POLITECNICO DI TORINO

Master degree course in Mathematical Engineering

Master Degree Thesis

Genetic Algorithms for Discrete Bayesian Optimization

Supervisors
Prof. Fabio Fagnani
Prof. Giacomo Como

Candidate
Roberta Raineri

A.A. 2019/2020

Contents

1 Introduction **1**

2 The Problem **3**

3 Background on Bayesian Optimization **6**
3.1 Continuous Bayesian Optimization 6
3.1.1 Gaussian Process Regression 7
3.1.2 Acquisition function 14
3.2 Discrete Bayesian Optimization 18
3.2.1 GP surrogate model 19
3.2.2 Alternative surrogate model 22

4 Background on Metaheuristics **27**
4.1 Introduction to Metaheuristics 28
4.1.1 Solution representation 28
4.1.2 Performance Analysis 30
4.2 Single-Solution Based Metaheuristics 32
4.2.1 Neighbourhood Definition 32
4.2.2 Local Search 35
4.3 Genetic Algorithm 37
4.3.1 Initial Population 39
4.3.2 Selection Methods 41
4.3.3 Reproduction 43
4.3.4 Replacement strategies 46

5 Proposed methods **47**
5.1 General Assumptions 48
5.2 LSBO Algorithm 49
5.3 GBO Algorithm 52

5.4 Application: Binary Quadratic Programming 56

6 Case Study: Mobile Networks **59**
6.1 General Mobile Network Structure 59
6.2 Case Study Problem Description 62
6.3 Results 63
6.3.1 Naive BO Algorithm 64
6.3.2 Other Existing Algorithms 66
6.3.3 LSBO Algorithm 67
6.3.4 GBO Algorithm 71
6.3.5 Algorithms Comparisons 75

7 Conclusions **78**

Chapter 1

Introduction

The growing complexity of business systems, combined with the huge amount of available data, has made it necessary to adopt a modelling and optimizing approach in order to solve high dimensional problems and automate decisions. The function to be optimized could be a measure of expended time, costs, profits, quality, but, in many cases, we may not know its analytical formulation and its evaluation could be expensive in terms of time and/or costs.
Bayesian Optimization(BO) is the main optimization strategy used to solve this class of problems characterized by an expensive black-box function. Besides, it has been shown to outperform other state of the art global optimization algorithms on a number of challenging benchmark functions [20]. It has been used in a wide range of different problems (e.g adaptively choosing the sequence of molecular compounds to test in drug discovery [17], tuning model hyperparameters in Deep Neural Networks, optimizing a robot's gait by the regulation of its control parameters [16]).
With the wide spread of Machine Learning and discrete events simulations during the last decade, researchers' interest in this field, firstly developed by Kushner, Mockus and Zilinskas in the 1970s, has increased.
Frazier [8], Shahriari [19] and Rasmussen & Williams' [18] works are taken in our dissertation as main literature references.

Bayesian Optimization algorithm is based on two main components, a Gaussian Process, used as surrogate model for the objective function, and an acquisition function that guides the decision about the next point to be evaluated.
The major constraint of this strategy is that the problem variables are typically supposed to be continuous.

Common approaches to overcome this limit are focused on the manipulation of BO main components. Bergstra [4], Garrido-Merchan and Hernandez-Lobato [9] base their studies on the surrogate model structure, whereas Luong [15] focuses on a proper acquisition function choice. On the other hand, Baptista [3], whose work is found on combinatorial structures, introduces a new approach which merges BO with Simulated Annealing strategy.

In this work we propose two alternative approaches, the **LSBO** (Local Search Bayesian Optimization) and the **GBO** (Genetic Bayesian Optimization) methods, both of which stand out for the introduction of Metaheuristics strategies in the manipulation of search space exploration of standard Bayesian Optimization.
The LSBO introduces the Local Search approach in the generation of the neighbourhood of the current solution and so in the definition of the subset of points on which the acquisition function is evaluated to detect the next candidate solution. The GBO uses, instead, a Genetic Algorithm approach to guarantee a wide and efficient space exploration. We don't focus in this case only on the current solution, but on a population of parents selected through a quasi-elitist strategy. The neighbours, offspring of the selected population, are generated by applying proper mutation and recombination operators on them. The aim of this method, compared to the former, is not to lose the information provided by the correlation among variables and thus exploit it in the reproduction phase.
The goodness of these proposed methods is supported by the results of the thesis case study. The problem analysed is the upshot of a collaboration between the Department of Mathematical Sciences of Politecnico di Torino and the TIM group and concerns a coverage and capacity optimization problem designed on a given high dimensional Mobile Network. The performance measures, related to solution quality and computational time, highlight how both LSBO and GBO methods outperform the traditional approach, chosen as a benchmark.

The dissertation is organized as follows. We start with a technical introduction to the general optimization problem in Chapter 2. In Chapter 3 we recall the theory behind Bayesian Optimization and we explain the main existing methods to solve both continuous and discrete variables problems. Then, in Chapter 4 we summarize the main concepts concerning Metaheuristics theory and Genetic Algorithm. Chapter 5 is the core of our thesis where the two proposed methods are illustrated from a technical point of view and an application on Binary Quadratic Programming is provided. In Chapter 6 we describe and fully analyse the thesis case study. In the final Chapter 7 we summarize and comment the results obtained and pave the way to future improvements.

Chapter 2

The Problem

Optimization methods represent a powerful instrument widely used in science, industry and economics applications where each process has a certain potential to be optimized. An **optimization problem** is characterized by few main components:

- the **variables** $x = \{x_i,\ i = 1, \ldots, d\}$ of the problem. They are the input parameters and represent the elements on which we take decisions. We highlight that the complexity of the optimization problem grows exponentially with the number of variables d.
- the **set of feasible solutions** $\mathcal{X}$, also referred to as the set of configurations or states. It represents all the possible values that the variable of interest x can assume. It could be a continuous space $\mathcal{X} \subset \mathbb{R}^d$ with d number of problem variables, or a discrete subset of points.
- the **objective function** $f : \mathcal{X} \longrightarrow \mathbb{R}$. It represents the function that we want to optimize (maximize/minimize). It assigns to each $x \in \mathcal{X}$ a real number indicating its value, defining so a total order relation between any pair of solutions in the configurations space. The objective function could be for example a measure of expended time, costs, profits or quality. The mathematical model will obviously be just an approximation of the real-life problem, so use a model that well fits reality is essential to determine interesting results.

An optimization problem is usually referred to as the couple $(\mathcal{X}, f)$. The main goal when we solve an optimization problem is to find a global optimal solution x^* in the configuration space $\mathcal{X}$.

Definition 2.0.1 (**Global optimum**). *A solution $x^* \in \mathcal{X}$ is a **global optimum** for f if $f(x^*)$ is the best value among all possible solutions in the configuration space $\mathcal{X}$, that is, assuming a maximization problem,*

$$f(x^*) \geq f(x), \quad \forall x \in \mathcal{X}$$

Definition 2.0.2 (**Local optimum**). *A solution $x' \in \mathcal{X}$ is a **local optimum** of f if there is a proper subset [1] $N \subset \mathcal{X}$ including x' such that, supposing to work with a maximization problem,*

$$f(x') \geq f(x), \quad \forall x \in N$$

but

$$f(x') \leq f(x^*)$$

with x^* global optimum of the optimization problem.

The simplest way to solve an optimization problem is to use **Exact Methods**. Despite that it is not always possible or appropriate to apply those methods due to inner complexity of the problem or to a limited available time to provide the solution. In many cases obtaining an accurate enough formulation of the problem might be too difficult or expensive and so approximate algorithms are adopted.
Many solutions have been proposed to face the problem of too large solution space. **Metaheuritics** (described in chapter 4) are broadly adopted to reduce the effective size of the space and to explore it efficiently without being stuck prematurely in some local optima.
However, the objective function could be expensive to evaluate in terms of time and/or costs and so the number of evaluations is severely limited, or furthermore it could be a black-box function.
A function $f : \mathcal{X} \longrightarrow \mathbb{R}$ is called a **black-box function** if it happens that:

- the domain $\mathcal{X}$ is known;
- it is possible to know the value of f in each point of $\mathcal{X}$ carrying on a simulation;
- no other information is available for function f.

[1] The subset N is properly known as **neighbourhood** of x'. This will be widely explained in section 4.2.1, both for continuous and discrete variables.

In all these cases the **Bayesian Optimization** is broadly adopted. As we will better see later in chapter 3, those methods are characterized by two main components: a *surrogate model* (generally supposed to be a Gaussian Process), which approximate the objective function of interest, and an *acquisition function*, built from the surrogate model, used to decide the next point to evaluate. Most of the BO methods assume the input variables x to be continuous as a consequence of the continuous domain of the defined acquisition function. However, in real-world application we often face problems described by **discrete variables**. When we talk about discrete variables we consider $x \in \mathcal{X}$ with $\mathcal{X}$ discrete subset of points, where the variable x could be both *categorical* or *integer-valued*.
This is a challenging problem because when BO samples the next point to evaluate, it suggests a continuous point that however would be an invalid input for function f. Furthermore we have exponentially many combinations of discrete values with respect to the number of variables and so the search space becomes soon too large and it is impractical to try all possible values.
In the following work we will focus on this main problem, analyse the existing methods and propose two alternative solution approaches built by the combination of Bayesian Optimization and Metaheuristics strategies.

Chapter 3

Background on Bayesian Optimization

3.1 Continuous Bayesian Optimization

Standard Bayesian Optimization (BO) is an efficient machine-learning-based optimization method used to find a global optimizer for the problem

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

when $f(x)$ is an expensive black-box function.
In Bayesian Optimization we usually consider the following underlying assumptions for the problem in (3.1) :

- The input vector $x \in \mathbb{R}^d$, with typically $d \leq 20$.
- The feasible set $\mathcal{X}$ is a simple set, typically a compact subset of $\mathbb{R}^d$ like a hyper-rectangle $\{x \in \mathbb{R}^d : a_i \leq x_i \leq b_i,\ a_i, b_i \in \mathbb{R}\ \forall i = 0, \ldots, d-1\}$ or a d-dimensional simplex.
- The objective function $f : \mathcal{X} = \mathbb{R}^d \longrightarrow \mathbb{R}$ is continuous but it has not got a closed-form to compute derivative information nor a known special structure like concavity or linearity.
- In addition f is expensive to evaluate. This means that only a little number of evaluations may be performed due to amount of time needed or monetary/opportunity cost.

- The objective function $f(x)$ could be considered with/without noise. In case of noisy function we introduce a stochastic noise $\epsilon \sim \mathcal{N}(0, \lambda^2)$. In this case we would not consider $f(x_i)$ but the noisy value $y_i = f(x_i) + \epsilon$.

Our goal is finding the global optimum in a minimum number of steps. To do this BO incorporates prior belief about the objective function f and then updates this prior with samples drawn from f to get a posterior that better approximates it. The Bayesian model used for approximating the objective function is called *surrogate model*. An *acquisition function* is used for deciding where to sample next. This is a heuristic that directs sampling to areas where an improvement over the current best observation is likely to trade off exploration and exploitation; the optimum is located where the uncertainty in the surrogate model is large (exploration) and/or where the model prediction is high (exploitation). The next point to evaluate is selected maximizing the acquisition function.
The algorithm could be summarized as follows ([19]):

Algorithm 1: Bayesian Optimization

Initialize $\mathcal{D}_1$ as the set of all training evaluated pairs (x, y)
for *n = 1,2, ...* **do**
  $x_{n+1} = argmax_x \alpha(x; \mathcal{D}_n)$
  Query objective function to obtain y_{n+1}
  Augment data $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (x_{n+1}, y_{n+1})\}$
  Update surrogate model
end

An example of the general procedure is illustrated in [Fig. 3.1].

3.1.1 Gaussian Process Regression

The Gaussian Process (GP) is the most widely adopted surrogate model.

Definition 3.1.1. *A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

In other words given a GP of the form $f : \mathcal{X} = \mathbb{R}^d \longrightarrow \mathbb{R}$ any finite set of k points $\{x_i \in \mathbb{R}^d\}_{i=1}^k$ induces a multivariate Gaussian distribution on $\mathbb{R}^k$. The GP is a non-parametric model completely specified by the mean function $\mu_0(x) : \mathcal{X} = \mathbb{R}^d \longrightarrow \mathbb{R}$ and the covariance function or kernel $k(x, x') : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ defined as follows:

$$\mu_0(x) = \mathbb{E}\big[f(x)\big]$$

$$k(x, x') = \mathbb{E}\big[(f(x) - \mu_0(x))(f(x') - \mu_0(x'))\big]$$

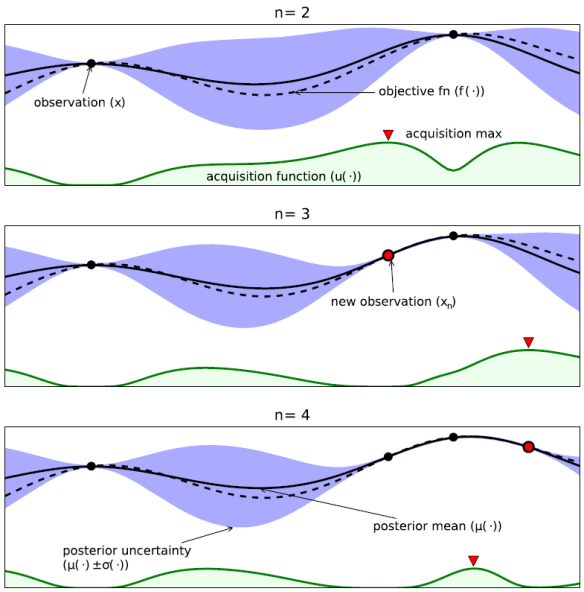


Figure 3.1: Example of the Bayesian Optimization procedure over three iterations. Mean and confidence intervals are estimated through a probabilistic model of the objective function. The next point to evaluate is chosen maximizing the acquisition function. [19]

So we rewrite the Gaussian Process as:

$$f(x) \sim \mathcal{GP}(\mu_0(x), k(x, x'))$$

The prior mean function μ_0 could potentially incorporate information from the objective function, but in the general case it is assumed constant for convenience. The covariance function (also denoted as kernel) k instead has to be chosen properly to well fit the structure of the response function.

