Universiteit Leiden
Computer Science

A New Approach Towards the Combined Algorithm Selection and Hyper-parameter Optimization Problem

Name: Xin Guo

Date: 28/01/2019

1st supervisor: Dr. Bas van Stein
2nd supervisor: Prof. Dr. Thomas Bäck

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Machine learning algorithms often have many hyper-parameters that can be tuned to improve empirical performance. However, manually exploring the complex hyper-parameter search spaces is tedious and cannot guarantee to find satisfactory outcomes. Recently, Bayesian optimization techniques for solving hyper-parameters optimization problem have shown substantial improvements, while also exposing some limitations. It is often the case that some hyper-parameters of a learning algorithm have a dependency on the values of other hyper-parameters or some conditions. These are called conditional hyper-parameters. Furthermore, this case can be expanded to include the choice of algorithm, resulting in the so-called combined algorithm selection and hyper-parameter optimization problem. In this thesis, we propose three strategies to extend Bayesian optimization to deal with such a hierarchical search space with conditional inputs, based on a state-of-the-art algorithm MiP-EGO [1]. We first evaluate MiP-EGO without conditional inputs on benchmark problems, and then empirically compare our proposed strategies with other methods on an algorithm configuration scenario. The results show MiP-EGO and its new variants can compete with other state-of-the-art methods, and they are promising for configuring and selecting learning algorithms in practice.

Acknowledgements

First and foremost, I would like to thank my supervisors, Dr. Bas van Stein and Prof. Dr. Thomas Bäck, for the patient guidance, encouragement and advice they have provided throughout my time doing this interesting project. Furthermore, I would like to thank my boyfriend and all my friends in Leiden for providing spiritual support and for distracting me from stress. Finally, I would like to thank my parents for their love and their financial supports during these years in Leiden. The accomplishment of the thesis would never be possible without any of you.

Contents

Acknowledgements ii

1 Introduction 1

2 Problem Definitions and Comparison 5
- 2.1 Black-box Optimization Problem . . . 6
- 2.2 Algorithm Configuration Problem . . . 7
- 2.3 Comparison between BBO and AC . . . 9
- 2.4 Combined Algorithm Selection and Hyper-parameter Optimization . . . 10

3 Related Approaches 12
- 3.1 Sequential Model-Based Methods . . . 12
- 3.2 Meta-Heuristic Methods . . . 13
- 3.3 Racing Procedures . . . 13

4 Bayesian Optimization 15
- 4.1 Initial Design . . . 16
- 4.2 Surrogate Models . . . 16
  - 4.2.1 Kriging . . . 17
  - 4.2.2 Random Forest . . . 22
- 4.3 Acquisition Function . . . 24
- 4.4 MiP-EGO . . . 25
  - 4.4.1 Handling Mixed Integer Input . . . 26
  - 4.4.2 A New Acquisition Function and Cooling Strategy . . . 26
  - 4.4.3 Parallel Execution . . . 27
- 4.5 SMAC . . . 27
  - 4.5.1 Maximizing EI and Intensification . . . 27

5 Extensions of MiP-EGO on Conditional Spaces 29

6 Experimental Evaluations 34
- 6.1 Artificial Test Problems . . . 34
  - 6.1.1 Barrier Function . . . 34
  - 6.1.2 Mixed Integer NK Landscapes . . . 36
- 6.2 Algorithm Configuration Scenario . . . 38
  - 6.2.1 Results and Discussions . . . 40

6.2.2 Visualization of Classifier Choices 41

7 Summary and Conclusions **45**

Bibliography **47**

Chapter 1

Introduction

Today, high-performance algorithms and models involve a large number of design choices and parameter settings. Design choices include alternative models, pre-processing, variable selection, value selection and so on. Many design choices have associated numerical parameters. A typical example in machine learning is Neural Networks (NNs), which have dozens of hyper-parameters (e.g. layer type, the total number of layers, the number of units in each layer) that need to be configured before training. The other examples include chemical plants optimization[2], engineering optimization [3, 4], computer simulation [5–7] and so on. The problem about parameter tuning and algorithm selection is common in science, engineering, finance, and other fields.

Challenges for designing algorithms and models involve three aspects basically. First of all, many alternative design choices are available but no one knows a prior which choice is optimal for a given problem. It is almost impossible for domain experts to test all the algorithms and it is also difficult for them to be familiar with all the specifics of existing methods. Secondly, high-performance algorithms often consist of various components but it is hard to figure out how these components and parameters interact with each other. A common practice is to treat the algorithm as a **black-box** where the internal working mechanism is unknown but inputs and outputs can be accessed. Thirdly, getting outputs or evaluating the performance of algorithms might be computationally expensive or resource intensive. This means that exhaustive enumeration or brute force search is not possible.

The traditional design approach is mainly based on trial-and-error which is guided by expertise or intuition. This simple approach generally attempts to find a solution, not the best solution and makes no attempt to generalize a solution to other problems.

To make this approach more systematical and automatic, a new concept called automatic algorithm configuration arises. Its workflow is illustrated in Figure 1.1. Note that this approach treats algorithm configuration as a black-box optimization problem. The configuration procedure executes the target algorithm, which is configured by a specific set of parameter settings, on one or more problem instances. And the configurator receives feedback (namely solution cost in Figure 1.1) about the algorithm's performance using this set of parameter settings. The configurator uses this information to decide which target algorithm runs to perform subsequently. The configurator does not have access to any internal state of the target algorithm. This leads to a clean interface and makes the configurator applicable to new target algorithms. In this thesis, we mainly discuss this configuration procedure and focus on the design of configurators.

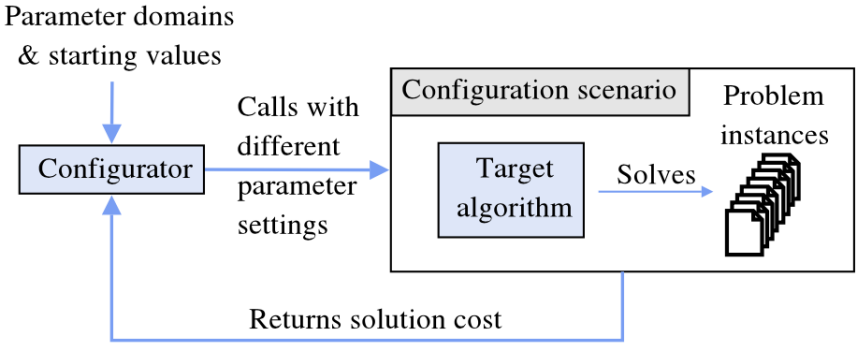


FIGURE 1.1: Visualization of automatic algorithm configuration proposed by Hutter [8].

The notion of automatic algorithm configuration can be roughly divided into two parts discussed in the following: hyper-parameter optimization and algorithm selection.

In machine learning, hyper-parameter optimization is the problem of choosing a set of optimal hyper-parameters for a learning algorithm. For example, a powerful learning algorithm, RBF-based support vector machine (SVM) [9] has two hyper-parameters that require tuning, C and gamma. From Figure 1.2, we see the response surface (AUROC) can be highly irregular for even simple learning tasks. It is non-smooth, non-convex and has many local minima. This is a general observation in the hyper-parameter optimization problems, and is one of the key reasons why we need a robust configurator.

If there were no local minima, a simple numerical optimization technique could do the trick.

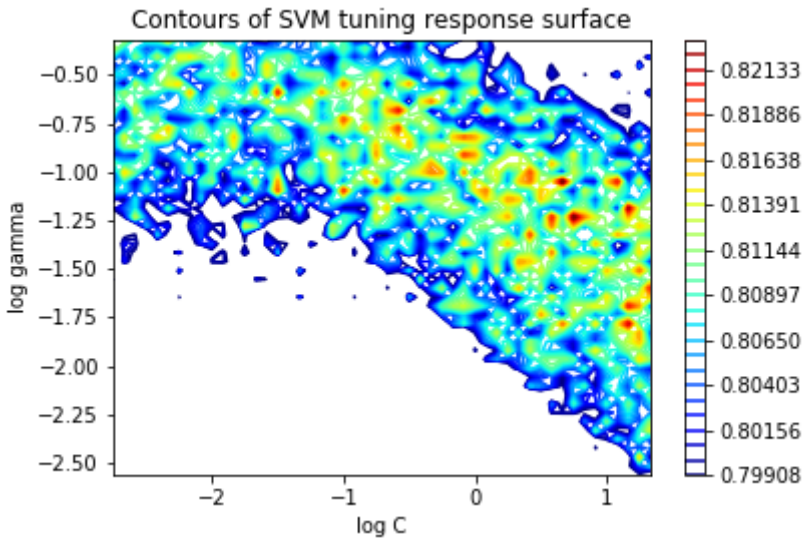


FIGURE 1.2: Response surface (AUROC) of SVM's hyper-parameters on a toy dataset.

Algorithm selection, sometimes also called per-instance algorithm selection or offline algorithm selection can be stated as follow:

Given a computational problem, a set of algorithms for solving this problem, and a specific instance that needs to be solved, determine which of the algorithms can be expected to perform best on that instance. [10]

It is motivated by the observation that an algorithm has different performances on many practical problems. That is, while one algorithm performs well on some instances, it performs poorly on others. If we can identify when to use which algorithm, the potential of algorithms can be maximized and problems can be solved effectively.

The remainder of the thesis is structured as follows: We first introduce the traditional standard black-box optimization problem and the emerging algorithm configuration problem, which emphasizes on costly objective function evaluations, randomness and complex search spaces, and discuss their relation and difference. Then we introduce a wide range of related approaches for solving the hyper-parameter optimization problems in chapter 3. In chapter 4, we introduce Bayesian optimization strategy to solve these problems, and discuss two state-of-the-art implementations of this strategy. We also extend MiP-EGO [11], in chapter 5, to solve the combined algorithm selection and

hyper-parameter optimization problem by constructing hierarchical search space properly. The experimental results in chapter 6 show that our proposed methods perform well and are not significantly worse than the other methods.

Chapter 2

Problem Definitions and Comparison

An important topic of this thesis is clarifying the standard *black-box optimization problem*, and the *algorithm configuration problem* and their relations between the two. Before we introduce these two problems formally, we first introduce some common notations shared by both and some unique notations for the algorithm configuration problem.

Let $\mathcal{A}$ denote an algorithm with parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$, where $\theta_i \in \Theta_i$ for $i = 1, \ldots, k$. The domain of possible values of θ_i is denoted as Θ_i. These domains can be finite and ordered (for continuous and integer parameters) or finite and unordered (for categorical parameters). We use $\boldsymbol{\Theta} \subseteq \Theta_1 \times \ldots \times \Theta_k$ to represent the space of all feasible parameter configurations, and $\mathcal{A}(\boldsymbol{\theta})$ to represent the instantiation of algorithm $\mathcal{A}$ with parameter configuration $\boldsymbol{\theta} \in \boldsymbol{\Theta}$.

Specifically for the algorithm configuration problem, let $\mathcal{D}$ denote a probability distribution over a space Π of problem instances, and denote an element of Π as π, *i.e.*, an individual problem instance. $\mathcal{D}$ is usually defined as the uniform distribution over Π [12].

2.1 Black-box Optimization Problem

Mathematically, the **deterministic black-box optimization (BBO)** problem is formulated as

$$\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} f(\boldsymbol{\theta})$$

where $f : \boldsymbol{\Theta} \to \mathbb{R}$ is a "black-box" function, that means no analytical or derivative information available. We can query f at arbitrary input $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ and the only information we can get about f is its function value at this queried point. In typical BBO problems, the domain is continuous, that is, $\boldsymbol{\Theta} \subseteq \mathbb{R}$. Other common names for f which appear in this thesis are *objective function, performance measure, fitness function or cost.*

In the **stochastic BBO** problem, the function f is replaced with a **stochastic process** $\{F_{\boldsymbol{\theta}} | \boldsymbol{\theta} \in \boldsymbol{\Theta}\}$, a collection of random variables indexed by $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. The goal of stochastic BBO is to find the configuration that optimizes a given statistical parameter τ of F_θ's distribution. This statistical parameter might be the expected value or median. Let $\mathbb{P}_{\{\boldsymbol{\theta}\}}$ denote the distribution of F_θ, then the stochastic BBO problem is formulated, in [8], as

$$\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \tau(\mathbb{P}_{\{\boldsymbol{\theta}\}})$$

Specifically, in algorithm configuration, $\mathbb{P}_{\{\boldsymbol{\theta}\}}$ is also called the configuration's *cost distribution*, an observed sample from that distribution is called *observed cost*, o, and the *overall cost* of a configuration is defined as $c(\boldsymbol{\theta}) := \tau(\mathbb{P}_{\{\boldsymbol{\theta}\}})$ [8]. The optimal configuration is thus defined as

$$\boldsymbol{\theta}^* \in \arg\min_{\boldsymbol{\theta}^*\in\boldsymbol{\Theta}} c(\boldsymbol{\theta})$$

Since the objective varies for different applications, the so-called overall cost could be defined as the median solution quality achieved within given computation time ,or as the expected computation time required to reach an optimal solution. In this thesis, we will consider the solution quality as the performance measure of a given target algorithm.

To intuitively illustrate notations in the stochastic BBO problem, we give an example shown in Figure 2.1. Here, the black circle denotes the observed cost o of running $\mathcal{A}(\boldsymbol{\theta})$ on instance $\pi \in \Pi$. The cost of configuration $\boldsymbol{\theta}$ is considered as a random variable $F_{\boldsymbol{\theta}}$

and $\mathbb{P}_{\{\boldsymbol{\theta}\}}$ is denoted as $F_{\boldsymbol{\theta}}$'s distribution. In this example, all of the observed cost values on configuration $\boldsymbol{\theta}$ are sampled from a normal distribution. A statistical parameter of $F_{\boldsymbol{\theta}}$ is denoted as τ. In this example, τ represents the expected value. If we connected all values of τ in configuration space in this figure, we can get a curve representing the objective function that we want to optimize, i.e. $c(\boldsymbol{\theta}) := \tau(\mathbb{P}_{\{\boldsymbol{\theta}\}})$.

