POLITECNICO DI TORINO

Master of Science in
Mathematical Engineering

Discrete Bayesian Optimization Algorithms and Applications

Supervisors

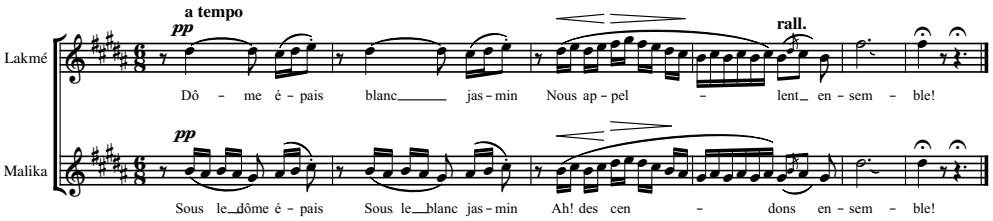
Prof. Giacomo Como

Prof. Fabio Fagnani

Candidate

Raffaele Damiano

Academic Year 2019-2020



Léo Delibes, Duo des fleurs

Abstract

Dealing with expensive-to-evaluate objective functions is a hard problem in optimization. Bayesian Optimization (BO) is a methodology allowing one to efficiently approximate the objective function and perform the optimization with as few evaluations as possible. This is achieved by introducing a surrogate model, i.e., a statistical model for the objective function, and an acquisition function that let us move through the feature space. The most common surrogate models are Gaussian Processes. While BO algorithms based on Gaussian Processes typically perform well over continuous domains, these techniques prove not so efficient when dealing with discrete or categorical variables and different approaches and settings are required.
In this thesis, the Separable Bayesian Optimization algorithm (SBO) is proposed to overcome the limitations of classical BO. It moves from the idea of considering the discrete variables as nodes of a graph, over which a statistical model is built. This model introduces a certain structure in order to seek approximations of the objective function. The main steps of BO are then adapted exploiting the properties of Gaussian Processes.
The thesis starts with an overview of Bayesian Optimization and its main components: surrogate model and acquisition function. In particular, the emphasis is on Gaussian Processes as surrogate models. Afterwards, the SBO algorithm is presented in all its theoretical steps and a description of its implementation using Python 3 is also provided. The second part of the thesis deals with some applications of SBO to discrete and black-box optimization problems. A case study proposed by the TIM group is then addressed. The problem is to design a self-optimizing mobile network that is capable of setting the best configuration of tilts for its antennas in order to get the best performance in terms of capacity and coverage (Capacity and Coverage Optimization).

Contents

Chapter 1

Introduction

The optimization of a black-box function is a challenging problem in many modern applications. Consider the following

$$\max_{x \in \mathcal{X}} F(x),$$

where $F(\cdot)$ is a costly function to evaluate. Problems of this type are really common in machine learning techniques such as neural networks. ML algorithms involve the tuning of learning parameters or model hyper-parameters, which may be hard to set and many tries may be required (Snoek et al. [35], Shahriari et al. [33], Bergstra et al. [8]). On the one side, it is possible to reduce the number of parameters to the minimum in order to decrease the number of tries. On the contrary, one can try to automatize the tuning. The second choice can be seen as a black-box optimization problem in which the objective function is the performance of the ML algorithm, obtained with a set of parameters. Clearly, this objective function is expensive to evaluate since each evaluation is done with a running of the ML algorithm itself.

In general, Artificial Intelligence is full of expensive-to-evaluate objective functions. As an example, some object detection systems, like the C-RNN framework [16], use expensive-to-evaluate detection score function. An object detection system allows one to locate objects in images and is mainly based upon neural networks. An object detection algorithm first tries to fill an image with many bounding-boxes. Then, for each region a classification algorithm produces a detection score, which explains how well the algorithm detected an object in the specified region. This step is repeated with different bounding-boxes arrangements. An object is detected when the detection score is maximized. Notice that each evaluation of the objective function

depends on the computational cost of the classification algorithm and more importantly, finding a global maximum is expensive due to the huge number of regions that the neural network generates.

Bayesian Optimization (BO) is a large class of methods that efficiently optimize black-box functions. It is useful when a closed-form for the objective function is not available and obtaining observations of this function at sampled values is expensive. BO methods are able to exploit the very little information we give to the algorithms at the initialization and try to obtain the optimal value asking for the smallest number of evaluations of the objective function. To achieve this goal, the cornerstone of Bayesian Optimization is to gain knowledge step-by-step. The approach is similar to that of Bayes' Theorem, after which BO is named: the posterior probability of a model M given evidence E is proportional to the prior probability of M multiplied by the likelihood of E given M:

$$p(M|E) \propto p(E|M) \cdot p(M).$$

In the Bayesian optimization, the prior represents our belief about the space of possible objective functions. The likelihood expresses our belief over the properties of the objective function, for example its smoothness, which lets us modify the prior and make some possible objective functions more plausible than others. Each time we gather new observations, the support of the distribution over the space of objective function becomes smaller, and centered on the true objective function. Bayesian Optimization models are made up of two building blocks: a surrogate model and an acquisition function. The *surrogate model* is a statistical model which has the role of approximating the black-box objective function. It provides the probability distribution over the search space defining a prior distribution and the likelihood of the model. The most used surrogate models are *Gaussian Processes* because of their properties of conditioning and marginalization, useful in defining a posterior distribution in closed form. Instead, the *acquisition function* is responsible of finding the next observation to be evaluated. The new observation allows one to update the information given by the surrogate model.

Brochu et al. [10] traces the roots of modern Bayesian Optimization approach in the works of Kushner (1964) [23], Mockus (1974) [28] and Krige (1951) [22]. However, BO received more attention after the work of Jones et al. (1998) [20] where the authors proposed the Efficient Global Optimization algorithm (EGO) to optimize expensive black-box functions. Since

then, several works have been produced and lots of BO models arose. For example in the field of material science, Zhang et al. (2015) [41] and Frazier et al. (2015) [13] applied Bayesian Optimization to the design of material systems. In robotics, Lizotte et al. (2007) [25] addressed gait optimization of quadrupedal and bipedal robots by using the BO approach. Instead, Brochu et al. (2010) [10] applied Bayesian Optimization to Active user modelling with preferences and hierarchical Reinforcement Learning. In Machine Learning Snoek et al. (2012) [35] were the first to notice the usefulness of Bayesian Optimization in the tuning of hyperparameters of ML algorithms, especially deep neural networks. Many other works investigate BO in Machine Learning applications: as an example Bergstra et al. (2013) [8], Wang et al. (2016) [39], Binois et al. (2018) [9] extended the approach to high-dimensional problems, while Swersky et al. (2013) [36] and Toscano-Palmerin et al. (2018) [38] developed multi-task Bayesian Optimization. The latter is an extension of multi-tasking Gaussian Processes to the BO approach: this model allows one to find optimal hyperparameter settings more efficiently and is useful for sequential design of experiments with random environmental conditions. Lastly, other theoretical treatments of Bayesian Optimization have been done by Huang et al. (2006) [18], Shahriari et al. (2016) [33], Frazier (2018) [12] and Oh et al. (2019) [30].

As experimented in the above-mentioned works, Bayesian Optimization works well when the features domain is a continuous subset of $\mathbb{R}^d$ ($d \leq 20$). However, for discrete or categorical domains, these techniques become unsuccessful and, in general, discrete optimization requires different approaches and settings. To deal with discrete or categorical variables in black-box optimization problems, several algorithms have been proposed in literature. As an example, Hutter et al. (2011) [19] proposed the SMAC algorithm (Sequential Model-based Algorithm Configuration) to solve *algorithm configuration* problems. It uses random forests as surrogate models and random walks to obtain local optimum. Notice that Bayesian Optimization is a particular class of sequential model-based algorithms. Garrido-Merchán et al. (2020) [15] suggested a model that is based on rounding to the closest integer value. They considered the objective to be constant in the interval of rounding of a discrete value. For example, if the discrete variable x varies in $\mathbb{N}$, then on the interval $(4.5, 5.5) \subset \mathbb{R}$ the objective is assumed to have the same value as in 5, since all the values of the interval are rounded to

5. Suddenly, this leads to a step-wise acquisition function, which is difficult to optimize. Other algorithms are provided by Bergstra et al. (2013) [7], Baptista et al. (2018) [5] with the BOCS algorithm, Oh et al. (2019) [31] with the COMBO algorithm and Luong et al. (2019) [26] (2019) with the Discrete-BO approach. The last three algorithms will be widely discussed in Chapter 3.

In this thesis we propose a tailor made algorithm that combines the idea of the Bayesian optimization algorithms with a structural setting that allows us to work over combinatorial domains. We will be referring to it as the SBO algorithm, where SBO stands for Separable Bayesian Optimization. The key point of the algorithm is assuming that the objective function has a defined structure. In particular, considering the complete graph whose nodes are the discrete variables involved, the objective function is assumed to be additively separable: substantially, it can be written as the summation of several random variables built onto the complete graph itself. These random variables are assumed to follow a Gaussian distribution, hence the surrogate model is still a Gaussian Process. However, Gaussian Processes do not work properly over discrete domains. In the SBO algorithm we will adjust the steps of Bayesian Optimization with Gaussian Processes, in order to tailor them to the discrete world.

In Chapter 2 we will explain how Bayesian Optimization works over continuous domains: we will discuss about Gaussian Processes as surrogate models and Acquisition functions.

In Chapter 3 the problem setting is outlined. We will present some examples of discrete black-box problems and then we will discuss some algorithms that have been proposed by researchers to overcome the limitation that BO has over discrete domains.

Chapter 4 will be dedicated to the SBO algorithm, our solution to the limitations of BO over discrete domains: there will be both a theoretical and an implementation part. Appendix B will complete the technical discussion, deepening into the implementation part with Python 3.

In Chapter 5 the SBO algorithm is tested over some benchmark problems proposed in literature.

Lastly, in Chapter 6 we will apply the SBO algorithm to a real applicative context, a mobile network problem, proposed by the TIM group.

Chapter 2

Background on Bayesian Optimization

In this chapter we will give an overview of Bayesian Optimization (BO), a powerful approach that allows one to optimize expensive-to-evaluate functions. Consider the following optimization problem:

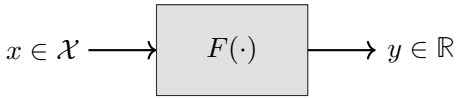
$$\max_{x \in \mathcal{X}} F(x) \tag{2.1}$$

in which the objective function $F(\cdot)$ has not a closed form expression or it is expensive to obtain observations at sampled points. These kind of objective functions are referred to as black-box functions. Several works demonstrated that Bayesian Optimization is able to optimize such problems and to find the optimum in few steps [10, 12, 28, 33, 35]. Bayesian optimization is made up of two main parts: a statistical model for modelling the objective function (a *surrogate model*) and an *acquisition function* to choose a point from the domain where the algorithm will sample next. Here we will focus on BO that uses Gaussian Processes as surrogate models.

The development of this chapter is based mainly on the works of Frazier [12], Borchu et al. [10] and Agnihotri [1].

2.1 The core idea of Bayesian Optimization

Suppose we have a black-box function $F(\cdot) : \mathcal{X} \to \mathbb{R}$ in problem (2.1) as in the following figure.



Black-box

We will refer to x as *input* of the black-box function, and to the couple (x, y) as *observation*. Bayesian Optimization aims at finding the maximum value that F can reach over the $\mathcal{X}$ domain. Since F is unknown or expensive to evaluate, BO needs to find the best approximation of F in the objective function space. To do that, BO relates on a step-by-step learning procedure. The initial information required by this procedure in each BO algorithm are:

- a train set of observations $(x_i, y_i = F(x_i))$, for $i = 1, \ldots, n$, which allows one to select only those objective functions that interpolates all the points of the train set;
- a prior distribution over the objective function space, which let us determine which objective functions are more plausible;
- a likelihood function, which expresses our beliefs over the properties of the objective function, for example its continuity and smoothness.

The last two items are specified by providing a surrogate model to the BO algorithm. A surrogate model is a statistical model that handles the probability distribution over the objective function space. The step-by-step learning procedure consists in re-shaping the prior distribution each time a new observation is added to the train set. The surrogate model also specifies the way BO updates the prior: it provides both the prior and the posterior distributions.

BO uses the re-shaping procedure in order to gather all possible informations from the minimum number of evaluations (i.e. observations). Moreover, it is the BO algorithm itself that chooses which point should be sampled next. This step is managed by the acquisition function and its maximization. The acquisition function looks at the shape of the posterior distribution over the objective function space and decides which is the most promising point. In essence, the core question of Bayesian optimization is "by what we know so far, where should we evaluate next?".

We can sum up BO algorithms into the following four steps:

1. at first choose a surrogate model for modelling the true objective function and define its prior;
2. given the set of observations, use Bayes rule to obtain the posterior;
3. use an acquisition function $\alpha(x)$, which is a function of the posterior, to decide the next sample point $x_{n+1} = \text{argmax}_x \alpha(x)$;
4. add the new sampled couple $(x_{n+1}, F(x_{n+1}))$ to the set of observations and go to step 2 until convergence or budget elapses.

2.2 Gaussian Processes

When $\mathcal{X}$ is a continuous subset of $\mathbb{R}^d$ in (2.1), then the most used surrogate models are *Gaussian Processes* (GP). Gaussian Processes simplify the computation of the posterior from the prior in the second step of BO algorithms. In fact, GP provides a closed form expression for the conditional distribution over the objective function space, conditioned to a new observation.

Firstly, let us introduce Gaussian Processes. A Gaussian process is a stochastic process $Z = \{Z_t : t \in T\}$, $T \subseteq \mathbb{R}$, such that $\forall k \in \mathbb{N}$ and for each $t_1, \ldots, t_k \in T$ we have that $(Z_{t_1}, \ldots, Z_{t_k}) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu} \in \mathbb{R}^k$ is the mean vector and $\Sigma \in \mathbb{R}^{k \times k}$ is the covariance matrix. In other words, a stochastic process Z is a Gaussian process if and only if each vector of random variables taken from the stochastic process Z has a finite multivariate Gaussian distribution. Therefore a GP is completely identified by the mean vector $\boldsymbol{\mu}$ and the covariance matrix Σ. The notation is the following:

$$Z \sim \mathcal{GP}(\boldsymbol{\mu}, \Sigma).$$

More in general, $\boldsymbol{\mu}$ and Σ may not be constant. We can identify a Gaussian process by specifying a mean function $\mu(t) = \mathbb{E}[Z_t]$, $t \in T$, and a covariance function $\Sigma(t, t') = Cov(Z_t, Z_{t'})$, $t, t' \in T$. The covariance function is also called *kernel* and denoted with $k(\cdot, \cdot)$. For further insights on Gaussian Processes in Machine Learning see Rasmussen et al. [32].

In the Bayesian optimization context, the surrogate model attempts to approximate the black-box function at each iteration of the BO loop. As

we said before, it provides the probability distribution over the objective functions space. This is equivalent to give a probability distribution to the output $F(x)$ for each input $x \in \mathcal{X}$. The random variable associated to this probability distribution will be referred to as $f(x)$. Notice that the sample space of $f(x)$ is the output space $\mathbb{R}$. Considering a Gaussian Process as surrogate model means that the collection $\mathcal{F} = \{f(x) : x \in \mathcal{X}\}$ is a Gaussian process. Therefore, $(f(x_1), \ldots, f(x_k))$ is a finite multivariate Gaussian distribution, for every choice of $x_1, \ldots, x_k$ in $\mathcal{X}$. Note that for a general Gaussian process, usually, the index t varies in a time domain T, while here the index set is the set of all possible inputs $\mathcal{X}$.

The main problem is now to define a mean function and a covariance function in order to identify the Gaussian process $\mathcal{F}$. We do this by giving a prior distribution to the process in terms of a mean function and a covariance function. Therefore we write

$$(f(x_1), \ldots, f(x_k)) \sim \mathcal{N}\Big(\big(\mu_0(x_i)\big)_{i=1}^k, \big(\Sigma_0(x_i, x_j)\big)_{i,j=1}^k\Big),$$

where $x_1, \ldots, x_k \in \mathcal{X}$. The function $\mu_0(\cdot) : \mathcal{X} \to \mathbb{R}$ is the mean function and $\Sigma_0(\cdot, \cdot) : \mathcal{X}^2 \to \mathbb{R}$ is the covariance function for the prior distribution. We used the following compact notation:

$$\big(\mu_0(x_i)\big)_{i=1}^k = \begin{pmatrix} \mu_0(x_1) \\ \vdots \\ \mu_0(x_k) \end{pmatrix},$$

$$\big(\Sigma_0(x_i, x_j)\big)_{i,j=1}^k = \begin{pmatrix} \Sigma_0(x_1, x_1) & \cdots & \Sigma_0(x_1, x_k) \\ \vdots & \ddots & \vdots \\ \Sigma_0(x_k, x_1) & \cdots & \Sigma_0(x_k, x_k) \end{pmatrix}.$$

Afterwards, when we obtain new observations we will modify those functions and update the distribution of the GP. In fact, suppose we obtain n observations $(x_i, y_i = F(x_i))$ from the black-box function, which will be considered as *train observations* of the GP. We want to re-shape the distribution of the Gaussian Process over the feature space. Namely, let $x \in \mathcal{X}$ be a new input point of the feature space, we would like to infer over the value of the outcome $F(x)$, given the information we collected and built with the n observations. What we can do is considering the random variable $f(x)$ instead

of the true value $F(x)$. From the GP assumption we know that

$$\begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \\ f(x) \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_0(x_1) \\ \vdots \\ \mu_0(x_n) \\ \mu_0(x) \end{pmatrix}, \begin{pmatrix} \Sigma_0(x_1,x_1) & \cdots & \Sigma_0(x_1,x_n) & \Sigma_0(x_1,x) \\ \vdots & \ddots & \vdots & \vdots \\ \Sigma_0(x_n,x_1) & \cdots & \Sigma_0(x_n,x_n) & \Sigma_0(x_n,x) \\ \Sigma_0(x,x_1) & \cdots & \Sigma_0(x,x_n) & \Sigma_0(x,x) \end{pmatrix} \right),$$

is still a multivariate Gaussian distribution. From the property of conditioning and marginalization of the multivariate Gaussian distribution it derives that

$$f(x) \mid (y_i)_{i=1}^n \sim \mathcal{N}(\mu_n(x), \sigma_n^2(x)), \tag{2.2}$$

where

$$\begin{aligned} \mu_n(x) &= \mu_0(x) + \big(\Sigma_0(x,x_j)\big)_{j=1}^n \Big(\big(\Sigma_0(x_i,x_j)\big)_{i,j=1}^n\Big)^{-1} \big(y_i - \mu_0(x_i)\big)_{i=1}^n, \\ \sigma_n^2(x) &= \Sigma_0(x,x) - \big(\Sigma_0(x,x_j)\big)_{j=1}^n \Big(\big(\Sigma_0(x_i,x_j)\big)_{i,j=1}^n\Big)^{-1} \big(\Sigma_0(x_i,x)\big)_{i=1}^n. \end{aligned} \tag{2.3}$$

A proof of the derivation of (2.2) and (2.3) is provided in Appendix A. This conditional distribution is the posterior distribution and allows us to evaluate the value $f(x)$ for every new observation in the features space. We highlighted the fact that n observations were used to update the prior distribution by modifying the subscript of the mean and covariance functions. Notice that equations (2.2) and (2.3) will be used every time a new observation (x, y) is obtained, by adding it to the train set of observations.