Prediction with noise-free observations

If we start our evaluation from a finite collection of points $\{x_i \in \mathbb{R}^d\}_{i=1}^n$ we compute the mean vector evaluating the mean function μ_0 at each x_i and we construct the covariance matrix by evaluating the kernel k at each pair x_i, x_j. The resulting prior distribution is

$$f(x_{1:n}) \sim \mathcal{N}(\mu_0(x_{1:n}), K(x_{1:n}, x_{1:n}))$$

where $f(x_{1:n}) = [f(x_1), \dots, f(x_n)]$, $\mu_0(x_{1:n}) = [\mu_0(x_1), \dots, \mu_0(x_n)]$ and $K(x_{1:n}, x_{1:n}) = [k(x_1, x_1), \dots, k(x_1, x_n); \dots; k(x_n, x_1), \dots, k(x_n, x_n)]$.

In our problem we are interested in incorporating the knowledge provided by observed data in order to update the surrogate function to get a posterior that better approximates the objective function f. Given that f is an expensive-to-evaluate function we suppose we will not be able to compute its value at a new point x and we want so to infer it.
We know that the joint distribution of the function f at $x_{1:n}$ and at the new point $\tilde{x}$ according to the prior is:

$$\begin{bmatrix} f(x_{1:n}) \\ f(\tilde{x}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_0(x_{1:n}) \\ \mu_0(\tilde{x}) \end{bmatrix}, \begin{bmatrix} K(x_{1:n}, x_{1:n}) & K(x_{1:n}, \tilde{x}) \\ K(\tilde{x}, x_{1:n}) & k(\tilde{x}, \tilde{x}) \end{bmatrix} \right) \tag{3.2}$$

Conditioning the joint Gaussian prior distribution on the given observations, we compute the posterior of $f(\tilde{x})$ given $f(x_{1:n})$:

$$f(\tilde{x})|f(x_{1:n}) \sim \mathcal{N}(\mu_n(\tilde{x}), \sigma_n^2(\tilde{x})) \tag{3.3}$$

with

$$\mu_n(\tilde{x}) = K(\tilde{x}, x_{1:n}) K(x_{1:n}, x_{1:n})^{-1} (f(x_{1:n}) - \mu_0(x_{1:n})) + \mu_0(\tilde{x})$$

$$\sigma_n^2(\tilde{x}) = k(\tilde{x}, \tilde{x}) - K(\tilde{x}, x_{1:n}) K(x_{1:n}, x_{1:n})^{-1} K(x_{1:n}, \tilde{x})$$

Prediction with noisy observations
We could generalize the previous explanation considering more realistic situations characterized by noisy observations $y = f(x) + \epsilon$. Assuming additive independent and identically distributed Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$, the prior covariance function $cov(\cdot, \cdot)$ on the observations becomes:

$$cov(y_i, y_j) = k(x_i, x_j) + \sigma_\epsilon^2 \delta_{ij}$$

where δ_{ij} is the Kronecker delta. Or equivalently using vector notation

$$cov(\mathbf{y}) = K(X, X) + \sigma_\epsilon^2 I$$

where $\mathbf{y}$ represents the vector $(y_1, \dots, y_n)$, X is the matrix $x_{1:n}$ where x_i is a d dimensional vector and I is the identity matrix.
In other words a diagonal matrix is added to the previous noise-free matrix.

The joint distribution computed in (3.2) takes the form:

$$\begin{bmatrix} y \\ f(\tilde{x}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_0(x_{1:n}) \\ \mu_0(\tilde{x}) \end{bmatrix}, \begin{bmatrix} K(x_{1:n}, x_{1:n}) + \sigma_\epsilon^2 I & K(x_{1:n}, \tilde{x}) \\ K(\tilde{x}, x_{1:n}) & k(\tilde{x}, \tilde{x}) \end{bmatrix} \right) \tag{3.4}$$

Deriving from this one the conditional distribution as in (3.3) we have:

$$f(\tilde{x})|y \sim \mathcal{N}\left(\overline{f(\tilde{x})}, Var(f(\tilde{x})) \right) \tag{3.5}$$

with

$$\overline{f(\tilde{x})} \triangleq \mathbb{E}\big[f(\tilde{x})|y\big] = K(\tilde{x}, x_{1:n})(K(x_{1:n}, x_{1:n}) + \sigma_\epsilon^2 I)^{-1}(y - \mu_0(x_{1:n})) + \mu_0(\tilde{x})$$

$$Var(f(\tilde{x})) = k(\tilde{x}, \tilde{x}) - K(\tilde{x}, x_{1:n})(K(x_{1:n}, x_{1:n}) + \sigma_\epsilon^2 I)^{-1} K(x_{1:n}, \tilde{x})$$

Suppose now prior means $\mu_0 = 0$. The previous expressions could be rewritten as follows:

$$\overline{f(\tilde{x})} = K(\tilde{x}, x_{1:n})(K(x_{1:n}, x_{1:n}) + \sigma_\epsilon^2 I)^{-1} y \tag{3.6}$$

$$Var(f(\tilde{x})) = k(\tilde{x}, \tilde{x}) - K(\tilde{x}, x_{1:n})(K(x_{1:n}, x_{1:n}) + \sigma_\epsilon^2 I)^{-1} K(x_{1:n}, \tilde{x}) \tag{3.7}$$

If we focus on the mean predictor in (3.6) we notice that it is a linear combination of observations y. Another way to see this equation is as a linear combination of n kernel functions, each one centred on an already observed point, by writing:

$$\overline{f(\tilde{x})} = \sum_{i=1}^{n} \xi_i k(x_i, \tilde{x})$$

where $\xi = (K + \sigma_\epsilon^2 I)^{-1} y$.
We note also that the variance in (3.7) does not depend on the observed targets, but only on the inputs, this is a property of the Gaussian distribution. The variance is the difference between two terms: $k(\tilde{x}, \tilde{x})$ that is the prior covariance and a positive term which collects the information the observations gives us about the function.

GPR Algorithm
An implementation of the GP regression is shown in the following algorithm.

Algorithm 2: Gaussian Process Regression

input: X(inputs), y(targets), k (kernel), σ_ϵ^2(noise level), $\tilde{x}$(test input)

Compute Cholesky decomposition $L := cholesky(K + \sigma_\epsilon^2 I)$

Compute ξ from $L^T L\xi = y$

Define $\overline{f(\tilde{x})} := K(\tilde{x}, x_{1:n})\xi$

Compute v from $Lv = K(\tilde{x}, x_{1:n})$

Define $Var(f(\tilde{x})) := k(\tilde{x}, \tilde{x}) - v^T v$

$\log p(y|X) := -\frac{1}{2} y^T \xi - \sum_i \log L_{ii} - \frac{n}{2} \log 2\pi$

return: $\overline{f(\tilde{x})}$ (mean), $Var(f(\tilde{x}))$ (variance), $\log p(y|X)$

We underline that the algorithm uses Cholesky decomposition instead of directly inverting the matrix since it is faster and numerically more stable. The algorithm, built for noise free observed data, returns mean and variance. To compute the prediction distribution for noisy observations we have to add the noise variance σ_ϵ^2 to the predictive variance of $f(\tilde{x})$.
The computational complexity is $n^3/6$ for the Cholesky decomposition and $n^2/2$ for solving the two following triangular systems (for each test case), where n is the number of observations. The Cholesky decomposition must be recomputed every time we change the kernel hyperparameters, i.e. at every iteration of our algorithm.

The covariance function or kernel

The choice of the proper kernel is important because it encodes all the assumptions about the function we want to learn and furthermore it determines the smoothness properties of samples drawn from the defined Gaussian Process. We can choose among *periodic* and *stationary* kernels according to the response function expected. A stationary covariance function is a function of $x - x'$ and so it is invariant to translations in the input space. In addition if the kernel is a function of $x - x'$ only then it is called *isotropic* and it is invariant to all rigid motions. Finally a *dot product* covariance function depends only on $x \cdot x'$ and it is invariant to a rotation of the coordinates about the origin but not to translations.
Generally speaking we call *kernel* a function k of two arguments mapping a pair of inputs $x, x' \in \mathcal{X}$ into $\mathbb{R}$. The term kernel comes from theory of integral operators where the operator T_k is defined as:

$$(T_k f)(x) = \int_{\mathcal{X}} k(x, x') f(x') d\mu(x')$$

where μ denotes a measure.
A kernel to be a covariance function must satisfy the following conditions:

- It must be symmetric $k(x, x') = k(x', x)$;
- Given a set of input points $\{x_i \in \mathcal{X} | i = 1, \dots, n\}$, the matrix K, with entries $K_{ij} = k(x_i, x_j)$, also known as Gram matrix, must be positive semidefinite.

We recall that a $n \times n$ matrix K is positive semidefinite if $v^T K v \geq 0$ for all vectors $v \in R^n$. A symmetric matrix is positive semidefinite (PSD) if and only if all of its eigenvalues are non-negative.
A kernel is said to be positive semidefinite if

$$\int k(x, x') f(x) f(x') d\mu(x) d\mu(x') \geq 0 \tag{3.8}$$

for all $f \in \mathcal{L}_2(\mathcal{X}, \mu)$. Equivalently a kernel function which gives rise to PSD Gram matrices for any choice of $n \in \mathbb{N}$ and $\mathcal{D}$, where $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ is the set of already observed point, is positive semidefinite.
To easily prove that the kernel function of interest satisfies PSD requirement we can take as f the weighted sum of delta functions at each x_i and, since these functions are limits of functions in $\mathcal{L}_2(\mathcal{X}, \mu)$, then (3.8) implies that the Gram matrix corresponding to any $\mathcal{D}$ is PSD.
As previously anticipated the choice of the kernel function affects the characteristics of sampled function f. To better understand this relation now we focus on mean square continuity and differentiability of stochastic processes.
Let $x_1, x_2, \dots$ be a sequence of points and x_* be a fixed point in $\mathbb{R}^d$ such that $|x_k - x_*| \longrightarrow 0$ as $k \longrightarrow \infty$. Then a process $f(x)$ is **continuous in mean square** at x_* if

$$\mathbb{E}\big[|f(x_k) - f(x_*)|^2\big] \longrightarrow 0 \quad \text{as } k \longrightarrow \infty$$

If this holds for all $x_* \in \mathcal{A}$ with $\mathcal{A} \subset \mathbb{R}^d$ then $f(x)$ is said to be continuous in mean square over $\mathcal{A}$. A random field is continuous in mean square at x_* if and only if its covariance function $k(x, x')$ is continuous at the point $x = x' = x_*$. For stationary covariance functions this reduces to checking continuity at $k(0)$, but we underline that mean square continuity doesn't necessarily imply sample function continuity.

The *mean square derivative* of $f(x)$ in the i-th direction is defined as:

$$\frac{\partial f(x)}{\partial x_i} = \lim_{h \to 0} \frac{f(x + he_i) - f(x)}{h}$$

when the limit exists, where e_i denotes the unit vector in the i-th direction. The covariance function of $\frac{\partial f(x)}{\partial x_i}$ is given by $\frac{\partial^2 k(x,x')}{\partial x_i \partial x_i'}$. For stationary processes if the $2k$-th order partial derivative $\frac{\partial^{2k} k(x)}{\partial^2 x_{i_1} \dots \partial^2 x_{i_k}}$ exists and is finite at $x = 0$ then the k-th order partial derivative $\frac{\partial^k f(x)}{\partial x_{i_1} \dots \partial x_{i_k}}$ exists for all $x \in \mathbb{R}^d$ as a mean square limit. So we deduce that the properties of kernel k around 0 determine the smoothness properties (**mean square differentiability**) of a stationary process.

Examples of covariance functions

A first example of covariance function is given by the **squared exponential** kernel.

$$k(x, x') = exp\left(-\frac{\|x - x'\|_2^2}{2l^2}\right)$$

where $\|\cdot\|_2^2$ is the Euclidean distance and l is a hyperparameter known as characteristic length scale. This one has the role of rescaling any point x by $1/l$ before computing the kernel value. A short length scale makes function values strongly correlated only if their respective inputs are very close to each other. In particular the covariance is almost one between variables whose corresponding inputs are very close and decreases as their distance in the input space increases. This kernel is infinitely differentiable and this implies that the GP is very smooth. This covariance function is isotropic, it depends only on $\|x - x'\|$ and for this reason it is also known as **radial basis function (RBF)**.

Another important example of kernel, widely used for GPs, is given by the **Matérn Kernel**, a stationary covariance function which incorporates a smoothness parameter ν which grants greater flexibility in modelling functions.