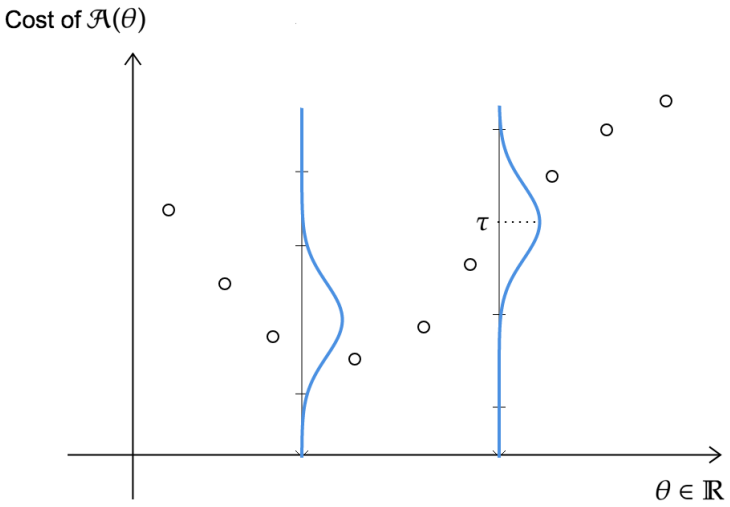


FIGURE 2.1: An example of stochastic BBO problem.

2.2 Algorithm Configuration Problem

Roughly speaking, the algorithm configuration problem (AC) can be stated as follows: given a parameterized algorithm $\mathcal{A}$, a set of problem instances Π and an overall cost metric c, find configuration $\boldsymbol{\theta}$ of $\mathcal{A}$ that minimizes c on Π. We see that AC is similar to the stochastic BBO. Both of them aim to find the value of a vector $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ that minimizes a scalar-valued function $F(\boldsymbol{\theta})$. From this angle, AC can be viewed as a special type of BBO problem.

Before we compare both of them in detail, it is necessary to introduce the formal definition of AC given in [8, 12].

Definition 2.1. (Algorithm configuration problem). An *instance* of the algorithm configuration problem consists of a 6-tuple $\langle \mathcal{A}, \boldsymbol{\Theta}, \mathcal{D}, \kappa_{max}, o, \tau \rangle$, where:

- $\mathcal{A}$ is a parameterized algorithm;
- $\boldsymbol{\Theta}$ is the parameter configuration space of $\mathcal{A}$;

- $\mathcal{D}$ is a distribution over problem instances with domain Π, usually assumed as uniform distribution;
- κ_{max} is a cutoff time (or captime), after which each run of $\mathcal{A}$ will be terminated if still running;
- o is a function that measures the observed cost of running $\mathcal{A}(\boldsymbol{\theta})$ on instance $\pi \in \Pi$ with captime $\kappa \in \mathbb{R}$; and
- τ is a statistical population parameter to be optimized.

From this definition, we can see that the algorithm configuration problem for deterministic algorithms and single instances can be seen as the deterministic BBO problem, while AC of randomized algorithms or AC on a set of instances Π would be a stochastic BBO problem. The relation between AC and BBO are summarized in Figure 2.2. Note that algorithm parameters can be discrete, so in the AC problems we have $\boldsymbol{\Theta} \neq \mathbb{R}^d$.

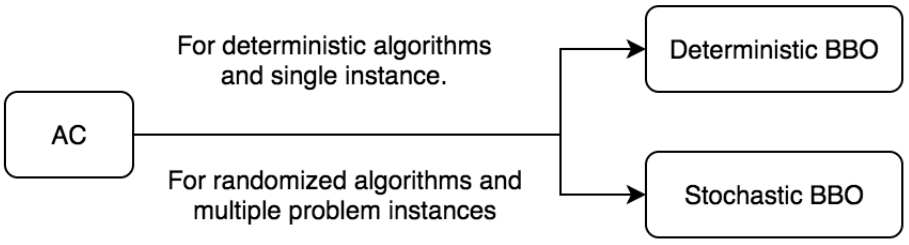


FIGURE 2.2: The relation of AC and BBO

It turns out that the cost of a configuration, $c(\boldsymbol{\theta})$, can not be optimized directly because we can not provide an analytical form for it. Instead we can only execute *a sequence of runs* $\mathbf{R}$ of the target algorithm $\mathcal{A}$ with different parameter configurations, that means we can only get empirical estimates of $c(\boldsymbol{\theta})$'s value. This derives the notion of the empirical cost estimate given in [8, 12].

Definition 2.2. (Empirical Cost Estimate). Given an algorithm configuration problem instance $\langle \mathcal{A}, \boldsymbol{\Theta}, \mathcal{D}, \kappa_{max}, \tau \rangle$, an empirical cost estimate of $c(\boldsymbol{\theta})$ based on a sequence of runs $\mathbf{R} = ((\boldsymbol{\theta}_1, \pi_1, s_1, \kappa_1, o_1), \ldots, (\boldsymbol{\theta}_n, \pi_n, s_n, \kappa_n, o_n))$ is defined as $\hat{c}(\boldsymbol{\theta}, \mathbf{R}) := \hat{\tau}(\{o_i | \boldsymbol{\theta}_i = \boldsymbol{\theta}\})$, where $\hat{\tau}$ is the sample statistic analogue to the statistical parameter τ.

Here s_i denotes the random number seed used in the run. If we denote $\mathbf{R}_{\boldsymbol{\theta}}$ as the sequence of runs with configuration $\boldsymbol{\theta}$ (i.e. all $\boldsymbol{\theta}_i$ equals $\boldsymbol{\theta}$ in these runs), we can see that

TABLE 2.1: Differences in terminology between BBO and AC.

BBO	AC
Parameter tuning	Algorithm configuration
Best configuration seen so far	The incumbent
A group of similar objective functions	Instances
A deterministic function to be optimized	A statistical parameter of an observation distribution to be optimized

the empirical estimate of $c(\boldsymbol{\theta})$ is solely based on $\mathbf{R}_{\boldsymbol{\theta}}$. This definition suggests that, on one hand, AC is exactly the stochastic BBO when configuring a randomized algorithm on a single instance; on the other hand, a fair comparison between two configurations $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ should be based on the same number of runs, i.e. $|\mathbf{R}_i| = |\mathbf{R}_j|$. With this notion in mind, when executing a sequence of runs $\mathbf{R}$, the configuration observed so far to have the lowest cost, is called the incumbent configuration, shortly the *incumbent*, $\boldsymbol{\theta}_{inc}$.

2.3 Comparison between BBO and AC

From the definition above, we can see a big difference between AC and BBO is about how the notion of runs is defined. In BBO problems, each function evaluation is assumed to take the same amount of time on a given instance. Thus the ith run in $\mathbf{R}$ can be described by three values: $(\boldsymbol{\theta}_i, s_i, o_i)$, that is, parameter configuration $\boldsymbol{\theta}_i$, the random number seed s_i used in the run and the observed cost o_i. In contrast, in the AC problem, the ith run is described by five values: $(\boldsymbol{\theta}_i, \pi_i, s_i, \kappa_i, o_i)$. Here, o_i measures the observed cost of running $\mathcal{A}(\boldsymbol{\theta}_i)$ on instance π_i with captime κ_i. This implies multiple instances with various hardness tend to be solved simultaneously in the overall configuring process. We are also free to terminate any runs after any cutoff time $\kappa \leq \kappa_{max}$, to avoid poor configuration wasting too much time on some difficult instances [8]. The different terminologies used in BBO and AC are summarized in Table 2.1

2.4 Combined Algorithm Selection and Hyper-parameter Optimization

BBO and AC are concerned with the hyper-parameters optimization of a single algorithm. However, it is also challenging to choose a suitable algorithm for a specific problem before tuning its hyper-parameters. It is infeasible to try all algorithms in practice, and also, the rankings of algorithms depend on whether their hyper-parameters are tuned properly. This dilemma is especially common when trying to select a suitable classification algorithm. Fortunately, according to the work in [13], these two problems can efficiently be tackled as a single, structured, joint optimization problem, which is formulated as the Combined Algorithm Selection and Hyper-parameters optimization (CASH) problem [13]. Given a set of classification algorithms $\mathcal{A} = \left\{A^{(1)}, A^{(2)}, \ldots, A^{(m)}\right\}$ with associated hyper-parameter spaces $\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2, \ldots, \boldsymbol{\Theta}_m$, the CASH problem is formally defined as

$$A^*_{\boldsymbol{\theta}^*} \in \underset{A^{(j)} \in \mathcal{A}, \boldsymbol{\theta} \in \boldsymbol{\Theta}_j}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(A^{(j)}_{\boldsymbol{\theta}}, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid}\right),$$

where $\mathcal{L}\left(A^{(j)}_{\boldsymbol{\theta}}, \mathcal{D}^{(i)}_{train}, \mathcal{D}^{(i)}_{valid}\right)$ is the loss (i.e. misclassification rate) achieved by $A^{(j)}$ with hyper-parameters $\boldsymbol{\theta}$ when trained on $\mathcal{D}^{(i)}_{train}$ and evaluated on $\mathcal{D}^{(i)}_{valid}$. K-fold cross-validation is used for estimating generalization performance of the classification algorithm $A^{(j)}$. The goal of the CASH problem is to find the joint algorithm and hyper-parameter setting that minimize the average loss $\mathcal{L}$ on a dataset.

To make this problem easier to solve, we reformulate it as the following, by adding an extra algorithm selection parameter $\theta_0 \in \Theta_0 = \{1, 2, \ldots, m\}$. It is written as

$$\theta^*_0, \boldsymbol{\theta}^* = \underset{A^{(j)} \in \mathcal{A}, \boldsymbol{\theta} \in \boldsymbol{\Theta}_j}{\operatorname{argmin}} \mathcal{L}\left(A^{(j)}_{\boldsymbol{\theta}}, \mathcal{D}\right).$$

Here, the loss defined previously is simplified to be $\mathcal{L}\left(A^{(j)}_{\boldsymbol{\theta}}, \mathcal{D}\right)$, denoting the cross-validation error achieved by $A^{(j)}_{\boldsymbol{\theta}}$ on a dataset $\mathcal{D}$. We combine the hyper-parameters and algorithm selector parameter all together. The resulting hybrid search space is written as $\boldsymbol{\Gamma} = \Theta_0 \times \boldsymbol{\Theta}_1 \times \cdots \times \boldsymbol{\Theta}_m$. We say that hyper-parameters $\boldsymbol{\theta}_j \in \boldsymbol{\gamma}$ of $A^{(j)}$ are active if parameter selector $\theta_0 = j$. The hierarchical structure of $\boldsymbol{\Gamma}$ is demonstrated in Figure 2.3.

In Figure 2.3, the conditionality of hybrid configuration $\boldsymbol{\gamma} = (\theta_0, \boldsymbol{\theta}_1^\intercal, \ldots, \boldsymbol{\theta}_m^\intercal)^\intercal$ is defined vertically in the graph, with subgroups separated by different values of root-level parameter. At any time, only a subset of $\boldsymbol{\gamma}$ is active, and the other hyper-parameters do not take values. In this thesis, we propose several approaches to deal with the CASH problem with conditional parameter, and the experimental results are discussed in section 6.2.

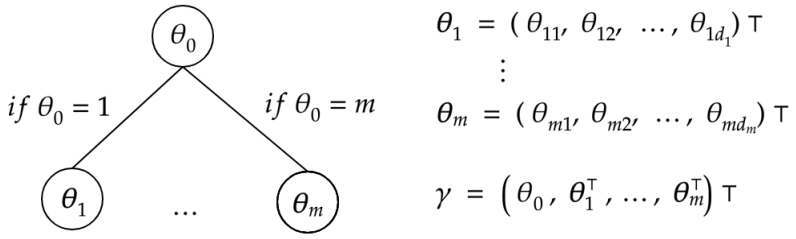


FIGURE 2.3: Generic hyper-parameter space with hierarchical structure.

Chapter 3

Related Approaches

In general, every black-box optimization method can be applied to the hyper-parameter optimization problem. Due to the non-convex nature of the problem, global optimization algorithms are usually preferred, but some local optimization is useful in order to make progress within a few function evaluations. In addition, some experimental design methods, such as grid search and random search, have been shown to be sufficiently efficient on hyper-parameter optimization problems although they are unreliable in some cases [14]. In this chapter, we first discuss sequential model-based optimization methods and then discuss meta-heuristic methods and the racing procedure (known as sequential statistical testing).

3.1 Sequential Model-Based Methods

Sequential model-based optimization, also known as Bayesian optimization, is a state-of-the-art technique for solving the expensive black-box optimization problem. It is recently widely used in tuning machine learning models for image classification [15, 16], speech recognition [17] and so on. In a nutshell, Bayesian optimization is an iterative algorithm with two key components: a probabilistic surrogate model which simulates the behaviour of the target function, and an acquisition function derived from the surrogate model to guide the location of next sample point. In each iteration, the promising sampling location is evaluated by the actual target function, and the surrogate model is re-trained on all observations of the target function made so far, then a new sampling position

is generated through maximizing the acquisition function. Compared to the original expensive target function, the acquisition function is much cheaper to be optimized.

Since searching promising candidate points is guided by a probabilistic model, Bayesian optimization often outperforms other model-free methods, such as random search and grid search, and is often able to find satisfactory result within a few function evaluation budgets. However, a major drawback of Bayesian optimization is that updating the surrogate model in each iteration can be costly, especially when the surrogate model is Gaussian processes. The time of computing a posterior prediction distribution of Gaussian processes grows cubically in the number of samples, as it needs to compute the inversion of a dense covariance matrix. Considering this limitation, Gaussian processes is replaced by other models in practice, such as deep neural networks [16] and random forest [18], and both of them show success in improving the scalability and reducing the computational cost of building the surrogate model.

