC)

Titre : Paramétrage automatisé d'algorithme par instance pour l'optimisation boîte noire
Mot clés : intelligence artificielle, optimisation, apprentissage statistique

Résumé : Cette thèse porte sur la configuration automatisée des algorithmes qui vise à trouver le meilleur paramétrage à un problème donné ou une catégorie de problèmes.
Le problème de configuration de l'algorithme revient donc à un problème de méta-optimisation dans l'espace des paramètres, dont le méta-objectif est la mesure de performance de l'algorithme donné avec une configuration de paramètres donnée.
Des approches plus récentes reposent sur une description des problèmes et ont pour but d'apprendre la relation entre l'espace des caractéristiques des problèmes et l'espace des configurations de l'algorithme à paramétrer.
Cette thèse de doctorat porte sur le CAPI (Configuration d'Algorithme Par Instance) pour résoudre des problèmes d'optimisation de boîte noire continus, où seul un budget limité d'évaluations de fonctions est disponible.
Nous étudions d'abord les algorithmes évolutionnaires pour l'optimisation continue, en mettant l'accent sur deux algorithmes que nous avons utilisés comme algorithme cible pour CAPI, DE et CMA-ES.
Ensuite, nous passons en revue l'état de l'art des approches de configuration d'algorithme, et les différentes fonctionnalités qui ont été proposées dans la littérature pour décrire les problèmes d'optimisation de boîte noire continue.
Nous introduisons ensuite une méthodologie générale pour étudier empiriquement le CAPI pour le domaine continu, de sorte que toutes les composantes du CAPI puissent être explorées dans des conditions réelles.
À cette fin, nous introduisons également un nouveau banc d'essai de boîte noire continue, distinct du célèbre benchmark BBOB, qui est composé de plusieurs fonctions de test multidimensionnelles avec différentes propriétés problématiques, issues de la littérature.
La méthodologie proposée est finalement appliquée à deux AEs. LA méthodologie est ainsi, validé empiriquement sur le nouveau banc d'essai d'optimisation boîte noire pour des dimensions allant jusqu'à 100.

Title: Per Instance Algorithm Configuration for Continuous Black Box Optimization
Keywords: Artificial intelligence, Optimization, Machine Learning

Abstract: This PhD thesis focuses on the automated algorithm configuration that aims at finding the 'best' parameter setting for a given problem or a class of problem.
The Algorithm Configuration problem thus amounts to a meta-optimization problem in the space of parameters, whose meta-objective is the performance measure of the given algorithm at hand with a given parameter configuration. However, in the continuous domain, such method can only be empirically assessed at the cost of running the algorithm on some problem instances.
More recent approaches rely on a description of problems in some features space, and try to learn a mapping from this feature space onto the space of parameter configurations of the algorithm at hand. Along these lines, this PhD thesis focuses on the Per Instance Algorithm Configuration (PIAC) for solving continuous black box optimization problems, where only a limited budget of function evaluations is available.
We first survey Evolutionary Algorithms for continuous optimization, with a focus on two algorithms that we have used as target algorithm for PIAC, DE and CMA-ES.
Next, we review the state of the art of Algorithm Configuration approaches, and the different features that have been proposed in the literature to describe continuous black box optimization problems.
We then introduce a general methodology to empirically study PIAC for the continuous domain, so that all the components of PIAC can be explored in real-world conditions.
To this end, we also introduce a new continuous black box test bench, distinct from the famous BBOB benchmark, that is composed of a several multi-dimensional test functions with different problem properties, gathered from the literature.
The methodology is finally applied to two EAs.
First we use Differential Evolution as target algorithm, and explore all the components of PIAC, such that we empirically assess the best. Second, based on the results on DE, we empirically investigate PIAC with Covariance Matrix Adaptation Evolution Strategy (CMA-ES) as target algorithm. Both use cases empirically validate the proposed methodology on the new black box testbench for dimensions up to 100.

Université Paris-Saclay
Espace Technologique/ Immeuble Discovery
Route de l'Orme aux Merisiers RD 128/91190 Saint-Aubin,France