The possibility of obtaining a close form posterior distribution over the objective function space, makes Gaussian Processes very popular as surrogate models in BO algorithms. Moreover, GPs define a space of smooth functions which are notoriously easier to handle with in optimization problems.

2.3 Acquisition Functions

Given a surrogate model and a distribution over the objective function space, Bayesian Optimization algorithms need an *acquisition function* to suggest the most promising point to sample next. The crucial idea in the acquisition function maximization step is the exploration-exploitation balance. In fact, a BO algorithm needs to balance between moving to unexplored

regions of the state space and focusing on regions where we obtained better results. The acquisition function allows us to combine the two ways of exploring the state space, expressing how desirable it is to evaluate a point, based on our present model.

Expected Improvement

The most commonly used acquisition function is the *expected improvement* (EI). The function to maximize is the following:

$$\mathrm{EI}_n(x) = \mathbb{E}_n\left[(f(x) - f_n^*)^+\right], \tag{2.4}$$

where f_n^* is the optimum obtained at the n-th iteration and $\mathbb{E}_n[\,\cdot\,] = \mathbb{E}[\,\cdot \mid x_1, \ldots, x_n, y_1, \ldots, y_n]$ is the expected value conditioned to the posterior distribution built over the $x_1, \ldots, x_n$ inputs and the $y_1, \ldots, y_n$ observations. This posterior distribution is a Gaussian distribution with mean $\mu_n(x)$ and variance $\sigma_n^2(x)$. The idea behind the definition is straightforward. Given the temporary optimum value f_n^* at the n-th iteration of the BO loop, we decide where to sample next by considering the improvement $f(x) - f_n^*$ we can add to the temporary optimum, and choosing the best improvement. This improvement could be negative, therefore the definition involves the positive part of the increment. Moreover, $f(x)$ is a random variable, therefore the best we can do is considering the expected improvement of the positive part of the increment, and so we obtain the definition (2.4). Hence, the maximization problem in point 3 of the Bayesian optimization loop is

$$x_{n+1} = \mathrm{argmax}_x \mathrm{EI}_n(x).$$

Applying the definition of expected improvement, we can write the expected improvement in closed form. In fact, the following result holds[1].

Proposition 2.1. *Let $\mathcal{F} = \{f(x) : x \in \mathcal{X}\}$ be a Gaussian Process with $\mu_n(\cdot)$ and $\sigma_n^2(\cdot)$ as mean and variance functions. Let f^* be the optimum obtained within n iterations of the BO loop that uses $\mathcal{F}$ as surrogate model. Then, for all $x \in \mathcal{X}$ it holds that*

$$EI_n(x) = \sigma_n(x)\varphi\left(\frac{\Delta_n(x)}{\sigma_n(x)}\right) + \Delta_n(x)\Phi\left(\frac{\Delta_n(x)}{\sigma_n(x)}\right), \tag{2.5}$$

[1] The proof is taken from Al-Dujaili [2].

where $\Delta_n(x) = \mu_n(x) - f_n^*$ *and* $\varphi(\cdot)$ *and* $\Phi(\cdot)$ *are the density function and the cumulative function of a standard normal distribution, respectively.*

Proof. Let us write $f(x)$ as $f(x) = \mu_n(x) + \sigma_n(x)z$ with $z \sim \mathcal{N}(0,1)$. Then, from the definition of expected improvement we obtain:

$$\begin{aligned}
\mathrm{EI}_n(x) &= \int (f(x) - f_n^*)^+ \mathrm{d}\varphi = \int_{-\infty}^{\infty} \max\left(f(x) - f_n^*, 0\right) \varphi(\varepsilon)\mathrm{d}\varepsilon = \\
&= \int_{-\infty}^{\infty} \max\left(\mu_n(x) + \sigma_n(x)\varepsilon - f_n^*, 0\right) \varphi(\varepsilon)\mathrm{d}\varepsilon = \\
&= \int_{-\frac{\mu_n(x) - f_n^*}{\sigma_n(x)}}^{\infty} (\mu_n(x) + \sigma_n(x)\varepsilon - f_n^*)\varphi(\varepsilon)\mathrm{d}\varepsilon = \\
&= (\mu_n(x) - f_n^*) \int_{-\frac{\Delta_n(x)}{\sigma_n(x)}}^{\infty} \varphi(\varepsilon)\mathrm{d}\varepsilon + \sigma_n(x) \int_{-\frac{\Delta_n(x)}{\sigma_n(x)}}^{\infty} \varepsilon \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{\varepsilon^2}{2}} \mathrm{d}\varepsilon.
\end{aligned}$$

Then, computing the integrals we obtain:

$$\begin{aligned}
\mathrm{EI}_n(x) &= \Delta_n(x) \left(1 - \Phi\left(-\frac{\Delta_n(x)}{\sigma_n(x)}\right)\right) - \frac{\sigma_n(x)}{\sqrt{2\pi}} \left[\mathrm{e}^{-\frac{\varepsilon^2}{2}}\right]_{-\frac{\Delta_n(x)}{\sigma_n(x)}}^{\infty} = \\
&= \Delta_n(x)\Phi\left(\frac{\Delta_n(x)}{\sigma_n(x)}\right) + \frac{\sigma_n(x)}{\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}\left(\frac{\Delta_n(x)}{\sigma_n(x)}\right)^2} = \\
&= \Delta_n(x)\Phi\left(\frac{\Delta_n(x)}{\sigma_n(x)}\right) + \sigma_n(x)\varphi\left(\frac{\Delta_n(x)}{\sigma_n(x)}\right),
\end{aligned}$$

which is the (2.5). □

Note that in the closed form, the Expected improvement is expressed as a summation of a term of exploration $\sigma_n\varphi(\Delta_n/\sigma_n)$, and a term of exploitation $\Delta_n\Phi(\Delta_n/\sigma_n)$. In fact, there is a trade-off in searching over the whole state space (exploration) and searching within a promising area nearby the temporary optimum, "exploiting" the information we already have. We can improve this trade-off by adding a parameter k that determine the amount of exploration during the optimization. In particular, we define the *modified expected improvement* (mEI) as in (2.5) but with

$$\Delta_n(x) = \mu_n(x) - f_n^* - k. \tag{2.6}$$

The k parameter has the effect of damping the contribution of the mean effect. Therefore, for bigger values of k the second term of (2.5) will be smaller, leading to more exploration. Instead, smaller value of k allows one to exploit more the neighbourhood of the temporary optimum f^*.

Upper Confidence Bound

The second proposed acquisition function is the *upper confidence bound* (UCB). In trying to combine the exploration/exploitation trade-off, the definition arises trivially:

$$\mathrm{UCB}_n(x) = \mu_n(x) + \beta\sigma_n(x). \tag{2.7}$$

In fact, exploitation and exploration result in the GP mean function $\mu_n(\cdot)$ and covariance function $\sigma_n(\cdot)$, respectively. The β hyperparameter weights the importance of both terms, the surrogate mean and the surrogate uncertainty, taking the role that the k parameter has for the expected improvement.

2.4 Further developments and key issues

We introduce Bayesian Optimization in its essential form. As a sequential model-base algorithm, it is adaptable to several variations. For example, many other acquisition functions have been proposed: knowledge-gradient, entropy search, predictive entropy search [12]. Alternatively, one can replace the acquisition function maximization step with other techniques, like random search or Thompson sampling [1].

In literature, many authors tried to get the most from Bayesian Optimization approach in various directions. As an example, some researchers adapted the BO approach to the parallelization technique in order to obtain multiple function evaluations in shorter time (Ginsbourger et al. [17]). Another extension can be done involving some constraints in the general problem (2.1), where also the constraints are expensive to evaluate (Gardner et al. [14]). Swersky et al. [36], Toscano-Palmerin et al. [38] developed multi-task Bayesian Optimization, also called optimization with random environmental conditions, which is applied to the following problems:

$$\max_{x\in\mathcal{X}} \int f(x,w)p(w)\mathrm{d}w \qquad \text{or} \qquad \max_{x\in\mathcal{X}} \sum_{w} f(x,w)p(w)\mathrm{d}w,$$

where f is expensive to evaluate.

Typically Bayesian Optimization is most successful when the dimension d of the domain is not greater than 20 [12]. Directions for research include developing BO models for high-dimensional domains. Some examples of

works that tried to extended the BO approach to high-dimensional problems are Bergstra et al. [8], Binois et al. [9], Wang et al. [39]. Other research directions include theoretical analysis over convergence for BO algorithms and the development of BO with new statistical models, since Gaussian Process are used in most work on Bayesian Optimization. New surrogate models can be useful for certain classes of problems where the objective could be better modelled through other approaches [12].

Last but not least, another development of BO involves discrete domains. Most Bayesian Optimization methods assume the input variables to be continuous, rather than combinatorial. One of the reason is the use of Gaussian Processes, which are built on continuous domains. Though there exists some kernel proposed for combinatorial structures (See Oh et al. [31]) some of the properties of GP are lost, like the smoothness of the objective function. In the following chapter, some recent works on Bayesian Optimization models over discrete domains are analysed.

Chapter 3

Bayesian Optimization over Discrete Domains

Suppose we have to solve the following black-box problem:

$$\max_{x \in \mathcal{X}} F(x), \tag{3.1}$$

where $\mathcal{X}$ is a discrete or categorical domain and the objective function is expensive-to-evaluate. In the previous chapter we analysed the BO approach as a method to optimize black-box objective functions. Bayesian Optimization works well when the features domain is a continuous subset of $\mathbb{R}^d$, with $d \leq 20$. However, for discrete or categorical domains, this technique becomes unsuccessful and different approaches are required.

3.1 Discrete and black-box problem examples

Natural sciences, engineering, Machine Learning are some of the fields in which researchers faces expensive-to-evaluate objective function over set of combinatorial structures. An example comes from the field of pharmacology: drug discovery (Negoescu et al. [29]). To discover a new medication, medical researchers often start with single molecule that shows some desirable therapeutic effect. Then they test many variations of that molecule to find one that produces the best results. These variations are obtained by substituting some atoms in the original molecule with different atoms. The number of variations increases exponentially in the number of atoms of the molecule.

Testing each variation is infeasible due to the time limitation (each variation has to be synthesized and analysed) or budget constraints.

Another example involves Multidisciplinary models. The design of complex systems like aircraft, spacecraft or wind turbines, involves the work of many specialists in different disciplines. As design tasks becomes more decentralized, disciplinary specialists are needed. Their job is to build a communication network in order to make the transfer of informations easier and optimized. Multidisciplinary analysis and optimization (MDAO) is a field of engineering that uses optimization methods to couples the multiple models involved in complex systems design. The coupling among disciplines typically contributes significantly to the computational cost of analysing the entire system. Therefore, the minimization problem consists in reduce the number of couplings without losing information in the communication network. Notice that some coupling may be embedded within optimization loop, therefore informations may need to travel back and forth. It is a typical practice to derive these discipline couplings using expert opinion and domain experience. However, lately several automated models have been proposed to solve MDAO problems (Baptista et al. [4]).

The most challenging example of discrete black-box optimization problems is the tuning of model hyperparameters in Machine learning algorithms. As an example, neural networks and deep learning methods notoriously require careful tuning of numerous hyperparameters. Some of them are related to the network architecture, like the number of hidden layers and of epoch, the batch size, the learning rate and the activation function. Other parameters are used for regularization, like the dropout parameter and the momentum[2]. Some of these hyperparameters are integer, others are categorical and others may be continuous if no alphabet is specified for the parameter. Currently the process of setting the hyperparameters requires expertise and extensive trial and error. A grid search or random search over the hyper-parameter space is computationally prohibitive and time consuming [34]. More in general, the tuning of hyperparameters in neural networks can be subject to time or memory constraints. On the first side, each function evaluation can require a variable amount of time. For example, training a small neural network with 10 hidden units will take less time than a bigger

[2]Refer to Alto [3] and Lau [24] for details.

network with 1000 hidden units. On the other side, the cost of requiring large-memory machines for learning may exceeds some budget constraint. In summary, the tuning of hyperparameters in ML algorithms is the best example of black-box function that is expensive to sample. Moreover, variables are discrete or categorical and, when dealing with continuous hyperparameters, typically a discrete set of variation is defined. For further details over the tuning of hyperparameters for neural networks see Shahriari et al. [33], Smith [34] and Snoek et al. [35].

3.2 Existing methods and related works

A simplistic approach is to use standard BO as-is to optimize over discrete domains. In this case, variables are treated as continuos and only at the end of each iteration the new suggested point is rounded to the closest discrete one. This approach will be refered to as *Naive-BO* as in [26]. Suddenly, this method may stumble in rounding the same continuous value into two different discrete values. This results in the repetition of the same values, without stopping when a global discrete optimum is reached.

Several researchers have tried to combine the effectiveness of Bayesian optimization with discrete problems, and multiple algorithms have been proposed. One of the most popular is the SMAC algorithm (Sequential Model-based Algorithm Configuration) proposed by Hutter et al. [19]. The algorithm uses random forests as surrogate models and random walks to obtain local optimum. Garrido-Merchán et al. [15] suggested a model that is based on rounding to the closest integer value. They considered the objective to be constant in the interval of rounding of a discrete value. For example, if the discrete variable x varies in $\mathbb{N}$, then on the interval $(4.5, 5.5) \subset \mathbb{R}$ the objective is assumed to have the same value as in 5, since all the values of the interval are rounded to 5. Suddenly, this leads to a step-wise acquisition function, which is difficult to optimize.

3.2.1 Discrete Bayesian Optimization

Luong et al. [26] tried to overcome the Naive BO and to solve the repetition problem of Naive-BO with the *Discrete-BO* algorithm. They wanted the algorithm to sample points that are different from the previous observa-

tions. To avoid sampling pre-existing observations, they focus on adjusting two hyperparameters. The first is the exploration-exploitation trade-off factor β of the Upper Confidence Bound Acquisition function. The second is the length scale l of the following covariance function (kernel):

$$k(x, x') = \sigma^2 \exp\left(-\frac{1}{2l^2}||x - x'||^2\right).$$

This kernel is called *squared exponential kernel*. It defines the Gaussian process used as surrogate model for the BO algorithm. To select the optimal values for β and l at each iteration of the BO loop, the algorithm solves a minimization problem with three objective:

$$\begin{aligned} \beta^*, l^* &= \underset{\Delta\beta\in[0,\beta_h], l\in(0,l_h]}{\operatorname{argmin}} \; g(\beta + \Delta\beta, l) \\ g(\beta + \Delta\beta, l) &= \Delta\beta + ||x_{t+1} - x'_{t+1}||_2 + P(x'_{t+1}). \end{aligned} \tag{3.2}$$

The first part minimizes the variation $\Delta\beta$ of the variable *beta*. The second part minimizes the distance between the suggested point x_{t+1} corresponding to the original β_t and the suggested point x'_{t+1} corresponding to the incremented $\beta_t + \Delta\beta$, in order to suggest a point close to the current potential area of exploration. The third objective is a penalty $P(x'_{t+1})$ that occurs only when also the suggested point x'_{t+1} has been already sampled.

In summary, Discrete-BO algorithm works as classic naive-BO. However, each time the suggested point is an already sampled point, then the algorithm solves problem (3.2) to chose another input point.

3.2.2 Bayesian Optimization of Combinatorial Structures

Baptista and Poloczek [5] proposed the *Bayesian Optimization of Combinatorial Structures* (BOCS) algorithm. For the sake of simplicity, the domain is the set $\mathcal{X} = \{0, 1\}^d$, however BOCS generalizes to integer-valued and categorical variables and to models of higher order. They gave a structure to the unknown function F, as a combination of the x_i variables, in order to model the interplay of the elements. In the set of all possible objective functions, they restricted the search of the theoretical one to second order models of the form:

$$f_{\boldsymbol{\alpha}}(x) = \alpha_0 + \sum_{0\le j\le d} \alpha_j x_j + \sum_{0\le i<j\le d} \alpha_{ij} x_i x_j. \tag{3.3}$$

The uncertainty boils down to the parameters $\boldsymbol{\alpha} = (\alpha_j, \alpha_{ij}) \in \mathbb{R}^p$ with $p = 1 + d + \binom{d}{2}$. They employed the *heavy-tailed horseshoe prior* as prior distribution

over $\boldsymbol{\alpha}$. Then, thanks to a re-parameterization of the horseshoe prior, they computed the conditional posterior distribution given in (2) of [5]. For the acquisition function maximization step, they used the following acquisition function:

$$f_{\boldsymbol{\alpha}}(x) - \lambda \mathcal{P}(x),$$

with $\mathcal{P}(x) = ||x||_1$ or $\mathcal{P}(x) = ||x||_2^2$ and λ is a penalty parameter. At each iteration the optimization problem can be rewritten in the form of a binary quadratic problem:

$$x_{t+1} = \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, x^\top A x + b^\top x. \tag{3.4}$$

Problem (3.4) is computational hard due to the discrete domain $\mathcal{X}$, therefore, the authors provided some strategies to obtain an approximation. Firstly, they relaxed the quadratic program into a vector program. Then, this vector program is rewritten in a semidefinite program (SPD) which is then approximated in polynomial time. The solution is converted back into the $\mathcal{X}$ domain. This version of the algorithm is therefore referred to as BOCS-SDP.

The authors proposed a variant of the above-descripted algorithms: the BOCS-SA algorithm. This version replaces semidefinite programming with stochastic local search, specifically with *simulated annealing* (SA). Instead of solving (3.4), simulated annealing chooses the next point to be sampled x_{t+1} by performing a random walk on the domain $\mathcal{X}$. It starts from a random point $x^{(0)} \in \mathcal{X}$. Then at each iteration k from 0 to K, SA randomly chooses a point $\bar{x}$ in the neighbourhood of $x^{(k)}$ which is the set of points with Hamming distance at most one from $\bar{x}$. The *Hamming distance* between $x', x'' \in \{0,1\}^d$ is the number of positions i at which $x'_i \neq x''_i$. If the chosen point $\bar{x}$ has an observed objective value that is better than the observed in $x^{(k)}$ then $x^{(k+1)} = \bar{x}$. Otherwise $x^{(k+1)}$ is set to $\bar{x}$ with probability $\exp((f_{\boldsymbol{\alpha}}(\bar{x}) - f_{\boldsymbol{\alpha}}(x^{(k)}))/T_{k+1})$, else $x^{(k+1)} = x^{(k)}$. T_{k+1} is a parameter that decreases with k in order to encourage exploration at first and to zoom on a good solution in the end. The last value $x^{(K)}$ is taken as next point to be sampled x_{t+1}. Essentially, simulated annealing completely substitute the acquisition function maximization step.

In summary, in both BOCS-SDP and BOCS-SA, surrogate models are specified by two elements: the model (3.3) for the objective function and the horseshoe prior for the coefficients $\boldsymbol{\alpha}$. The acquisition function step is instead different for the two algorithms. BOCS-SDP uses the idea that we

want to obtain an objective function of the form (3.3), therefore $f_{\boldsymbol{\alpha}}(x)$ is included in the definition of the acquisition function. Instead, BOCS-SA is based on the graph structure induced by the discrete variables.