3.2 Meta-Heuristic Methods

Meta-heuristic methods, such as genetic programming , evolutionary strategies, particle swarm optimization and local search, have been applied to hyper-parameter optimization for a long time and shown competitive performances [12, 19–23]. One of the best known population-based methods is the covariance matrix adaption evolutionary strategy (CMA-ES). It is a state-of-the-art variant of evolutionary strategy. New candidate solutions are derived based on a multivariate Gaussian whose mean and covariance matrix are updated in each generation according to the operation of recombination and mutation of individuals. Recent research shows that it tends to perform best for larger function evaluation budgets, while Bayesian optimization often performs best for small function evaluation budgets [24].

3.3 Racing Procedures

The racing algorithm was first proposed by Maron et al.[25] in 1997 for solving the model selection problem, after that, Birattari et al. [26, 27] improved the approach and applied it to automatically configuring meta-heuristic algorithms such as ant colony optimization

(ACO) on travelling salesman problem (TSP). The concept of racing procedure is based on a simple but effective idea: given a number of candidate solvers for a given problem, sequentially evaluate a set of candidates on one problem instance at once, and discard solvers that are shown to perform significantly worse than the current best ones. As a result, the computational overhead of evaluating solvers with different configuration will gradually bias to those promising candidates instead of wasting on bad solvers. And the final best candidate solvers will be evaluated most of times.

The racing algorithm has evolved many variants in recent years [28], such as F-Race, Sampling F-Race and Iterative F-Race. However, even the most efficient method, I/F-Race, still requires a relatively large evaluation budget. For example, optimizing an algorithm with 22 parameters needs at least 500 algorithm runs, which might be time-consuming for some expensive algorithms.

Chapter 4

Bayesian Optimization

In this chapter, we will introduce a powerful strategy, Bayesian optimization (BO), to solve the algorithm configuration problem, which can be seen as noisy and costly BBO problem in general. To adapt BO to more complex situations, two implementation of BO are proposed, namely SMAC and MiP-EGO. We will discuss their differences and experimentally test both of them.

Bayesian optimization is a powerful strategy for optimizing computationally expensive functions with small evaluation budget [29]. The efficiency of this approach comes from two aspects: estimate the original expensive function with a cheap surrogate model, and use a cheaper so-called acquisition function to decide where to sample next.

Algorithm 1 Bayesian Optimization

1: Generate the initial samples $X = \{ \mathbf{x}_i \}_{i=1}^{p}$.
2: Evaluate X and collect results to the data set $\mathcal{D} = \{ (\mathbf{x}_i, y_i) \}_{i=1}^{p}$.
3: Construct the surrogate model $\mathcal{M}$ on $\mathcal{D}$.
4: **for** t = 1,2,$\cdots$ **do**
5: Select new $\mathbf{x}_t$ by maximizing the acquisition function α:

$$\mathbf{x}_t = \underset{\mathbf{x}}{\operatorname{argmax}}\, \alpha(\mathbf{x}; \mathcal{M}, \mathcal{D}_{1:p+t-1})$$

6: Evaluate $\mathbf{x}_t$: $y_t = f(\mathbf{x}_t)$.
7: Augment the data $\mathcal{D}_{1:p+t} = \{ \mathcal{D}_{1:p+t-1}, (\mathbf{x}_t, y_t) \}$.
8: Update the surrogate model $\mathcal{M}$.
9: **end for**

The generic BO procedure (see Algorithm 1) starts with an initial evaluation data set and an initial surrogate model. In the following loop, a promising point $\mathbf{x}_t$ is selected by

maximizing the acquisition function α, and then evaluate $\mathbf{x}_t$ using the objective function f, finally update evaluation history data and corresponding surrogate model. Each step is explained in more detail in the following subsections.

4.1 Initial Design

To construct the surrogate model, some initial samples, $X = \{ \mathbf{x}_i \}_{i=1}^{p}$ are generated via a certain sampling technique, such as uniform random sampling and Latin Hyper-cube sampling (LHS) [30], and then evaluated by objective function or target algorithm in the algorithm configuration scenario. The number of the initial samples p may affect the beginning phase of optimization process. A small p may result in poor fitting of surrogate model and consequently a suboptimal point is selected by maximizing the acquisition function which is derived from that surrogate model. On the other hand, a large design may waste too many evaluations on those poor samples and optimization stage is forced to shorten within limited evaluation budget. In practice, obtaining an evaluation of a sample point $\mathbf{x}$ is usually expensive and time-consuming. In this case, the initial design size p can be set as 1 (only applicable for random forest surrogate model), so that optimization stage can be started as soon as possible.

4.2 Surrogate Models

When obtaining a function value is costly, it is natural to consider looking for a model to simulate the input-output relation of that function efficiently, and the output of this model should be as close to the function as possible. Some models in the machine learning community are good choices, which usually map an input to an output based on a set of data pairs. Since BO is a numerical iterative algorithm, where new input-output pairs are generated in each iteration, the surrogate model (also called response surface model or meta-model) needs to update itself in each iteration as well in order to simulate the function more and more accurately. In this subsection, we will elaborate two most commonly used surrogate models, Kriging and Random Forest (RF).

4.2.1 Kriging

Kriging is a spatial estimation method originated from Geostatistics, which is named after a South African engineer Krige in 1950s. Kriging is modeled by a stochastic process and gives the *best linear unbiased prediction* at an unsampled location. It has been widely used in spatial statistics, environmental science and computer simulation experiments, and is still important today [29].

The general aim of Kriging is to predict the value of an underlying random function (also called stochastic process or random field) $Z = Z(\mathbf{x})$ at an unsampled location of interest $\mathbf{x}_0$, based on a set of observations. In Kriging, the random function can be characterized by two major components, large scale variation (trend) and small scale error random function, represented as $Z(\mathbf{x}) = \mu(\mathbf{x}) + Y(\mathbf{x})$.

Suppose a random function $Z = Z(\mathbf{x})$ with $x \in D \subseteq \mathbb{R}^d$, where $Z(\mathbf{x})$ also denotes a random variable for $\forall \mathbf{x} \in D$. We assume that there are n distinct data locations $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and associated data values $Z(\mathbf{x}_1), \ldots, Z(\mathbf{x}_n)$ evaluated by the underlying random function. The *random vector* is thereby defined as $\mathbf{Z} := (Z(\mathbf{x}_1), \ldots, Z(\mathbf{x}_n))^T \in \mathbb{R}^n$, i.e. a collection of values of random variable $Z(\mathbf{x}_i)$. Based on proper prior assumption and random vector, Kriging is able to predict the value of random variable $Z(\mathbf{x}_0)$ at an arbitrary location $\mathbf{x}_0 \in D$.

The main idea of coefficient estimation in Kriging is that the observations which are close to the location being estimated tend to get more weight in the prediction so as to improve the accuracy of the estimate. In other words, Kriging relies on the Euclidean distances of observations and test location of interest, which are modeled by covariance of the underlying random function $Z(\mathbf{x})$.

Model Assumptions

(i) Assume that $Z(\mathbf{x})$ can be decomposed into a deterministic *trend function* $\mu(\mathbf{x})$, and a residual (random function) from the trend $Y(\mathbf{x})$, such that $Z(\mathbf{x}) = \mu(\mathbf{x}) + Y(\mathbf{x})$ [31].

(ii) $Y(\mathbf{x})$ is typically assumed to be *intrinsically stationary* with zero mean [31], namely

$$\mathbb{E}[Y(\mathbf{x})] = 0 \ , \forall \mathbf{x} \in D$$

and the covariance function $Cov(Y(\mathbf{x}_i), Y(\mathbf{x}_j))$ only depends on $\mathbf{x}_i - \mathbf{x}_j$. The covariance function is required to be a positive-definite kernel. We take Gaussian kernel as an example in this section.

$$k(\mathbf{x}_i, \mathbf{x}_j) := b^2 \exp\left(-\sum_{k=1}^{d} \frac{(x_{ik} - x_{jk})^2}{2a_k^2}\right) \text{ for } |\mathbf{x}_i - \mathbf{x}_j| \geq 0$$

According to Assumption (i), we have

$$\mathbb{E}[Z(\mathbf{x})] = \mathbb{E}[\mu(\mathbf{x})] + \underbrace{\mathbb{E}[Y(\mathbf{x})]}_{=0} = \mu(\mathbf{x})$$

(iii) Let $f_0, f_1, ..., f_L$ be known basic function of $\mathbf{x}$. We assume $\mu(\mathbf{x})$ to be a linear combination of these functions evaluated at $\mathbf{x}$:

$$\mu(\mathbf{x}) = \sum_{l=0}^{L} \beta_l f_l(\mathbf{x})$$

with unknown coefficients $\beta_l \in \mathbb{R}\backslash\{0\}$ for $l = 0, \ldots, L$. Here $f_0(\mathbf{x}) = 1$ by convention.

In fact, the trend function $\mu(\mathbf{x})$ depends on the type of Kriging. The assumption (iii) actually defines *universal kringing*. If the global mean $\mu \in \mathbb{R}$ of the random function $Z(\mathbf{x})$ is known and constant, the model is called *simple Kriging*; if the global, constant mean μ is unknown, it is called *ordinal Kriging*.

According to the assumptions above, Cressie observed [32]

$$Z(\mathbf{x}_i) = \mu(\mathbf{x}_i) + Y(\mathbf{x}_i) = \sum_{l=0}^{L} \beta_l f_l(\mathbf{x}_i) + Y(\mathbf{x}_i) = (F\boldsymbol{\beta} + \mathbf{Y})_i, i = 1, \ldots, n$$

Hence, we obtain a matrix form for the random vector $\mathbf{Z}$:

$$\mathbf{Z} = \begin{pmatrix} Z(\mathbf{x}_1) \\ \vdots \\ Z(\mathbf{x}_n) \end{pmatrix} = \underbrace{\begin{pmatrix} f_0(\mathbf{x}_1) & \cdots & f_L(\mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ f_0(\mathbf{x}_n) & \cdots & f_L(\mathbf{x}_n) \end{pmatrix}}_{=F} \underbrace{\begin{pmatrix} \beta_0 \\ \vdots \\ \beta_L \end{pmatrix}}_{=\boldsymbol{\beta}} + \underbrace{\begin{pmatrix} Y(\mathbf{x}_1) \\ \vdots \\ Y(\mathbf{x}_n) \end{pmatrix}}_{=\mathbf{Y}} = F\boldsymbol{\beta} + \mathbf{Y}$$

Kriging Predictor

The Kriging predictor is essentially a *best linear unbiased predictor* (BLUP) which means it is unbiased and has minimal prediction variance among all linear unbiased predictors [33]. The Kriging predictor $Z_\omega^*(\mathbf{x}_0)$ of the value of $Z(\mathbf{x})$ at the test point $\mathbf{x}_0$ is the linear combination of Kriging weights and random vector:

$$Z_\omega^*(\mathbf{x}_0) := \sum_{i=1}^{n} \omega_i Z(\mathbf{x}_i) = \omega^T \mathbf{Z}$$

with $\boldsymbol{\omega} := (\omega_1, \ldots, \omega_n)^T \in \mathbb{R}^n$. Combining this definition with $\mathbf{Z} = F\boldsymbol{\beta} + \mathbf{Y}$, we obtain

$$Z_\omega^*(\mathbf{x}_0) = \omega^T (F\boldsymbol{\beta} + \mathbf{Y})$$

Unbiasedness Condition

Once we obtain our linear predictor $Z_\omega^*(\mathbf{x}_0)$, we need to make sure that it is unbiased in any situation, i.e. $\mathbb{E}[Z_\omega^*(\mathbf{x}_0) - Z(\mathbf{x}_0)]$ should equal to 0. Following the work of Cressie [32] and Wackernagel [34], we can get:

$$\mathbb{E}[Z_\omega^*(\mathbf{x}_0) - Z(\mathbf{x}_0)] = \sum_{i=1}^{n} \omega_i \left(\mathbb{E}[\mu(\mathbf{x}_i)] + \underbrace{\mathbb{E}[Y(\mathbf{x}_i)]}_{=0} \right) - \left(\mathbb{E}[\mu(\mathbf{x}_0)] + \underbrace{\mathbb{E}[Y(\mathbf{x}_0)]}_{=0} \right)$$

$$= \sum_{i=1}^{n} \omega_i \mu(\mathbf{x}_i) - \mu(\mathbf{x}_0) \stackrel{!}{=} 0$$

$$\Leftrightarrow \sum_{l=0}^{L} \beta_l \left(\sum_{i=1}^{n} \omega_i f_l(\mathbf{x}_i) - f_l(\mathbf{x}_0) \right) = 0$$

Together with $\beta_l \neq 0$ in model assumption (iii) and $\mathbf{f}_0 := (1, f_1(\mathbf{x}_0), \ldots, f_L(\mathbf{x}_0))^T$, the general unbiasedness of $Z_\omega^*(\mathbf{x}_0)$ is satisfied if and only if

$$\sum_{i=1}^{n} \omega_i f_l(\mathbf{x}_i) = f_l(\mathbf{x}_0) \text{ for } l = 0, \ldots, L \Leftrightarrow F^T \omega = \mathbf{f}_0$$

which are called *universality conditions* [35].