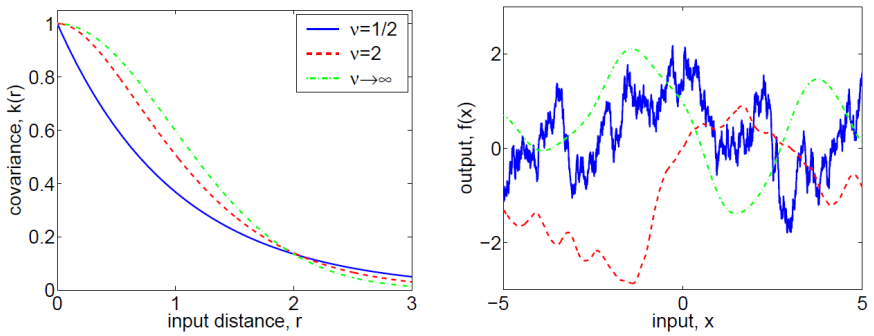
$$k(x, x') = \frac{1}{2^{\nu-1}\Gamma(\nu)} \left(\frac{2\sqrt{\nu}\|x - x'\|_2^2}{l}\right)^{\nu} H_\nu \left(\frac{2\sqrt{\nu}\|x - x'\|_2^2}{l}\right)$$

where ν and l are two hyperparameters and $\Gamma(\cdot)$ and $H(\cdot)$ are respectively the Gamma function and the Bessel function of order ν. Also this expression depends on $\|x - x'\|$ and so it is a Radial Basis Function. As before, l is the length-scale

parameter and it affects the smoothness of the GP. Instead ν is a hyperparameter related to the number of times that the GP samples can be differentiated. When $\nu \longrightarrow \infty$ the Matérn kernel shrinks to the squared exponential kernel and when $\nu = 0.5$ it is reduced to the unsquared exponential kernel. Generalizing this covariance function becomes very simple when ν is half-integer: $\nu = p + 1/2$ where $p \in \mathbb{N}$ is a non-negative integer. In this case the function is reduced to a product of an exponential and a polynomial of order p. The most widely adopted values in machine learning community are $\nu = 3/2$ and $5/2$:

$$k_{\nu=3/2}(x, x') = \left(1 + \frac{\|x - x'\|\sqrt{3}}{l}\right) e^{-\frac{\|x-x'\|\sqrt{3}}{l}}$$

$$k_{\nu=5/2}(x, x') = \left(1 + \frac{\|x - x'\|\sqrt{5}}{l} + \frac{(x - x')^2}{3l^2}\right) e^{-\frac{\|x-x'\|\sqrt{5}}{l}}$$



(a) (b)

Figure 3.2: In (a) the covariance functions for different parameter of Matérn kernel, in (b) random functions drawn from GPs with this kernel choice. [18]

3.1.2 Acquisition function

The acquisition function α is a heuristic used to decide where to sample next. In our choice we would like to encourage exploration of never seen areas and exploitation of areas near the actual optimum.

Generally speaking we say that a point is **repeated** (i.e. it has already been observed) if it holds

$$x_{n+1} = [\underset{x \in \mathcal{X}}{\operatorname{argmax}}\, \alpha_n(x)] \in \mathcal{D}_n$$

where $\alpha_n(x)$ is the chosen acquisition function and $\mathcal{D}_n$ is the set of all observations (x, y) up to iteration n. Given that query the function f on a new point x_{n+1} is expensive, in the selection of α we pay attention to avoid repetitions.
Then, the acquisition function will be characterized by:

- high value for points expected to be optimal;
- high value for points we have not explored;
- low values for repeated points.

We will describe now some of the most commonly used acquisition functions. In Figure 3.3 are displayed some examples for different values of the hyperparameters.

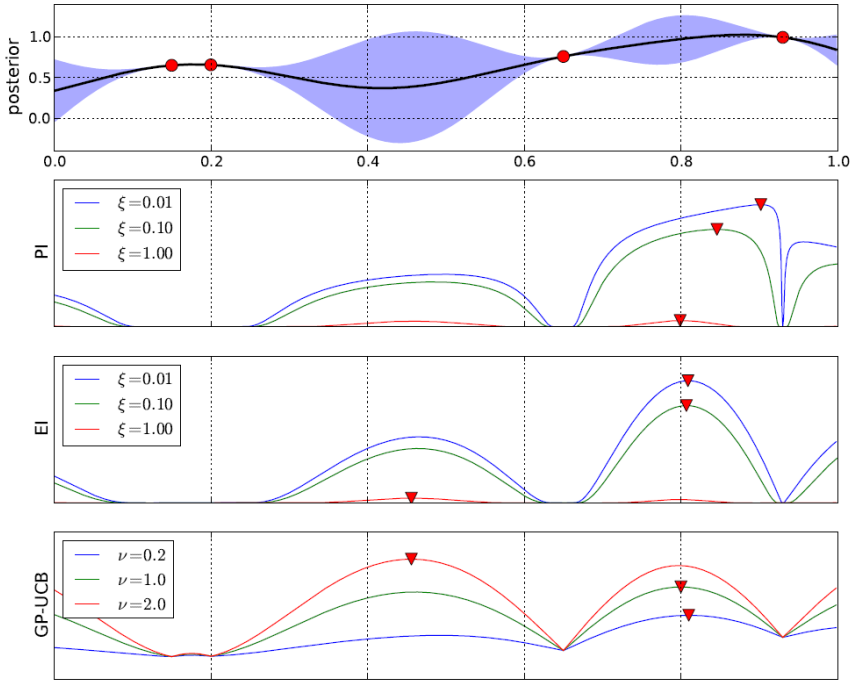


Figure 3.3: Examples of the acquisition functions described (PI, EI, UCB) for different values of the parameter. The triangle marker indicates the maximum of each function. [6]

Probability of improvement (PI)

The Probability of improvement acquisition function focuses on maximizing the probability of improvement over $f(x^*)$ where x^* is the current optimum value $x^* = \underset{x_i \in x_{1:k}}{argmax} f(x_i)$.

$$PI(x) = \mathbb{P}\big[f(x) \geq f(x^*)\big] = \Phi\left(\frac{\mu(x) - f(x^*) - \xi}{\sigma(x)}\right)$$

The choice of ξ is left to the user: usually it starts fairly high at the beginning to drive exploration and decreases towards zero as the algorithm continues.
The main drawback of this acquisition function is that it is a pure exploitation method and it doesn't take into account the potential magnitude of the improvement.

Expected Improvement (EI)

The Expected Improvement acquisition function takes into account both the probability and the magnitude of the improvement the point can potentially yield. The improvement I can be defined as follow:

$$I(x) = \max\{0, f_{n+1}(x) - f(x^*)\}$$

So $I(x)$ is positive if the prediction makes better than the current best value and it is zero otherwise. The new point x_{n+1} to query is found maximizing the expected improvement:

$$x_{n+1} = \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, \mathbb{E}\big[\max\{0, f_{n+1}(x) - f(x^*)\} | \mathcal{D}_n\big]$$

This can be evaluated analitically as:

$$EI(x) = \begin{cases} (\mu(x) - f(x^*))\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases}$$

where $Z = \frac{\mu(x) - f(x^*)}{\sigma(x)}$ and $\phi(\cdot)$ and $\Phi(\cdot)$ denote respectively the PDF and the CDF of the standard normal distribution.
In order to generalize the function to control the trade-off between exploitation (local search) and exploration (global search), a new formulation has been

introduced:

$$EI(x) = \begin{cases} (\mu(x) - f(x^*) - \xi)\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases}$$

where $Z = \frac{\mu(x) - f(x^*) - \xi}{\sigma(x)}$.
The parameter ξ determines the amount of exploration during optimization. Higher ξ values lead to more exploration: if we increase ξ we reduce the weight of the first addend of EI expression which is associated to the improvements predicted by the posterior mean.
The parameter ξ is very similar to the one introduced in PI acquisition function but as a whole EI is a valid alternative to PI as it exceeded the limits of this one.

Upper Confidence Bound (UCB)

The Upper Confidence Bound acquisition function takes the form

$$UCB(x; \beta) = \mu(x) + \sqrt{\beta}\sigma(x)$$

where $\sigma(x) = \sqrt{\Sigma(x, y)}$ is the marginal standard deviation of $f(x)$.
It is based on the principle of being optimistic in the face of uncertainty: for every query x it corresponds to effectively use a fixed probability best case scenario. The parameter β regulates trade-off between exploration and exploitation in the algorithm.
The next point to explore is so chosen as

$$x_{n+1} = \underset{x \in \mathcal{X}}{argmax}\ \mu_n(x) + \sqrt{\beta_{n+1}}\sigma_n(x) \tag{3.9}$$

A natural interpretation of this strategy is that it greedily selects points x in such a way that $f(x)$ should be a reasonable upper bound on $f(x^*)$ since the argument in (3.9) is an upper quantile of the marginal posterior $\mathbb{P}\big[f(x)|y_n\big]$. Another intuition about this sampling rule is given in [Fig.3.4]. Since the upper and lower confidence bounds correspond to percentile points for f, the function values are suboptimal with high probability at points where the UCB is smaller than the highest lower confidence bound. The UCB sampling rule implicitly cuts those regions out of the decision set.

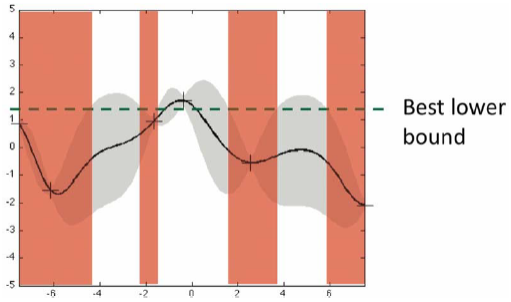


Figure 3.4: The UCB acquisition function implicitly rules out regions of the decision set where the upper confidence bound is less than the maximum lower confidence bound. [21]

Looking at (3.9) we highlight that UCB acquisition function is a combination of $\mu_n(x)$ and $\sigma_n(x)$; so, its maximum value is determined by one of these scenarios:

- $\mu_n(x)$ *dominates* $UCB(x;\beta)$. In this case the maximizer is completely determined by μ_n while σ_n has no effects on the solution. Increasing β_n the influence of σ_n on the solution rises and as a consequence it introduces some extra exploration in the optimization.

- $\sigma_n(x)$ *dominates* $UCB(x;\beta)$. In this case μ_n has no effects on the solution. Repetitions of already observed points are not possible because σ_t is small near existing observations.

- $\mu_n(x)$ *and* $\sigma_n(x)$ *are balanced*. Adjusting β_n can make σ_n dominant and so it consequently stops repetitions. In this case it is also possible to adjust the parameter l of the GP kernel which allows us not to use too high values for β_n. Changing the length scale l may cause slight misspecification of the GP prior.

Ideally the hyperparameter β_n should be chosen dynamically to decrease monotonically with the function evaluation: this avoid sampling already observed points.

3.2 Discrete Bayesian Optimization

In the previous section we have assumed the input variables x to be continuous, i.e. $x \in \mathcal{X} \subset \mathbb{R}^d$, and have described the general structure of the standard Bayesian Optimization method. Despite that, in real-world application it is not

always the case, we could face with variables which take categorical or integer values and so extra approximation became necessary. The configuration space $\mathcal{X}$ can be more generally defined in those cases as a discrete subset of points with a particular structure inferred. In the following work we will focus in particular on integer-valued variables and will analyse some of the solutions currently developed for this kind of problems.

3.2.1 GP surrogate model

Naive Approach

We have said before that the main problem is that $f(\cdot)$ cannot be evaluated at all potential input locations, but only at those ones which are compatible with the integer-valued variables. A naive approach to account for this is to optimize the acquisition function $\alpha(\cdot)$ assuming all variables taking values on the real line and then replace them by the closest integer before evaluation. This method, known as Naive BO, might not perform well because it can produce situations in which the BO method always evaluates the objective at a repeated point. We can find an example of this in [Fig. 3.5]. The function represented in the picture admits only values on the grid $x \in [-2, 10]$. The black dots represent the already evaluated points. Maximizing the acquisition function $\alpha(\cdot)$ suggests a continuous point (represented by the red star) $x = 0.3$. Since it is a continuous point, it doesn't belong to the grid and therefore it must be approximated to its closest neighbour $x = 0$ which has already been evaluated.

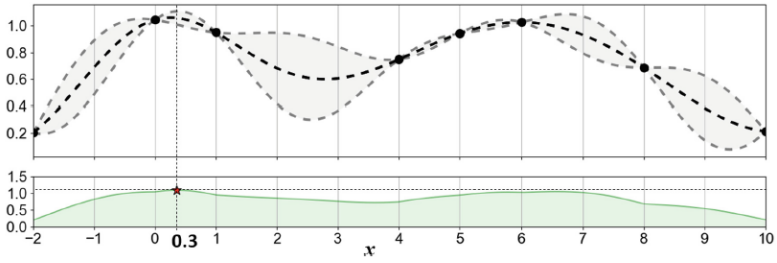


Figure 3.5: **Naive BO.** Illustration of the repetition of already evaluated points in Naive BO approach. [15]

The problem associated to this approach is that the actual objective is constant in the intervals that are rounded to the same integer value but it is ignored by the GP model and so generally it leads to sub-optimal results.

Transformation approach

The Transformation method, developed by Garrido-Merchan and Hernandez-Lobat [15], tries to solve the previously described problem. It considers the objective to be constant in those regions of the input space that lead to the same input variable configuration in which the objective has to be evaluated. This property is introduced into the GP modifying the kernel $k(\cdot, \cdot)$. In section 3.1.1 we have seen RBF kernels which only depends on the distance between the input points: if the distance between two points is zero, then correlation between them is equal to one because the function values at both points will be the same. On the bases of this fact we can build an alternative covariance function

$$k'(x_i, x_j) = k(T(x_i), T(x_j))$$

where $T(x)$ is a transformation in which the input variables corresponding to an integer-value input variable are rounded to the closest integer value.

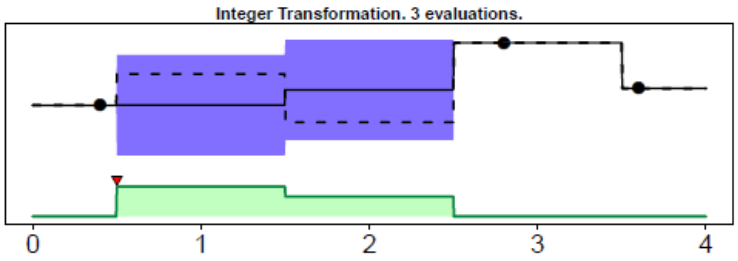


Figure 3.6: **Transformation approach.** The transformation applied to the covariance function makes it a step-wise function. The GP identifies a step-wise objective function. [9]

In this way, the GP correctly identifies that the objective function is constant inside intervals of real values that are rounded to the same integer and that the uncertainty is the same on those intervals as reflected by the new covariance function defined.
The problem of this method is that it makes the acquisition function a step-wise function which is difficult to optimize.