3.2.3 Combinatorial Bayesian Optimization

Oh et al. [31] proposed the COMBO algorithm in which they focused the attention on *combinatorial graphs*. To define a combinatorial graph, let $\mathcal{G}(\mathcal{X}_i)$ be a sub-graph for the variable $x_i \in \mathcal{X}_i$ defined as follows: if x_i is categorical then $\mathcal{G}(\mathcal{X}_i)$ is a complete graph; if x_i is an ordinal variable then $\mathcal{G}(\mathcal{X}_i)$ is a path graph. Then the combinatorial graph $\mathcal{G}$ is defined as the Cartesian product of the sub-graphs built for each variable:

$$\mathcal{G} = \square_{i=1}^{d} \mathcal{G}(\mathcal{X}_1). \tag{3.5}$$

Therefore, the vertex set contains all possible combinations of the discrete variables and two vertices are adjacent if and only if they differ by the value of only one variable. The Hamming distance defined in the previous paragraph, is a natural choice of metric on categorical variables. In a combinatorial graph, the length of the shortest path between two vertices is equivalent to the Hamming distance of the two configuratons. As surrogate model they chose the *automatic relevance determination* (ARD) diffusion kernel for the Gaussian Process[3]. The computation of the diffusion kernel on the graph relies on the eigendecomposition of the graph Laplacian. The graph Laplacian is given by the the difference of the degree matrix and the adjacency matrix of the graph:

$$L(\mathcal{G}) = D_{\mathcal{G}} - A_{\mathcal{G}}.$$

The computation is sped up thanks to the structure of $\mathcal{G}$ as graph Cartesian product of the sub-graphs $\mathcal{G}(\mathcal{X}_i)$. In fact, we can write the ASD kernel as the Kronecker product of the individual kernels per sub-graph:

$$\mathbf{K} = \bigotimes_{i=1}^{d} \exp\big(-\beta_i L(\mathcal{G}(\mathcal{X}_i))\big).$$

They gave a Horseshoe prior to the parameters β_i, in order to encourage sparsity. In the acquisition function maximization step, the authors balanced exploration with exploitation. On one side they took 20 000 random

[3]For further details see par. 2.3 in [31].

vertices of $\mathcal{G}$ to be explored. On the other, they choose 20 randomly chosen vertices in the neighbourhood of the best input found. Among those 20 020 configurations, COMBO chooses 20 inputs with highest acquisition function value. Then further optimization is done with breadth-first local search (BFLS): at a given vertex of the 20 inputs, the acquisition value is compared with that of the adjacent vertices. Then BFLS moves to the vertex with highest acquisition value and repeat until no adjacent vertices brings an improvement to the acquisition function value.

Chapter 4

Separable Bayesian Optimization

The SBO algorithm is our proposal to solve black-box, discrete optimization problems. SBO stands for *Separable Bayesian Optimization* since we resemble the Bayesian Optimization approach. "Separable" stands for the class of functions we use to approximate the unknown objective function. In fact, SBO algorithm uses a surrogate model that considers the small subset of *additively separable functions* as plausible objective functions. A function f is said to be additively separable if there exist N functions f_i such that

$$f(\cdot) = \sum_{i=1}^{N} f_i(\cdot).$$

Moreover, we will consider only objective functions in which each addend f_i is a function of only some of the d variables $x_1, \ldots, x_d$, namely

$$f(x) = \sum_{i=1}^{N} f_i(x_{A_i}), \quad \forall x \in \mathcal{X},$$

where $A_i \subset \{1, ..., d\}$ is a set of indexes, with i from 1 to N. In such a way, we focus on the single addends rather than the objective function in its entirety.

Our proposal moves from the idea of Baptista and Poloczek [5] of considering a structured objective function. In the first section, the separable structure given to the objective function is built by using a graphical interpretation. In the second section, the property of conditioning of Gaussian Processes is adjusted for our discrete case. The third section deals with the

acquisition function maximization step, while in the last section we developed some details over the implementation in Python 3.

4.1 Statistical model

To build the structured objective function, we begin by treating each discrete variable involved in the optimization problem (3.1) as a node of a complete graph[4]. Let us assume a common finite, discrete domain $\mathcal{A}$ for the variables, namely the values a node can assume. Let $a = |\mathcal{A}|$ be the size of $\mathcal{A}$. We will refer to $\mathcal{A}$ also as the *alphabet* of the problem. The basic idea is to equip each element of the graph (nodes and links) with a function, defined on $\mathcal{A}$ for the nodes and on $\mathcal{A}^2$ for the links. In particular, considering a graph with d nodes, we define the following functions:

- $\lambda_i : \mathcal{A} \longrightarrow \mathbb{R}$, for $i = 1, \dots, d$,
- $\lambda_{ij} : \mathcal{A} \times \mathcal{A} \longrightarrow \mathbb{R}$, for $i, j = 1, \dots, d$ with $i < j$.

Let $\Lambda = (\lambda_i, \lambda_{ij})$ be a vector collecting all these functions. We then associate a functional $f_\Lambda : \mathcal{X} = \mathcal{A}^d \longrightarrow \mathbb{R}$ to the whole graph by taking the summation of the above-defined functions:

$$f_\Lambda(x) = \sum_{1 \le i \le d} \lambda_i(x_i) + \sum_{1 \le i < j \le d} \lambda_{ij}(x_i, x_j). \tag{4.1}$$

Notice that (4.1) is a generalization of (3.3), conceiving more general functions depending on one or two variables. For the sake of simplicity of notation, we assume $\lambda_i = \lambda_{0i}$, for all $i = 1, \dots, d$, as if we add an extra stubborn node $i = 0$ to the complete graph, whose value x_0 is fixed. In this way, the functional f_Λ takes the easier form of

$$f_\Lambda(x) = \sum_{0 \le i < j \le d} \lambda_{ij}(x_i, x_j).$$

Therefore, we obtained an additively separable function as objective function.

[4]Let us recall that a complete graph K_d is a simple graph (undirected, unweighted and with no self-loops) made up of d nodes each connected to every other node.

In summary, the SBO algorithm assumes a generic separable function as model for the objective function. To reduce the complexity we let the addends of the objective function be at most in two variables, which leads to a functional defined over pairwise connections of the graph. While Baptista et al. [5] consider exclusively second order objective function, in the SBO algorithm we give no restrictions on the form of the addends of the objective function. The only hypotesis is that the objective function can be decomposed in the contribution of both single variables and paired variables.

4.2 Computation of the posterior distribution

We use a Gaussian prior over the unknown factors of the objective function. In particular, each quantity $\Lambda_{ij}(s,t)$ with $i,j \in 0,...,d$, $i < j$, and $s,t \in \mathcal{A}$, is modelled with a Gaussian random variable:

$$\Lambda_{ij}(s,t) \sim \mathcal{N}(\mu_{ijst}, \sigma^2_{ijst}).$$

We then combine all this random variables in a random vector Λ, which follows a multivariate Gaussian distribution. The dimension of Λ is

$$N = da + \frac{d(d-1)}{2}a^2. \tag{4.2}$$

where we recall that a is the cardinality of the alphabet $\mathcal{A}$.

Let $F_\Lambda(\cdot) : \mathcal{X} \to \mathbb{R}$ be the true black-box function. We recall that $\forall\, x \in \mathcal{X}$, the true objective function value $F_\Lambda(x)$ is a realization of the random variable $f_\Lambda(x)$. Moreover, we suppose that the measured output in $x \in \mathcal{X}$ can be subjected to a noise ε_x, which is a random variable $\varepsilon_x \sim \mathcal{N}(0, \sigma^2_x)$ that we consider independent from the random vector Λ. In essence we write

$$y = F_\Lambda(x) = f_\Lambda(x) + \varepsilon_x, \quad \forall\, x \in \mathcal{X}.$$

Let $(x^{(n+1)}, y^{(n+1)})$ be a new observation that for the sake of simplicity will be referred to as (x,y)[5]. To update the prior distribution of $f_\Lambda(z)$ for

[5]Note that here we changed the subscript index n with a superscript, since notations will become pretty rough in the following. Let us recall that the superscript index (n) refers to the size of the train set for the GP, that is the number of observations we got from the black-box and we used to update the prior distribution.

all the unvisited inputs $z \in \mathcal{X}$ we need to update the mean and covariance functions for the $\Lambda_{ij}(s,t)$ random variables. In particular, we can make use of equations (2.2) and (2.3) to condition these distributions over the new observation. Specifically, let

$$\mu_{ij}^{(n)}(s,t) = \mathbb{E}[\Lambda_{ij}(s,t)],$$

and

$$\begin{aligned}\Omega_{ij,hk}^{(n)}((s,t),(u,v)) &= \mathrm{Cov}(\Lambda_{ij}(s,t), \Lambda_{hk}(u,v)) = \\ &= \mathbb{E}[\Lambda_{ij}(s,t) \cdot \Lambda_{hk}(u,v)] - \mu_{ij}^{(n)}(s,t) \cdot \mu_{hk}^{(n)}(u,v)\end{aligned}$$

be the means and covariances of the $\Lambda_{ij}(s,t)$ random variables given n observations, $i,j,h,k \in \{0,\dots,d\}$, $i<j$, $h<k$, and $s,t,u,v \in \mathcal{A}$. To compute the updated means and covariances $\mu_{ij}^{(n+1)}(s,t)$ and $\Omega_{ij,hk}^{(n+1)}((s,t),(u,v))$, let us define the following quantities:

- the covariance between the model $f_\Lambda^{(n)}(x)+\varepsilon_x$ and the random variables $\Lambda_{hk}(u,v)$ as

$$\begin{aligned}\rho_{hk}^{(n)}(u,v) &= \mathrm{Cov}\left(\left(f_\Lambda^{(n)}(x)+\varepsilon_x\right), \Lambda_{hk}(u,v)\right) = \\ &= \mathbb{E}\left[\left(f_\Lambda^{(n)}(x)+\varepsilon_x\right)\cdot\Lambda_{hk}(u,v)\right] - \mathbb{E}\left[f_\Lambda^{(n)}(x)+\varepsilon_x\right]\cdot\mu_{hk}^{(n)}(u,v) = \\ &= \sum_{0\le i<j\le d}\left(\mathbb{E}[\Lambda_{ij}(x_i,x_j)\cdot\Lambda_{hk}(u,v)] - \mathbb{E}[\Lambda_{ij}(x_i,x_j)]\cdot\mu_{hk}^{(n)}(u,v)\right) = \\ &= \sum_{0\le i<j\le d}\left(\mathbb{E}[\Lambda_{ij}(x_i,x_j)\cdot\Lambda_{hk}(u,v)] - \mu_{ij}^{(n)}(x_i,x_j)\cdot\mu_{hk}^{(n)}(u,v)\right) = \\ &= \sum_{0\le i<j\le d}\Omega_{ij,hk}^{(n)}((x_i,x_j),(u,v));\end{aligned}$$

- the mean value of the measured output

$$\begin{aligned}\mu^{(n)} &= \mathbb{E}\left[f_\Lambda^{(n)}(x)+\varepsilon_x\right] = \sum_{0\le i<j\le d}\mathbb{E}[\Lambda_{ij}(x_i,x_j)] = \\ &= \sum_{0\le i<j\le d}\mu_{ij}^{(n)}(x_i,x_j);\end{aligned} \tag{4.3}$$

- the variance of the measured output

$$\begin{aligned}
\left(\rho^2\right)^{(n)} &= \mathbb{E}\left[\left(f_\Lambda^{(n)}(x)+\varepsilon_x\right)^2\right] - \mathbb{E}\left[f_\Lambda^{(n)}(x)+\varepsilon_x\right]^2 = \\
&= \sum_{i'<j'}\sum_{i''<j''} \mathbb{E}\left[\Lambda_{i'j'}(x_{i'},x_{j'})\Lambda_{i''j''}(x_{i''},x_{j''})\right] + \sigma_x^2 - \left(\mu^{(n)}\right)^2 = \\
&= \sum_{i'<j'}\sum_{i''<j''} \Omega^{(n)}_{i'j',i''j''}((x_{i'},x_{j'}),(x_{i''},x_{j''})) + \\
&\quad + \sum_{i'<j'}\sum_{i''<j''} \mu^{(n)}_{i'j'}(x_{i'},x_{j'})\mu^{(n)}_{i''j''}(x_{i''},x_{j''}) + \sigma_x^2 - \left(\mu^{(n)}\right)^2 = \\
&= \sum_{i'<j'}\sum_{i''<j''} \Omega^{(n)}_{i'j',i''j''}((x_{i'},x_{j'}),(x_{i''},x_{j''})) + \sigma_x^2 = \\
&= \sum_{i<j} \rho_{ij}^{(n)}(x_i,x_j) + \sigma_x^2. \qquad (4.4)
\end{aligned}$$

To make use of (2.3), we consider the following $(N+1)$-variate Gaussian distribution:

$$\begin{pmatrix} F_\Lambda(x^{(n+1)}) \\ \Lambda \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu^{(n)} \\ \bar{\boldsymbol{\mu}}^{(n)} \end{pmatrix}, \begin{pmatrix} \left(\rho^2\right)^{(n)} & \boldsymbol{\rho}^{(n)\top} \\ \boldsymbol{\rho}^{(n)} & \Omega^{(n)} \end{pmatrix}\right), \qquad (4.5)$$

where $\bar{\boldsymbol{\mu}}^{(n)}$ and $\boldsymbol{\rho}^{(n)}$ are N dimensional vectors collecting respectively means $\mu_{hk}^{(n)}(u,v)$ and covariances $\rho_{hk}^{(n)}(u,v)$ with $h,k \in \{0,\dots,d\}$, $h<k$ and $u,v \in \mathcal{A}$, in the same order given to the Λ vector. In similar fashion we defined the $N \times N$ covariance matrix $\Omega^{(n)}$ collecting the covariances $\Omega^{(n)}_{ij,hk}((s,t),(u,v))$, with $i,j,h,k \in \{0,\dots,d\}$, $i<j$, $h<k$, and $s,t,u,v \in \mathcal{A}$, except when $i=0$ or $h=0$, because in that case $s=x_0$ or $u=x_0$ respectively. With this in mind, since $F_\Lambda(x^{(n+1)}) = y^{(n+1)}$, we can easily compute the posterior with equation (2.3) as follows:

$$\begin{aligned}
\bar{\boldsymbol{\mu}}^{(n+1)} &= \bar{\boldsymbol{\mu}}^{(n)} + \frac{1}{\left(\rho^2\right)^{(n)}}\boldsymbol{\rho}^{(n)}\left(y^{(n+1)} - \mu^{(n)}\right), \\
\Omega^{(n+1)} &= \Omega^{(n)} - \frac{1}{\left(\rho^2\right)^{(n)}}\boldsymbol{\rho}^{(n)}\boldsymbol{\rho}^{(n)\top}.
\end{aligned}$$

Component-wise it results in:

$$\begin{aligned}
\mu_{hk}^{(n+1)}(u,v) &= \mu_{hk}^{(n)}(u,v) + \frac{\rho_{hk}^{(n)}(u,v)}{\left(\rho^2\right)^{(n)}}\left(y^{(n+1)} - \mu^{(n)}\right), \\
\Omega^{(n+1)}_{ij,hk}((s,t),(u,v)) &= \Omega^{(n)}_{ij,hk}((s,t),(u,v)) - \frac{\rho_{ij}^{(n)}(s,t)\rho_{hk}^{(n)}(u,v)}{\left(\rho^2\right)^{(n)}}.
\end{aligned} \qquad (4.6)$$

Finally, with these two equations we can compute the posterior distribution of Λ given a new observation $(x, y) = (x^{(n+1)}, y^{(n+1)})$, in terms of its mean vector and covariance matrix.

4.3 Acquisition function

The second main step of Bayesian optimization is solving an optimization problem involving an acquisition function in order to find the next input to sample. Since it is not an easy task to solve a maximization problem over a discrete domain we choose to do a local search over a subset of the features space. In fact, the number of configurations in the input space is a^d, which grows exponentially in the dimension of the domain, hence a global search is infeasible. For this reason, at each iteration of the BO loop we need to define a *neighbourhood* of the previous configuration. In particular, at each iteration we solve the following problem:

$$x^{(n+1)} = \underset{z \in \mathcal{X}_{x^{(n)}}}{\operatorname{argmax}}\, \alpha(z) \tag{4.7}$$

where $\alpha(\cdot)$ is an acquisition function and $\mathcal{X}_{x^{(n)}} \subset \mathcal{X} = \mathcal{A}^d$ is the neighbourhood of $x^{(n)}$.

There are several types of neighbourhood. For example we can use the neighbourhood defined by Baptista et al. [5] in the BOCS-SA algorithm discussed in Section 3.2. The algorithm defines the neighbourhood of a configuration $x^{(n)}$ as the set of points with Hamming distance at most 1 from $x^{(n)}$. Also Oh et al. [31] used the Hamming distance in their COMBO algorithm (also discussed in Section 3.2). On a combinatorial graph, two points have an Hamming distance equal to 1 if the shortest path between the two points consists in a single edge. Therefore the neighbourhood defined by Baptista et al. coincides with the neighbourhood (as defined in graph theory) on the combinatorial graph.

Other types of neighbourhood will be introduced in the Implementation section below. In the applications we will use all the types of neighbourhood and acquisition function (EI, mEI, UCB), as defined in Section 2.3, in order to compare the results and choose the one that performs better for each problem.

4.4 Implementation

The programming language used to implement the SBO algorithm was Python 3. We defined several classes and function that implements all the steps of the algorithm. Algorithm 1 is a brief overview of the algorithm in pseudo-code.

Algorithm 1 SBO algorithm

Input Dimension d of the feature space; set $\mathcal{A}$ of values a variable can take, coded from 1 to $a = |\mathcal{A}|$; training set of couples (x, y); number of iterations `n_iter`.
Initialize r.v. $\Lambda_{ij}(s,t)$ with sample mean and covariance.
for (x, y) in training set **do**
    Update $\Lambda_{ij}(s,t)$
    Add x to a vector `X` of inputs.
    Add y to a vector `Y` of outputs.
for i **from** 1 **to** `n_iter` **do**
    Define neighbourhood of last element of `X`.
    Maximize AF over the neighbourhood and get `x_new`.
    Sample the new cople (`x_new`, `y_new`).
    Update $\Lambda_{ij}(s,t)$.
    Add `x_new` to the vector `X`.
    Add `y_new` to the vector `Y`.
Output Max and Argmax of the vector `Y`.

4.4.1 Initialization and arrangement of Λ_{ij} r.v.

First of all we decided to use sample mean and sample covariances to initialize the $\Lambda_{ij}(s,t)$ random variables. In particular, given a vector of training outputs Y, we computed the sample mean $\bar{\mu}$ and sample variance $\bar{\rho}^2$. Then, since the output is a sum of the $\Lambda_{ij}(s,t)$ for s and t fixed, we divide these values for the number of distinct Λ_{ij}, according to the following

equations:

$$\mu_{ij}^{(0)}(s,t) = \frac{2}{(d+1)d}\bar{\mu},$$

$$\Omega_{ij,hk}^{(0)}((s,t),(u,v)) = \begin{cases} \frac{2}{(d+1)d}\bar{\rho}^2 & \text{if } (i,j) = (h,k) \text{ and } (s,t) = (u,v), \\ 0 & \text{otherwise,} \end{cases}$$

since

$$\binom{d+1}{2} = \frac{(d+1)d}{2}$$

is the number of distinct combination of indexes (i,j), namely the number of edges of the graph (including the stubborn node).

More sophisticated choices can be made to initialize these parameters, like for example not choosing to give the same mean and variance to each variable, but differentiate them according to some criterion. Alternatively, a tuning using some ML algorithm can be done or simply estimating the parameters maximizing a likelihood function (MLE).