Variance of the Prediction Error

The variance of the prediction error can be used as a measurement of the accuracy of the predictor $Z_\omega^*(\mathbf{x}_0)$. Based on the work of Cressie [32], it is derived as:

$$\sigma_E^2 := \mathrm{Var}\left(Z_\omega^*(\mathbf{x}_0) - Z(\mathbf{x}_0)\right) = \mathbb{E}\left[\left(Z_\omega^*(\mathbf{x}_0) - Z(\mathbf{x}_0)\right)^2\right] = \mathbb{E}\left[\left(\sum_{i=1}^{n}\omega_i Z(\mathbf{x}_i) - Z(\mathbf{x}_0)\right)^2\right]$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n}\omega_i\omega_j\mathbb{E}\left[Y(\mathbf{x}_i)\,Y(\mathbf{x}_j)\right] - 2\sum_{i=1}^{n}\omega_i\mathbb{E}\left[Y(\mathbf{x}_0)\,Y(\mathbf{x}_i)\right] + \mathbb{E}\left[\left(Y(\mathbf{x}_0)\right)^2\right]$$

$$= b^2 + \boldsymbol{\omega}^\top\mathbf{K}\boldsymbol{\omega} - 2\mathbf{k}_0^\top\boldsymbol{\omega}$$

Here $k(\mathbf{x}_0,\mathbf{x}_0) = b^2$, $\mathbf{k}_0 = \mathbf{k}(\mathbf{x}_0,\cdot) = \left(k(\mathbf{x}_0,\mathbf{x}_1), k(\mathbf{x}_0,\mathbf{x}_2), \ldots, k(\mathbf{x}_0,\mathbf{x}_n)\right)^\top$ and

$$\mathbf{K}_{(b^2,\boldsymbol{a})} = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & k(\mathbf{x}_1,\mathbf{x}_2) & \ldots & k(\mathbf{x}_1,\mathbf{x}_n) \\ k(\mathbf{x}_2,\mathbf{x}_1) & k(\mathbf{x}_2,\mathbf{x}_2) & \ldots & k(\mathbf{x}_2,\mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n,\mathbf{x}_1) & k(\mathbf{x}_n,\mathbf{x}_2) & \ldots & k(\mathbf{x}_n,\mathbf{x}_n) \end{bmatrix}$$

Minimal Prediction Variance

In order to minimize the prediction error variance σ_E^2, we have to solve the following constrained optimization problem given by

$$\text{minimize } b^2 + \boldsymbol{\omega}^\top\mathbf{K}\boldsymbol{\omega} - 2\mathbf{k}_0^\top\boldsymbol{\omega} \text{ subject to } \boldsymbol{\omega}^T F = \mathbf{f}_0^T$$

This optimization problem can be solved using *Lagrange Multipliers*, so we get

$$\begin{bmatrix} \mathbf{K} & \mathbf{F} \\ \mathbf{F}^\top & \mathbf{O} \end{bmatrix}\begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\mathbf{k}_0 \\ \mathbf{f}_0 \end{bmatrix}$$

with Lagrange parameter vector $\boldsymbol{\lambda} := (\lambda_0, \lambda_1, \ldots, \lambda_L)^T \in \mathbb{R}^{L+1}$.

Solving this linear system, we have

$$\boldsymbol{\omega} = \mathbf{K}^{-1}\left(\mathbf{k}_0 - \mathbf{F}\boldsymbol{\lambda}\right)$$
$$\boldsymbol{\lambda} = \left(\mathbf{F}^\top \mathbf{K}^{-1}\mathbf{F}\right)^{-1}\left(\mathbf{F}^\top \mathbf{K}^{-1}\mathbf{k}_0 - \mathbf{f}_0\right)$$

Plugging $\boldsymbol{\omega}$ back, we have the **Kriging predictor**:

$$Z^*_\omega(\mathbf{x}_0) = \boldsymbol{\omega}^T\mathbf{Z} = \left[\mathbf{k}_0 - \mathbf{F}\left(\mathbf{F}^\top \mathbf{K}^{-1}\mathbf{F}\right)^{-1}\left(\mathbf{F}^\top \mathbf{K}^{-1}\mathbf{k}_0 - \mathbf{f}_0\right)\right]^\top \mathbf{K}^{-1}\mathbf{Z}$$

and corresponding Kriging estimate $z^*_\omega(\mathbf{x}_0)$ at the location of interest $\mathbf{x}_0$:

$$z^*_\omega(\mathbf{x}_0) = \boldsymbol{\omega}^T\mathbf{z} = \left[\mathbf{k}_0 - \mathbf{F}\left(\mathbf{F}^\top \mathbf{K}^{-1}\mathbf{F}\right)^{-1}\left(\mathbf{F}^\top \mathbf{K}^{-1}\mathbf{k}_0 - \mathbf{f}_0\right)\right]^\top \mathbf{K}^{-1}\mathbf{z}$$

We can also get the minimal prediction variance through inserting $\boldsymbol{\omega}$ and $\boldsymbol{\lambda}$ to σ^2_E.

Estimation of Trend Function Coefficients and Residual Covariance

(i) First of all, we have to estimate the coefficients of the trend function $\mu(\mathbf{x}) = F\boldsymbol{\beta}$, since we do not have it at hand. This could be done using, for instance, the **generalized-least-squares** estimator. It turns out that the estimate of the coefficient vector $\boldsymbol{\beta}$ can be derived as $\hat{\boldsymbol{\beta}} = \left(\mathbf{F}^\top \mathbf{K}^{-1}\mathbf{F}\right)^{-1}\mathbf{F}^\top \mathbf{K}^{-1}\mathbf{Z}$ (cf. [32])

(ii) Furthermore, the hyper-parameters of $k(\mathbf{x}_i, \mathbf{x}_j)$, such as b and $\mathbf{a}$ in Gaussian kernel defined previously can be fixed or estimated from the data.

Kriging and Gaussian Processes

The Gaussian process (GP) is a stochastic process with infinite dimensions where any finite set of random variables follows a multivariate Gaussian distribution [36]. The GP is specified by its mean function and covariance function. The simple Kriging with zero mean is exactly the same as GP with zero mean.

The Kriging estimator can also be derived via Bayesian inference [37]. From the perspective of Bayesian statistics, it turns out that Kriging and Gaussian process regression

are so closely related that they are often used interchangeably. There are some key differences in practice, though, in terms of model assumption and fitting method. For a more in-depth introduction to Kriging and GP, see, i.e.[29, 38, 39].

4.2.2 Random Forest

An alternative surrogate model is Random Forest, which can be used for both classification and regression. It enables us to explore the complex input-output dependency without any prior assumption on the data or any limitation on variable type. This means it is a non-parametric model and can handle both numerical and categorical input variables. It is an ensemble method, consisting of several independent base estimators (Decision Trees) and using their average as a prediction. We first briefly introduce Decision Tree and then discuss how to build a Random Forest for regression task.

The general but infeasible goal of Decision Tree Regression is to choose a collection of regions $\{ R_M \}_{m=1}^{M}$ to

$$\min_{\{ R_M \}_{m=1}^{M}} \frac{1}{N_m} \sum_{i \in R_m} (y_i - \bar{y}_m)^2$$

where region (also called group) R_i has N_m data points and $\bar{y}_m$ is the average values of the dependent variable in region R_i.

This problem is NP-hard, because there are $\binom{N}{M}$ possible groups in total. A feasible problem solution is that we do not search over all possible groups, instead only group on hyper-cubes using *recursive binary splits*. The idea of this method is demonstrated graphically and geometrically in Figure 4.2 and Figure 4.1 respectively.

At the root node we start with whole training set and we ask a question which partition divides the data into two subsets. The aim of splitting is to achieve more homogeneous partitions. We iteratively continue splitting until we obtain good enough partitions or reach the corresponding leaf nodes. There are various metrics for measuring goodness of a split. For classification, we can use information gain or Gini impurity. For regression task, a widely-used metric is mean squared error (MSE):

$$MSE = \frac{1}{N_m} \sum_{i \in R_m} (y_i - \bar{y}_m)^2$$

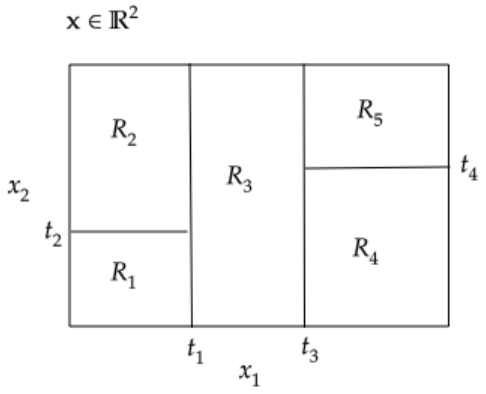


FIGURE 4.1: An example of decision tree demonstrating recursive binary splitting geometrically.

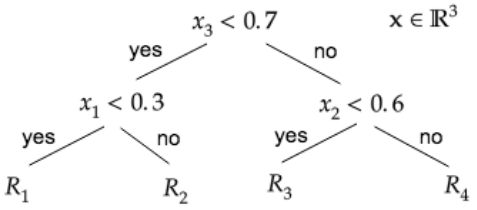


FIGURE 4.2: Another graphical example of decision tree.

Once we can build one decision tree, it is easy to repeat that process to build a random forest. An algorithm of a random forest for regression can be found in [40]:

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{ T_b \}_1^B$
3. To make a prediction at a new point $\mathbf{x}$: $\hat{f}_{rf}^B(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} T_b(\mathbf{x})$

Here *bootstrapping* is a re-sampling method. The basic idea is to randomly draw N samples with replacement from the training set. This is done B times, producing B bootstrap data sets. We notice that the randomness of this random forest algorithm comes from two aspects. One is concerned with bootstrapped data, that is, each tree sees a different data set. Another involves only m variables can be considered when looking for the best split. Randomness of trees turns out to be very useful and makes random forest work well. Each tree is correct but missing a lot of information because each tree only sees a little bit of data in a little bit of features. Therefore we consider each

tree to be a predictor with very high variance and low bias. One common technique to reduce variance of low-bias predictors is *bagging*. When making a prediction at a new point, the predictions of all trees are combined and averaged to form an overall prediction.

Note that when using random forest under Bayesian optimization framework, a predictive uncertainty estimate needs to be computed. However, random forest does not provide such uncertainty estimate of predictions. An alternative method is using empirical variance of predictions across all trees in the ensemble. This simple method works well in practice, because intuitively, if the individual tree predictions $y_1, \ldots, y_B$ get closer, we are more certain about the overall prediction.

4.3 Acquisition Function

The global optimum is approached by iteratively maximizing a so-called acquisition function. An acquisition function (also called infill-criterion) can be seen as measuring the usefulness (or utility) of candidate points. It incorporates mean and standard deviation of surrogate model prediction and it is able to trade-off between exploitation and exploration. One of popular acquisition function is Expected Improvement (EI). The expected improvement can be defined as

$$\mathrm{EI}(\mathbf{x}) = \mathrm{E}(I(\mathbf{x})),$$

where the random variable $I(\mathbf{x})$ represents the possible improvement evaluating at $\mathbf{x}$ over the current optimum. In minimization problem, $I(\mathbf{x})$ is defined as [41]

$$I(\mathbf{x}) = \max\left\{y_{\min} - Y(\mathbf{x}|\mathcal{D}), 0\right\},$$

where $y_{\min} = \min\left\{y_i | \left(\mathbf{x}_i, y_i\right) \in \mathcal{D}\right\}$, and $Y(\mathbf{x}|\mathcal{D})$ is a random variable representing the posterior predictive distribution at $\mathbf{x}$. When using Gaussian Process as surrogate, $Y(\mathbf{x}|\mathcal{D})$ follows a Gaussian distribution, i.e., $Y(\mathbf{x}|\mathcal{D}) \sim \mathcal{N}\left(\hat{\mu}(\mathbf{x}|\mathcal{D}), \hat{s}^2(\mathbf{x}|\mathcal{D})\right)$. Therefore, EI can be expressed as

$$\mathrm{EI}(\mathbf{x}) = (y_{\min} - Y(\mathbf{x}|\mathcal{D})\Phi\left(\frac{y_{\min} - \hat{\mu}(\mathbf{x}|\mathcal{D})}{\hat{s}(\mathbf{x}|\mathcal{D})}\right) + \hat{s}(\mathbf{x}|\mathcal{D})\phi\left(\frac{y_{\min} - \hat{\mu}(\mathbf{x}|\mathcal{D})}{\hat{s}(\mathbf{x}|\mathcal{D})}\right),$$

where $\Phi(\cdot)$ and $\phi(\cdot)$ denote the cumulative distribution function and probability density function of the standard normal random variable. It turns out that EI is highly effective and well-balanced among all of the existing acquisition functions, and it can even ensure global convergence [42]. The other popular acquisition functions include the generalized expected improvement (GEI) [43], the probability of improvement (PI) [44], the lower confidence bound (LCB) [45], the moment-generating function of the improvement (MGFI) [46], and so on.

To propose the next point for evaluation, the acquisition function needs to be optimized. Since the acquisition function is much cheaper to evaluate than the original objective function, we can use various optimization techniques. For example, the branch and bound algorithm [47], the multi-start approach [48], mixed-integer evolutionary strategy [11], local search [18], and so on.

4.4 MiP-EGO

Kriging has been applied to the field of Design and Analysis of Computer Experiments (DACE), which is developed by Sacks et al .[7, 49]. The efficient global optimization (EGO) algorithm, proposed by Jones et al. [47], is the combination of Kriging with the sequential expected improvement acquisition criterion in the context of DACE . The goal is to find a design point or points that optimizes a black-box function of interest. EGO and Bayesian optimization share the exact same idea but derived from different fields, so in this section they will be used interchangeably for the sake of simplicity.

EGO is limited in optimizing noise-free functions with real-valued input, and it only selects one sample point to evaluate in each step. However, in the context of algorithm configuration, randomized algorithms often have mixed categorical/numerical parameters and a considerably large search space of parameter settings, and multiple parameter settings are expected to be evaluated simultaneously in each step in order to speed up the automatic tuning procedure. In this case, the Mixed-Integer Parallel Efficient Global Optimization (MiP-EGO) [11] is proposed, as an extension of EGO, to overcome these limitations.

4.4.1 Handling Mixed Integer Input

Random forest is used as surrogate model in MiP-EGO and Mixed-Integer Evolution Strategy (MI-ES) [50] is adopted to optimize the infill-criterion, considering that MI-ES is well-suited for finding multiple local optima from a complicated multimodal infill-criterion function. Although MI-ES has shown a significant strong performance on parameter tuning tasks, it is still limited to the demand of large evaluation budget. Therefore,it is better to apply MI-ES on acquisition function optimization than on the original algorithm configuration problem.