Discrete-BO

The Discrete-BO Algorithm, proposed by Luong et al. [15], wants to solve the repetition problem that occurs in the Naive BO method through the definition of a new version of the UCB acquisition function (vd. Section 3.1.2). From general

theory we know that an accurate choice of the parameters β and l is fundamental: a large value of β causes more exploration, making the algorithm less efficient, while the length scale l controls the smoothness of the surrogate model. At the same time we know that a large misspecification leads to a requirement of more samples to get accurate function estimation and this is not a negligible drawback in case of expensive to evaluate functions. Choosing parameters β and l through random search or grid search would be computationally expensive, thus Luong proposes a systematic approach. Its method finds the new value of β and l by solving the following optimization problem:

$$\begin{aligned} \beta^*, l^* &= \underset{\Delta\beta \in [0,\beta_h],\ l \in (0,lh]}{\operatorname{argmin}} g(\beta_t + \Delta\beta, l) \\ g(\beta_t + \Delta\beta, l) &= \Delta\beta + \|x_{t+1} - x'_{t+1}\|_2 + P(x'_{t+1}) \end{aligned} \tag{3.10}$$

where $\Delta\beta$ is the increment applied on β_t as $\beta_t \leftarrow \beta_t + \Delta\beta$; x_{t+1} is the new point suggested by the original β_t and l_t, while x'_{t+1} is the one suggested by $\beta_t + \Delta\beta$ and adjusted l. The term $P(x'_{t+1})$ is set to a constant C if $round(x'_t + 1) \in \mathcal{D}_t$ where $\mathcal{D}_t$ is the dataset of already visited points, otherwise is set to zero. β_h and l_h are the upper limits for the two variables.
The problem in 3.10 has three goals:

- to minimize $\Delta\beta$ in order not to exceed too much previous β_t to avoid inefficiency;
- to minimize the distance between x_{t+1} and x'_{t+1} (represented by the Euclidean norm term) because the algorithm should suggest a point close to the current potential area for exploitation;
- to minimize a penalty factor that is given to make sure not to sample again pre-existing observations.

The Discrete-BO algorithm is then summarized in the following Algorithm taken from [15]:

Algorithm 3: Discrete-BO Algorithm

input: GP model, initial data $\mathcal{D}_0 = \{(x_0, y_0)\}$, β_h, l_h
for $t = 0, \dots, n$ **do**
    Calculate β_t as suggested for GP-UCB, estimate l_t using $\mathcal{D}_t$
    Select $x_{t+1} = \text{argmax}_{x \in \mathcal{X}} \alpha_t^{UCB}(x)$ with β_t and l_t
    $x_{t+1} = round(x_{t+1})$
    if $x \in \mathcal{D}_t$ **then**
        Find the optimal β^* and l^* using 3.10
        $x_{t+1} = \text{argmax}_{x \in \mathcal{X}} \alpha_t^{UCB}(x)$ with β^*, l^*
        $x_{t+1} = round(x_{t+1})$
    Query the objective function to get y_{t+1}
    Augment $\mathcal{D}_{t+1} = \{\mathcal{D}_t, (x_{t+1}, y_{t+1})\}$ and update GP
end

3.2.2 Alternative surrogate model

SMAC

The Sequential model-based optimization (SMAC) [11] [12] uses a random forest (RF) as surrogate model of the black-box objective, instead of commonly used GP. In random forest T random regression trees are iteratively fitted using each time a bootstrap sample of training data, each obtained by drawing with replacement from the observed data N instances. Then we compute the random forest's predictive mean μ_x and variance σ_x^2 for a new configuration $x \in \mathcal{X}$ as the empirical mean and variance of its individual trees predictions for x.
The main steps of the algorithm are then the ones we have already analysed for standard BO:

- Based on the data collected thus far, construct a model that predicts a probability distribution for f's value at arbitrary point $x \in \mathcal{X}$.
- Use the model to quantify the desirability of learning $f(x)$ at each $x \in \mathcal{X}$ (the commonly used acquisition function is the EI) and select

$$x^* = \underset{x \in \mathcal{X}}{\text{argmax}}\, \alpha(x)$$

- Evaluate $f(x^*)$ resulting in new data point $(x^*, f(x^*))$.

SMAC uses as EI function the $\mathbb{E}[I_{exp}]$ criterion for log-transformed costs. Given the predictive mean μ_x and the variance σ_x^2 of the log-transformed cost of a

configuration x, it is defined as

$$EI(x) := \mathbb{E}\big[I_{exp}(x)\big] = f_{min}\phi(v) - e^{\frac{1}{2}\sigma_x^2+\mu_x} \cdot \Phi(v - \sigma_x)$$

where $v = \frac{ln(f_{min})-\mu_x}{\sigma_x}$, Φ denotes the cumulative distribution function of a standard normal distribution and $f_{min} = \mu(x^*) + \sigma(x^*)$ denotes the empirical mean performance of the current best optimum x^*.
The definition is based on log-transformed cost because it has been found that the logarithmic transformation of runtime data improves model quality. So when we consider an observed data in SMAC we refer to the couple (x_i, o_i) where $o_i = ln(y_i)$. This transformation could have the drawback of changing the cost metric that users aim to optimize, but this problem could be avoided by computing the prediction in the leaf of a tree by "untransforming" the data, computing the user-defined cost metric and then transforming the result again. The main advantage of this algorithm is that RF has a low computational cost and can naturally deal with discrete variables due to the tree-based structure and improve performance in this kind of optimization problems.

TPE

The Tree Parzen estimator (TPE) [4] is a tree-based method which evaluates the densities of good and bad candidates points in the search space. Instead of evaluating $p(y|x)$, it fits $p(x|y)$ and $p(y)$. Applying Bayesian rule we could in fact represent $p(y|x)$ as

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

TPE redefines $p(x|y)$ by using two densities: $l(x)$ which consider the observations that are lower than a chosen threshold y^* and $g(x)$ that is estimated using the rest of observations.

$$p(x|y) = \begin{cases} l(x) & \text{if} \quad y \leq y^* \\ g(x) & \text{if} \quad y > y^* \end{cases}$$

where y^* is set as a quantile γ of the observed y values, so that $p(y < y^*) = \gamma$. The two densities $l(x)$ and $g(x)$ are modelled using Parzen Estimators (also known as kernel density estimators). The Adaptive Parzen Estimator yields a model over $\mathcal{X}$ by placing density in the vicinity of K observations $\mathcal{B} = \{x_1, \ldots, x_K\}$. If we work with discrete variables, assuming the prior a vector of N probabilities p_i, the posterior vector elements were proportional to $Np_i + C_i$ where C_i counts

the occurrences of choice i in $\mathcal{B}$. If instead the variable is continuous then it is specified either by a uniform prior over some interval (a, b), or by a Gaussian or by a log-uniform distribution. Hence, the TPE substitutes an equally-weighted mixture of that prior with Gaussian centred at each of the $x_i \in \mathcal{B}$ and with standard deviation set to the greater of the distances to the left and right neighbour, but clipped to remain in a reasonable range.
The main cons is that in order to model these distribution well it needs a large number of observations at the beginning.
The parametrization of $p(x, y)$ as $p(x|y)p(y)$ is chosen to facilitate the optimization of the EI function:

$$\alpha(x) = \int_{-\infty}^{y^*} (y^* - y)p(y|x)dy = \int_{-\infty}^{y^*} (y^* - y)\frac{p(x|y)p(y)}{p(x)}dy \propto (\gamma + \frac{g(x)}{l(x)}(1-\gamma))^{-1}$$

where $\gamma = p(y < y^*)$. The aim is to maximize the EI with respect to x that means finding the best $x \in \mathcal{X}$ under the surrogate function $p(y|x)$.
Looking at this expression we deduce that the TPE-EI criterion is easily maximized by choosing a point with high probability under $l(x)$ and low probability under $g(x)$. The tree-structured form on l and g makes it easy to draw many candidates according to l and evaluate them in terms of $\frac{g(x)}{l(x)}$. On each iteration the algorithm returns the best candidate x^* with the greatest EI.
Using this kind of model the objective function improves much more rapidly than with random or grid search, reducing running time of the algorithm and getting better scores.

BOCS-SA

The BOCS-SA algorithm developed by Baptista-Poloczek [3] is an evolution of the standard Bayesian Optimization of Combinatorial Structures (BOCS) Algorithm.
Firstly we recall the general structure of the **BOCS Algorithm** and we consider a problem like

$$\underset{x \in \mathcal{X}}{\operatorname{argmax}} f(x)$$

with $\mathcal{X}$ discrete structured domain of feasible points. We consider in particular $\mathcal{X} = \{0, 1\}^d$. The function $f : \mathcal{X} \longrightarrow \mathbb{R}$ is the expensive to evaluate function of interest. A general model of f is thus given by $\sum_{S \in 2^{\mathcal{X}}} \beta_S \Pi_{i \in S} x_i$ where $2^{\mathcal{X}}$ is the power set of the domain and β_S is a real-valued coefficient. BOCS-SA considers restricted models that contains only monomials up to order k because otherwise the model would be impractical due to the exponential number of monomials. A

higher order increases expressiveness but reduces accuracy if we have limited data. A good trade-off is considering second-order models:

$$f_\beta(x) = \beta_0 + \sum_j \beta_j x_j + \sum_{i,j>i} \beta_{ij} x_i x_j$$

In this way the interaction terms are quadratic in $x \in \mathcal{X}$ but are linear in $\beta = (\beta_i, \beta_{ij}) \in \mathbb{R}^p$ with $p = 1 + d + \binom{d}{2}$. Since some form of regularization on x is often required, the problem is usually restated as follows:

$$\underset{x \in \mathcal{X}}{\operatorname{argmax}} f_\beta(x) - \lambda \mathcal{P}(x)$$

where $\mathcal{P}(x) = \|x\|_1$ or $\mathcal{P}(x) = \|x\|_2^2$.
The general algorithm structure is the following:

Algorithm 4: Bayesian Optimization of Combinatorial Structures

input: $f(x) - \lambda\mathcal{P}(x)$; N_{max} (sample budget); N_0(initial dataset)
Sample initial dataset $\mathcal{D}_0$
Compute the posterior on β given the prior and $\mathcal{D}_0$
for $t = 1$ **to** $N_{max} - N_0$ **do**
    Sample coefficients $\beta_t \sim P(\beta|X, y)$
    Compute approximate solution x_t for $max_{x \in \mathcal{X}} f_{\beta_t}(x) - \lambda\mathcal{P}(x)$
    Evaluate $f(x_t)$ and append y_t to y
    Update the posterior $P(\beta|X, y)$
end for
return: $\operatorname{argmax}_{x \in \mathcal{X}} f_{\beta_t}(x) - \lambda\mathcal{P}(x)$

In order to compute the next to evaluate solution x_t, defined as

$$x_t = \underset{x \in \mathcal{X}}{\operatorname{argmax}} f_{\beta_t}(x) - \lambda \mathcal{P}(x),$$

given the characteristic of the objective function of interest, the algorithm applies semidefinite programming.
The variant of the algorithm proposed in [3], denoted as **BOCS-SA**, replaces semidefinite programming by stochastic local search, in particular by Simulated Annealing. This performs a random walk on $\mathcal{X}$ starting from a point chosen uniformly at random. The next point x_{t+1} is selected in the neighbourhood $N(x_t)$ that contains all points with Hamming distance at most one from x_t.

Definition 3.2.1 (**Hamming distance**). *The Hamming distance or $\mathcal{L}_0$-distance between two vectors $x, y \in \mathcal{X}$ is the number of positions at which the corresponding elements are different:*

$$d_0(x, y) = \|x - y\|_0 = \sum_i \mathbb{1}(x_i \neq y_i)$$

where $\mathbb{1}(x_i \neq y_i) = \begin{cases} 1 & \text{if } x_i \neq y_i \\ 0 & \text{otherwise} \end{cases}$.

SA picks $x \in N(x_t)$ uniformly at random and evaluates $obj(x)$. Then if the observed objective value is better than the observation for x_t, SA sets $x_{t+1} = x$. Otherwise the point is adopted with probability $exp\left(\frac{obj(x)-obj(x_t)}{T_{t+1}}\right)$ where T_{t+1} is the current temperature. SA starts with a high T that encourages exploration and cools down over time to zoom in on a good solution.

Chapter 4

Background on Metaheuristics

Metaheuristics are a family of optimization methods which don't guarantee the optimality of the obtained solutions but provide "acceptable" solutions in a reasonable time for solving hard and complex problems. Metaheuristics are commonly applied in many classes of problems like:

- Engineering design, topology and structural optimization in electronics, telecommunications, auto motive, and robotics.
- Machine learning and data mining in bioinformatics, computational biology and finance.
- System modelling, simulation and identification in chemistry, physics, and biology; control, signal, and image processing.
- Planning in routing problems, robot planning, scheduling and production problems, logistics and transportation, supply chain management.

As for previously described optimization methods also in this case we have to take into account the trade-off between the exploration of the search space (diversification) and the exploitation of the best solutions founded (intensification). In intensification the promising regions, determined by the obtained good solutions, are explored more thoroughly in the hope of improving them; in diversification, instead, non explored regions must be visited to be sure that the search is not confined only to a reduced number of regions in the configuration space.

The main advantage of using metaheuristics is a restrictive assumption in formulating the model. In fact, some optimization problems cannot be formulated with an unambiguous analytical mathematical notation, or furthermore an analytical formulation should not exist at all, like for black-box functions. In those cases the problem cannot be solved in an exhaustive manner and so the use of metaheuristics provides an efficient alternative.