We now explain the implementation part. For the random variables used in the algorithm we defined two Python classes: a class `Lambdas` corresponding to the random vector Λ and a class `lambdas_ijst` for the single $\Lambda_{ij}(s,t)$ random variable[6]. An element of the class `Lambdas` is equipped with a vector of element of class `lambdas_ijst`, by definition of Λ, a mean vector `MuVector` and a covariance matrix `CovMatrix`, which are $\bar{\boldsymbol{\mu}}$ and Ω of equation (4.5) respectively. Moreover we defined two functions:

- `F_Lambda`, that given an input x^* returns the mean and variance of $f_\Lambda(x^*)$, obtained with (4.3) and (4.4);
- `posterior`, which updates `MuVector` and `CovMatrix` given a new pair of observation (`x_new`, `y_new`), according to equation (4.6); this function uses the `F_Lambda` function to obtain the new mean and variance of the output given the new observations.

Instead, the class `lambdas_ijst` is equipped with a mean `mu` and a covariance vector `cov` that collects the variance of the $\Lambda_{ij}(s,t)$ r.v. and its covariances with all the others; therefore the covariance vector has size N (see (4.2)).

[6]For further details concerning the arrangement of the $\Lambda_{ij}(s,t)$ inside the vector Λ, see Appendix B.

We defined the `update` method which changes the value of mean and covariances given the new ones calculated in the `Lambdas.posterior` method. The following chunk is the initialization part.

```
N = int(n_vals*n_vars*(1+n_vals*(n_vars-1)/2))
X = X_train
Y = Y_train
n_train = np.shape(Y)[0]
Lambdas_obj = Lambdas()
for t in range(n_train):
    Lambdas_obj.posterior(X[t],Y[t])
```

4.4.2 BO loop and neighbourhoods

The core of the Bayesian optimization algorithm is handled by the main part of the code. Specifically, the following code chunk is used:

```
for i in range(n_iter):
    x_new = next_x(X[-1], y_best, n_vals, Lambdas_obj, neigh_type,
                   AF, AFparameter=0)
    y_new = oracolo(x_new)
    Lambdas_obj.posterior(x_new, y_new)
    X = np.vstack((X, x_new))
    Y = np.hstack((Y, y_new))
```

The element `Lambdas_obj` is an object of the class `Lambdas`. The function `next_x` takes care of finding the next input to be sampled. In essence, it generates the neighbourhood of the last input `X[-1]` according to the criterion `neigh_type` and then it solves the maximization of the Acquisition function problem. The Acquisition function is specified by a string `AF` and its parameter is `AFparameter`: this parameter is null for the classic Expected improvement, while takes the role of k for the modified Expected improvement (see (2.6)) and of β for the Upper confidence bound (see (2.7)). These Acquisition function use the mean and variance provided by the method `F_Lambda` of the class `Lambdas`, and the implementation follows equations (2.5) and (2.7). Afterwards, the new output corresponding to the suggested input is sampled, then the distribution of the $\Lambda_{ij}(s,t)$ is updated thanks to the method `Lambdas.posterior` and finally the new sampled couple of input-output is stored.

The last building block of the algorithm is the function involved in creating the neighbourhood of a given x, namely `GraphOfTentatives`. The different neighbourhood types of an input $x \in \mathbb{R}^d$ are the followings (for each type the corresponding figure is the neighbourhood $\mathcal{X}_x$ of $x = (1,1,1,1)^\top \in \{1,2,3\}^4$, therefore $a = 3$ and $d = 4$):

- `neigh_type = 1`: $y \in \mathcal{X}_x$ if and only if $\exists\, j \in \{1,\dots,d\}$ such that $y_j \neq x_j$ and $y_i = x_i$, $\forall\ i \neq j$;

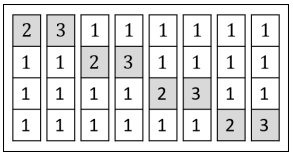


Figure 4.1: Neighbourhood type 1.

- `neigh_type = 2`: $y \in \mathcal{X}_x$ if and only if $\exists\, j_1, j_2 \in \{1,\dots,d\}$ such that $y_i = x_i$, $\forall\ i \neq j_1, j_2$ and

$$y_{j_1} = \begin{cases} x_{j_1} + 1 \pmod a \\ x_{j_1} - 1 \pmod a \end{cases} \quad \text{or} \quad \text{and} \quad y_{j_2} = \begin{cases} x_{j_2} + 1 \pmod a \\ x_{j_2} - 1 \pmod a; \end{cases} \quad \text{or}$$

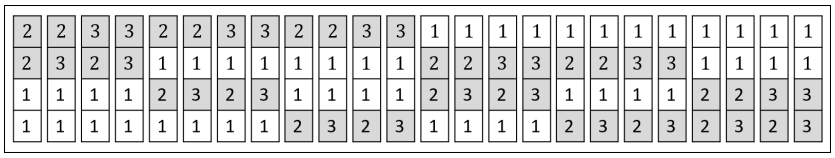


Figure 4.2: Neighbourhood type 2: notice that $1 - 1 = 3 \pmod 3$.

- `neigh_type = 3`: given the parameter m, we say $y \in \mathcal{X}_x$ if and only if $\exists\, j_1, \dots, j_m \in \{1,\dots,d\}$ such that $y_{j_k} \neq x_{j_k}$, for $k = 1,\dots,m$ and $y_i = x_i$, $\forall\ i \neq j_1,\dots,j_m$; moreover we take only `num_neigh` of this configurations, since the number of combinations grows exponentially;

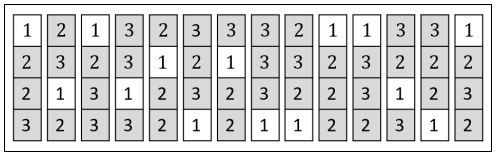


Figure 4.3: Neighbourhood type 3 with $m = 3$ and `num_neigh` $= 14$.

- `neigh_type = 4`: $y \in \mathcal{X}_x$ if and only if one of the following occurs:

  1. $\exists j \in \{1, \dots, d\}$ such that $y_j = x_j + 1 \pmod a$ and $y_i = x_i, \forall\, i \neq j$,
  2. $\exists j_1, j_2 \in \{1, \dots, d\}$ such that $y_{j_k} = x_{j_k} + 1 \pmod a$ for $k = 1, 2$ and $y_i = x_i, \forall\, i \neq j_1, j_2$,
  3. $\exists j_1, j_2, j_3 \in \{1, \dots, d\}$ such that $y_{j_k} = x_{j_k} + 1 \pmod a$ for $k = 1, 2, 3$ and $y_i = x_i, \forall\, i \neq j_1, j_2, j_3$,
  4. $\exists j_1, \dots, j_4 \in \{1, \dots, d\}$ such that $y_{j_k} = x_{j_k} + 1 \pmod a$ for $k = 1, \dots, 4$ and $y_i = x_i, \forall\, i \neq j_1, \dots, j_4$;

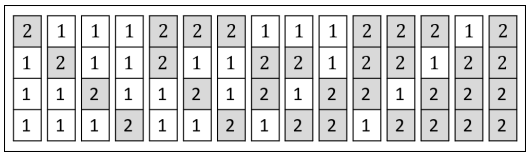


Figure 4.4: Neighbourhood type 4.

- `neigh_type = 5`: at each iteration three indexes are randomly chosen, namely j_1, j_2, j_3, then $y \in \mathcal{X}_x$ if and only if $y_{j_k} \neq x_{j_k}$ for $k = 1, 2, 3$ and $y_i = x_i, \forall\, i \neq j_1, j_2, j_3$;

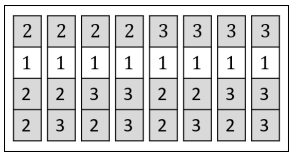


Figure 4.5: Neighbourhood type 5: suppose we extracted indexes 1, 3, and 4.

- `neigh_type = 6`: at each iteration one of the two following rules is applied with probability of 50%:

  1. the set of neighbours is like in `neigh_type = 2` but with just one index,
  2. a random input $z \in \mathcal{X}$ is generated, then, given the M previously obtained best inputs $x^1, \dots, x^M$, we build the neighbourhood of x taking each of the M best input x^k to get two neighbours y^{1k}

and y^{2k} as follows:

$$y_j^{1k} = \begin{cases} x_j^k & \text{for } j = 1, \dots, \lfloor \frac{d}{2} \rfloor \\ z_j & \text{for } j = \lfloor \frac{d}{2} \rfloor + 1, \dots, d \end{cases}$$

$$y_j^{2k} = \begin{cases} z_j & \text{for } j = 1, \dots, \lfloor \frac{d}{2} \rfloor \\ x_j^k & \text{for } j = \lfloor \frac{d}{2} \rfloor + 1, \dots, d. \end{cases}$$

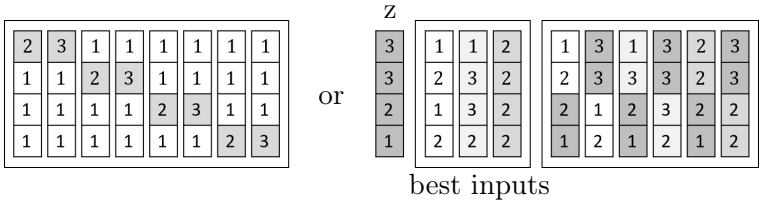


Figure 4.6: Neighbourhood type 6 with $M = 3$.

The last type of neighbourhood is inspired by the *Genetic Algorithm* problem solving method (see Thengade [37]). In fact, two main steps can be made: *mutations* and *crossovers*. A mutation occurs when only a selected set of alleles changes its value in the given chromosome; this happens in an easier way in the first rule. On the other hand, the second rules refers to single-point crossovers: in a single-point crossover, a point of splitting is chosen (the middle point in our rule) and then two chromosomes swap their alleles after that point, so that the two children take one section of the chromosome from each parent. In our case the two parents are the random input z and one of the best inputs. An important difference between our rule and the Genetic algorithm is the probability of occurrence of events, because in Genetic algorithms single mutations occurs with a small chance (1-2%); moreover there is also a chance of failure for the crossovers, since only 60-70% of times two children are generated.

We have defined the neighbourhood types keeping in mind that we get different cardinality of the $\mathcal{X}_x$ set with each rule. In fact, here is the cardinality of the neighbourhood in each case:

1. $|\mathcal{X}_x| = d(a-1)$
2. $|\mathcal{X}_x| = \binom{d}{2} \cdot 4 = 2d(d-1)$
3. $|\mathcal{X}_x| =$ `num_neigh`

4. $|\mathcal{X}_x| = d + \binom{d}{2} + \binom{d}{3} + \binom{d}{4} =$
$$= d + \frac{d(d-1)}{2} + \frac{d(d-1)(d-2)}{3 \cdot 2} + \frac{d(d-1)(d-2)(d-3)}{4 \cdot 3 \cdot 2} =$$
$$= \frac{d^4 - 2d^3 + 11d^2 + 14d}{24}$$

5. $|\mathcal{X}_x| = (a-1)^3$

6. $|\mathcal{X}_x| = \begin{cases} 2d & p = 1/2 \\ 2M & p = 1/2 \end{cases}$

Let us plot these cardinalities as functions of d and of a. According to the dimensionality of the problem, both in the feature space and in the domain, one can choose to set the remaining parameters accordingly. On one hand, one can prefer to use a neighbour of smaller size to improve the maximization problem of the Acquisition function; on the other hand, this would lead to more iterations of the BO loop to reach the optimum, while bigger neighbours can analyse more input simultaneously, slowing down the algorithm.

Note that for `neigh_type`= 6 we plot the average value $d + M$.

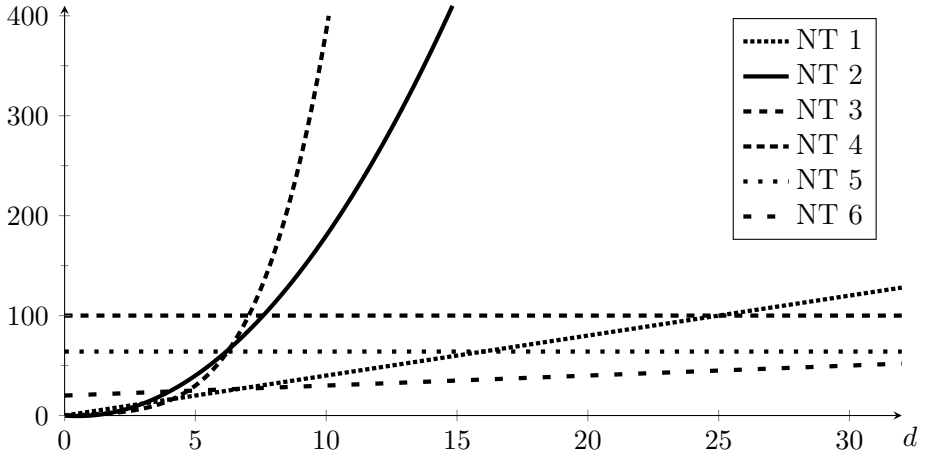


Figure 4.7: Number of neighbours as a function of the dimension d of the feature space, with $a = 5$, $M = 20$ and `num_neigh`= 100 fixed.

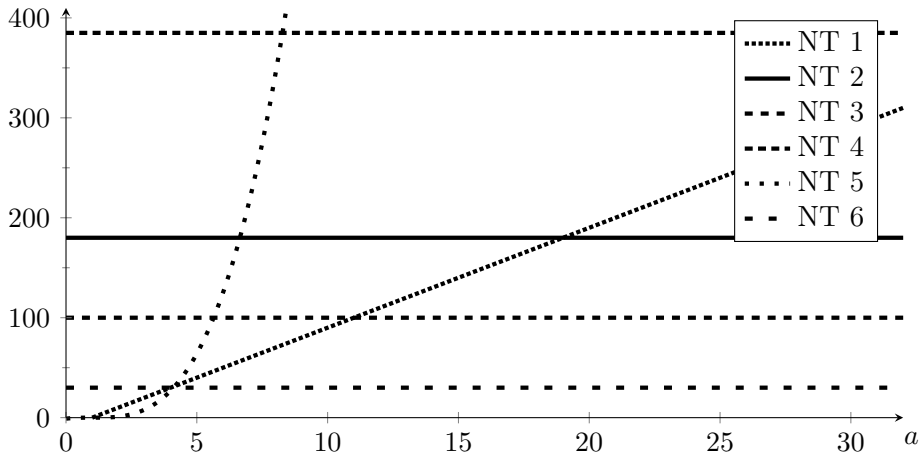


Figure 4.8: Number of neighbours as a function of $a = |\mathcal{A}|$, with $d = 10$, $M = 20$ and `num_neigh`$= 100$ fixed.

Chapter 5

Numerical Results with binary problems

In this chapter we will test the SBO algorithm against two benchmarks proposed in Baptista et al. [5] and Oh et al. [31]. The first application is the Binary Quadratic Programming (BQP) problem. BQP has became a central model in combinatorial optimization, due to its great variety of applications. Problems on graphs, logistic problems, resources allocation or clustering and ordering problems, can be considered as BQP models.

The second application deals with the sparsification of Ising models. Sparsify a graph means approximate it with another one with fewer edges (or nodes). Thus, the benchmark objective is to find the best sparsification of the reticular graph that is used to assemble Ising models, which are models built to describe the thermodynamic properties of magnetic systems.

5.1 Binary Quadratic Programming problem

Binary quadratic programming problem is an optimization problem defined over the combinatorial domain $\{0,1\}^d$ (or $\{-1,1\}^d$ for certain applications) and it has the following objective function:

$$F(x) = x^\top Q x,$$

where $Q \in \mathbb{R}^{d\times d}$ is a random symmetric positive semidefinite matrix. As done by Baptista and Poloczek [5], we add a regularization term $\mathcal{P}(x)$ that

may be either $||x||_1$ or $||x||_2$ to the objective function. Hence, the problem can be formulated as follows:

$$\max_{x \in \{0,1\}^d} x^\top Q x - \eta ||x||_1, \tag{5.1}$$

where η is a regularization parameter. Notice that this objective function is an additively separable function and can be rewritten as a summation of pairwise connections:

$$F(x) = \sum_{1 \leq i,j \leq d} q_{ij} x_i x_j - \sum_{1 \leq i \leq d} \eta |x_i|, \quad \forall\, x \in \{0,1\}^d.$$

Therefore, the separability assumption of the SBO algorithm is satisfied. In particular, in this case, we have

$$\lambda_{0j}(x_0, x_j) = q_{jj} x_j^2 - \eta |x_j| \qquad \text{and} \qquad \lambda_{ij}(x_i, x_j) = 2 q_{ij} x_i x_j,$$

for all $i, j \in \{1, \ldots, d\}$ with $i < j$. The number N of distinct $\lambda_{ij}(s,t)$, that is also the size of the vector Λ, is

$$N = 2d + 4\frac{d(d-1)}{2}.$$

On the other hand, the number of distinct functions λ_{ij}, which is equal to the number of terms of the separable function F, is $d + \frac{d(d-1)}{2}$.

We will study the performance of the algorithm with $d = 12$ and for different values of η. Clearly in this case $a = 2$ since $\mathcal{A} = \{0, 1\}$.

5.1.1 Results

To carry out a simulation we firstly need to set the value of the hyper-parameters:

- the neighbourhood type NT and its parameters (m and `num_neigh` for the NT 3, M for the NT 6),
- the number of executions of the SBO algorithm,
- the acquisition function and its parameter, if any,
- the number of training observations for the algorithm,
- the number of iterations each run of the algorithm has available.

Due to the great number of parameters, we decided to fix some of them and for the others we defined a proper range of variation. In particular, we set $m = 4$ and `num_neigh`=100 for NT 3 and $M = 10$ for NT 6. We decided to run the algorithm 20 times for each combination of parameters, and in each run we made available 100 iterations to try reaching the optimum. Moreover, the number of training observations given to initialize the algorithm is set to 20. On the other hand, we let the AF and the NT vary. In particular, the acquisition function is chosen between Expected Improvement, modified Expected Improvement and Upper confidence bound, while the neighbourhood type varies from 1 to 6. Moreover, we let the parameter of exploration-exploitation k of the modified Expected improvement vary in the set $\{0.1, 1, 10\}$; the same set is used also for the hyperparameter β of the Upper confidence bound acquisition function.

Let us recall that each neighbourhood type (NT) produces a set of neighbours $\mathcal{X}_x$ with different cardinality. In particular, for $d = 12$ and $a = 2$ we got:

NT:	**1**	**2**	**3**	**4**	**5**	**6**
$\lvert\mathcal{X}_x\rvert$	12	264	100	793	1	24 or 20
$\frac{\lvert\mathcal{X}_x\rvert}{\lvert\mathcal{X}\rvert}$	0.29%	6.44%	2.44%	19.36%	0.02%	0.53%

Table 5.1: Cardinality of the neighbourhood and corresponding percentage of domain, for each NT. The cardinality of $\mathcal{X}$ is $a^d = 4\,096$.

Clearly, the neighbourhood type 5 will not lead to successful performance, since at each iteration of the BO loop we can move only to a specific configuration.

For the design specification of the problem, as we already said, we put $d = 12$, while the symmetric, positive semidefinite matrix Q is chosen randomly with the function `make_spd_matrix`.