4.4.2 A New Acquisition Function and Cooling Strategy

In MiP-EGO, a novel acquisition function based on the moment-generating function (MGF) of the improvement [46] is adopted. A real-valued parameter t (called "temperature") is introduced to explicitly control the exploration-exploitation trade-off. For detailed introduction and complete mathematical expression of this acquisition function, see [46]. Here we use a general expression of an acquisition function, i.e. $u(\mathbf{x};t) = \mu(\mathbf{x}) + t\sigma(\mathbf{x})$, to describe its working mechanism. When t goes up, the prediction error $\sigma(\mathbf{x})$ of the model is assigned more weight so that $u(\mathbf{x};t)$ tends to explore the region with high uncertainty; when t decreases, $u(\mathbf{x};t)$ prefers to exploit the region with high prediction expectation $\mu(\mathbf{x})$. Notice that the functionality of parameter t is analogous to that of the temperature in simulated annealing [51]. Thus the idea of temperature cooling strategies are borrowed from simulated annealing to improve the Bayesian optimization.

The cooling strategy is applied on parameter t, this means, t goes down when iteration i increases. One of commonly used cooling strategies is the exponential strategy, written as $t_{i+1} = \alpha t_i$, $0 < \alpha < 1$, where α denotes the cooling speed. It is required to pre-specify the initial temperature t_0 and cooling speed α manually, as well as the maximal number of iterations N_{max}, for this iterative strategy. Since $\alpha = (t_f/t_0)^{1/N_{\max}}$ is given in [51], actually we only need to set t_0 and t_f (the temperature at the final iteration of BO) by hand and then α will be automatically determined during optimization procedure. It tuns out the cooling strategy makes the Bayesian optimization more explorative in the beginning and more exploitative in the final convergence stage [46].

4.4.3 Parallel Execution

Due to the typically large execution time of a black-box function or a randomized algorithm, parallelized execution is preferred. In other words, multiple different candidate points are required to propose in each iteration of Bayesian optimization. In MiP-EGO, q ($q > 1$) different temperatures t are sampled from the log-normal distribution $Lognormal(0, 1)$ and q different acquisition functions $\{u_i(\mathbf{x}; t_i)\}_i^q$ are instantiated using corresponding temperatures. Consequently, q candidate points can be generated by simultaneously optimizing those q acquisition functions. Since log-normal is a long-tailed distribution, most of sample values of t are relatively small and few of them are large. This leads to most of $u(\mathbf{x}; t)$ being exploited and a few of them being very explorative [1].

4.5 SMAC

Sequential Model-based Algorithm Configuration (SMAC) method is another state-of-the-art implementation of Bayesian optimization for solving algorithm configuration problem. It also uses random forest as surrogate model to handle with mixed categorical/numerical input. Although SMAC has other characteristics, such as solving algorithm selection problems with multiple instances, in this thesis, we will only focus on its novel approach to maximizing the infill-criterion function.

4.5.1 Maximizing EI and Intensification

In SMAC, the expected improvement (EI) is not directly optimized by a systematic algorithm, but by an ad hoc strategy. Concretely, the procedure of finding promising configurations θ with large $EI(\theta)$ is as follows: (1) It computes EI for all configurations used in previous target algorithm runs, and picks 10 configurations with maximal EI, and initializes a local search at each of them. If there were less than 10 configurations in running history, all of them are picked. In local search, discrete/numerical variables are treated separately. The discrete variable is "mutated" by a randomized one-exchange neighbourhood, while the numerical variable v has four "neighbouring" values which are sampled from a univariate Gaussian distribution with mean v and standard deviation 0.2.

Each numerical variable is normalized to $[0, 1]$ in advance, so new values outside the interval $[0, 1]$ will be rejected. EI for all configurations are computed at once; each local search stops, once none of the neighbours has larger EI. (2) It computes EI for 10,000 randomly-sampled configuration and then sort all 10,010 configurations in descending order of EI. The ten results of local search typically had larger EI than all randomly sampled configurations [18]. (3) SMAC alternates between configurations from the list, which contains 10,010 sorted configurations, and another 10,000 configurations sampled uniformly at random. The resulting list (denoted as $\mathbf{\Theta}_{new}$ in [18]) of 20,010 configurations would look like $\mathbf{\Theta}_{new} = [\theta_{sorted,1}, \theta_{rand,1}, \theta_{sorted,2}, \theta_{rand,2}, \ldots, \theta_{sorted,10010}, \theta_{rand,10000}]$. This means, SMAC generates a sequence of new configurations, instead of one or multiple promising configurations using infill-criterion (e.g. in MiP-EGO and general Bayesian optimization).

Next, the best configuration so far (called *the incumbent*), are "compared" with element in $\mathbf{\Theta}_{new}$ one by one in order to determine new incumbent. This special comparison procedure is called *intensification* in SMAC, i.e. racing challengers against an incumbent. The aim of intensification is to guarantee two configurations being compared fairly in the sense of statistics, since they assume the objective black-box function is noisy. The side effect is that more runs are required for evaluating configurations which will greatly increase the computational overhead. For detailed introduction of intensification, see [18, 52].

Chapter 5

Extensions of MiP-EGO on Conditional Spaces

Since MiP-EGO can only solve hyper-parameter optimization so far, it is useful to extend MiP-EGO to solve the combined algorithm selection and hyper-parameter optimization problem. We propose three variants of MiP-EGO for searching on conditional spaces, abbreviated as EGO-impute, EGO-ss and EGO-ws respectively. The basic idea is to construct individual surrogate model for each learning algorithm.

First of all, we define the black-box objective function, which will be optimized assisted by a surrogate model $\mathcal{M}$, as $f(\boldsymbol{\theta})$. Here, $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_d)^\intercal$ denotes a vector containing d hyper-parameters of a machine learning algorithm and $f(\boldsymbol{\theta})$ represents a specific metric to evaluate the algorithm, which might be cross-validation error, F1 score, AUROC or so on.

In order to optimize m learning algorithms simultaneously and let MiP-EGO select the best algorithm, we simply combine their hyper-parameters all together and introduce an additional parameter θ_0 which indicates the choice of the learning algorithm. For simplicity, θ_0 is represented by an integer ranging from 1 to m, namely, $\theta_0 \in \Theta_0 = \{1, \ldots, m\}$. The resulting hybrid configuration is denoted as $\boldsymbol{\gamma} = (\theta_0, \boldsymbol{\theta}_1^\intercal, \ldots, \boldsymbol{\theta}_m^\intercal)^\intercal$ and its corresponding search space is $\boldsymbol{\Gamma} = \Theta_0 \times \boldsymbol{\Theta}_1 \times \cdots \times \boldsymbol{\Theta}_m$. The dimensionality of $\boldsymbol{\gamma}$ is $m + 1$. For a specific hybrid configuration $\boldsymbol{\gamma}$, we call $\boldsymbol{\theta}_i \in \boldsymbol{\gamma}$ active when $\theta_0 = i$, while the others in $\boldsymbol{\gamma}$ are called inactive, and the so-called performance of $\boldsymbol{\gamma}$ is the value of $f_i(\boldsymbol{\theta}_i)$.

Now we can say the input of EGO-X algorithm includes $\boldsymbol{\gamma}$'s search space (Γ) and a collection of black-box functions, that is, $\{ f_i \}_{i=1}^{m}$, while the output is expected to be the vector $\boldsymbol{\gamma}$.

The simplest strategy, EGO-impute, is based on *imputation*, as was done in [53, 54]. In statistics, imputation is a process of replacing missing data with an estimated value. Here, we make an analogy between missing values in a dataset and inactive hyper-parameters in a hybrid configuration $\boldsymbol{\gamma}$. Before training a single surrogate model based on the data of hybrid configuration and its associated performance, we give some default values to those inactive hyper-parameters in $\boldsymbol{\gamma}$. Typically, each default value is taken from the middle of its range. The overall optimization procedure of EGO-impute is similar with MiP-EGO, but the way of evaluating a sample $\boldsymbol{\gamma}$ is changed, as shown in Algorithm 2 as well as Algorithm 3 and 4. Specifically, if there was a sample point $\boldsymbol{\gamma} = (1, \boldsymbol{\theta}_1^{\intercal}, \ldots, \boldsymbol{\theta}_m^{\intercal})^{\intercal}$ with active hyper-parameter $\boldsymbol{\theta}_1$ and associated evaluation $y_1 = f_1(\boldsymbol{\theta}_1)$, then the data of $\boldsymbol{\gamma}_{imputed} = (1, \boldsymbol{\theta}_1^{\intercal}, \boldsymbol{\theta}_2^{\intercal,(\text{default})}, \ldots, \boldsymbol{\theta}_m^{\intercal,(\text{default})})^{\intercal}$ and y_1 are used to train the surrogate model. As the amount of data grows, the surrogate model will fit better theoretically. This can be interpreted that forcing inactive parameters as fixed values actually helps the surrogate model, especially random forest, to pick the most important features from $\boldsymbol{\gamma}$, that is, active parameters. Just as imputing missing values in statistics helps reducing the bias, imputing inactive hyper-parameters in Bayesian optimization also conduces to obtaining properly behaving surrogate models.

Algorithm 2 EGO-impute

Generate the initial samples $\{ \boldsymbol{\gamma}_i \}_{i=1}^{p}$;
for $i = 1, \ldots, p$ **do**
    $g_i, \boldsymbol{\gamma}_i \leftarrow$ *partially_evaluate* $(\boldsymbol{\gamma}_i, \{ f_i \}_{i=1}^{m})$;
    $\boldsymbol{\gamma}_{imputed,i} \leftarrow$ *impute* $(\boldsymbol{\gamma}_i)$;
end for
Construct the surrogate model $\mathcal{M}$ on $\mathcal{D} = \{ (\boldsymbol{\gamma}_{imputed,i}, g_i) \}_{i=1}^{p}$;
while the stopping criterion is not fulfilled **do**
    Maximize the infill-criterion using *MI-ES*:

$$\boldsymbol{\gamma} \leftarrow \underset{\boldsymbol{\gamma} \in \boldsymbol{\Gamma}}{\operatorname{argmax}} \mathcal{M}(\boldsymbol{\gamma})$$

    $g, \boldsymbol{\gamma} \leftarrow$ *partially_evaluate* $(\boldsymbol{\gamma}, \{ f_i \}_{i=1}^{m})$;
    $\boldsymbol{\gamma}_{imputed} \leftarrow$ *impute* $(\boldsymbol{\gamma})$;
    Augment $\mathcal{D}$ with $(\boldsymbol{\gamma}_{imputed}, g)$;
    Update the surrogate model $\mathcal{M}$ on $\mathcal{D}$.
end while
Return $\boldsymbol{\gamma}_{imputed}, g$

Algorithm 3 Function partially_evaluate $(\boldsymbol{\gamma}, \{ f_i \}_{i=1}^{m})$

1: **for** $i = 1, \ldots, m$ **do**
2: **if** $i = \theta_0 \in \boldsymbol{\gamma}$ **then**
3: $g \leftarrow f_i(\boldsymbol{\theta}_i)$;
4: **end if**
5: **end for**
6: **Return** g;

Algorithm 4 Function impute $(\boldsymbol{\gamma})$

1: **for** $i = 1, \ldots, m$ **do**
2: **if** $i = \theta_0 \in \boldsymbol{\gamma}$ **then**
3: Replace $\boldsymbol{\theta}_j, j \neq i$, with the default values;
4: Generate a new hybrid configuration: $\boldsymbol{\gamma}_{imputed}$;
5: **end if**
6: **end for**
7: **Return** $\boldsymbol{\gamma}_{imputed}$;

The other two strategies, EGO-ss and EGO-ws, have the common idea shown in Algorithm 5, except for the way of determining a new candidate point. The overall optimization procedure tends to construct m surrogate models for corresponding learning algorithms. In this case, we need to get the evaluation value for each $\boldsymbol{\theta}$ in $\boldsymbol{\gamma}$, and maintain m sub-datasets $\mathcal{D}_i$, $i = 1, 2, \ldots, m$. Also, we need to keep a dataset that records $\boldsymbol{\gamma}$ and its associated performance value in each iteration. Here the so-called performance value is just minimum among $\{ y_i = f_i(\boldsymbol{\theta}_i) \}_{i=1}^{m}$ for $\boldsymbol{\theta}_i \in \boldsymbol{\gamma}$. The process of initializing and updating an individual surrogate model can be reflected by two for-loops in Algorithm 5.

The difference between EGO-ss and EGO-ws depends on the way of determining a new sample point $\boldsymbol{\gamma}$. More specifically, the internal optimization algorithm (MI-ES) has distinct input in these two strategies. In EGO-ss (shown in Algorithm 6), MI-ES behaves in the standard way. It searches on a single search space $\boldsymbol{\Theta}$ using one associated infill-criterion function at one time. And the algorithm selector θ_0 is determined by the maximum among m *EI* values in each iteration. While, in EGO-ws (shown in Algorithm 7), the inputs of MI-ES are the search space $\boldsymbol{\Gamma}$ of the hybrid configuration, and a set of the infill-criterion function. In order to adapt MI-ES to this conditional search space, we modify the evaluation procedure in MI-ES, as shown in Algorithm 3. In MI-ES, the fitness of $\boldsymbol{\gamma}$ is determined by the infill-criterion value of associated active parameters. Notice that it is not necessary to impute inactive parameters in this evaluation procedure,

because a new population of $\boldsymbol{\gamma}$ will be sampled from the entire search space $\boldsymbol{\Gamma}$ in each iteration of MI-ES. The procedure of MI-ES in EGO-ws is shown in Algorithm 8.