4.1 Introduction to Metaheuristics

Deterministic versus stochastic metaheuristics. A deterministic metaheuristic solves an optimization problem through deterministic decisions; examples are local search and tabu search methods. On the contrary, in stochastic metaheuristics some random rules are applied during search; examples are simulated annealing and genetic algorithm. In the former case, using the same initial solution, we will obtain the same final solution, whereas in stochastic metaheuristics this is not guaranteed (i.e. the same initial solution could lead to different final solutions).

Population-based search versus single-solution based search. Single-solution based algorithms like simulated annealing and local search manipulate a single solution during the search, while in population-based algorithms, like genetic algorithms, a whole population of solution is evolved. From a technical point of view we can say that single-solution based metaheuristics are exploitation oriented because they intensify the search in local regions, whereas population-based ones are exploration oriented, providing a better diversification in the space.

4.1.1 Solution representation

When we work with a problem we have to provide a structure to its solutions. This is not unique, the same problem could be represented in many different way, but it has to respect some characteristics:

- **Completeness**. It should be able to represent all problem solutions.
- **Connexity**. Given two feasible solutions, it must exist a search path to connect them.
- **Efficiency**. The representation must be easy to manipulate by the search operators reducing space and time complexity.

According to their structure we can distinguish two main classes of representations: linear and non linear.

Linear representations Linear representations may be viewed as strings of symbols of a given alphabet. In many classical optimization problems, where the variables denote the presence or absence of an element, binary encoding may be used. It is for example the case of the well-known knapsack problem. In the 0/1 knapsack problem with k objects each solution is represented by a vector x pf size k where

$$x_i = \begin{cases} 1 & \text{if object } i \text{ is in the knapsack} \\ 0 & \text{otherwise} \end{cases} \qquad \forall i = 1, \ldots, k$$

This representation can be generalized to any discrete values based encoding using an n-ary alphabet. In this case, each variable takes its value over the defined alphabet and the encoding will be a vector of discrete variables. This encoding may be used for those problems where the variables can take a finite number of values, such as combinatorial optimization problems.

Non linear representations Non linear representations are more complex encoding structures generally based on graphs and trees. The tree encoding is used mainly for hierarchical structured optimization problems, in this case a solution is represented by a tree of objects. The tree may encode an arithmetic expression, a first-order predicate logic formula or a program. An example of use is given by the tree encoding for regression problems. In this case given some input and output values regression problem consist in finding the function that will provide the best output for all inputs. Any $\mathcal{X}$-expression can be drawn as a tree of functions and terminals where the functions could be for example sine, cosine, sum, whereas the terminals (i.e. leaves of the tree) represent constants or variables of the problem.

Chosen the best fitting representation, we have to define the global structure of the problem of, called **Search space**.

Definition 4.1.1. *The **search space** is defined by a directed graph $\mathcal{G} = (\mathcal{X}, E)$ where the set of vertices $\mathcal{X}$ corresponds to the possible solutions of the problem defined by the representation used to solve the problem, and the set of edges E corresponds to the move operators used to generate new solutions.*

In the graph will exist an edge between solutions x_i and x_j if the solution x_j can be generated from the solution x_i using a move operator (i.e. x_i and x_j are neighbours). The graph is *directed* because the existence of a link (i, j) doesn't imply that also the link with reversed direction (j, i) exists. In addition we highlight that using different neighbouring definitions we will generate different search spaces.
One of the main properties that has to be verified when we describe a search space is **connectivity**: for any two solutions x_i and x_j there should be a path from x_i and x_j. This implies in particular that from any solution x_i there will be a path from x_i to the global optimum of the problem x^*.

4.1.2 Performance Analysis

Exact optimization methods guarantee the global optimality of solutions so the efficiency in terms of search time is the main indicator to evaluate the performances of the algorithms. Instead, if we consider metaheuristic search methods we have to evaluate also indicators related to the quality of the solution, the computational effort and the robustness.

Solution Quality

The indicator used to measure the quality of the solution is based on measuring the distance or the percent deviation of the obtained solution to one of the following ones (graphically explained in [Fig. 4.1]):

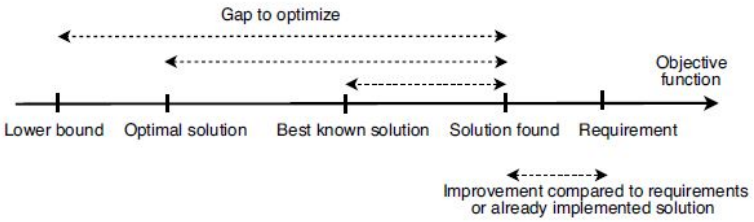


Figure 4.1: Performance of the solution quality. Graphical explanation of the different distances and percent deviations for a minimization problem. [23]

- **Global optimal solution.** This allows a more absolute performance evaluation of the different metaheuristics. The difference can be evaluated as

$$|f(s) - f(s^*)| \qquad \text{or} \qquad \frac{|f(s) - f(s^*)|}{f(s^*)}$$

where s is the obtained solution and s^* the global optimal one. A drawback of these formulas is that they are not invariant under different scaling of the objective function. For this reason are usually preferred

$$\frac{|f(s) - f(s^*)|}{|f_{worst} - f(s^*)|} \quad \text{or} \quad \frac{|f(s) - f(s^*)|}{|E_{unif}(f) - f(s^*)|}$$

where f_{worst} represents the worst objective function and $E_{unif}(f)$ denotes expectation with respect to the uniform distribution of solutions.
The global optimal solution may be deduced by an exact algorithm or it could be known a priori using constructed instances.

- **Lower/upper bound solution.** If we consider an optimization problem for which the optimal solution is not available, tight lower/ upper bounds may be considered as an alternative to global optimal solutions respectively for a minimization/ maximization problem. These could be known or easily computed using relaxation techniques like continuous relaxation or Lagrangian relaxation.

- **Best known solution.** If we are working with a classical problem usually there exist libraries of standard instances to compare with available ones in literature. The best available solution is updated each time an improvement is found.

- **Requirements or actual implemented solution.** For real-life problems we could be in the condition to have a threshold to satisfy.

Computational Effort

The **time complexity** of an algorithm is the number of steps required to solve a problem of size n and it is defined in terms of the worst-case analysis. It is evaluated in terms of CPU time or wall clock time, with or without input/output and preprocessing/postprocessing time. The main drawback of computation time measure is that it is influenced by computer characteristics such as the hardware, operating system, language and compiler on which the metaheuristic is executed. For this reason it is preferred to use indicators such as the number of objective function evaluations, which is independent of the used computer system. It is an acceptable measure for time-intensive and constant objective functions.

Robustness

In literature we can find many different alternative definitions for robustness. It can be used to measure the insensitivity against small deviations in the input instances (data) or the parameters: the lower the variability the better the robustness. It could also be used to measure the performance of the algorithms according to different types of input instances and/or problems using the same parameters: it could be overfitted for some instances and be less efficient for others. In stochastic algorithms it may be also related to the average/deviation behaviour of the algorithm over different runs of the algorithm on the same instance.

4.2 Single-Solution Based Metaheuristics

The Single-Solution Based Metaheuristics are optimization methods which explore the search space using "walks" or trajectories through neighbourhoods, performed by iterative procedures that move from the current solution to another admissible one. They are characterized by two main phases:

- In the **generation phase** a set of candidate solutions $C(x)$ is generated from the current solution x using local transformations;
- In the **replacement phase** a selection is performed from the candidate set $C(x)$ to replace the current solution. The new solution $x' \in C(x)$ would be so the new solution for the next iteration.

4.2.1 Neighbourhood Definition

The first essential step to define a Single-Solution Based Metaheuristic is the definition of the neighbourhood.

Definition 4.2.1 (**Neighbourhood**). *A neighbourhood function N is a mapping $N : \mathcal{X} \longrightarrow 2^{\mathcal{X}}$ that assigns to each solution $x \in \mathcal{X}$ a set of solutions $N(x) \subset \mathcal{X}$.*

A solution x' in the neighbourhood of x is called *neighbour* of x and it is generated by the application of a move operator that performs a small perturbation to the solution x. When we define a neighbourhood we also study its locality that is the effect on the solution when performing a perturbation. When small changes in the representation reveal small changes in the solution the neighbourhood is said to have a strong locality, otherwise we will refer to as weak locality.

Before providing some further definitions on the neighbourhood we recall some concepts about the notion of distance.

Definition 4.2.2. *Let $\mathcal{X}$ be a set, a **distance function** or metric on $\mathcal{X}$ is any mapping $d : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ such that, $\forall x, y, z \in \mathcal{X}$:*

- $d(x, y) \geq 0$*;*
- $d(x, y) = 0 \iff x = y$ *(separative property);*
- $d(x, y) = d(y, x)$ *(symmetrical property);*
- $d(x, z) \leq d(x, y) + d(y, z)$ *(triangle inequality).*

Definition 4.2.3. *A function $\|\cdot\| : \mathbb{R}^d \longrightarrow \mathbb{R}$ defines a **norm** if it respect the following properties:*

- $\|x\| \geq 0 \qquad \forall x \in \mathbb{R}^d$ *and* $\|x\| = 0 \iff x = 0$
- $\|\alpha x\| = |\alpha| \|x\|$ *for any vector $x \in \mathbb{R}^d$ and any scalar $\alpha \in \mathbb{R}$*
- $\|x + y\| \leq \|x\| + \|y\| \qquad \forall x, y \in \mathbb{R}^d$ *(triangle inequality)*

The most commonly used norms are the $l_p - norms$ which are defined by

$$\|x\|_p = \left(\sum_{i=1}^{d} |x_i|^p \right)^{1/p}$$

Hereinafter we will focus on the $l_1 - norm$

$$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_d|$$

and the $l_2 - norm$, broadly known as *Euclidean norm*

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_d^2} = \sqrt{x^T x}$$

Given a norm $\|\cdot\|$ and any two vectors $x, y \in \mathbb{R}^d$ the distance between x and y with respect to this norm is defined as $\|x - y\|$ and it is called **induced distance**.

Definition 4.2.4. *The neighbourhood $N(x)$ of a solution in a **continuous space** is the ball with center x and radius equal to ϵ with $\epsilon > 0$.*

$$N(x) = \{x' \in \mathcal{X} : \|x' - x\| < \epsilon\}$$

with $\|\cdot\|$ selected norm (e.g. Euclidean norm).

Definition 4.2.5. *In a **discrete** optimization problem the neighbourhood $N(x)$ of a solution x is represented by the set*

$$N(x) = \{x' \in \mathcal{X} : d(x', x) \leq \epsilon\}$$

where d represents a given distance that is related to the move operator.

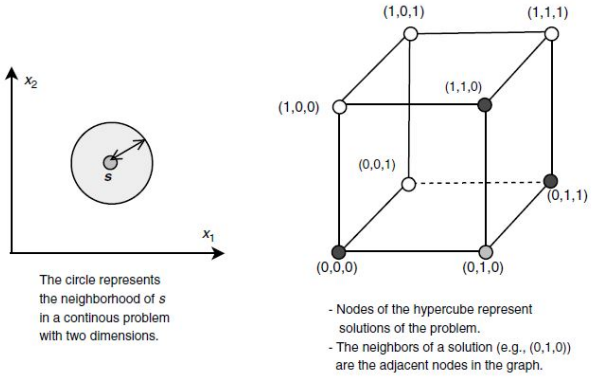


Figure 4.2: Example of neighbourhood respectively for a continuous and a discrete (binary) space. [23]

Some graphical examples of neighbourhoods in continuous and discrete spaces are shown in [Fig. 4.2].
As already seen in BOCS-SA the natural neighbourhood for binary representations is based on the Hamming distance. The Hamming neighbourhood for binary encodings may be then extended to any discrete vector representation using a given alphabet Σ. In this case the substitution can be generalized by replacing the discrete value of a vector elements by one of the other characters available in the alphabet. So if the cardinality of the alphabet Σ is k then the size of the neighbourhood will be $(k-1) \cdot n$ for a discrete vector of size n.
If we take back the notions about search space we could similarly provide a notion of distance on the graph G, useful in the design of the search mechanism of the algorithm.

Definition 4.2.6. *Given two solutions $x_i, x_j \in \mathcal{X}$, the distance $d(x_i, x_j)$ is defined as the length of the shortest path in the graph G, i.e. the minimum number of applications of the move operator needed to move from x_i to x_j.*

When we define the neighbourhood we have to trade-off between its size (or diameter) and the computational complexity to explore it. The size of the

neighbourhood for a given solution x is the number of neighbours of x. Designing large neighbourhoods may improve the quality of the obtained solutions since more neighbours are considered at each iteration, but in the meanwhile it requires an additional computational time to generation and evaluation. The impact of the neighbourhood size in local search is displayed in [Fig. 4.3]

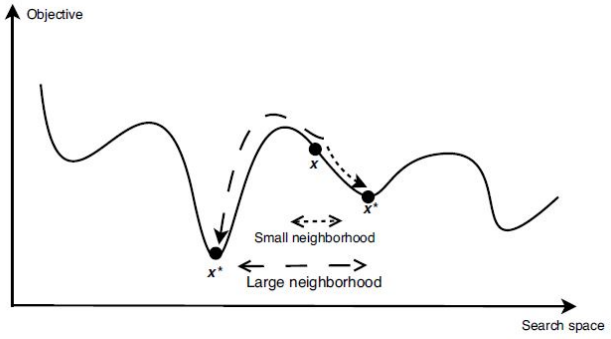


Figure 4.3: Large neighborhoods improve the quality of the search because they allow to explore much more solutions at each iteration. [23]

A commonly used strategy is to generate variable distance moves (k-distance or k-exchange) with a distance of 2 or 3. Given a neighbourhood N defined by neighbours of distance $k = 1$ from a solution x

$$N_1(x) = \{x' \in \mathcal{X} : d(x, x') = 1\}$$

similarly a larger neighbourhood $N_k(x)$ of distance k is defined as the set

$$N_k(x) = N_{k-1}(x) \cup \{x'' | \exists x' \in N_k(x) : x'' \in N_1(x')\}$$

The distance k could be almost equal to n, where n is the size of the problem (i.e. it is the maximum distance between any two solutions). Since $N_n(x)$ represents the whole search space finding the best solution in this neighbourhood is NP-hard if the original problem is NP-hard.