Table 5.2 contains a summary of the results obtained with the 20 executions of the BO loop using the SBO algorithm, with the different sets of parameters. Here we report the results for the problem using $\eta = 10^{-4}$ as regularization parameter. The best performance for each acquisition function is set in bold. We immediately notice that the neighbourhood type 6, which we recall to be the one involving the Genetic algorithm, outperforms the others when dealing with expected improvement. In fact, in each of the

		NT:		
		1	**2**	**3**
EI		51.13 ± 3.61	51.48 ± 1.43	52.18 ± 1.35
mEI	0.1	52.49 ± 1.08	51.61 ± 1.27	52.06 ± 1.35
	1	51.81 ± 2.48	51.05 ± 3.50	51.74 ± 1.07
	10	51.45 ± 2.69	52.17 ± 1.14	51.50 ± 1.22
UCB	0.1	28.89 ± 8.97	26.75 ± 9.22	26.77 ± 6.46
	1	27.87 ± 6.00	28.86 ± 7.05	26.17 ± 5.12
	10	29.23 ± 7.86	26.00 ± 6.26	26.54 ± 6.62

		NT:		
		4	**5**	**6**
EI		52.83 ± 0.88	38.97 ± 6.16	**53.20**$\pm$0.00
mEI	0.1	52.73 ± 0.93	38.72 ± 6.65	**53.20**$\pm$0.00
	1	52.72 ± 0.97	37.31 ± 5.51	**53.20**$\pm$0.00
	10	52.73 ± 0.93	40.16 ± 7.00	**53.20**$\pm$0.00
UCB	0.1	29.31 ± 7.95	**38.03**$\pm$6.82	29.66 ± 7.98
	1	29.26 ± 9.24	**39.35**$\pm$6.45	31.03 ± 8.22
	10	31.27 ± 8.05	**37.52**$\pm$6.88	29.32 ± 8.79

Table 5.2: Results after 20 executions of the algorithm: for each combination of parameters the mean value of the optimal outputs is reported, with one standard deviation error estimate.

20 re-run of the algorithm we reach the optimal configuration and the optimum 51.42 (hence the standard deviation is null). The performance gets worse with upper confidence bound, and the one obtaining the greatest values of the objective function is the neighbourhood type 5, that as we said is useless since no optimization is involved because we are moving into the features space according to a specific rule.

Table 5.3 contains the percentage of success of obtaining the theoretical optimum with each setting of parameters. The algorithm performs poorly with Upper confidence bound acquisition function.

Lastly, Figure 5.1 is a plot of the average error made with each combination of parameters, defined as follows:

$$\overline{\text{err}} = \frac{1}{20} \sum_{i=1}^{20} |F(x^*) - f_i^*|, \tag{5.2}$$

		NT:					
		1	**2**	**3**	**4**	**5**	**6**
EI		55%	35%	60%	85%	0%	100%
mEI	0.1	70%	35%	55%	80%	0%	100%
	1	60%	50%	35%	80%	0%	100%
	10	55%	55%	30%	80%	5%	100%
UCB	0.1	5%	5%	0%	0%	0%	5%
	1	0%	0%	0%	5%	5%	0%
	10	0%	0%	0%	0%	0%	0%

Table 5.3: Percentages of obtaining the optimum.

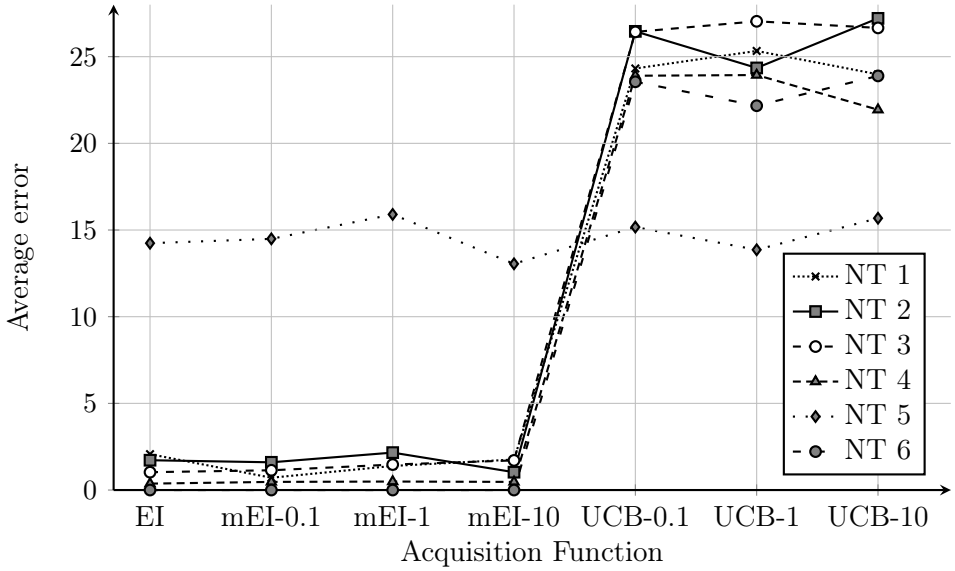


Figure 5.1: Plot of the errors averaged over 20 repetitions of the BO loop, for each acquisition function.

where x^* is the optimal input, i.e. the argmax of (5.1), while f_i^* is the optimum obtained at the i-th run of the algorithm for a certain set of parameters. We notice again that the performance of NT 5 is not affected by the changing in the acquisition function, while the others performs better with EI and modified EI. Similar results were obtained with different matrices Q and with η varying in the set $\{10^{-4}, 10^{-2}, 0\}$.

The great advantage of the SBO algorithm is its speed. In fact, measuring

NT:	**1**	**2**	**3**	**4**	**5**	**6**
avg time (in ms)	5.1	27.62	38.57	307.54	1.02	4.23
$\lvert\mathcal{X}_x\rvert$	12	264	100	793	1	24 or 20
$\frac{\lvert\mathcal{X}_x\rvert}{\lvert\mathcal{X}\rvert}$	0.29%	6.44%	2.44%	19.36%	0.02%	0.53%

Table 5.4: Average times to compute the `next_x` step, compared to the cardinality of the neighbourhood at each iteration.

NT:	**1**	**2**	**3**	**4**	**5**	**6**
avg total time with EI (in s)	1.325	4.177	5.886	41.44	0.796	1.178
avg total time with UCB (in s)	0.991	2.629	3.364	21.607	0.754	1.034
total number of explored input	1 200	26 400	10 000	79 300	100	2 200

Table 5.5: Average total time of execution of the algorithm, for each acquisition function (EI and mEI are gathered under EI since they have similar performance).

the time the algorithm takes step-by-step, and recalling that in this scenario $N = 288$, on average we get the following results:

- the update the prior distribution over Λ given a new observation takes $6.86 \cdot 10^{-3}$ seconds (i.e. 6.86 milliseconds) on average;
- the different times to compute the `next_x` step, involving creating the neighbourhood and maximizing the acquisition function over it, are collected in Table 5.4, for each neighbourhood type;
- lastly, Table 5.5 contains the average total time of execution of the SBO algorithm, compared to the total number of input of the domain that have been analysed[7].

[7]Notice that $|\mathcal{X}| = |\{0,1\}^d| = 4\,096$, therefore many inputs have been analysed more than once.

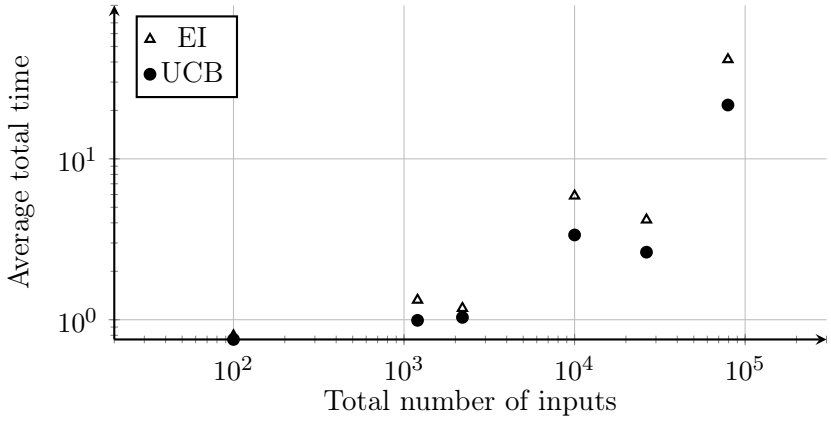


Figure 5.2: Plot of the errors averaged over 20 repetitions of the BO loop, for each acquisition function.

We notice that the algorithm is generally faster with UCB since it is easier to calculate and less expensive. Plot 5.2 is a visualisation of the previous table that let us compare the results in term of space and time. For a more understandable visualization, we plot on a logarithmic scale. Though the different scale, we notice a certain linear relation between number of configurations and time of execution.

5.1.2 Comparison with BOCS

We compared the performance of the SBO algorithm with those of the BOCS. Both BOCS-SA and BOCS-SDP are considered (see Section 3.2). The comparison was performed considering the same matrix Q and parameter η but with different train observation sets. Moreover, we focused on neighbourhood type 6 and acquisition function EI and mEI with parameter 0.1, 1 and 10. Figure 5.3 shows the average best output obtained at each iteration. In particular, we executed 20 times each algorithm with a random train set of 20 observations. We collected the best output at each of the 100 iterations of each algorithm. Lastly, we averaged over the 20 repetitions, separately for each iteration and algorithm. We notice that the performs of the SBO algorithm are competitive with those of the BOCS algorithm. BOCS on average is slightly faster in reaching the optimum value with respect to the SBO algorithm.

Figure 5.4 shows the average best output with one standard deviation

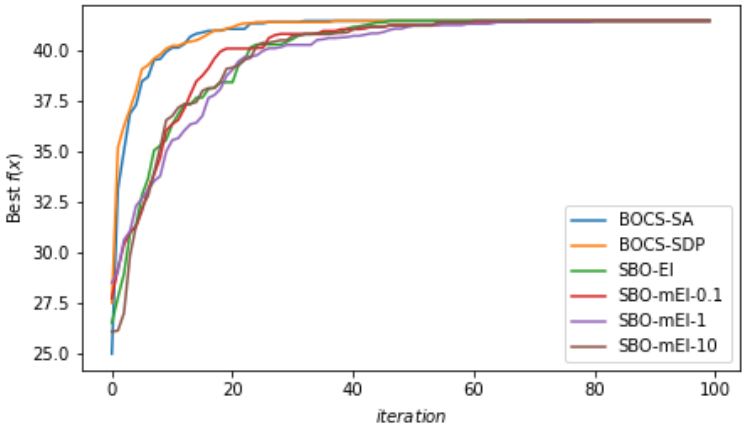


Figure 5.3: Average best output at each iteration, for each algorithm.

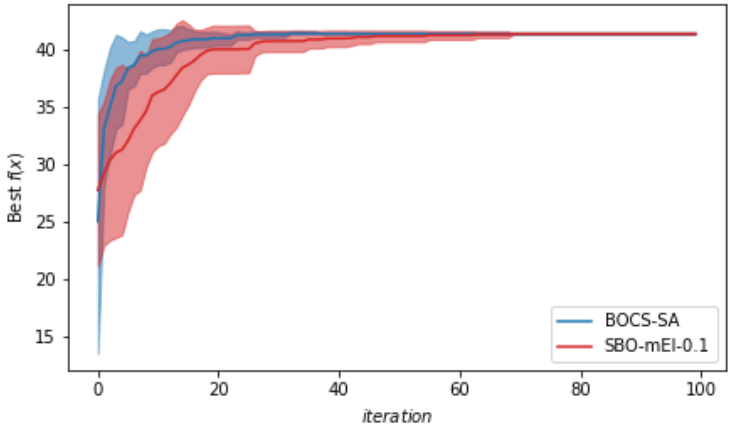


Figure 5.4: Average best output $\pm$ 1 standard deviation error estimate, for BOCS-SA and SBO with mEI and parameter 0.1.

error estimate. For easy understanding we chose to plot only the performance of BOCS-SA and of SBO with mEI and parameter 0.1. The performance of BOCS-SDP are similar to that of BOCS-SA, and the performance of the SBO algorithm are similar for each acquisition function choice. We notice that the SBO algorithm has a higher variation at the beginning, while BOCS converges faster. All the algorithms eventually reaches the optimum after the 100 iterations.

Lastly, we notice again that the performance of the SBO is not affected

by the choice of the acquisition function parameter (regarding expected improvement), for a fixed neighbourhood type.

5.2 Sparsification of Ising model problem

The second benchmark proposed by Baptista and Poloczek [5] deals with *Ising models*. The Ising model is a prototypical system that describes the thermodynamic properties of magnetic systems from a microscopic point of view. In particular, it is helpful in describing phase transitions from ferromagnetism to paramagnetism. It is one of the most heavily studied model in statistical mechanics and it is extremely important also because it does not only apply to magnetic systems: many other systems, like fluids, neural networks or binary alloys, can be shown to be equivalent to Ising models.

For further information regarding Ising models, see Baxter [6] or McCoy and Wu [27].

5.2.1 Problem description

A 2-dimensional Ising model consists of a 2-dimensional lattice with N sites. We assume a square lattice, hence N is a perfect. Other types of 2-dimensional lattices are triangular lattices, honeycomb lattices and Bethe lattices. The square lattice can be represented with a grid graph $\mathcal{G} = (\{1, \dots, N\}, \mathcal{E})$ as in figure 5.5. For each site we define a discrete variable σ_k that can only assume the value -1 or $+1$, for all k from 1 to N. Therefore we have a total number of 2^N configurations for the lattice. The σ_k variables represent the spins of the atoms that form the metallic lattice. If all the spins are align, hence $\sigma_k = 1$ or $\sigma_k = -1$ for all $k = 1, \dots, N$, then the material exhibits a ferromagnetic property. We assume that for any two adjacent sites i and j, $(i, j) \in \mathcal{E}$, there is an interaction J_{ij} among the spin variables σ_i and σ_j, hence we build a symmetric *interaction matrix* $J \in \mathbb{R}^{N \times N}$.

Defining $\mathcal{S} = \{-1, 1\}^N$, the energy of a spin configuration $\sigma = (\sigma_1, \dots \sigma_N) \in \mathcal{S}$ is given by the following Hamiltonian:

$$\mathcal{H}(\sigma) = -\sum_{(i,j)\in\mathcal{E}} J_{ij}\sigma_i\sigma_j.$$

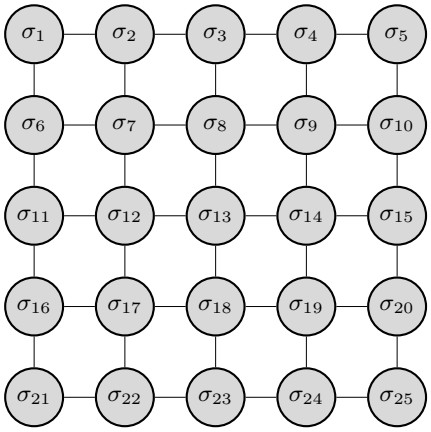


Figure 5.5: Square lattice $\mathcal{G}$ when $N = 25$; the σ_k variables are specified on each node.

In addition, a spin may interact with an external magnetic field h_k. In that case, the Hamiltonian $\mathcal{H}$ would contain the additional term $\sum_{k=1}^{N} h_k \sigma_k$. Here we focus on *zero-field* Ising models, hence $h_k = 0$ for all $k = 1, \ldots, N$.

The probability of the system of being in a configuration $\sigma \in \mathcal{S}$ is given by the *Boltzmann distribution*:

$$p(\sigma) = \frac{1}{Z^p} \exp\left(-\frac{1}{k_B T}\mathcal{H}(\sigma)\right) = \frac{1}{Z^p} \exp\left(\beta \sigma^\top J \sigma\right), \tag{5.3}$$

where k_B is the Boltzmann's constant, T is the temperature, $\beta = (k_B T)^{-1}$ is called *coldness* and Z_p is the normalization constant also referred to as *partition function*:

$$Z^p = \sum_{\sigma \in \mathcal{S}} \mathrm{e}^{-\beta \mathcal{H}(\sigma)}. \tag{5.4}$$

For simplicity let us fix $\beta = 1$. This probability distribution is useful because if A is some observable property of the system, such as its total energy or magnetization, and we can observe its value $A(\sigma)$ in each state $\sigma \in \mathcal{S}$, then we can compute its observed average thermodynamic value as

$$\langle A \rangle = \sum_{\sigma \in \mathcal{S}} A(\sigma) p(\sigma).$$

The basic problem of statistical mechanics is to calculate the sum in (5.4), since the number of states configuration arises exponentially and the calculation is infeasible for any realistic interacting system of macroscopic

size. Therefore we may try to approximate the real system with an idealization or rather make some approximation in the definition. One way is to approximate the density function (5.3) with a distribution

$$q_x(\sigma) = \frac{1}{Z^x}\exp\left(\sigma^\top J^x \sigma\right), \quad \forall\, \sigma \in \mathcal{S} \tag{5.5}$$

built over a graph $\mathcal{G}^x$ with the same nodes as $\mathcal{G}$ but with fewer of edges. The variables $x \in \{0,1\}^{|\mathcal{E}|}$ that indicate if an edge of $\mathcal{G}$ is present in $\mathcal{G}^x$. The matrix J^x in (5.5) is defined as follows:

$$J^x_{ij} = \begin{cases} x_{(i,j)} J_{ij} & \text{if } (i,j) \in \mathcal{E} \\ J_{ij} & \text{otherwise,} \end{cases}$$

for every $x \in \mathcal{X} = \{0,1\}^{|\mathcal{E}|}$. In essence, we want to do a *sparsification* of the graph $\mathcal{G}$, considering $\mathcal{G}^x$ a good approximation if the distribution q_x is close to the distribution p. We can measure the distance between two distributions p and q_x with the Kullback-Leibler (KL) divergence:

$$\begin{aligned} D_{KL}(p||q_x) &= \sum_{\sigma\in\mathcal{S}} p(\sigma)\log\left(\frac{p(\sigma)}{q^x(\sigma)}\right) = \\ &= \sum_{(i,j)\in\mathcal{E}} (J_{ij} - J^x_{ij})\mathbb{E}_p[\sigma_i\sigma_j] + \log\left(\frac{Z^x}{Z^p}\right). \end{aligned}$$

Finally, the optimization problem that we want to solve is

$$\min_{x\in\mathcal{X}} D_{KL}(p||q_x) + \lambda||x||_1, \tag{5.6}$$

where the second term of the objective function is responsible of finding an approximated graph with the minimum number of edges and λ is a penalty parameter. Eventually, we change the sign of the objective function in order to deal with a maximization problem.

We want to use the SBO algorithm to find the best sparsification of the original model, although the hypothesis of additively separability is not held. The SBO algorithm will find the best approximation of the objective function onto the space of separable functions. Notice that here the objective function is known, but Bayesian optimization is helpful since the KL divergence is an expensive to evaluate function: it is not feasible to evaluate the objective function over all the domain in order to find the optimum solution directly due to the Z^x normalization constant.

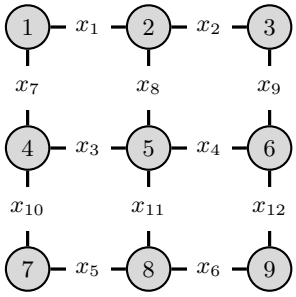


Figure 5.6: Mapping of the configuration $x \in \{0,1\}^{12}$ onto the square lattice of side 3.