Algorithm 5 The common framework of EGO-ss and EGO-ws.

for $i = 1, \ldots, m$ **do** // Initialize each surrogate model.
  Generate the initial samples $\{ \boldsymbol{\theta}_i^k \}_{k=1}^p$;
  Construct surrogate model $\mathcal{M}_i$ on data set $\mathcal{D}_i = \{ (\boldsymbol{\theta}_i^k, y_i^k) \}_{k=1}^p$;
end for
Initialize $\mathcal{D} = \varnothing$;
while the stopping criterion is not fulfilled **do**
  Select a new hybrid configuration using:

$$\boldsymbol{\gamma} \leftarrow \textit{argmax\_SingleSpace}(\{ \mathcal{M}_i \}_{i=1}^m)$$

$$\textbf{or } \boldsymbol{\gamma} \leftarrow \textit{argmax\_WholeSpace}(\{ \mathcal{M}_i \}_{i=1}^m);$$

  $g \leftarrow \textit{partially\_evaluate}(\boldsymbol{\gamma}, \{ f_i \}_{i=1}^m)$;
  Add $(\boldsymbol{\gamma}, g)$ to $\mathcal{D}$;
  for $i = 1, \ldots, m$ **do** // Re-train respective surrogate model.
    Evaluate $\boldsymbol{\theta}_i \in \boldsymbol{\gamma}$: $y_i = f_i(\boldsymbol{\theta}_i)$;
    Augment $\mathcal{D}_i$ with $(\boldsymbol{\theta}_i, y_i)$;
    Re-train model $\mathcal{M}_i$ on $\mathcal{D}_i$;
  end for
end while
Return $\boldsymbol{\gamma}, g$

Algorithm 6 Function argmax_SingleSpace($\{ \mathcal{M}_i \}_{i=1}^m$ **)**

Initialize $u_{max} \leftarrow 0, i^* \leftarrow 1$;
for $i = 1, \ldots, m$ **do**
  Construct the infill-criterion function $U_i \leftarrow \textit{EI}(\boldsymbol{\theta}_i; \mathcal{M}_i)$;
  $\boldsymbol{\theta}_i^*, u_i^* \leftarrow$ *MI-ES* ($\boldsymbol{\theta}_i$*'s search space* $\boldsymbol{\Theta}_i, U_i$);
  if $u_i^* > u_{max}$ **then**
    $u_{max} \leftarrow u_i^*$ and $i^* \leftarrow i$;
  end if
end for
$\boldsymbol{\gamma} \leftarrow (i^*, \boldsymbol{\theta}_1^*, \ldots, \boldsymbol{\theta}_m^*)$;
Return $\boldsymbol{\gamma}$;

Algorithm 7 Function argmax_WholeSpace($\{ \mathcal{M}_i \}_{i=1}^m$ **)**

for $i = 1, \ldots, m$ **do**
  Construct the infill-criterion function $U_i \leftarrow \textit{EI}(\boldsymbol{\theta}_i; \mathcal{M}_i)$;
end for
$\boldsymbol{\gamma} \leftarrow$ *MI-ES*($\boldsymbol{\gamma}$*'s search space* $\Gamma, \{ U_i \}_{i=1}^m$);
Return $\boldsymbol{\gamma}$;

Algorithm 8 MI-ES in EGO-ws

1: $t \leftarrow 0$
2: **Initialize** population P_t, including μ individuals randomly generated within the space $\mathbf{\Gamma}$
3: **Partially_evaluate** P_t
4: **while** termination criterion not fulfilled **do**
5: **Generate** λ offspring
6: **Partially_evaluate** λ offsping
7: **Select** the μ best individuals for P_{t+1} from λ offspring
8: $t = t + 1$
9: **end while**

Chapter 6

Experimental Evaluations

In this chapter, we will empirically evaluate the performance of MiP-EGO and SMAC on two artificial test problems, and test the performance of three variants of MiP-EGO on a algorithm configuration scenario where several machine learning models are configured on a set of datasets, furthermore comparing them to other existing methods.

6.1 Artificial Test Problems

In this section, empirical results demonstrate the convergence behaviour of both MiP-EGO and SMAC on two parameterized problems. Both algorithms are allowed to perform maximal 200 evaluations and have 15 input variables. They both use EI as infill-criterion and select one point at each step.

6.1.1 Barrier Function

Barrier function is a parameterized multi-modal problem generator where the degree of ruggedness (problem difficulty) is controlled by parameter C through constructing an integer array A. Higher values of C result in more rugged landscapes with many barriers. Algorithm 9 [50] shows the construction procedure of barrier function with respect to one input variable.

Algorithm 9 Barrier Function

1: $\mathrm{A}[i] = i, i = 0, \ldots, 19$
2: **for** $k \in \{1, \ldots, C\}$ **do**
3: $j \leftarrow$ uniform random number out of $\{0, \ldots, 18\}$
4: swap values of $\mathrm{A}[j]$ and $\mathrm{A}[j+1]$
5: **end for**

To deal with mixed discrete/numerical input variables, we can apply array A on each variable and make a summation:

$$f_{\text{barrier}}(\mathbf{r}, \mathbf{z}, \mathbf{d}) = \sum_{i=1}^{n_r} \mathrm{A}\left[\lfloor r_i \rfloor\right]^2 + \sum_{i=1}^{n_z} \mathrm{A}\left[z_i\right]^2 + \sum_{i=1}^{n_d} \mathrm{B_i}\left[d_i\right]^2 \rightarrow \min$$

$$\mathbf{r} \in [0, 19]^{n_r} \subset \mathbb{R}^{n_r}, \mathbf{z} \in [0, 19]^{n_z} \subset \mathbb{Z}^{n_z}, \quad \mathbf{d} \in \{0, \ldots, 19\}^{n_d} \subset \mathbb{D}^{n_d}$$

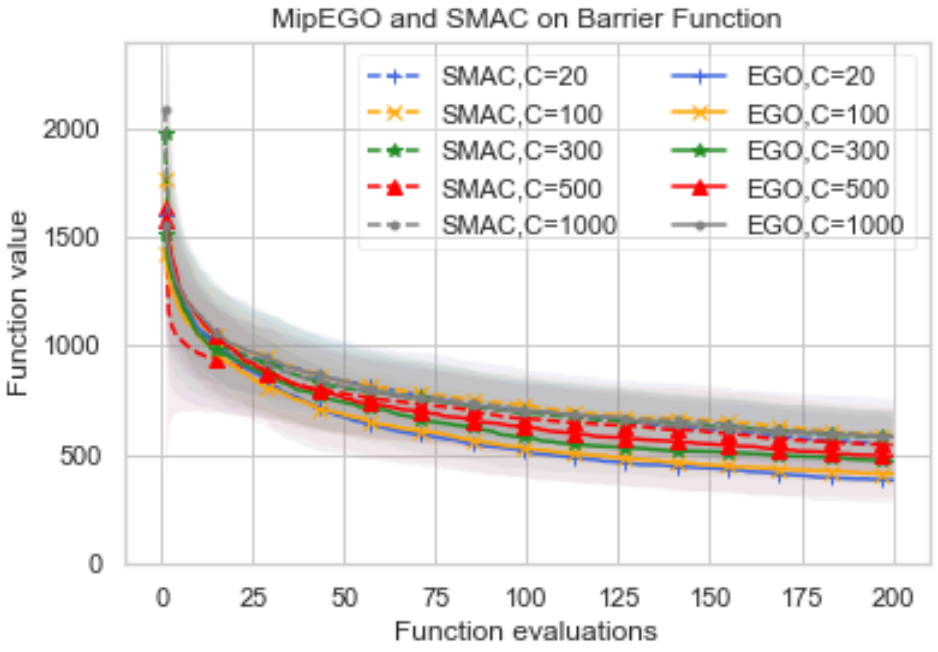
$$n_r = n_z = n_d = 5$$

Here, $B_i\,(i = 1, \ldots, n_d)$ denotes a set of i permutations of the sequence $0, \ldots, 19$. The random permutation of B_i is controlled by a random seed and is fixed before the run. This construction prevents the value of nominal variable d_i from being quantitatively correlated with the value of objective function f.

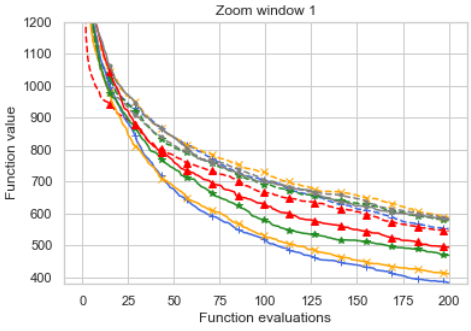
6.1.1.1 Results

We tested MiP-EGO and SMAC on multiple barrier functions identified by parameter C. Since barrier function is stochastic, in the empirical experiments, we created 10 problem instances (by using 10 random seeds) for each control parameter C, and we let the algorithm perform 20 repeated runs on each instance. That is, $20 \times 10 = 200$ runs for each value of C.

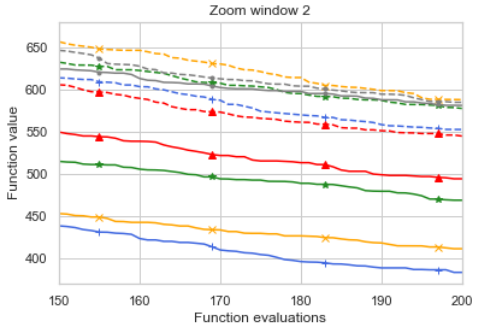
Figure 6.1(a) shows averaged best function values found by MiP-EGO and SMAC, and corresponding standard deviation represented by shaded area. We can see, it is more difficult for both MiP-EGO and SMAC to find the global optimum on barrier functions with a higher C value. If we zoom Figure 6.1(a) and ignore shaded area, in Figure 6.1(b) and Figure 6.1(c), we can see MiP-EGO perform better than SMAC on all five classes of problems identified by C. Especially on relatively simple problems (where $C = 20, 100, 300$), the overall performance of MiP-EGO is significantly better than that



(a)



(b) Zoom ignoring shaded area.



(c) Zoom into the last 50 evaluations.

FIGURE 6.1: Comparison between average error values found by MiP-EGO, SMAC, random search for problem $C = 20, 100, 300, 500, 1000$.

of SMAC. The corresponding box plot for best function values in the last evaluation found by MiP-EGO, SMAC and pure random search is shown in Figure 6.3.

6.1.2 Mixed Integer NK Landscapes

Mixed integer NK landscapes [55] are test functions where correlation between variables of different types can be controlled by parameter K. Similar to experiments for barrier functions, we generated 10 problem instantiations for each $K \in \{1, 3, 5, 10, 14\}$. Each

generated problem consists of $N = 15$ variables, defined as:

$$\mathbf{r} \in [-10, 10]^{n_r} \subset \mathbb{R}^{n_r}, \mathbf{z} \in [0, 19]^{n_z} \subset \mathbb{Z}^{n_z}, \quad \mathbf{d} \in \{0, 1\}^{n_d} \subset \mathbb{D}^{n_d}$$

$$n_r = n_z = n_d = 5$$

We ran both MiP-EGO and SMAC 20 times on each problem instance, that is, each algorithm performs $20 \times 10 = 200$ runs for each value of K. Note that the global minimum of NK landscape can be obtained through brute force search beforehand, so the objective function value is represented by error rather than fitness value, where error = best found fitness - best possible fitness.

6.1.2.1 Results

The results are shown in Figure 6.2. As can be seen, when the problem difficulty increases with K, errors of both algorithms increase. And MiP-EGO perform better than SMAC on 4 classes of problems with $K = 1, 3, 5, 10$. The corresponding box plot for the last evaluation's errors of both algorithms can be seen in Figure 6.4. Note that the horizontal line at the beginning of run history of MiP-EGO indicates 5 samples in the initial design.

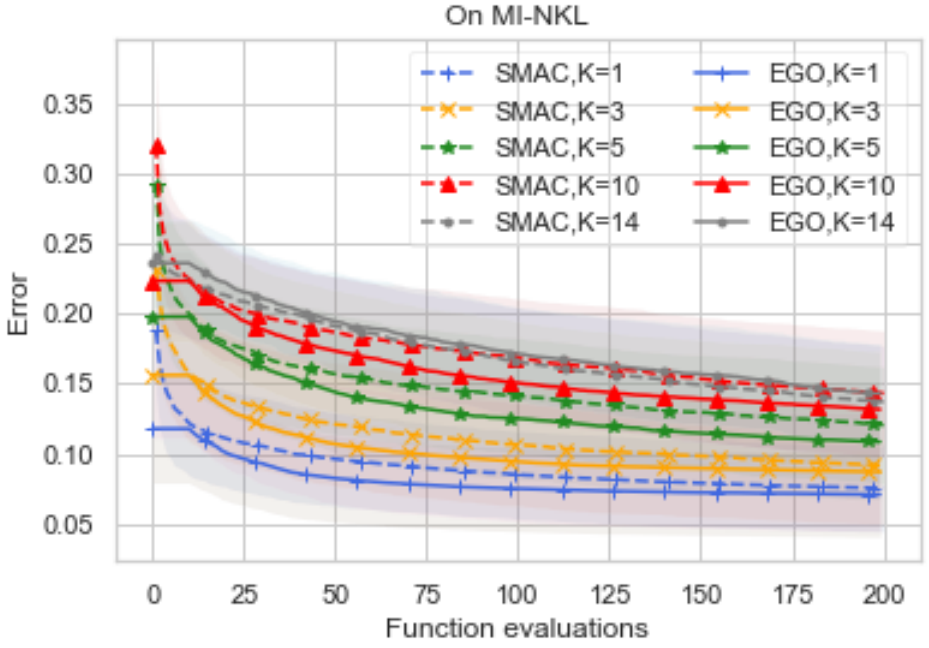


FIGURE 6.2: Comparison between average error values found by MiP-EGO, SMAC, random search on mixed integer NK landscapes problems $K = 1, 3, 5, 10, 14$.