4.2.2 Local Search

The local search is the simplest metaheuristic method. It starts at a given initial solution, then, at each iteration, the heuristic replaces the current solution by a neighbour that improves the objective function. It stops when there are no more improving solutions among the neighbours of the current one, that is the

local optimum is reached. The algorithm will generate a sequence of solutions $x_1, \ldots, x_k \in \mathcal{X}$ starting from an initial solution x_0. The length k of this sequence is unknown a priori and it depends on the time needed to satisfy the termination criterion (i.e. reaching the local optimum). The elements of this sequence satisfy the following properties:

- $x_{i+1} \in N(x_i) \quad \forall i \in [0, k-1]$, where $N(x_i)$ is the neighbourhood of x_i
- $f(x_{i+1}) > f(x_i) \quad \forall i \in [0, k-1]$ (maximization problem supposed)
- x_k is a local optimum: $f(x_k) \geq f(x) \quad \forall x \in N(x_k)$.

The whole algorithm should be summarized as follows:

Algorithm 5: Local Search

```
Generation of the initial solution x = x_0
while not Termination Criterion do
    Generate neighbourhood N(x) from current solution x
    if there is no better neighbour then
        Stop
    end
    x = x' with x' better neighbour in N(x)
end
Output Final solution found (local optima)
```

The better neighbour in $N(x)$ can be selected using different strategies:

- **Best improvement(steepest descent)**: in this case the best neighbour is chosen as the one that improves the most the cost function. The exploration of the neighbourhood is exhaustive and fully deterministic. For this reason this type of exploration is time consuming for large neighbourhoods.
- **First improvement**: in this case the net solution is the one selected as the first improving neighbour of the current one. This method implies a partial evaluation of the neighbourhood.
- **Random selection**: here a random selection is applied to those neighbours which improves the current solution.

The choice of the best strategies depends on each particular problem. In many applications the first improving strategy leads to the same quality of solutions as the best improving strategy using a smaller computational time and with a lower probability of premature convergence to local optima.

The convergence to local optima is the main drawback of the local search methods. They work well if there are not too many local optima in the search space or if the quality of the different local optima is more or less similar. In order to avoid this problem we could apply different strategies:

- **Iterating from different initial solutions.** This strategy is applied for example in multistart local search search ad iterated local search.
- **Accepting nonimproving neighbours.** In this case we enable moves that degrade the current solution making it possible to move out the basin of attraction of a given local optimum. Examples of this class of algorithms are simulated annealing and tabu search.
- **Changing the neighbourhood.** This strategy consists in changing the neighbourhood structure during the search.
- **Changing the objective function or the input data of the problem.** In this case the problem is transformed by perturbing either the input data or the objective function or the constraints of our problem. This approach has been implemented in the guided local search and in noising methods.

4.3 Genetic Algorithm

The Genetic Algorithm is an optimization method which belongs to the class of **Population-Based Metaheuristcs**. It takes inspiration from natural evolution and genetics and reflects the process of natural selection where the fittest individuals are selected in order to produce the offspring of the next generation. This class of algorithms plays a main role in combinatorial problems in order to avoid local optima, in particular it has been widely applied in problems of business, engineering, science.
Instead of classical single solution algorithms, it starts and works at each step with a population of candidate solutions. The main advantage of this is linked to the possibility of better simultaneous exploration. Point-to-point methods are a perfect prescription for locating false peaks in multimodal search spaces, while working with a rich dataset simultaneously allows climbing many peaks in parallel so that the probability of finding a false peak is reduced.
In the Genetic Algorithm each possible solution, denoted as **chromosome**, is represented by a string of finite length. Each element of the chromosome is called **gene** and it corresponds to a problem variable. The values that a gene can assume are called **alleles** and belong to a finite alphabet of symbols. The

initial population, whose size depends on the nature of the problem, is usually generated randomly according to a uniform distribution over all admissible solutions. All these solutions are evaluated through a **fitness function** f that represents a measure of profit, utility or goodness that we want to maximize (objective function of the problem). A fitness-based process is so applied to select individuals to be used as parents of the next generation, generated through mutations and recombinations. The new population is finally created selecting some chromosome from the old generation and some of the new solutions. All the process is repeated until a certain satisfaction condition is reached. Both the generation and the replacement phases are memoryless: they are only influenced by the current population, the previous history has no effect on it.
To better understand the algorithm we recall here the general pseudo-code:

Algorithm 6: Genetic Algorithm

```
Choose an initial population of individuals P(0)
Evaluate the fitness f of all the individuals in P(0)
Choose a maximum number of generations t_max
while t < t_max do:
    t ← t + 1
    Select parents for offspring production
    Apply reproduction and mutation operators
    Create a new population P(t)
Return the best individual of P(t)
```

The following are the key elements on which we have to take some decisions:

- the way the parents are selected to produce offspring (**selection methods**);
- the way selected parents are recombined and which kind of mutation is applied on the individuals (**reproduction**);
- the probability of crossover and mutation which conditions exploration and exploitation of research space;
- the population size and how to adjust it towards iterations.

The **selection operator** makes the difference with respect to traditional random methods. It focuses the research on most promising areas of the search space: individuals with higher fitness are more likely to be selected for the next generation. Also this operator takes on a main role in the trade-off between exploration and exploitation: strong selection pressure may in fact lead to convergence to a

local optimum.
Another important peculiarity of this class of algorithms, related to the fact that we haven't worked so far with strings as strings alone, is given by the concept of **schemata** or **similarity templates**[10]. A schema is a similarity template describing a subset of strings with similarities at certain string positions. For example if we create strings over the ternary alphabet $\{0, 1, *\}$, a schema matches a particular string if at every location in the schema either a 1 matches a 1 in the string, or a 0 matches a 0, or a $*$ matches $*$ (i.e. the schema $*000$ matches $\{1000, 0000\}$). The information provided by similarities, as the one that comes from fitness values, help direct our search. Short-defining-length schemata, also called *building blocks*, are propagated generation by generation by giving exponentially increasing samples to the observed best.

4.3.1 Initial Population

An appropriate choice of the initial population is one of the core problems, in fact it plays a crucial role in the effectiveness and efficiency of the algorithm. The main criterion to deal with is diversification. If the initial population is not well diversified a premature convergence could occur.
The diversification criterion could be taken into account explicitly through maximizing the minimum distance between any two solution in the population:

$$\max_{i=1,\dots,n} \left(\min_{j=1,\dots,i-1} \{d_{ij}\} \right)$$

where d_{ij} represents the distance in the decision space between two solutions x_i and x_j and n is the size of the population. More in general the diversification criterion could be classified into four categories: random generation, sequential diversification, parallel diversification and heuristic initialization.

Random Generation. A very common strategy is to generate occurrences randomly in the feasible range. In continuous optimization each variable x_{ij} is defined to be in a given range $[l_j, u_j]$, with $l_j, u_j \in \mathbb{R}$. Each solution x_i of the population is a k-dimensional real vector where k is the number of variables. Each element of the vector x_{ij} is generated randomly as follows:

$$x_{ij} = l_j + rand_j[0, 1] \cdot (u_j - l_j) \qquad \text{for } i \in [1, n], j \in [1, k]$$

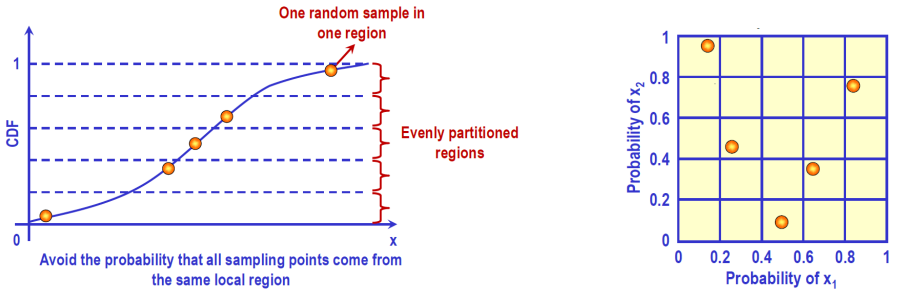
where $rand_j$ is a uniformly distributed random variable in the range $[0, 1]$.
In discrete optimization the same uniform random initialization may be applied

to binary vectors, discrete vectors and permutations.
The random generation is usually performed according to pseudo-random or quasi-random sequence of numbers. This happens because we could prove randomness for a sequence of infinite size but it is impossible for a finite sequence. The most popular random generation is the pseudo-random one which uses various classical generators like congruential quadratic. In quasi -random sequence the goal of the generator is related not only to the independence between the successive numbers but also to their dispersion.

Sequential Diversification. In a sequential diversification the solutions are generated in such a way that the diversity is optimized. The most common sequential strategy used is the Simple Sequential Inhibition (SSI) process. In this case, given a sub-population $\mathcal{Q}$, initially composed by only one random picked solution, any new selected solution must be at a minimum distance Δ to all other solutions of the current sub-population $\mathcal{Q}$. The process is repeated iteratively until the specified number of solutions isn't reached. This strategy guarantees diversity due to the fact that the minimum distance between any two solutions is at least Δ. Meanwhile the main cons is related to the high computational cost.

Parallel Diversification. In this case solutions are generated in a parallel independent way. A widely used method is based on the Latin hypercube sampling. In this case given n the size of the population and k the number of variables, the variable range of each variable x_j is divided into n equal segments of size $\frac{u_j - l_j}{n}$, with $l_j, u_j \in \mathbb{R}$ respectively the lower and upper bounds of x_j. A random real number is then generated in each segment. An example of this strategy is explained in [Figure 4.4] (taken from Xin Li works) which represents both a one dimensional and a two dimensional Latin Hypercube sampling. As seen in the pictures the sampling is random in each grid and we can extract one sample in each row and column. The procedure can be generalized to higher dimensions.

Heuristic Initialization. In this case any heuristic (e.g. local search) can be used to initialize the population. A greedy heuristic can be used to determine the initial solution. This strategy depends on the fitness landscape of the optimization problem and for this reason it is so more effective and efficient than random initialization. The main drawback of this method is that it lacks in population diversity and it could generate premature convergence and stagnation



(a) One dimension (b) Two dimensions

Figure 4.4: Explanation of the Latin hypercube strategy. In 4.4a is shown the one-dimensional Latin hypercube sampling: the CDF is divided into N regions and one sampling point is randomly picked up in each region. In 4.4b is explained the two-dimensional LHS generated by the randomly combination of two one-dimensional samples.

of the population.

In Table 4.1 are summarized the comparisons among the different initialization strategies. The evaluation is better with more plus (+) signs.

Strategy	Diversity	Computational Cost	Quality
Pseudo-random	++	+++	+
Quasi-random	+++	+++	+
Sequential diversification	++++	++	+
Parallel diversification	++++	+++	+
Heuristic	+	+	+++

Table 4.1: Performance comparisons of different initialization strategies. [23]

Sequential and parallel diversification methods provide in general the best diversity. The heuristic initialization provides in general better solutions in terms of quality of initial solutions but with the expense of a higher computational cost and a reduced diversity.
In our work we will generally use pseudo-random generators.

4.3.2 Selection Methods

The selection strategy is one of the main components of the Genetic Algorithms. All the proposed solutions to this matter are based on the idea that "the better is an individual, the higher it is the chance of being parent". We recall now briefly some of the most used selection mechanisms.

Roulette Wheel Selection. According to this strategy we should assign to each individual a selection probability that is proportional to its relative fitness. Given f_i the fitness of the individual i in population P, the probability to be selected is:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}.$$

We then consider a pie graph where the space reserved to each individual is proportional to its fitness. The selection of μ individuals is performed by μ independent spins of the roulette wheel: at each spin we select one individual. The general principle is shown in [Figure 4.5].
In the roulette wheel selection better individuals have more space ans so more chance to be chosen but at the same time outstanding individuals will create a bias and could cause a premature convergence.

Stochastic Universal Sampling. This is a variant of the previous method. In this case μ individuals are selected simultaneously by a single spin of the roulette wheel. The general principle is shown in [Figure 4.5].

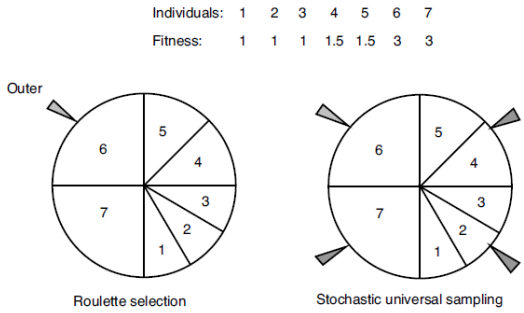


Figure 4.5: Roulette wheel strategies. The first picture describes the standard *Roulette Wheel Selection* which select one single individual at each spin of the roulette wheel. The second one instead represents the *Stochastic Universal Sampling* with $\mu = 4$, so 4 individuals are selected simultaneously. [23]

Tournament Selection. This method consists in randomly selecting k individuals, where k is called the size of the tournament group. A tournament is applied to select among the k individuals the best one. In order to select μ individuals this procedure must be iterated μ times.

4.3.3 Reproduction

In the previous section we have described how the individuals could be selected to be parents of the next generation. After this selection has been performed, variation operators are applied to generate the offspring. Those operators are subdivided into two groups: **mutation** (unary operator) and **recombination** or **crossover** (binary operator).