5.2.2 Simulation and results

First of all notice that the number of edges in a lattice graph of side $s = \sqrt{N}$ is given by

$$|\mathcal{E}| = 2 \cdot s \cdot (s-1).$$

We decided to run two different set-ups: at first we used $N = 9$ and then we simulated with $N = 16$. The reason is that finding the optimal theoretical solution when $N = 16$ is infeasible since the computation time would last for more that 100 days[8], due to the great number of configurations: $2^{|\mathcal{E}|} = 2^{24} = 16\,777\,216$. Instead, when $N = 9$ we have that $|\mathcal{E}| = 2 \cdot 3 \cdot 2 = 12$, therefore $2^{12} = 4\,096$ and the computation time to find the theoretical solution is about 20 seconds. We used the scenario with $N = 9$ to find the best set of parameters and then we will use that set to compare the performance of the SBO algorithm with those of the Baptista and Poloczek algorithm BOCS.

The edge parameters J_{ij} are chosen randomly with uniform distribution over the interval $[0.05, 5]$. Moreover their sign will be chosen randomly with probability 50%. Given an interaction matrix J we can compute once for all the elements that constitute the probability distribution p. In particular, we will use a function `ising_model_moments` that will compute the partition function Z^p and the moments $\mathbb{E}[\sigma_i \sigma_j]$, $\forall i, j = 1, \ldots, N$, involved in the computation of the objective function.

Considering a 3×3 zero-field Ising model, $N = 9$, we have that the reticular grid has 12 edges, therefore $d = 12$ and $x \in \{0,1\}^{12}$. Notice that we

[8]When $N = 16$ one evaluation of the objective function in a configuration σ takes approximately 0.6 seconds.

		NT:		
		1	**2**	**3**
EI		1.016 ± 0.005	1.071 ± 0.050	1.051 ± 0.050
mEI	0.1	1.022 ± 0.015	1.080 ± 0.044	1.074 ± 0.049
	1	1.021 ± 0.017	1.084 ± 0.049	1.045 ± 0.047
	10	1.019 ± 0.004	1.051 ± 0.050	1.049 ± 0.048
UCB	0.1	1.943 ± 0.509	1.808 ± 0.602	1.899 ± 0.489
	1	1.757 ± 0.477	1.746 ± 0.494	1.858 ± 0.535
	10	1.671 ± 0.418	1.691 ± 0.544	1.905 ± 0.512

		NT:		
		4	**5**	**6**
EI		**1.016**$\pm$0.004	1.200 ± 0.186	1.025 ± 0.031
mEI	0.1	**1.017**$\pm$0.004	1.256 ± 0.195	1.022 ± 0.016
	1	**1.016**$\pm$0.005	1.192 ± 0.163	1.022 ± 0.030
	10	**1.015**$\pm$0.005	1.314 ± 0.250	1.025 ± 0.027
UCB	0.1	1.964 ± 0.507	**1.300**$\pm$0.197	2.132 ± 0.428
	1	1.787 ± 0.561	**1.241**$\pm$0.208	1.736 ± 0.503
	10	1.834 ± 0.489	**1.345**$\pm$0.273	1.975 ± 0.569

Table 5.6: Results after 20 executions of the algorithm: for each combination of parameters the mean value of the optimal outputs is reported, with one standard deviation error estimate.

decided to map a configuration vector x onto the grid from left to right, from top to bottom and labelling the horizontal edges at first, as shown in Figure 5.6. To run a simulation we decided to chose the parameter λ to be 10^{-1}, since a smaller value would obviously lead to choose $q_x = p$, hence to the configuration $x = \mathbb{1} \in \{0,1\}^{12}$ that corresponds to approximate $\mathcal{G}$ with itself. The size of the train observation set is set to 20; the number of executions of the algorithm for each set of parameters is also 20 and the number of iterations for each run is 100. Since the dimensionality of this setting is the same as the Binary Quadratic Programming problem discussed in Section 5.1 ($d = 12$ and $a = 2$), the cardinalities of the neighbourhood $\mathcal{X}_x$ for each NT are the same as those collected in Table 5.1.

Table 5.6 contains a summary of the results obtained with the 20 executions of the BO loop using the SBO algorithm, with the different sets of parameters; the best performance is set in bold. Notice that here we changed

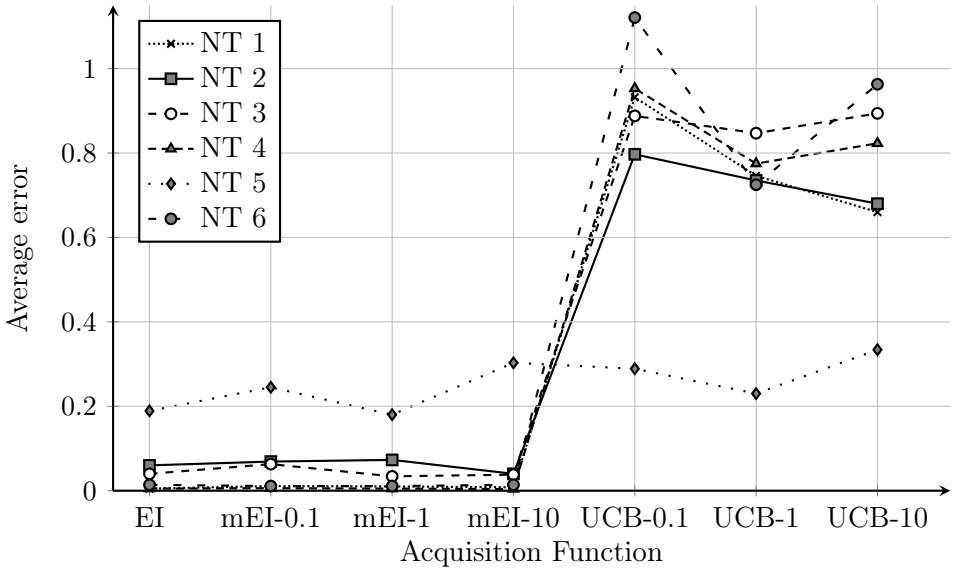


Figure 5.7: Plot of the errors averaged over 20 repetitions of the BO loop, for each acquisition function.

the sign of the results since the original problem was a minimization problem. Instead, Figure 5.7 is a plot of the average error (5.2) made with each set of parameter: the optimal minimum value is 1.011.

We immediately notice the same behaviour of the Binary Quadratic Programming problem, namely better performance with EI and mEI regarding UCB. Morevore, NT 5 is again inefficient since the cardinality of the neighbourhood is always 1. The best performance is obtained with neighbourhood type 4 and with Expected Improvement, though the other NTs (except for the fifth) are close to the best performance in terms of the average error.

When dealing with a 4×4 Ising model built on a square lattice with $N = 16$ nodes and $|\mathcal{E}| = 24$ edges, the computation of the theoretical optimum is infeasible. We will then choose the sets of parameters that performed at best with $N = 9$ and compare the results with the BOCS algorithm. In particular, we will consider EI and mEI as Acquisition functions. The best performing NT parameter was 4. However, when $d = 24$, the cardinality of the neighbourhood $\mathcal{X}_x$ is equal to 12 950, and it is too expensive using NT 4 in this case. Therefore, we will run the SBO algorithm with the second most performing NT parameter, hence the neighbourhood type 1. For NT 1, we

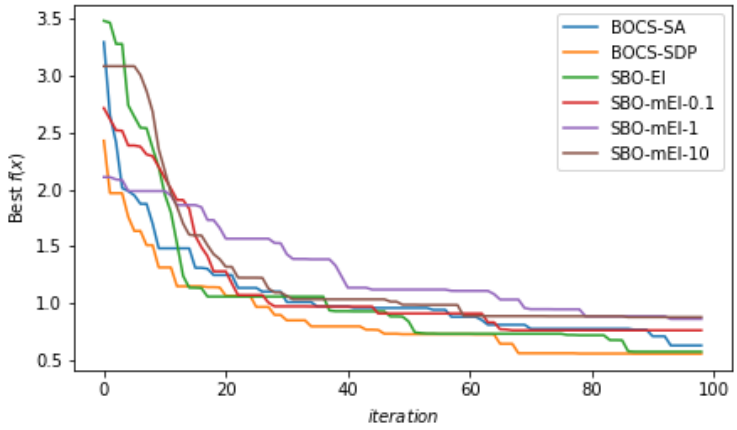


Figure 5.8: Average best output at each iteration, for each algorithm.

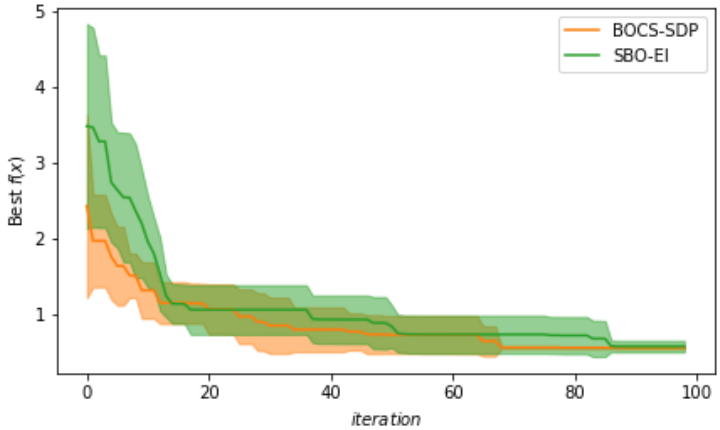


Figure 5.9: Average best output ± 1 standard deviation error estimate, for BOCS-SDP and SBO with EI.

have that $|\mathcal{X}_x| = 24$.

Figure 5.8 shows the average best output at each iteration of the analysed algorithms. The average is computed on 7 executions of each algorithm. We notice that again the performance of the SBO algorithm is competitive with those of the BOCS algorithm, in both variations, BOCS-SDP and BOCS-SA. Differently from the BQP problem, where on average BOCS converged faster, here we don't see a clear difference in the two algorithms.

Figure 5.9 shows the average best output with one standard deviation

BOCS-SA		0.628 ± 0.097
BOCS-SDP		0.555 ± 0.015
SBO-EI		0.573 ± 0.076
SBO-mEI	$k = 0.1$	0.761 ± 0.355
	$k = 1$	0.861 ± 0.335
	$k = 10$	0.878 ± 0.340

Table 5.7: The mean value of the optimal outputs is reported for each algorithm, with one standard deviation error estimate.

error estimation. We chose to plot only the performance of BOCS-SDP and of SBO with classic Expected improvement, since they reached the smallest values of output, on average. The two algorithms have similar behaviour in terms of variance, however, BOCS is closer to the optimal solutions some iterations before SBO.

Lastly, Table 5.7 shows the optimum found with each algorithm averaged over 7 repetitions. Although the best performance is obtained with BOCS-SDP, the SBO algorithm is able to reach similar results in fewer time.

Chapter 6

Self-Optimizing Mobile Networks Case Study

This Chapter is dedicated to the employment of the SBO algorithm in a case study proposed by the TIM S.p.A company. The problem is to realize a dynamic solution to the self-optimization of next-generation mobile networks. In particular, given a network of antennas (also referred to as *cells*), they focus on create an AI algorithm that automatically set the orientation of each antenna, in order to obtain better performance in terms of capacity and coverage (Capacity and Coverage Optimization). In fact, unlike other networks made of omni-directional antennas, this kind of mobile networks uses *sector antennas* which are directional antennas that radiates in specific directions, typically circular sectors of 60, 90 or 120 degrees. The coverage area can be adjusted by two types of tilting, mechanical or electronic. The tilt angle is intended to be vertical with respect to the horizon (Figure 6.1).

In the case study the adjustment can be done only to the electronic tilts of the antennas, and specifically, TIM defined a fixed set of tilts value for each antenna. The problem arose since no automatic optimization have been done till now: in fact, TIM designers of the networks plan the antennas setting according to their experience, to the configurable parameters and to regulatory constraints. They asked for a model that is capable of exploring the inputs space and finding the optimal solutions, while interacting continuously with the reference environment. Therefore, the algorithm should observe an outcome and find the subsequent optimal solution, changing the tilt of each antenna accordingly. This is exactly the modus operandi of

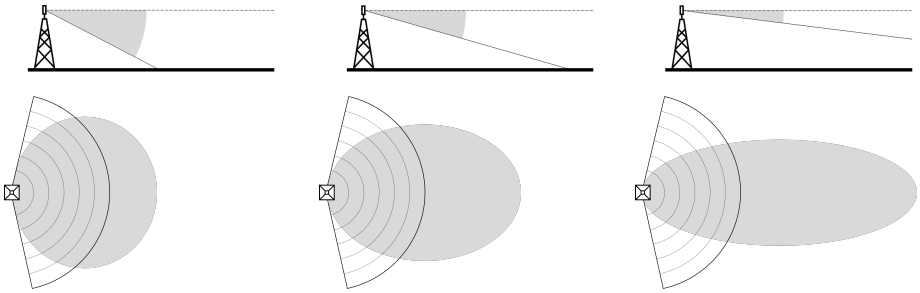


Figure 6.1: Qualitative representation of how a sector antenna works. The gray angles are the tilt angles and the gray ellipses corresponds to the coverage areas: a smaller angle lead to the coverage of further areas, while a bigger angle has more wide coverage but for closer areas.

Bayesian optimization, and in particular, since the domain is discrete (the set of potential tilts is discrete), we will apply the SBO algorithm.

6.1 Problem specification

The problem at hand deals with a mobile network of cells located in a small size town and surrounding area. The network is made of 126 cells of which only 12 are modifiable. Therefore, the SBO algorithm will handle with 12 variables $x_1, \ldots, x_{12}$ ($d = 12$). The other cells are not modifiable and their contribution will be gathered into the stubborn node x_0. Moreover, for each antenna x_i, with i from 1 to 12, TIM defined an alphabet $\mathcal{A}$ of 5 distinct tilts, which is the discrete domain of x_i. These domains will all be codified from 1 to 5. In the end, the feature space $\mathcal{X}$ has cardinality $|\mathcal{X}| = 5^{12}$.

In order to use a generic Machine Learning algorithm, TIM firstly needed to combine together several KPIs monitored by the company. Therefore, TIM mobile planning experts defined an appropriate Target function $F_{target}(\cdot)$ that allows us to aggregate the outputs given by the system in a unique response value $y = F_{target}(x)$, for all tilt configuration x in the feature space. Notice that this function does not explain how the output varies with the input; the black-box part of the problem is to obtain the outputs from the system: the only purpose of the function $F_{target}(\cdot)$ is to combine these outputs together.

Since we don't know directly how the output depends on the input, we

NT:	**1**	**2**	**3**	**4**	**5**	**6**
$\lvert\mathcal{X}_x\rvert$	48	264	`num_neigh`	793	64	24 or $2M$

Table 6.1: Cardinality of the neighbourhood for each NT. Note that the cardinality of $\mathcal{X}$ is $a^d = 244\,140\,625$.

apply the SBO algorithm to find a projection of the unknown black-box function onto the space of additively separable function over pairwise connections. Namely, we give the usual structure to the surrogate model built over the complete graph:

$$f_\Lambda(x) = \sum_{0 \le i < j \le 12} \lambda_{ij}(x_i, x_j), \quad \forall\, x \in \{0, \dots, 5\}^{12}.$$

The number N of distinct $\lambda_{ij}(s,t)$, which is also the size of the random vector Λ, can be computed from (4.2), obtaining $N = 1710$.

6.2 Results

To run a simulation we let the acquisition function and the neighbourhood type vary, as done for the previous examples. We fixed the number of executions of the algorithm to 15 for each combination of parameters, and fixed the number of iterations for each loop to 600. In summary, we defined the following domains:

$$\begin{aligned}
&\text{AF} \in \{\text{EI}, \text{mEI}, \text{UCB}\},\\
&\text{AF parameter} \in \{0, 0.1, 1, 10\},\\
&\text{NT} \in \{1, 2, 3, 4, 5, 6\}.
\end{aligned}$$

Each neighbourhood type produces a set of neighbours $\mathcal{X}_x$ with different cardinality, see Table 6.1: in this case $d = 12$ and $a = 5$.

We set `num_neigh`$= 100$ and $M = 20$ and we choose to use 50 training observations to initialize the algorithm.

Table 6.2 contains a summary of the results obtained with 15 executions of the SBO algorithm. The values are multiplied for 10^2 for better confrontation; the best performance for each acquisition function is set in bold. Figure 6.2 is a visualization of the previous tables. We notice that apart from NT 1, the performance of the other sets of parameters is quite similar and there

		NT:		
		1	**2**	**3**
EI		77.25 ± 9.17	87.51 ± 0.32	86.46 ± 1.56
mEI	0.1	81.50 ± 7.73	87.12 ± 0.62	86.98 ± 1.51
	1	82.59 ± 7.89	85.80 ± 4.49	87.03 ± 1.06
	10	82.25 ± 5.03	87.33 ± 0.48	86.85 ± 1.16
UCB	0.1	77.30 ± 9.46	86.72 ± 1.53	86.57 ± 1.47
	1	82.29 ± 7.43	87.13 ± 0.50	86.34 ± 1.79
	10	86.06 ± 4.27	86.72 ± 1.29	86.69 ± 1.49

		NT:		
		4	**5**	**6**
EI		85.28 ± 4.64	**87.57**± 0.32	86.91 ± 1.67
mEI	0.1	84.25 ± 6.06	**87.50**± 0.28	87.33 ± 1.19
	1	85.86 ± 4.48	**87.58**± 0.16	86.46 ± 2.17
	10	85.34 ± 4.57	**87.59**± 0.16	86.54 ± 1.83
UCB	0.1	86.06 ± 1.90	**87.43**± 0.23	86.32 ± 1.92
	1	84.61 ± 5.58	**87.67**± 0.23	86.59 ± 2.04
	10	85.99 ± 2.07	87.49 ± 0.22	**87.78**± 0.14

Table 6.2: For each combination of parameters the mean value of the optimal outputs is reported, with one standard deviation error estimate.

is no affection by the choice of Acquisition function. Unlike the previous applications, here the UCB acquisition function can still be taken into account. However, the best results are obtained with the fifth neighbourhood type, also in terms of smallest variance. Focusing on NT 5, plot 6.3 contains the density plot of the ouputs for each acquisition function. We also plot the density of the outputs when dealing with NT 6 since it obtains the best average reward when dealing with Upper confidence bound with $\beta = 10$. We notice that with neighbourhood type 5 the shape of the densities are almost similar, in some cases exhibiting a bi-modal distribution. Instead, NT 6 performs way worse with the others acquisition function, in terms of lower averages and bigger variances.