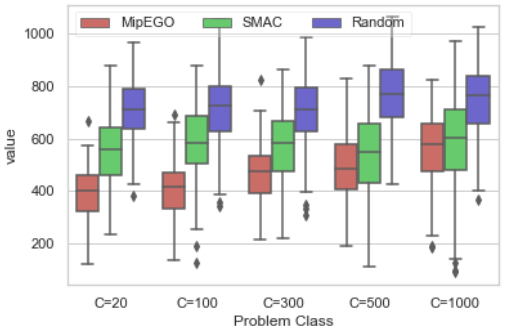


FIGURE 6.3: Box plot for function value on the last evaluation of **barrier functions** with different control parameter $C = 20, 100, 300, 500,$ and 1000 by using MiP-EGO, SMAC, and random search.

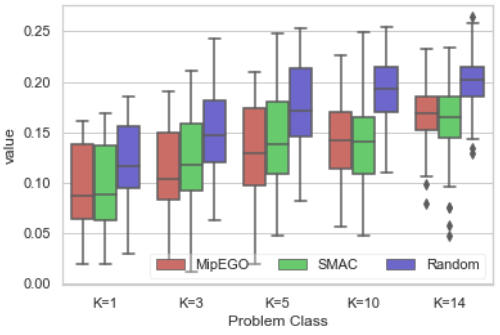


FIGURE 6.4: Box plot for error value on the last evaluation of **mixed integer NK landscape** problems with different control parameter $K = 1, 3, 5, 10, 14$ by using MiP-EGO, SMAC, and random search.

6.2 Algorithm Configuration Scenario

In order to assess the performance of proposed variants of MiP-EGO, we test them on 7 learning algorithms which have 14 hyper-parameters in total, with an additional parameter as algorithm selection indicator. That means the search space of variants of MiP-EGO consisting of 15 hyper-parameters and one level of conditionality. The learning algorithms and their hyper-parameters, as well as default values, implemented in the scikit-learn library [56] include the following:

- **K nearest neighbours** (knn): the number of neighbours n_neighbours *(default=5)* in $\{1, 2, \ldots, 30\}$.
- **RBF SVM** (svm): the penalty C *(default=1.0)* logarithmically scaled in $\left[10^{-5}, 10^{5}\right]$, the width of the RBF kernel γ_{RBF} *(default=1/n_features)* logarithmically in $\left[10^{-5}, 10^{5}\right]$.
- **Linear SVM** (linsvm): the penalty C *(default=1.0)* logarithmically scaled in $\left[10^{-5}, 10^{5}\right]$.
- **Decision tree** (dt): the maximal depth max_depth *(default=None* [1]*)* in $\{1, 2, \ldots, 10\}$, the minimum number of samples required to split an internal node min_samples_split *(default=2)* in $\{2, 3, \ldots, 100\}$, the minimum number of samples required to be at a leaf node. min_samples_leaf *(default=1)* in $\{2, 3, \ldots, 100\}$.

- **Random forest** (rf): the number of trees n_estimators *(default=10)* in $\{1, 2, \ldots, 30\}$, the domain of max_depth, min_samples_split and min_samples_leaf are same as those in decision tree.
- **AdaBoost** (adab): the number of weak learners n_estimators *(default=50)* in $1, 2, \ldots, 30$.
- **Quadratic Discriminant Analysis** (qda): the regularization reg_param *(default=0.0)* logarithmically in $\left[10^{-3}, 10^{3}\right]$.

The algorithm selection hyper-parameter that determines which underlying hyper-parameter should be active or inactive, is represented as an integer.

All of the learning algorithms are used to solve a series of 14 classification problems taken from UCI and OpenML [2]. The size of those data sets ranges from 500 to 60K and the feature dimension range from 3 to 256. All the data sets were pre-processed to have zero mean and unit standard deviation [3]. The performance of chosen hyper-parameters is measured with 5-fold cross-validation. Each optimization trial is repeated 10 times with different random seeds.

As for the experimental setup of the optimization algorithms that will be compared, we use the expected improvement as the infill-criterion and allocate a total of 200 observations for each method. In addition to three variants of MiP-EGO introduced in the previous chapter, we design another two strategies for comparison. The first one is called **EGO-baseline**, that means, hyper-parameters of each learning algorithm are optimized by MiP-EGO separately within $\lfloor 200 \div 7 \rfloor = 28$ evaluation budgets, and the best result among all learning algorithms is recorded. In this case, the algorithm selection procedure is replaced by a manual implementation, instead of introducing an extra parameter. The second one is **Sklearn**, that is, we run all learning algorithms with scikit-learn default hyper-parameters and record their best result. Repeat this procedure 10 times because

[3] From UCI: Adult (adlt), Bank (bnk), Letter (ltr), Magic (mgic), Page-blocks (p-blk), Pima (pim), Semeion (sem), Spambase (spam), Stat-german-credit (s-gc), Stat-image (s-im), Stat-shuttle (s-sh), Titanic (tita). Datasets identified by numbers were taken from OpenML, the numbers represents their ID in the OpenML database.

[3] All UCI datasets are already processed by Delgado et al. [57], so Lévesque et al. [54] and we use the exact same UCI data for algorithms comparision. For OpenML dataset, we just use same preprocessing technique.

[3] "If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples." (see https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)

algorithms are stochastic. The complete codes and experimental data of each method can be found in the github [4].

The other class of methods which we will compare are denoted as GP-X-(ls), proposed by [54]. They are GP-based Bayesian optimization, combined with imputation strategy for inactive values in conditional search space. Each method is different in terms of kernel type, whether using local search (ls) to optimize *EI* and applying imputation or not. Their GP-based methods include the use of priors on GP hyper-parameters (i.e. length-scales ℓ, the kernel amplitude σ_f and the noise σ_n), which are then tuned with slice sampling.

In the work of Lévesque et al. [54], random search (RS), Spearmint [15], SMAC, CMA-ES [58] are also taken into account for comparison, so we will keep comparing them with our proposed methods. As for random search, hyper-parameters are sampled uniformly in the search space. For SMAC, inactive hyper-parameter is imputed to a default value (which is roughly the middle value of each space). For CMA-ES, the population size used is $\lambda = 4 + 3 \log d$, which is suggested as a rule of thumb in [59]. No imputation of inactive parameters is performed.

Based on the work of Demšar et al. [60], we use a set of robust non-parametric tests for statistical comparisons of all methods: the Wilcoxon signed ranks test for comparison of two methods and the Friedman test with the corresponding post-hoc tests for comparison of more methods over multiple data sets. Results of the latter will also be presented with CD (critical difference) diagrams.

6.2.1 Results and Discussions

Table 6.1 shows the empirical validation error rate of each method on the dataset, averaged over 10 repetitions. The best performance per dataset is highlighted in bold, and for each dataset the methods that are significantly worse than the best according to a Wilcoxon signed-rank test with $p < 0.05$ are underlined.

[4] https://github.com/guoxin1122/EGO-X

We can see that GP-cond-ls is only significantly worse than the best on 3 datasets, while the member of EGO-X is worse on at least 6 datasets. From this perspective, GP-cond-ls seems better than the others. With respect to average ranks, we can see the best method is EGO-ss (5.11), followed by EGO-baseline (5.18), and GP-cond-ls (5.39).

When compared all together, the methods are significantly different according to a Friedman's test ($p = 3.67 \times 10^{-6}$). In this case, a post-hoc multiple comparisons test need to be performed. Figure 6.5 shows the result of a post-hoc Nemenyi test with the significance level $p = 0.05$. The performance of two methods is significantly different if the corresponding average ranks differ by at least CD (which is computed based on the formula in [60]).

From these tests, we can only conclude that four methods (i.e. EGO-ss, EGO-baseline, GP-cond-ls, EGO-ws) all perform significantly better than Spearmint. Methods which are linked by one bold line seem to have equivalent performances. The data is not sufficient to conclude whether methods crossed by two bold lines are equivalent to Spearmint or the best four methods. Similarly, it is not sufficient to conclude that SMAC's performance is same as that of EGO-ss or Spearmint. It should be pointed out that the Nemenyi test is rather conservative in its estimates.

Table 6.2 shows the result of pairwise Wilcoxon signed-rank tests. Methods highlighted in bold show a significant difference between row and column, and underlines highlight when the method on the corresponding row performs worse than the method in the corresponding column. From these tests, it is interesting to note that EGO-ss significantly outperforms 8 methods and it is never worse than any of the others. From this perspective, we can say EGO-ss is the best, and the second best is GP-cond-ls which is significantly outperformed by only two methods, while the worst is Spearmint which is significantly outperformed by all others.

6.2.2 Visualization of Classifier Choices

The frequency of the final sampled classifier choice can provide insights into the behaviour of combined algorithm selection and hyper-parameter optimization, once the optimization procedure is over. Here we look at the results for a single optimization method for simplicity, EGO-ss. Figure 6.6 shows the distribution of classifier choices

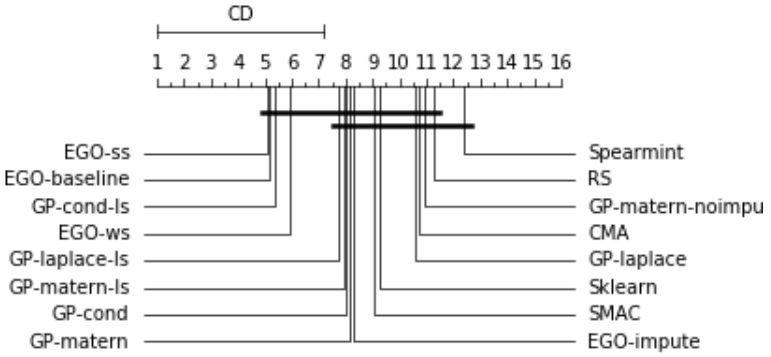


FIGURE 6.5: Comparison of all methods against each other with the Nemenyi test. Groups of methods that are not significantly different ($p > 0.05$) are connected. The average ranks of methods are drawn on the axis. Here CD=6.17 .

TABLE 6.1: Average validation error (%) of each method. Boldface highlights the best performing method per dataset, and underlined methods highlight methods that are significantly different from the best method according to a Wilcoxon signed-rank test ($p < 0.05$).

	389	772	adlt	bnk	ltr	mgic	p-blk	pim	s-gc	s-im	s-sh	sem	spam	tita	Rank
Sklearn-default	17.15	44.44	**14.0**	10.44	5.42	13.07	4.0	22.68	23.28	2.93	**0.03**	4.71	7.22	26.26	9.25
RS	14.20	46.61	14.44	10.70	3.14	12.76	3.34	23.83	23.25	3.77	0.09	6.07	5.67	24.06	11.25
Spearmint	20.32	44.63	15.53	10.83	8.81	13.61	3.17	23.44	23.10	4.52	0.86	9.47	6.86	21.50	12.39
SMAC	14.22	44.63	14.42	10.59	2.83	12.66	3.48	23.38	23.45	3.31	0.08	4.40	5.84	22.70	9.04
CMA	14.26	44.66	14.48	10.64	3.04	12.66	3.09	23.96	23.75	3.66	0.07	6.10	5.69	23.74	10.75
GP-matern-noimpute	17.52	44.63	14.50	10.61	3.87	12.40	3.05	23.51	23.50	3.42	0.08	8.98	5.87	23.79	10.93
GP-matern	14.10	45.32	14.36	10.72	3.16	12.53	2.55	23.70	23.35	3.51	0.05	4.98	5.27	21.36	8.18
GP-cond	15.48	44.91	14.33	10.51	3.47	12.44	2.50	24.29	23.75	2.84	0.04	4.89	5.28	21.77	8.0
GP-laplace	17.08	44.89	14.39	10.51	8.18	13.03	2.72	24.68	23.45	3.27	0.09	7.43	5.28	20.93	10.57
GP-matern-ls	20.10	45.60	14.39	10.35	3.48	12.49	2.60	24.61	23.10	2.71	0.03	5.29	5.01	21.52	7.96
GP-cond-ls	14.12	44.56	14.34	10.45	2.62	**12.22**	2.54	23.70	23.05	**2.64**	0.57	4.74	**4.86**	21.72	5.39
GP-laplace-ls	14.68	44.89	14.38	10.20	4.19	12.72	**2.49**	23.51	23.30	3.23	0.04	7.59	5.17	**20.45**	7.75
EGO-baseline	14.02	43.53	14.3	10.07	2.42	12.46	3.9	22.04	22.01	2.82	0.07	4.69	7.04	21.65	5.18
EGO-impute	**14.01**	43.48	14.35	10.27	3.29	13.06	3.97	22.08	22.54	2.85	0.1	6.28	7.04	22.16	8.32
EGO-ws	14.17	**43.47**	14.35	10.07	2.38	12.45	3.92	21.99	21.99	2.72	0.09	4.32	7.3	22.04	5.93
EGO-ss	14.17	43.49	14.35	**10.04**	**2.38**	12.35	3.87	**21.73**	**21.72**	2.7	0.09	**4.3**	7.24	21.87	5.11

over all datasets and repetitions. We can see that the best performing classifier found by EGO-ss is SVM with an RBF kernel. Other classifiers are picked far less than SVM.

Figure 6.7 provides further details, showing the classifier choices for each dataset in the benchmark. In this case, each column represents the 10 final classifier choices made for a single dataset. It is interesting to note that there are 6 out of 14 datasets only benefiting from SVM classifier, while others benefit from at least two types of classifiers. It should not be surprising that SVM does perform competitively well on a big proportion of UCI datasets, according to the work of Delgado et al [57]. They evaluate 179 classifiers belonging to a wide collection of 17 families on 121 UCI datasets, and conclude that the best and most general classifiers are random forest and SVM.

TABLE 6.2: Pairwise Wilcoxon signed-rank tests. Boldface indicates statistical significant difference between row and column, while underline indicates row worse than column.