Mutation

Mutation operators are unary operators that work on a single individual and that could also affect only one gene at a time. The probability of mutation p_m is often set to $1/k$ where k is the number of decision variables, which corresponds to only one mutated variable in average. When a mutation operator is designed it must respect some characteristics:

- **Ergodicity**: it should allow to reach every solution of the search space;
- **Validity**: the solutions produced by the operator must be valid;
- **Locality**: the mutation should produce a minimal change.

When we work with finite alleles we could distinguish three main classes of operators: in **binary representation** the flip operator is the most commonly used; in **discrete representation** the mutation consists in changing the value associated with an element with another value of the alphabet; in **order-based representations** are usually applied permutations based on swapping, inversion and insertion.
When instead we work with real-valued vectors the most used class of mutation operators has the form:

$$x' = x + M$$

where M is a random variable. The value M could have different forms. In case of **uniform random mutation** the value is chosen uniformly random within the interval $[a, b]$, with $a, b \in \mathbb{R}$. In case of **normally distributed mutation** it is used a Gaussian distribution $M = \mathcal{N}(0, \sigma)$ where $\mathcal{N}(0, \sigma)$ is a vector of independent random Gaussian numbers with mean 0 and standard deviation σ. Other mutation operators could involve polynomial probability distributions or other well-known distributions like the Cauchy or Laplace.

Recombination or Crossover

The crossover is a binary operator whose main role is to inherit some characteristics of the two parents to generate the offspring. Also in this case we can list some characteristics that should be respected:

- **Heritability**: some genetic material from both the parents should be inherited. If we work with a pure recombination operator strong heritability is satisfied (i.e. two identical individuals generate identical offspring). In our case instead we work with both mutation and crossover and so some diversification is introduced in the parents.
- **Validity**: the solutions generated should be valid.

A crossover operator Ox is *respectful* if the common decisions in both parents are preserved in the offspring. It is denoted as *assorting* if the distance between the parents (p_1, p_2) and the offspring o is lower or equal to the distance between the parents:

$$d(p_1, o) \leq d(p_1, p_2) \wedge d(p_2, o) \leq d(p_1, p_2) \quad \forall o \in O(p_1, p_2, Ox)$$

where $O(p_1, p_2, Ox)$ is the set of all possible offspring generated by the crossover operator Ox.
When we define a crossover operator we have to choose a *crossover rate* $p_c \in [0, 1]$ that represents the proportion of parents on which the operator will be applied. We recall now some of the widely used crossover techniques.

n-Point crossover. The simpler crossover operator is the **1-Point crossover**. In this case a site t in the string is randomly chosen, then two offspring are created by interchanging the two segments of the parents. For example if we have two individuals $ABC|DEFG$ and $abc|defg$ the children will be $ABC|defg$ and $abc|DEFG$.
Generalizing in the **n-Point crossover**, n crossover sites are selected. Thus if we apply for example a 2-point crossover to the strings $A|BCD|E$ and $a|bcd|e$, we will obtain $A|bcd|E$ and $a|BCD|e$ as offspring. A general example in binary case is displayed in [Fig.4.6].
Working with crossover we could easily fall into disruption. If we consider a 1-point crossover and suppose that the elements A and E are coadapted, namely their presence provide benefits to the individuals, then if we consider the string ABCDE we highlight that in most of the offspring the coadaptation will be lost because the cutting point will fall between the two elements. Another

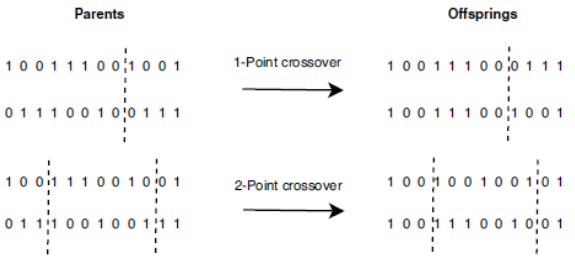


Figure 4.6: **n-Point crossover**. In the picture are shown respectively an example of 1-Point crossover (on the top) and 2-Point crossover on a binary string. The cross sites are chosen randomly. [23]

example could be given by $AA|BBCC$ and $BB|AACC$ which could produce the individual $AAAACC$ that contains only two groups out of three. The solution generated doesn't share the same good schemata of the parents and so it may produce very low-quality individuals from high-quality solutions.

Uniform crossover. In the case of uniform crossover two individuals are recombined without choosing a cut point. Each parent contributes equally to generate offspring, each gene is selected randomly between the respective gene of the two parents. A simple example is shown in [Fig. 4.7]

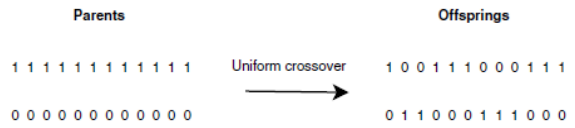


Figure 4.7: **Uniform crossover.** Example of simple uniform crossover with binary strings. Each gene is selected randomly between the respective genes of the two parents. [23]

Permutation crossover operators. The previous crossover will not always generate feasible solutions. If we consider strings that could only be permutations of the previous one, applying classical crossover operators we will easily generate solutions that are not permutations and are so unacceptable. To avoid this problem some permutation crossover operators have been introduced, based on the definition of a certain mapping system between the two parents (e.g. Order crossover, Partially mapped crossover). We won't dwell on this class of operators

further since they are not useful for our work, but more details should be found in [23].

4.3.4 Replacement strategies

Since we consider a constant size of the population, after the generation of the offspring we have to choose which elements will belong to the new generation and so become the initial population of the following step. We can use here some of the selection strategies explained in section 4.3.2 to withdraw individuals. The extreme replacement strategies are:

- **Generational replacement.** In this case the parents population is completely replaced by the offspring one.
- **Steady-state replacement.** In this case at each iteration only one offspring is generated and added to the population in place of the worst individual among the parents.

Between these two cases there are many intermediate solutions in which we replace a given number λ of individuals of the parents population with $1 < \lambda < n$, where n is the population size. The individuals from parents and offspring populations should be chosen according to elitist strategy, which means that only best individuals are selected. This approach leads to a faster convergence. Even if it could cause premature convergence, it may be necessary to avoid the sampling error problem.

Chapter 5

Proposed methods

As anticipated in chapter 2, we focus our attention on the optimization problems characterized by black-box and expensive to evaluate objective functions with discrete variables. Starting from Bayesian Optimization literature, summarized in chapter 3, we combine it with Metaheuristics strategies to design two possible improving algorithms. In the first one, called **LSBO**, we join BO with Local Search Metaheuristics to provide a better exploration of the search space. In the second one, called **GBO**, we mix BO and Genetic Algorithm to take into account both the space structure and the correlation among variables.
From a technical point of view both the algorithms are implemented in Python and are based on two libraries:

- **BoTorch**, a library for Bayesian Optimization research built on top of PyTorch. It provides a modular and easily extensible interface for composing Bayesian Optimization primitives, including probabilistic models, acquisition functions and optimizers.
- **GpyTorch**, a library for efficient and scalable GPs implemented in PyTorch and integrated in BoTorch library. We use it to be able to define all the GP parameters independently.

The employment of those libraries is fundamental in terms of scalability, modularity and speed, related to the fact that they utilize GPU acceleration and state-of-the-art inference algorithms.

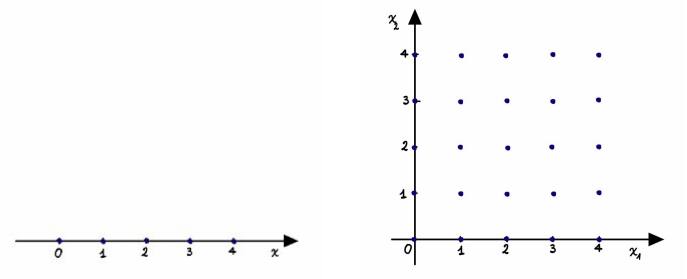
5.1 General Assumptions

Before describing the algorithms structure we summarize the main assumptions we take on the optimization problem

$$\max_{x \in \mathcal{X}} f(x)$$

- The **objective function** $f : \mathcal{X} \longrightarrow \mathbb{R}$ is a black-box function, expensive to evaluate in terms of time and cost;
- We work with d variables and so $x = (x_1, \ldots, x_d) \in \mathcal{X}$ is a d-dimensional vector which takes values from the set of feasible solutions $\mathcal{X}$;
- Each variable x_i is supposed to take values from a discrete finite alphabet $\mathcal{A}$ of cardinality t. Therefore, the configuration space $\mathcal{X} = \mathcal{A}^d$ consists of t^d points.

We infer on the configuration space a simple **grid structure**, that is so reflected in a d-dimensional hypercube. The graphical representation for one (line) and two variables (square grid) with $t = 4$ is given in [Fig. 5.1].



(a) (b)

Figure 5.1: Graphical representation of the configuration space in the case of one variable x (a) and two variables x_1 and x_2 (b). Each blue point indicates a feasible configuration. The integers $\{0, \ldots, 4\}$ represent the admissible alphabet.

The search space structure will be defined in details later for each of the following algorithms since it depends on the neighbourhood definition.
When we define the neighbourhood for a given point we have to define a proper metric. In our algorithms we will consider the distance induced by the $l_1 - norm$, i.e. the L_1**-distance**, defined as follows.

Definition 5.1.1 (L_1**-distance**). *Given two vectors $x, y \in \mathcal{X}$, the L_1-distance $5d_1 : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}^+$ is defined as:*

$$d_1(x, y) = \|x - y\|_1 = \sum_i |x_i - y_i|$$

Considering a Cartesian system it is the sum of the lengths of the projections of the segments between the two points onto the coordinate axes. This metric is also known as **Manhattan distance** or **Taxicab Geometry** since it is the distance you would need to walk in a city like Manhattan since must stay on the streets and can't cut through buildings.
In addition to take into account the correlations among variables we infer on them a **graph structure** $\mathcal{C} = (\mathcal{V}, \mathcal{E}, W)$, where:

- $\mathcal{V}$ is the set of nodes. Each node corresponds to a variable x_i of the problem, and so $|\mathcal{V}| = d$.
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. The presence of a link between variables i and j in the graph indicates that x_i influences x_j.
- $W \in \mathbb{R}_+^{\mathcal{V} \times \mathcal{V}}$ is the weight matrix. We know that $W_{ij} > 0$ if and only if $(i, j) \in \mathcal{E}$ and in particular in our case W_{ij} indicates the correlation among the variables.

We consider a directed and weighted graph, so the matrix W is not symmetric, i.e. the correlations between the variables could be different in the two directions.

In the algorithm definition we suppose that our a priori knowledge includes only the configuration space $\mathcal{X}$ and the matrix of weights W, no training data are provided. Since f is a black-box function we have no information about its properties but it is always possible to know its value at each point $x \in \mathcal{X}$.

5.2 LSBO Algorithm

The first algorithm we introduce is the **Local Search Bayesian Optimization** (**LSBO**) in which we maintain the general structure of the BO Algorithm but we introduce a different space exploration in acquisition function optimization. As known from the theory described in chapter 3 the BO Algorithm is characterized by two main components:

- the **surrogate model**, which we assume to be a Gaussian Process

$$f(x) \sim \mathcal{GP}(\mu_0(x), k(x, x'))$$

with constant mean μ_0 and RBF kernel

$$k(x, x') = exp\left(-\frac{\|x - x'\|_2^2}{2l^2}\right)$$

where $\|\cdot\|_2^2$ is the Euclidean distance and l is the characteristic length scale.

- the **acquisition function**. We will evaluate both the Expected Improvement(EI) defined as

$$EI(x) = \begin{cases} (\mu(x) - f(x^*))\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases}$$

where $Z = \frac{\mu(x) - f(x^*)}{\sigma(x)}$ and $\phi(\cdot)$ and $\Phi(\cdot)$ denote respectively the PDF and the CDF of the standard normal distribution,
and the Upper Confidence Bound(UCB) defined as

$$UCB(x; \beta) = \mu(x) + \sqrt{\beta}\sigma(x).$$

In standard BO Algorithms with discrete variables the next point to evaluate is chosen maximizing the acquisition function $\alpha(x)$ on its continuous domain region $\mathbb{R}^d$:

$$\tilde{x}_{n+1} = \underset{\tilde{x} \in \mathbb{R}^d}{\operatorname{argmax}}\, \alpha(\tilde{x}; \mathcal{D}_n)$$

where $\mathcal{D}_n = \{(x_i, y_i) : i = 0, \ldots, n\}$, and then rounding it to the nearest feasible value $x \in \mathcal{X}$.
On the contrary in LSBO Algorithm we evaluate the chosen acquisition function at a discrete finite set of possible configurations and then take the maximum among them. This set of points is defined as the neighbourhood of the current solution $current_x$. The choice of a proper neighbourhood is important because it affects the exploration/exploitation of the search space and as a consequence the general algorithm performance. The neighbourhood definition is implemented in our algorithm by the function *GraphOfTentatives(x,diameter)* where x represents the current solution and *diameter* indicates the maximum allowed L_1-distance between a neighbour and $current_x$: e.g. if $diameter = 2$ in the neighbourhood we find all the configurations with distance at most 2 from the current solution,

so supposing to work with 3 variables and that $current_x = [0, 0, 0]$ then both $[1, 1, 0]$ and $[2, 0, 0]$ are possible neighbours. Changing the parameter *diameter* we obtain different search spaces.
Since we have defined the neighbourhood of interest, we have to evaluate the acquisition function at all its points and choose the one which achieve the maximum. This is carried out by *next_config* function in our algorithm.