As regards computation times, the algorithms takes on average $88.46 \cdot 10^{-3}$ seconds to update the prior distribution over Λ. The average total times of execution is summed up in Table 6.3 (again compared to the total number

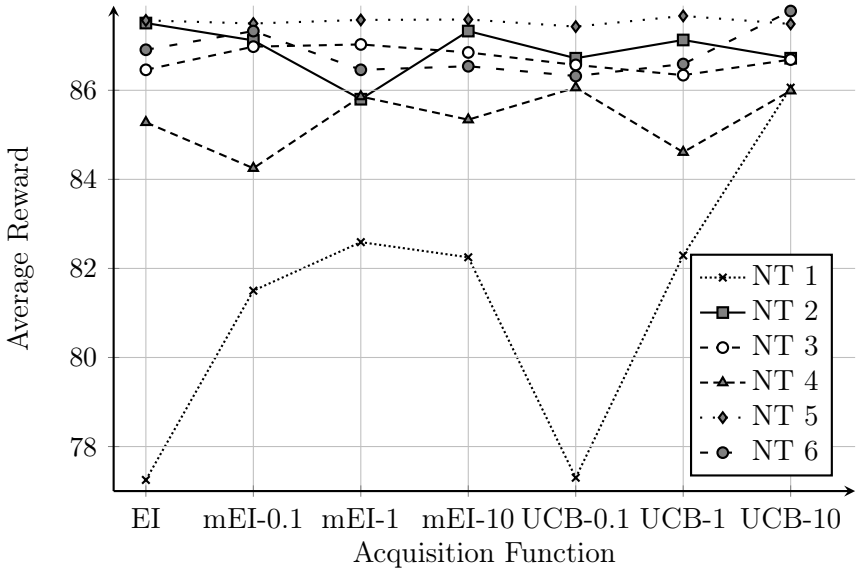


Figure 6.2: Plot of the average reward obtained in 15 repetitions of the BO loop, for each acquisition function.

of analysed inputs[9]).

Notice that though one may think that a greater number of analysed inputs would lead to better results since the domain is more explored (for example with NT 4), actually some inputs are included in a neighbourhood more than once because typically the search in the feature space get stuck in a subset of $\mathcal{X}$ where x_i is fixed for some $i \in \{1, \dots, d\}$. On the contrary, Neighbourhood type 5 and 6 were conceived to shake and change all the tilts of a configuration at least once in order to move occasionally further in the search space.

Lastly, let us recall that we initialized prior mean and covariance matrix with sample mean and variance computed from the observations train set. In particular, the prior variance obtained from the train sets typically varies in the interval $[2 \cdot 10^{-4}, 5 \cdot 10^{-4}]$. Table 6.4 sums up the average result when changing the prior variance, considering NT 5 and only EI and mEI. In particular we tried with values of different magnitude $\{10^{-8}, 10^{-1}, 10\}$ and we also reported the values obtained with the sample variance (10^{-4}). We

[9]The total number of possible configurations is $|\mathcal{X}| = 5^{12} = 244\,140\,625$.

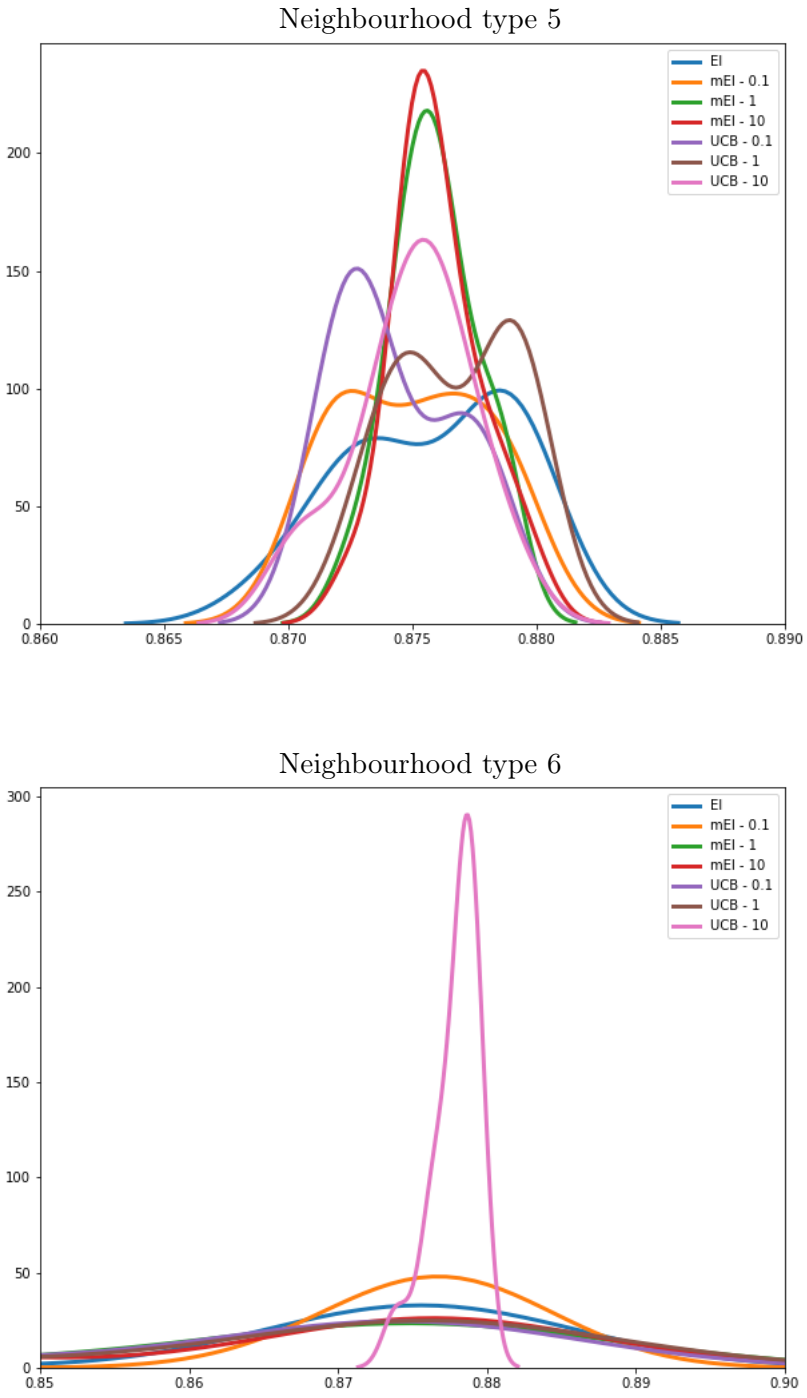


Figure 6.3: Density plot built over the 15 outputs, for each acquisition function and with NT 5 and NT 6.

NT:	**1**	**2**	**3**	**4**	**5**	**6**
avg total time with EI (in s)	71.49	161.78	426.81	432.55	84.11	66.55
avg total time with UCB (in s)	63.46	122.25	335.40	259.27	72.97	63.67
total number of explored inputs	28 800	158 400	60 000	475 800	278 400	19 200
percentage of explored inputs	0.01%	0.06%	0.02%	0.19%	0.11%	0.007%

Table 6.3: Average total time of execution of the algorithm, for each acquisition function (EI and mEI are gathered under EI since they have similar performance).

		Prior variance			
		10^{-8}	10^{-4}	10^{-1}	10
EI		87.51	87.57	87.75	87.64
mEI	0.1	87.44	87.50	87.76	87.62
	1	87.55	87.58	87.78	87.58
	10	87.43	87.59	87.69	87.62

Table 6.4: Average optimal output for each combination of acquisition function and prior variance.

notice that these results vary almost with the same rate for each value, although the variation is millesimal. This suggests to us that the algorithm quite depends on the initialization of the prior distribution. As suggested in Section 4.4.1, one may think of doing some tuning of these hyperparameters or rather giving a better estimate to them, instead of simplify the discussion as we did, considering sample mean and variance.

Chapter 7

Conclusion

This work explored the difficulties of optimizing black-box or expensive-to-evaluate objective functions. Bayesian Optimization is the state of the art in optimizing this kind of problems. BO algorithms are able to exploit the very little information the black-box function provides. This exploitation leads to a successful searching in the feature space, asking for the smallest number of evaluations of the objective function. Surrogate models and acquisition functions are the responsible of the success of Bayesian Optimization. The first ones are responsible of approximate the objective function, providing a distribution over the objective function space and other desirable properties like smoothness. The second ones are in charge of choosing the next point in the feature space to be sampled. In the literature, the power of these two building blocks have been exploited in different directions, in order to adapt the BO approach to several applications.

When dealing with discrete variables, classic Bayesian Optimization models need to be redesigned and modified in order to be successful also on discrete domains. In particular, Gaussian Processes as surrogate models are no longer performing, since they are not able to provide the property of smoothness over a discrete space. Moreover, maximizing an acquisition function over a discrete space is an infeasible task when the feature space has high dimensions. Several works and algorithms have been proposed in the literature to take the strengths of Bayesian Optimization algorithm and project them onto the discrete world.

Our proposal, the Separable Bayesian Optimization algorithm, turned out to be competitive with the others in terms of performance on classic benchmark problems. The main idea of the algorithm is to give a structure to

the unknown objective function. We assumed an additively separable model for the objective function, whose addends are functions of single variables or pairwise combinations of variables. Moreover, SBO was developed in order to bring the properties of Gaussian Processes in the discrete world. We decomposed Bayesian Optimization with Gaussian Processes in its steps, and bearing in mind the structure we gave to the objective function, we reassembled them for the discrete optimization.

We presented a simplistic version of the SBO algorithm. However, several adjustments can be done to enhance the results. This can lead to better performance also for more complex problems, like the TIM application. As an example, the tuning of hyperparameters has been done in a simple way: more refined approximation or tunings can be done. To name some, we used sample means and covariances to initialize the random variables involved in the SBO algorithm. Instead, these parameters can be estimated in different ways, using the observations train set. For example we can use a maximum likelihood estimation or we can consider them as random variables. In fact, we can provide a prior distribution over the hyperparameters that would be updated with new observation, as done with the Horseshoe prior in BOCS and COMBO. Another improvement can be done by considering different kernel functions or different acquisition functions. Here we focused on Expected Improvement and Upper Confidence Bound, which are defined for continuous domains. A maximization of these two functions over a discrete domain is infeasible. We got around the problem by maximizing the acquisition functions only on a small subset of the feature space. However, other techniques can be used in the acquisition function maximization step. For example we can use a random walk as done with simulated annealing in BOCS-SA, or we can optimize the search like in COMBO with the breadth-first local search. Moreover, we focused on the six neighbourhood types defined in Section 4.4: obviously infinite many other types can be implemented, according to the structure of the problem and to the faults that need adjustments. For example, NT 5 and 6 were conceived to move occasionally further since the search tended to stuck in small subsets of the feature space.

Lastly, the simplicity of the SBO algorithm as describe in this thesis positively affects the execution times of the algorithm itself. In fact, the SBO as implemented in Chapter 4 is faster than the other algorithms we used on

same applications, though the optimum is reached with more iterations. In summary, the Separable Bayesian Optimization algorithm is a valid means to solve optimization problems with black-box functions. Some adjustments can be done to enhance the performance, however we proved that also in its simplistic version it has comparable performance with respect to other algorithms proposed in literature.

Appendix

A Conditioning of a multivariate Gaussian distribution

In the following, the property of conditioning of a generic multivariate Gaussian distribution is enunciated and proved[10].

Proposition A.1. *Let $n, m \in \mathbb{N}$. If X is an n-dimensional random vector and Y an m-dimensional random vector that jointly have an $(n+m)$-variate Gaussian distribution*

$$(X, Y) \sim \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}_X \\ \boldsymbol{\mu}_Y\end{pmatrix}, \begin{pmatrix}\Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY}\end{pmatrix}\right), \tag{A.1}$$

then $Y \mid X$ has the following m-variate Gaussian distribution:

$$Y \mid X \sim \mathcal{N}\left(\boldsymbol{\mu}_Y + \Sigma_{YX}\Sigma_{XX}^{-1}(X - \boldsymbol{\mu}_X), \Sigma_{YY} - \Sigma_{YX}\Sigma_{XX}^{-1}\Sigma_{XY}\right). \tag{A.2}$$

Proof. The density function of an N-multivariate Gaussian random variable $Z \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is

$$f(\mathbf{z}) = (2\pi)^{-\frac{N}{2}}|\Sigma|^{-\frac{1}{2}}\mathrm{e}^{-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu})^\top\Sigma^{-1}(\mathbf{z}-\boldsymbol{\mu})}, \quad \forall\, \mathbf{z} \in \mathbb{R}^N.$$

Moreover, given the joint Gaussian distribution (A.1), the marginal distribution of the random vector X is still a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}_X$ and covariance matrix Σ_{XX}. By using the definition of conditional distribution, $\forall\, \mathbf{x} \in \mathbb{R}^n$ and $\forall\, \mathbf{y} \in \mathbb{R}^m$ realizations of X and Y

[10] The proof is taken from Dablander [11].

respectively, we have that

$$
\begin{aligned}
f(\mathbf{y} \mid X = \mathbf{x}) = \frac{f(\mathbf{x}, \mathbf{y})}{f(\mathbf{x})} = \\
&= \frac{(2\pi)^{-\frac{n+m}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \begin{pmatrix} \mathbf{x}-\boldsymbol{\mu}_X \\ \mathbf{y}-\boldsymbol{\mu}_Y \end{pmatrix}^\top \Sigma^{-1} \begin{pmatrix} \mathbf{x}-\boldsymbol{\mu}_X \\ \mathbf{y}-\boldsymbol{\mu}_Y \end{pmatrix}\right)}{(2\pi)^{-\frac{n}{2}} |\Sigma_{XX}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x-\boldsymbol{\mu}_X)^\top \Sigma_{XX}^{-1}(x-\boldsymbol{\mu}_X)\right)} = \\
&= (2\pi)^{-\frac{m}{2}} |\Sigma|^{-\frac{1}{2}} |\Sigma_{XX}|^{\frac{1}{2}} \cdot \\
&\quad \cdot \exp\left(-\frac{1}{2}\left[\begin{pmatrix} \mathbf{x}-\boldsymbol{\mu}_X \\ \mathbf{y}-\boldsymbol{\mu}_Y \end{pmatrix}^\top \Sigma^{-1} \begin{pmatrix} \mathbf{x}-\boldsymbol{\mu}_X \\ \mathbf{y}-\boldsymbol{\mu}_Y \end{pmatrix} - (\mathbf{x}-\boldsymbol{\mu}_X)^\top \Sigma_{XX}^{-1}(\mathbf{x}-\boldsymbol{\mu}_X)\right]\right).
\end{aligned}
$$

To make the notation easier let us define $\bar{\mathbf{x}} = \mathbf{x} - \boldsymbol{\mu}_X$ and $\bar{\mathbf{y}} = \mathbf{y} - \boldsymbol{\mu}_Y$. The inverse of a 2×2 block matrix is given by the following formula:

$$
\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{pmatrix} \tag{A.3}
$$

where $S = D - CA^{-1}B$ is the Schur complement[11] of the block A of the matrix. In this case, the Schur complement of the block Σ_{XX} of the variance matrix Σ is $\Sigma_{YY} - \Sigma_{YX}\Sigma_{XX}^{-1}\Sigma_{XY}$, but to make the notation easier let us define the following matrix:

$$
\Omega = \begin{pmatrix} \Omega_{XX} & \Omega_{XY} \\ \Omega_{YX} & \Omega_{YY} \end{pmatrix} = \Sigma^{-1}
$$

Doing the computation, we get:

$$
\begin{aligned}
f(\mathbf{y} \mid X = \mathbf{x}) &= (2\pi)^{-\frac{m}{2}} |\Sigma|^{-\frac{1}{2}} |\Sigma_{XX}|^{\frac{1}{2}} \cdot \\
&\cdot \exp\left(-\frac{1}{2}\left[\bar{\mathbf{x}}^\top \Omega_{XX}\bar{\mathbf{x}} + \bar{\mathbf{y}}^\top \Omega_{YX}\bar{\mathbf{x}} + \bar{\mathbf{x}}^\top \Omega_{XY}\bar{\mathbf{y}} + \bar{\mathbf{y}}^\top \Omega_{YY}\bar{\mathbf{y}} - \bar{\mathbf{x}}^\top \Sigma_{XX}^{-1}\bar{\mathbf{x}}\right]\right) = \\
&= (2\pi)^{-\frac{m}{2}} |\Sigma|^{-\frac{1}{2}} |\Sigma_{XX}|^{\frac{1}{2}} \cdot \\
&\cdot \exp\left(-\frac{1}{2}\left[\bar{\mathbf{x}}^\top (\Omega_{XX} - \Sigma_{XX}^{-1})\bar{\mathbf{x}} + 2\bar{\mathbf{y}}^\top \Omega_{YX}\bar{\mathbf{x}} + \bar{\mathbf{y}}^\top \Omega_{YY}\bar{\mathbf{y}}\right]\right)
\end{aligned}
$$

Let us add and subtract the following quantity at the exponent:

$$
\bar{\mathbf{x}}^\top \Omega_{XY}\Omega_{YY}^{-1}\Omega_{YY}\Omega_{YY}^{-1}\Omega_{YX}\bar{\mathbf{x}}.
$$

[11] See Zhang [40], Chap. 1.1.

In such a way, we get a quadratic form at the exponent, as follows:

$$\begin{aligned} f(\mathbf{y} \mid X = \mathbf{x}) &= (2\pi)^{-\frac{m}{2}} |\Sigma|^{-\frac{1}{2}} |\Sigma_{XX}|^{\frac{1}{2}} \cdot \\ &\cdot \exp\left(-\frac{1}{2} (\bar{\mathbf{y}} + \Omega_{YY}^{-1} \Omega_{YX} \bar{\mathbf{x}})^\top \Omega_{YY} (\bar{\mathbf{y}} + \Omega_{YY}^{-1} \Omega_{YX} \bar{\mathbf{x}}) \right) \cdot \\ &\cdot \exp\left(-\frac{1}{2} \left[\bar{\mathbf{x}}^\top (\Omega_{XX} - \Sigma_{XX}^{-1} - \Omega_{XY} \Omega_{YY}^{-1} \Omega_{YY} \Omega_{YY}^{-1} \Omega_{YX}) \bar{\mathbf{x}} \right] \right) \end{aligned}$$