	RS	Spearmint	SMAC	CMA	GP-matern-noimpute	GP-matern	GP-cond	GP-laplace	GP-matern-ls	GP-cond-ls	GP-laplace-ls	EGO-baseline	EGO-impute	EGO-ss	EGO-ws	Sklearn-default
RS		**0.16**	0.07	0.36	0.83	**0.01**	0.20	0.86	0.14	**0.00**	0.20	0.02	0.25	**0.02**	**0.02**	0.64
Spearmint	0.16		**0.04**	0.05	**0.04**	**0.01**	**0.02**	**0.02**	**0.04**	**0.00**	**0.00**	**0.00**	**0.01**	**0.00**	**0.00**	0.10
SMAC	0.07	**0.04**		0.05	0.06	1.00	0.92	0.42	0.88	0.06	0.88	0.07	0.71	**0.04**	**0.04**	0.25
CMA	0.36	0.05	0.05		0.88	**0.03**	0.31	0.68	0.35	**0.01**	0.30	**0.03**	0.32	**0.03**	**0.03**	0.55
GP-matern-noimpute	0.83	**0.04**	0.06	0.88		0.10	0.06	0.51	0.30	**0.01**	0.05	**0.02**	0.07	**0.02**	**0.03**	0.92
GP-matern	**0.01**	**0.01**	1.00	**0.03**	0.10		0.88	0.16	0.43	0.08	0.73	0.16	0.92	0.25	0.33	0.40
GP-cond	0.20	**0.02**	0.92	0.31	0.06	0.88		0.06	0.59	**0.01**	0.55	0.14	0.97	0.14	0.25	0.27
GP-laplace	0.86	**0.02**	0.42	0.68	0.51	0.16	0.06		0.31	**0.02**	**0.01**	0.06	0.18	0.07	0.07	0.35
GP-matern-ls	0.14	**0.04**	0.88	0.35	0.30	0.43	0.59	0.31		0.05	0.88	0.20	0.92	0.16	0.22	0.58
GP-cond-ls	**0.00**	**0.00**	0.06	**0.01**	**0.01**	0.08	**0.01**	**0.02**	0.05		0.17	0.36	0.47	0.43	0.51	0.12
GP-laplace-ls	0.20	**0.00**	0.88	0.30	0.05	0.73	0.55	**0.01**	0.88	0.17		0.18	0.78	0.18	0.27	0.47
EGO-baseline	**0.02**	**0.00**	0.07	**0.03**	**0.02**	0.16	0.14	0.06	0.20	0.36	0.18		**0.01**	0.33	0.97	**0.01**
EGO-impute	0.25	**0.01**	0.71	0.32	0.07	0.92	0.97	0.18	0.92	0.47	0.78	**0.01**		**0.02**	**0.04**	0.05
EGO-ss	**0.02**	**0.00**	**0.04**	**0.03**	**0.02**	0.25	0.14	0.07	0.16	0.43	0.18	0.33	**0.02**		**0.01**	**0.01**
EGO-ws	**0.02**	**0.00**	**0.04**	**0.03**	**0.03**	0.33	0.25	0.07	0.22	0.51	0.27	0.97	**0.04**	**0.01**		**0.01**
Sklearn-default	0.64	0.10	0.25	0.55	0.92	0.40	0.27	0.35	0.58	0.12	0.47	**0.01**	0.05	**0.01**	**0.01**	

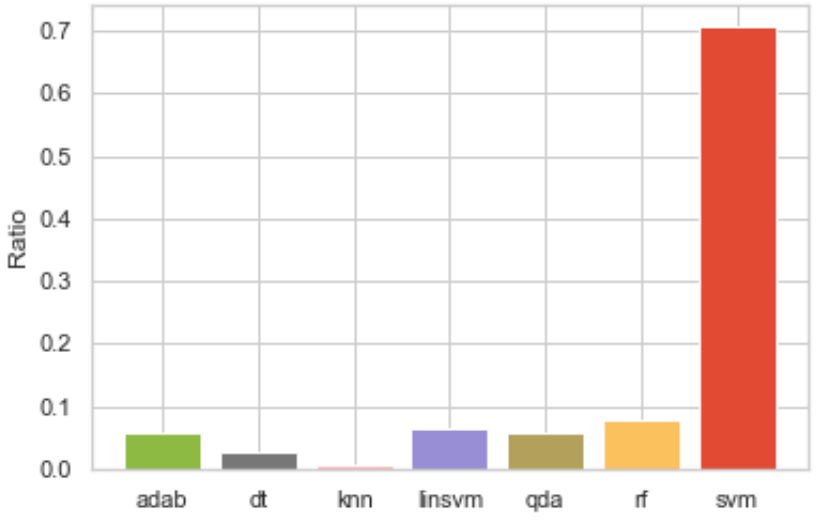


FIGURE 6.6: Resulting classifier choices for an optimization with EGO-ss over all 14 datasets and 10 repetitions.

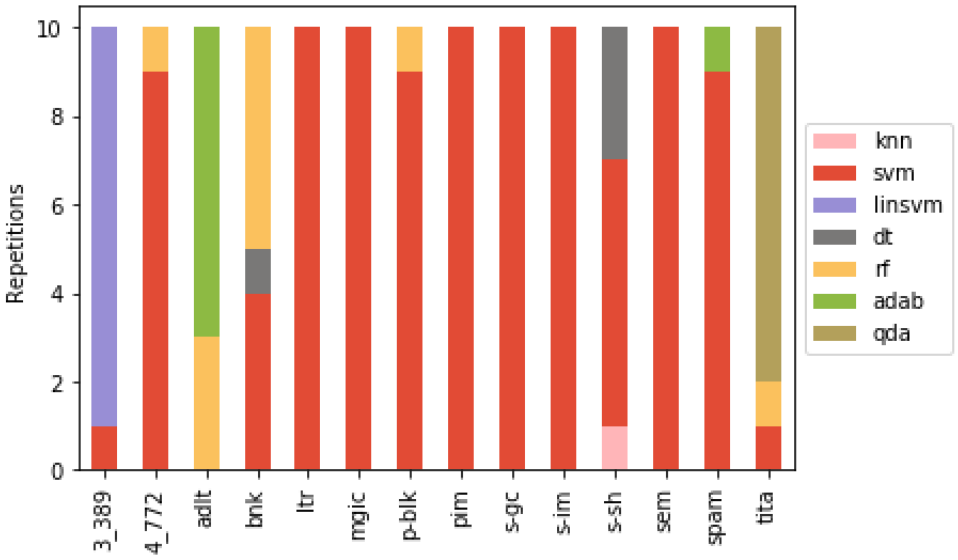


FIGURE 6.7: Resulting classifier choices for an optimization with EGO-ss. Each column represents the distribution of classifier choices across the 10 repetitions on a single dataset.

Chapter 7

Summary and Conclusions

We have investigated the difference and relation between the black-box optimization problem and algorithm configuration problem in the context of automatically tuning algorithm parameters. And we empirically evaluate two state-of-the-art algorithms, MiP-EGO and SMAC, which are originated from the field of black-box optimization and algorithm configuration, respectively, on some artificial test problems. Our results show that MiP-EGO outperforms SMAC on mixed integer search spaces within a small evaluation budget, and both algorithms cost roughly the same amount of time to approach respective near-optimum.

Not just optimizing parameters of one single algorithm, we extend our scope to include the choice of the algorithm, in order to solve the problem about automatically selecting the best one from a large number of highly optimized algorithms, for an unknown practical application. We combine the algorithm selection procedure and parameter optimization on a hierarchical search space, and propose three extensions of MiP-EGO to solve this hybrid problem.

We apply the variants of MiP-EGO, namely EGO-X, on a algorithm configuration scenario where a best highly optimized learning algorithm is determined automatically for solving a classification task. We use EGO-X to tune 7 learning algorithms simultaneously and test them on 14 classification datasets. In multi-comparison with other methods, the results show that EGO-ss, EGO-ws and EGO-baseline are not significantly worse than the state-of-the-art. In pairwise comparison, our results show that the best one is EGO-ss, which is never outperformed by all others, followed by EGO-baseline,

GP-cond-ls and EGO-ws. The better performance of EGO-ss can be explained by the ability of sufficiently and thoroughly optimizing hyper-parameters of learning algorithm before selection.

Although EGO-ss outperforms others on most datasets and has highest ranking, we do not tend to use these results for claiming a general presumed superiority of EGO-ss against the other algorithms. Rather, we wish to present the idea of applying MiP-EGO on hierarchical search space, and we want to interpret our results as an evidence that this idea is promising for configuring and selecting learning algorithms, and can be further applied on more extensive and practical problems.

Bibliography

[1] Bas van Stein, Hao Wang, and Thomas Bäck. Automatic configuration of deep neural networks with EGO. *arXiv preprint arXiv:1810.05526*, 2018.

[2] Michael Emmerich, Monika Grötzner, Bernd Groß, and Martin Schütz. Mixed-integer evolution strategy for chemical plant optimization with simulators. In *Evolutionary Design and Manufacture*, pages 55–67. Springer, 2000.

[3] Yolanda Mack, Tushar Goel, Wei Shyy, and Raphael Haftka. Surrogate model-based optimization framework: a case study in aerospace design. In *Evolutionary computation in dynamic and uncertain environments*, pages 323–342. Springer, 2007.

[4] Andrew J Booker, JE Dennis, Paul D Frank, David B Serafini, and Virginia Torczon. Optimization using surrogate objectives on a helicopter test example. In *Computational Methods for Optimal Design and Control*, pages 49–58. Springer, 1998.

[5] Sigrún Andradóttir. Simulation optimization. *Handbook of simulation: Principles, methodology, advances, applications, and practice*, pages 307–333, 1998.

[6] Timothy W Simpson, Timothy M Mauery, John J Korte, and Farrokh Mistree. Kriging models for global approximation in simulation-based multidisciplinary design optimization. *AIAA journal*, 39(12):2233–2241, 2001.

[7] Jerome Sacks, William J Welch, Toby J Mitchell, and Henry P Wynn. Design and analysis of computer experiments. *Statistical science*, pages 409–423, 1989.

[8] Frank Hutter. *Automated configuration of algorithms for solving hard computational problems*. PhD thesis, University of British Columbia, 2009.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[10] Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, pages 1–47, 2018.

[11] Bas van Stein, Hao Wang, and Thomas Bäck. Automatic configuration of deep neural networks with EGO. *CoRR*, abs/1810.05526, 2018. URL http://arxiv.org/abs/1810.05526.

[12] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[13] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 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*, pages 847–855. ACM, 2013.

[14] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[15] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[16] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pages 2171–2180, 2015.

[17] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

[18] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[19] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 497–504. ACM, 2017.

[20] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

[21] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[22] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 481–488. ACM, 2017.

[23] Shih-Wei Lin, Kuo-Ching Ying, Shih-Chieh Chen, and Zne-Jung Lee. Particle swarm optimization for parameter determination and feature selection of support vector machines. *Expert systems with applications*, 35(4):1817–1824, 2008.

[24] Ilya Loshchilov and Frank Hutter. CMA-ES for hyperparameter optimization of deep neural networks. *arXiv preprint arXiv:1604.07269*, 2016.

[25] Oden Maron and Andrew W Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1-5):193–225, 1997.

[26] 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*, pages 108–122. Springer, 2007.

[27] Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 11–18. Morgan Kaufmann Publishers Inc., 2002.

[28] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-Race and iterated F-Race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

[29] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[30] Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

[31] Richard Webster and Margaret A Oliver. *Geostatistics for environmental scientists*. John Wiley & Sons, 2007.

[32] Noel Cressie. Statistics for spatial data. *Terra Nova*, 4(5):613–617, 1992.

[33] Michael L Stein. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 2012.

[34] Hans Wackernagel. *Multivariate Geostatistics: An Introduction with Applications*. Springer Science & Business Media, 2013.

[35] G Matheron. The theory of regionalised variables and its applications. *Les Cahiers du Centre de Morphologie Mathématique*, 5:212, 1971.

[36] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[37] Carla Currin, Toby Mitchell, Max Morris, and Don Ylvisaker. Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, 86(416):953–963, 1991.

[38] Michael James Sasena. *Flexibility and efficiency enhancements for constrained global design optimization with kriging approximations*. PhD thesis, Citeseer, 2002.

[39] Hao Wang et al. *Stochastic and deterministic algorithms for continuous black-box optimization*. PhD thesis, 2018.

[40] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:, 2001.

[41] Jonas Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

[42] Emmanuel Vazquez and Julien Bect. Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. *Journal of Statistical Planning and inference*, 140(11):3088–3095, 2010.

[43] Matthias Schonlau, William J Welch, and Donald R Jones. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series*, pages 11–25, 1998.

[44] Antanas Žilinskas. A review of statistical models for global optimization. *Journal of Global Optimization*, 2(2):145–153, 1992.

[45] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[46] Hao Wang, Bas van Stein, Michael Emmerich, and Thomas Back. A new acquisition function for Bayesian optimization based on the moment-generating function. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 507–512. IEEE, 2017.

[47] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4): 455–492, 1998.

[48] Daniel James Lizotte. *Practical Bayesian optimization*. University of Alberta, 2008.

[49] Thomas J Santner, Brian J Williams, and William I Notz. *The design and analysis of computer experiments*. Springer Science & Business Media, 2013.

[50] Rui Li, Michael TM Emmerich, Jeroen Eggermont, Thomas Bäck, Martin Schütz, Jouke Dijkstra, and Johan HC Reiber. Mixed integer evolution strategies for parameter optimization. *Evolutionary computation*, 21(1):29–64, 2013.

[51] Hao Wang, Michael Emmerich, and Thomas Bäck. Cooling strategies for the moment-generating function in Bayesian global optimization. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

[52] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Kevin Murphy. Time-bounded sequential parameter optimization. In *International Conference on Learning and Intelligent Optimization*, pages 281–298. Springer, 2010.

[53] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

[54] Julien-Charles Lévesque, Audrey Durand, Christian Gagné, and Robert Sabourin. Bayesian optimization for conditional hyperparameter spaces. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 286–293. IEEE, 2017.

[55] Rui Li, Michael TM Emmerich, Jeroen Eggermont, Ernst GP Bovenkamp, Thomas Bäck, Jouke Dijkstra, and Johan HC Reiber. Mixed-integer NK landscapes. In *Parallel Problem Solving from Nature-PPSN IX*, pages 42–51. Springer, 2006.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[57] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[58] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[59] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[60] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.