Notice that $\Omega_{XX} - \Sigma_{XX}^{-1} = \Omega_{XY} \Omega_{YY}^{-1} \Omega_{YY} \Omega_{YY}^{-1} \Omega_{YX}$. In fact, using (A.3) we have

$$\begin{aligned} &\Omega_{XY} \Omega_{YY}^{-1} \Omega_{YY} \Omega_{YY}^{-1} \Omega_{YX} - \Omega_{XX} = \Omega_{XY} \Omega_{YY}^{-1} \Omega_{YX} - \Omega_{XX} = \\ &\quad = \left(\Sigma_{XX}^{-1} \Sigma_{XY} S^{-1} S S^{-1} \Sigma_{YX} \Sigma_{XX}^{-1} \right) - \left(\Sigma_{XX}^{-1} + \Sigma_{XX}^{-1} \Sigma_{XY} S^{-1} \Sigma_{YX} \Sigma_{XX}^{-1} \right) = \\ &\quad = \left(\Sigma_{XX}^{-1} \Sigma_{XY} S^{-1} \Sigma_{YX} \Sigma_{XX}^{-1} \right) - \left(\Sigma_{XX}^{-1} \Sigma_{XY} S^{-1} \Sigma_{YX} \Sigma_{XX}^{-1} \right) - \Sigma_{XX}^{-1} = \\ &\quad = -\Sigma_{XX}^{-1}, \end{aligned}$$

where $S = \Sigma_{YY} - \Sigma_{YX} \Sigma_{XX}^{-1} \Sigma_{XY}$. Moreover, note that the determinant of a block matrix factors as follows:

$$\begin{vmatrix} A & B \\ C & D \end{vmatrix} = |D - C A^{-1} B| \cdot |A|,$$

therefore $|\Sigma| = |S| \cdot |\Sigma_{XX}|$. Eventually, we obtain an m-dimensional Gaussian density function for the conditional distribution:

$$\begin{aligned} &f(\mathbf{y} \mid X = \mathbf{x}) = (2\pi)^{-\frac{m}{2}} |\Sigma_{YY} - \Sigma_{YX} \Sigma_{XX}^{-1} \Sigma_{XY}|^{-\frac{1}{2}} \cdot \\ &\quad \cdot \exp\left(-\frac{1}{2} (\bar{\mathbf{y}} + \Omega_{YY}^{-1} \Omega_{YX} \bar{\mathbf{x}})^\top \Omega_{YY} (\bar{\mathbf{y}} + \Omega_{YY}^{-1} \Omega_{YX} \bar{\mathbf{x}}) \right) = \\ &= (2\pi)^{-\frac{m}{2}} |\Sigma_{YY} - \Sigma_{YX} \Sigma_{XX}^{-1} \Sigma_{XY}|^{-\frac{1}{2}} \cdot \\ &\quad \cdot \exp\left(-\frac{1}{2} (\bar{\mathbf{y}} + S(-S^{-1} \Sigma_{YX} \Sigma_{XX}^{-1}) \bar{\mathbf{x}})^\top S^{-1} (\bar{\mathbf{y}} + S(-S^{-1} \Sigma_{YX} \Sigma_{XX}^{-1}) \bar{\mathbf{x}}) \right) = \\ &= (2\pi)^{-\frac{m}{2}} |\Sigma_{YY} - \Sigma_{YX} \Sigma_{XX}^{-1} \Sigma_{XY}|^{-\frac{1}{2}} \cdot \\ &\quad \cdot \exp\left(-\frac{1}{2} (\bar{\mathbf{y}} - \Sigma_{YX} \Sigma_{XX}^{-1} \bar{\mathbf{x}})^\top S^{-1} (\bar{\mathbf{y}} - \Sigma_{YX} \Sigma_{XX}^{-1} \bar{\mathbf{x}}) \right), \end{aligned}$$

hence, we have the thesis

$$Y \mid X = x \sim \mathcal{N}\left(\boldsymbol{\mu}_Y + \Sigma_{YX} \Sigma_{XX}^{-1} (x - \boldsymbol{\mu}_X), \Sigma_{YY} - \Sigma_{YX} \Sigma_{XX}^{-1} \Sigma_{XY} \right).$$

□

B Implementation details: handling of indexes

This appendix is dedicated to describe the functions whose task is to code the indexes of the random variables $\Lambda_{ij}(s,t)$ into the vector Λ. Specifically, they are the following: `indx_Matrices`, `indx_x`, `indx_VariablesToVector`, and `indx_VectorToVariables`. Here is the code for the first function (`n_vars` corresponds to d and `n_vals` is a):

```
def indx_Matrices(n_vars, n_vals):
    n_lambda = int(n_vars*(n_vars+1)/2)
    Var_indx = np.zeros([n_vars, n_vars])
    Var_indx_list = np.arange(n_vars, n_lambda)
    Var_indx[np.triu_indices(n_vars,1)]=Var_indx_list
    np.fill_diagonal(Var_indx, np.arange(n_vars))

    n_complete = int(n_vars*(n_vars-1))/2
    Vect_indx = np.zeros([n_vars, n_vars])
    Vect_indx_list = (n_vals**2)*np.arange(n_complete)
            +n_vals*n_vars
    Vect_indx[np.triu_indices(n_vars,1)]=Vect_indx_list
    np.fill_diagonal(Vect_indx, n_vals*np.arange(n_vars))

    X_indx_list = np.arange(n_vals**2)
    X_indx = X_indx_list.reshape(n_vals, n_vals)
    return Var_indx, X_indx, Vect_indx
```

The output of this function are the following three matrices (for the examples we set $d = 6$ and $a = 4$):

- `Var_indx` is an upper triangular matrix that contains an integer for each distinct Λ_{ij}. In particular, the main diagonal contains the Λ_{0j}, $j = 1\ldots, d$, while the others are organized in the upper part of the matrix. Here is an example of the arrangement of the variables for $d = 6$ and $a = 4$:

$$\begin{bmatrix} 0 & 6 & 7 & 8 & 9 & 10 \\ 0 & 1 & 11 & 12 & 13 & 14 \\ 0 & 0 & 2 & 15 & 16 & 17 \\ 0 & 0 & 0 & 3 & 18 & 19 \\ 0 & 0 & 0 & 0 & 4 & 20 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix}.$$

In fact, the six Λ_{0j} are placed on the diagonal, while the others 15 are arranged on the upper triangular matrix, row by row. To order the Λ_{ij} we can consider i and j as the row index and column index of the matrix: in particular, apart from those on the diagonal, ordered as $\Lambda_{01}, \ldots, \Lambda_{0d}$, the generic Λ_{ij} is placed in the position (i, j) of the matrix, considering i, j from 1 to d.

- `Vect_indx` is also an upper triangular matrix that contains the first index of a subvector of Λ corresponding to a couple of nodes of the graph (ether 0 and j or i and j). In particular, on the main diagonal there are the indexes of $\Lambda_{0j}(0, 1)$, $j = 1 \ldots, d$, and above the main diagonal it contains the indexes of $\Lambda_{i,j}(1, 1)$, $i, j = 1, \ldots, d$, $i < j$; the arrangement of the random variables is the same as the `Var_indx` matrix, and in the above-mentioned example we have:

$$\begin{bmatrix} 0 & 24 & 40 & 56 & 72 & 88 \\ 0 & 4 & 104 & 120 & 136 & 152 \\ 0 & 0 & 8 & 168 & 184 & 200 \\ 0 & 0 & 0 & 12 & 216 & 232 \\ 0 & 0 & 0 & 0 & 16 & 248 \\ 0 & 0 & 0 & 0 & 0 & 20 \end{bmatrix}.$$

Notice that the distance from the values of the diagonal is exactly a, while the elements in the upper side have a difference of a^2, since it is the number of combinations of values (s, t). For example, the value 136 in position $(2, 5)$ means that the element $\Lambda[136]$ of the Λ vector corresponds to $\Lambda_{2,5}(1, 1)$. Instead, element $\Lambda_{04}(0, 1)$ can be found in position 12 of the Λ vector.

- `X_indx` is an $a \times a$ matrix that contains an index for each combination of value (s, t). It is useful since we can give an order to these combinations: in fact, we can consider s and t as the row index and column index of the matrix `X_indx`, indexed from 1 to a; in the example:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix}.$$

Notice that the (s,t) are ordered by incrementing the second value first, and then the first: in essence we have the following arrangement of the values for each Λ_{ij}

$$\begin{array}{c}(1,1)\\(1,2)\\\vdots\\(1,a)\\(2,1)\\\vdots\\(a,a)\end{array}$$

The following functions allows us to combine together the information we gathered separately from the previous matrices. `indx_VariablesToVector` maps the four indexes (i,j,s,t) into an integer.

```
def indx_VariablesToVector(var1, var2, x1, x2):
    if var1>var2:
        return print('ERROR: var1>var2')
    ij = int(Vect_indx[int(var1), int(var2)])
    if var1==var2:
        if x1!=x2:
            return print('ERROR: if var1=var2 then x1=x2')
        else:
            st = x1
    else:
        st = int(X_indx[int(x1), int(x2)])
    return int(ij+st)
```

At first it takes the index of the $\Lambda_{ij}(1,1)$ variable, namely `ij`, then it looks for the combination (s,t) into the matrix `X_indx`, obtaining the value `st`. The function returns the sum of the two indexes `ij` and `st`. Notice that for the Λ_{0j} random variables, this function must be called as `indx_VariablesToVector(j,j,t,t)`, specifying twice the index j. In fact, the element $\Lambda_{0j}(0,t)$ is in position (j,j) of the matrix `Vect_indx`. Moreover, the value `st` corresponds trivially to the parameter t.

The function `indx_VectorToVariables` does the opposite job: given an index k of the Λ vector it returns the combination of indexes (i,j,s,t).

```
def indx_VectorToVariables(k, n_vars, n_vals):
    if k<(n_vars*n_vals):
        r = k % n_vals
        q = k // n_vals
        if q==0:
            var1, var2 = 0, 0
            x1, x2 = r, r
        else:
            var1, var2 = q, q
            x1, x2 = r, r
    else:
        r = (k-n_vars*n_vals) % (n_vals**2)
        if r==0:
            var1, var2 = np.where(Vect_indx == k)
            var1, var2 = var1[0], var2[0]
            x1, x2 = 0, 0
        else:
            var1, var2 = np.where(Vect_indx == k-r)
            var1, var2 = var1[0], var2[0]
            x1, x2 = np.where(X_indx == r)
            x1, x2 = x1[0], x2[0]
    return var1, var2, x1, x2
```

The first `if` condition is dedicated to the Λ_{0j}: since these variable are $d \cdot a$ and are the first ones in the vector, the check is on $k < d \cdot a$. In this case, the quotient of k/d gives the index j of the second variable[12], while the reminder of k/d gives the value t of the variable x_j. In the general case, we take the reminder r of $(k - d \cdot a)/a^2$ to find the index of (s, t) in the `X_indx` matrix. If the reminder is null then we can directly find the value k into the `Vect_indx` matrix, since $(s, t) = (0, 0)$ (that in Python means $(1, 1)$). This gives us the indexes (i, j) and therefore k would correspond to the variable $\Lambda_{ij}(1, 1)$. If the reminder is not null, then we can find the value $k - r$ into the `Vect_indx` matrix to get the (i, j) indexes.

The last function is `indx_x`: given an input x, it returns the list of indexes of combinations (i, j, x_i, x_j) inside the vector Λ. For each combination (i, j)

[12]Note that the indexes of vectors and matrices in Python start from 0, and not from 1: with this in mind, the element $\Lambda_{0j}(0, 1)$ is in position $(j - 1, j - 1)$ of the `Vect_indx` matrix.

with $i < j$ it calls the `indx_VariablesToVector` function to get the index k.

```
def indx_x(x):
    n_var = np.max(x.shape)
    indices = []
    for k in range(n_var):
        indices.append(indx_VariablesToVector(k, k, x[k], x[k]))

    for var_i in range(n_var):
        for var_j in range(n_var):
            if var_i<var_j:
                indices.append(indx_VariablesToVector(var_i,
                  var_j, x[var_i], x[var_j]))
    return indices
```

Bibliography

[1] Agnihotri A., Batra N. (2020) - **Exploring Bayesian Optimization** (`https://distill.pub/2020/bayesian-optimization`);

[2] Al-Dujaili A. (2018) - **Expected Improvement for Bayesian Optimization: A Derivation** (`http://ash-aldujaili.github.io/blog/2018/02/01/ei/`);

[3] Alto V. (2019) - **Neural Networks: parameters, hyperparameters and optimization strategies** (`https://towardsdatascience.com/neural-networks-parameters-hyperparameters-and-optimization-strategies-3f0842fac0a5`);

[4] Baptista R., Marzouk Y., Willcox K., Peherstorfer B. (2018) - **Optimal Approximations of Coupling in Multidisciplinary Models**, in *AIAA Journal*, vol 56, n 6, pp 2412-2428;

[5] Baptista R., Poloczek M. (2018) - **Bayesian Optimization of Combinatorial Structures**, arXiv:1806.08838 [stat.ML];

[6] Baxter R.J. (1982) - **Exactly solved models in statistical mechanics**, Dover Publications;

[7] Bergstra J., Yamins D., Cox D.D. (2013) - **Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms**, in *Proceedings of the 12th Python in Science Conference*, pp 13-19;

[8] Bergstra J., Yamins D., Cox D.D. (2013) - **Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures**, in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, pp I–115–I–123;

[9] Binois M., Ginsbourger D., Roustant O. (2018) - **On the choice of the low-dimensional domain for global optimization via random embeddings**, arXiv:1704.05318 [math.OC];

[10] Brochu E., Cora V.M., de Freitas N. (2010) - **A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning**, arXiv:1012.2599 [cs.LG];

[11] Dablander F. (2019) - **Two properties of the Gaussian distribution** (`https://fabiandablander.com/statistics/Two-Properties.html`);

[12] Frazier P.I. (2018) - **A Tutorial on Bayesian Optimization**, arXiv:1807.02811 [stat.ML];

[13] Frazier P.I., Wang J. (2015) - **Bayesian Optimization for Materials Design**, in *Springer Series in Materials Science*, pp 45-75, Springer International Publishing;

[14] Gardner J., Kusner M., Xu E., Weinberger K., Cunningham J. (2014) - **Bayesian Optimization with Inequality Constraints**, in *31st International Conference on Machine Learning, ICML 2014*, vol 3;

[15] Garrido-Merchán E.C., Hernández-Lobato D. (2018) - **Dealing with categorical and integer-valued variables in Bayesian Optimization with Gaussian processes**, in *Neurocomputing*, vol 380, pp 20-35, Elsevier BV;

[16] Girshick R., Donahue J., Darrell T., Malik J. (2014) - **Rich feature hierarchies for accurate object detection and semantic segmentation**, arXiv:1311.2524 [cs.CV];

[17] Ginsbourger D., Le Riche R., Carraro L. (2008) - **A Multi-points Criterion for Deterministic Parallel Global Optimization based on Gaussian Processes**, hal-00260579;

[18] Huang D., Allen T.T., Notz W.I., Zeng N. (2006) - **Global Optimization of Stochastic Black-Box Systems via Sequential**

Kriging Meta-Models, in *Journal of Global Optimization*, vol 34, n 3, pp 441-466;

[19] Hutter F., Hoos H.H.,, Leyton-Brown K. (2011) - **Sequential Model-Based Optimization for General Algorithm Configuration**, in *Learning and Intelligent Optimization*, pp 507-532, Springer Berlin Heidelberg;

[20] Jones D.R., Schonlau M., Welch W.J. (1998) - **Efficient Global Optimization of Expensive Black-Box Functions**, in *Journal of Global Optimization*, vol 13, n 4, pp 455-492;

[21] Kochenberger G., Hao J., Glover F. et al. (2014) - **The unconstrained binary quadratic programming problem: a survey**, in *Journal of Combinatorial Optimization*, vol 28, pp 58-81, Springer;

[22] Krige D.G. (1951) - **A statistical approach to some basic mine valuation problems on the Witwatersrand**, in *Journal of the Southern African Institute of Mining and Metallurgy*, vol 52, n 6, pp 119-139, Southern African Institute of Mining and Metallurgy;

[23] Kushner H.J. (1964) - **A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise**, in *Journal of Basic Engineering*, vol 86, n 1, pp 97-106;

[24] Lau S. (2017) - **A Walkthrough of Convolutional Neural Network - Hyperparameter Tuning** (`https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network-7f474f91d7bd`);

[25] Lizotte D., Wang T., Bowling M. Schuurmans D. (2007) - **Automatic Gait Optimization with Gaussian Process Regression**, pp 944-949;

[26] Luong P., Gupta S., Nguyen D., Rana S., Venkatesh S. (2019) - **Bayesian Optimization with Discrete Variables**, in: Liu J., Bailey J. (eds) - *AI 2019: Advances in Artificial Intelligence*, part of *AI 2019. Lecture Notes in Computer Science*, vol 11919, Springer;

[27] McCoy B.M., Wu T.T. (2014) - **The Two-Dimensional Ising Model: second edition**, Dover Publications;

[28] Mockus J. (1974) - **On Bayesian Methods for Seeking the Extremum**, in: Marchuk G.I. (ed) - *Optimization Techniques*, pp 400-404, Springer;

[29] Negoescu D., Frazier P.I., Powell W.B. (2011) - **The Knowledge-Gradient Algorithm for Sequencing Experiments in Drug Discovery**, in *INFORMS Journal on Computing*, vol 23, n 3, pp 346-363;

[30] Oh C., Gavves E., Welling M. (2019) - **BOCK: Bayesian Optimization with Cylindrical Kernels**, arXiv:1806.01619 [stat.ML];

[31] Oh C., Tomczak J.M., Gavves E., Welling M. (2019) - **Combinatorial Bayesian Optimization using the Graph Cartesian Product**, arXiv:1902.00448 [stat.ML];

[32] Rasmussen C.E., Williams C.K.I. (2006) - **Gaussian Processes for Machine Learning**, in *Adaptive Computation and Machine Learning series*, The MIT Press;

[33] Shahriari B., Swersky K., Wang Z., Adams R.P., de Freitas N. (2016) - **Taking the Human Out of the Loop: A Review of Bayesian Optimization**, in *Proceedings of the IEEE*, vol 104, n 1, pp 148-175;

[34] Smith L.N. (2018) - **A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay**, arXiv:1803.09820 [cs.LG];

[35] Snoek J., Larochelle H., Adams R.P. (2012) - **Practical Bayesian Optimization of Machine Learning Algorithms**, arXiv:1206.2944 [stat.ML];

[36] Swersky K., Snoek J., Adams R.P. (2013) - **Multi-Task Bayesian Optimization**, in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, pp 2004-2012;

[37] Thengade A., Dondal R. (2012) - **Genetic Algorithm – Survey Paper**, in *IJCA Proc National Conference on Recent Trends in Computing, NCRTC*, vol 5;

[38] Toscano-Palmerin S., Frazier P.I. (2018) - **Bayesian Optimization with Expensive Integrands**, arXiv:1803.08661 [cs.LG];

[39] Wang Z., Hutter f., Zoghi M., Matheson D., de Freitas N. (2016) - **Bayesian Optimization in a Billion Dimensions via Random Embeddings**, arXiv:1301.1942 [stat.ML];

[40] Zhang F. (2005) - **The Schur Complement and Its Applications**, Springer;

[41] Zhang Y., Apley D.W., Chen W. (2020) - **Bayesian Optimization for Materials Design with Mixed Quantitative and Qualitative Variables**, in *Scientific Reports*, vol 10, n 1, pp 4924-4936;

[42] Zhang Y., Sohn K., Villegas R., Pan G., Lee H. (2015) - **Improving object detection with deep convolutional networks via Bayesian optimization and structured prediction**, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp 249-258.

Acknowledgements

Vorrei ringraziare i Professori Giacomo Como e Fabio Fagnani per aver reso possibile la realizzazione di questo elaborato. Ringrazio il team di ricerca che ha permesso la nascita dell'algoritmo SBO: in particolare Luca e Roberta e il gruppo TIM, nelle veci di Roberta ed Ennio.
Non posso che ringraziare i miei familiari, mia madre e le mie sorelle, che mi hanno sempre sostenuto, supportato con gesti, attenzioni, un paccodagiù in più, abbracci e poche parole, perché la mia famiglia si sa, è di poche parole.
Un grazie ai miei compagni di viaggio, Sofia, Silvia, Andrea, Laura, Irene, Francesca, Marilina, con cui ho condiviso molto più di un banco o un caffè da 25 cent.
Grazie a Serena e Fulvia, amiche, compagne di tisane tuttigusti+1 e alleate per cantate a squarciagola.
Grazie a Matteo, sempre presente, nonostante gli alti e bassi della vita.
Grazie ai miei coinquilini, fonte inesauribile di pasti scroccati e perle di saggezza (dopo il sesto bicchiere).
Grazie a Marco, più che amico fratello acquisito e come per i familiari, non c'è bisogno di parole per volerci bene.
Grazie ai miei amici di sempre, Ilaria, Stefano, Angela, Gaia, Betta, Olga, Fabiana, Sara, Serena, punto fermo, lì ad aspettarmi per l'ennesima rimpatriata fatta di pizza e Just Dance.
Infine, grazie a tutti quelli che anche per un breve istante mi hanno voluto bene, e volontariamente o involontariamente hanno arricchito la mia vita di inaspettati momenti di felicità.