Algorithm 7: next_config function for LSBO Algorithm

next_config($current_x, diameter$):
$neigh \leftarrow GraphOfTentatives(current_x, diameter)$
for *(x **in** elements in the neigh)* **do**
  Evaluate the acqusition function α at x
end
$next\_x \leftarrow \text{argmax}(\alpha(x)\ evaluations)$
Return $next\_x$

The described modification made on BO Algorithm represents the main improvement provided by our LSBO Algorithm. To design it we take inspiration from **Local Search Metaheuristics** where:

- the *diameter* of the neighbourhood must be chosen as a trade-off between the quality of the solutions reached thanks to better exploration and the consequent computational complexity.
- Best Improvement is adopted as selection strategy.

The main steps of the LSBO Algorithm are summarized in Algorithm 8.

Algorithm 8: LSBO Algorithm

$x_{train} \leftarrow n_{train}$ randomly selected training points
$y_{train} \leftarrow$ Evaluate the function f at x_{train}
Train the GP model on (x_{train}, y_{train})
$current_x \leftarrow$ current best configuration
for $k < n_{iterations}$ **do**
  Generate $current_x$ neighbourhood $neigh$
  $x_{next} \leftarrow \text{argmax}_{neigh}(acq.function)$
  $y_{next} \leftarrow$ Evaluate f at x_{next}
  Fit and update model and data with (x_{next}, y_{next})
  $current_x \leftarrow x_{next}$
end
Return configuration x which maximizes the objective function f

Since we suppose that no initial training dataset has been provided, the algorithm is characterized by an initial training phase in which we randomly select n_{train} configurations x_{train}, we evaluate the black-box function f at them to obtain y_{train} and finally we train the Gaussian Process on the couple (x_{train}, y_{train}).
Before starting algorithm iterations we set as starting point the current best configuration, selected among the ones evaluated in training phase. Then, at each iteration, we generate the neighbourhood of the current solution through *GraphOfTentatives* function and we use *next_config* function to select the next to evaluate point. The model is updated with the couple $(next_x, f(next_x))$ and $next_x$ becomes the starting point for the following iteration.
The procedure is iterated for a prefixed number of times $n_{iterations}$.
To avoid waste of computational time, before evaluating the function f at the new candidate solution, we check if it has already been evaluated in the past, in that case we skip to the next iteration. Finally the best configuration is selected as the one which maximizes the objective function.

5.3 GBO Algorithm

The second algorithm we introduce is the **Genetic Bayesian Optimization** (**GBO**). We know that a proper space exploration is a fundamental component in Optimization Algorithms. In this case to improve it we base on both the grid structure of the configuration space and the graph structure of the variables to take into account the correlation among them.
As we have done for the LSBO Algorithm we keep the general structure of the Bayesian Optimization with Gaussian Process surrogate model and EI / UCB acquisition function. Also in this case our contribution occurs in the choice of the next to evaluate configuration, for which we introduce some notions about Genetic Algorithms. As before we don't want the acquisition function to be evaluated and maximized on a continuous domain region, thus we have to define a proper neighbourhood. In LSBO Algorithm, based on Local Search theory and so on Single-Solution Based Metaheuristics, it was simply generated starting from a single point $current_x$. Instead in GBO Algorithm, based on GA and so on Population-Based Metaheuristics, we work, at each step, with a subset of parents on which we apply some variation operators to generate offspring. All these individuals build the next generation which represents in this case the set of configurations at which we will evaluate the acquisition function to choose then the next to evaluate point.

We describe at first the method used to select the configurations that will be parents of the next generation (**selection method**). To do this we have to identify a fitness function to evaluate the quality of a given solution, in fact, as known from theory, "*the better is an individual, the higher is the chance of being parent*". Given that we consider a general problem, for the sake of simplicity, we assume the fitness function to coincide with the objective function f.
As selection method we focus on a nearly **elitist strategy**. We select the parents as follows:

- we take the best $best\_k$ configurations from the already evaluated ones in terms of fitness;
- one configuration randomly generated;
- the current solution $current_x$.

To avoid premature convergence into local optima, we always consider the current solution and a randomly generated configuration in order to guarantee a wider exploration of the space. In our simulations we focus on tuning $best\_k$ parameter considering a trade off between evaluation time and algorithm performance and finally we set it to $best\_k = 10$.

The selected configurations are the parents of the next generation and produce the offspring (**reproduction phase**) thanks to the application of mutation and recombination operators.

Mutation. Mutation operators are unary operators which work on one single individual and that can affect only one gene. From theory we know that they must respect the properties of *ergodicity* (every solution of the search space should be reached), *validity* (the solutions produced must be valid) and *locality* (the mutation should produce a minimal change). In our case the mutation applied to a selected parent is the same that we describe in the generation of solution neighbourhood for the LSBO Algorithm by the function *GraphOfTentatives*. So given a parent all the children generated by the mutation operator are the configurations which are distant only one from the $current_x$ in the L_1 metric.

Recombination or Crossover. Crossover operators work instead with two parents to generate the offspring and have the peculiarity to preserve some of their characteristics or schemata. When we define a crossover operator it has to respect the properties of *validity* (the solutions must be valid) and *heritability*

(some genetic material from both the parents should be inherited). We work in particular with a **n-Point crossover**, in which n crossover sites are chosen (e.g. if we consider $A|BCD|E$ and $a|bcd|e$ as parents, the offspring will be $A|bcd|E$, $a|BCD|e$). The crossover sites are not chosen randomly but take account of the graph structure associated to problem variables. The basic idea is to merge together the nodes of the graph with high correlation so that they create a schemata that must be inherited by the offspring generation not to lose the quality improvements, provided by this one, cause to space exploration. The idea is implemented in few steps:

- We generate a transformed weight matrix $W' \in \mathbb{R}_+^{\mathcal{V}\times\mathcal{V}}$ such that

$$W'_{ij} = \begin{cases} W_{ij} & \text{if } W_{ij} \geq C \\ 0 & \text{if } W_{ij} < C \end{cases} \qquad \forall i, j \in \{1, \dots, d\}$$

  where $C \in \mathbb{R}_+$ is the chosen threshold for correlation.

- We define the connected components of the new graph $\mathcal{C}' = (\mathcal{V}, \mathcal{E}, W')$, that are the maximal subsets of nodes $\mathcal{V}_1, \dots, \mathcal{V}_h$ of the node set $\mathcal{V}$ such that for every pair of nodes (i, j) in the same connected component there exists a path from i to j. Then we collapse all the nodes belonging to the same connected component in a 'supernode'. Each of these supernodes represents a building block that we want to preserve.

Therefore, for each pair of parents, the children are generated taking one supernode at a time and switching the corresponding schemata (traditional 2-point crossover strategy). A simple graphical example is represented in [Fig. 5.2].
The **neighbourhood** of our problem coincides with the new generation made by all the children generated both from mutation and crossover operators. The parents are completely replaced by the offspring and so we are in the case of a generational replacement strategy.
The next to evaluate point x_{next} is chosen as the neighbour which maximizes the acquisition function $\alpha(x)$.
Both the steps described, the neighbourhood generation and the next candidate solution selection, are implemented in our algorithm in *next_config* function. Apart from this, the general algorithm for GBO coincides with the pseudo-code described in Algorithm 8 for LSBO.
Going more into detail we have to highlight that we implement two versions of the *next_config* function. The first one, summarized in Algorithm 9, applies at each algorithm iteration, all the steps previously described for the neighbourhood

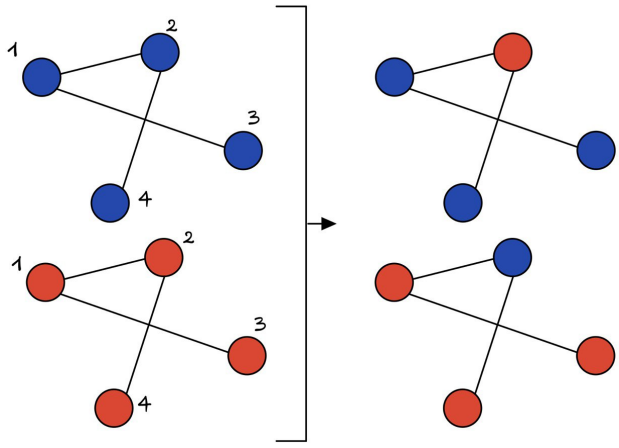


Figure 5.2: Graphical representation of 2-point crossover strategy applied on the graph $\mathcal{C}'$. In this examples the two children generated by the recombination applied on supernode 2.

generation. It requires many acquisition function evaluations and so it is more expensive with respect to the LSBO Algorithm.

Algorithm 9: next_config function - All Genetic

next_config($current_x, diameter, best_k$):
$parents \leftarrow$ ($best_k$ configurations, $current_x$, random configuration)
Mutation: $children_{mut} \leftarrow GraphOfTentatives(parents, diameter)$
for *(each pair* (x_1, x_2) ***in*** *parents* **do**
  | Generate $children_{cross}$ through crossover
end
$neigh \leftarrow (children_{mut}, children_{cross})$
for *(*x ***in*** *elements in the neigh)* **do**
  | Evaluate the acqusition function α at x
end
$next\_x \leftarrow$ argmax($\alpha(x)$ *evaluations*)
Return $next\_x$

The second one, summarized in Algorithm 10, alternates the LSBO and GBO strategies of neighbourhood generation. Since the Genetic method implies a wider space exploration and a consequentially higher computational cost, we applies it only every 10 steps. All the other times we use the Local Search method which simply analyses the neighbourhood of the current solution with $diameter = 1$.

Algorithm 10: next_config function - Partially Genetic

next_config($current_x, diameter, best_k, iteration$):
Return *next_x* **if** *iteration is a multiple of 10* **then**
  $parents \leftarrow$ ($best_k$ configurations, $current_x$, random configuration)
  Mutation: $children_{mut} \leftarrow GraphOfTentatives(parents, diameter)$
  for *(each pair* (x_1, x_2) ***in*** *parents)* **do**
    Generate $children_{cross}$ through crossover
  end
  $neigh \leftarrow (children_{mut}, children_{cross})$
end
else
  $neigh \leftarrow GraphOfTentatives(current_x, diameter)$
end
for *(x* ***in*** *elements in the neigh)* **do**
  Evaluate the acqusition function α at x
end
$next\_x \leftarrow \text{argmax}(\alpha(x)\ evaluations)$

5.4 Application: Binary Quadratic Programming

We first apply our methods on a Binary Quadratic Programming (BQP) problem. The aim of BQP problem is to maximize a quadratic function with l_1 regularization:

$$f(x) = x^T Q x - \lambda \|x\|_1$$

over $\mathcal{X} = \{0, 1\}^d$ with d number of variables.
The matrix $Q \in \mathbb{R}^{d \times d}$ is a measure of correlation among variables, so it must be a PSD matrix. If we infer a graph structure on the problem variables we deduce that such graph will be indirect if the matrix is symmetric PDS, direct otherwise. The parameter λ represents a penalization term.
Given that the problem search space is quite small (2^d points, with d number of variables), we can easily solve the real problem to find the real expected optimum. In order to make proper comparisons we implement:

- Standard Naive BO algorithm with EI acquisition function, chosen as reference benchmark;
- LSBO Algorithm with diameter equal to 1 and UCB acquisition function;
- LSBO Algorithm with diameter equal to 2 and UCB acquisition function;

- GBO Algorithm with All Genetic Steps and EI acquisition function;
- GBO Algorithm with Partially Genetic Steps and EI acquisition function.

The selection of algorithm parameters is not arbitrary, but it is the result of wider tuning and analysis led during experimental phase. Given that BQP wants to be just a theoretical example, we report here just the most significant results. In the simulations we made we considered $n_{vars} = 12$ binary variables, an initial training set of 20 configurations, randomly chosen, and we fixed the maximum number of algorithm iterations to $n_{iter} = 50$. The Q matrix of the problem is generated as a random symmetric positive-definite matrix, built using *scikit-learn* Python library. We recall that the Q_{ij} element is referred to the correlation between variables i and j and it can assume values in the interval $[-1, 1]$. In order to build the transformed weight matrix W' required by the GBO algorithm, we choose as threshold $C = 0$ to eliminate the edges of the graph associated to negatively correlated variables. Then the supernodes are designed extracting the connected components of the new graph, thanks to *scipy* Python library.
The results displayed refer to 100 simulations of the same algorithm, in order to make some proper statistical analysis. The real optimum is reached by the selected algorithms with the following frequency:

Naive BO	81/100
LSBO diameter 1	99/100
LSBO diameter 2	99/100
GBO Partially Genetic	94/100
GBO All Genetic	100/100

The plot in [Fig. 5.3] displays the **optimum reached** by the algorithms when the real optimum, indicated by the dashed blue line, is not found. Here we highlight that when the LSBO misses the real optimum the error is higher than in the other cases (46.8%), but the frequency is extremely low, equal to 1 out of 100. On the other hand the average error reached by the standard NaiveBO is about 1.9%, while the one of GBO with Partially Genetic steps is about 0.55%. Another important parameter is given by the **number of iterations** required by the algorithm to reach the optimum. The objective function of the problem is supposed to be an expensive to evaluate black-box function and so it is essential to minimize the number of evaluations. The results are shown in [Fig. 5.4]. Looking to this performance parameter the best algorithms are the LSBO and secondly the GBO with all genetic steps. On the contrary the GBO with Partially Genetic steps reaches results comparable to the ones of standard NaiveBO.

Finally we have to compare the algorithms in terms of **computational time** required. The benchmark is given by the NaiveBO characterized by an average computational time of 2.97s (the results refer to 50 iterations of the algorithm trained on a training set of 20 data points). The LSBO algorithm, with diameter equal either to 1 or 2, performs better than the NaiveBO, with an average time of 2.75s. The GBO is characterized by a wider space exploration compared to the other algorithms and it is reflected on the average computational time, equal to 6.38s for the Partially Genetic and to 36.19s for the All Genetic.

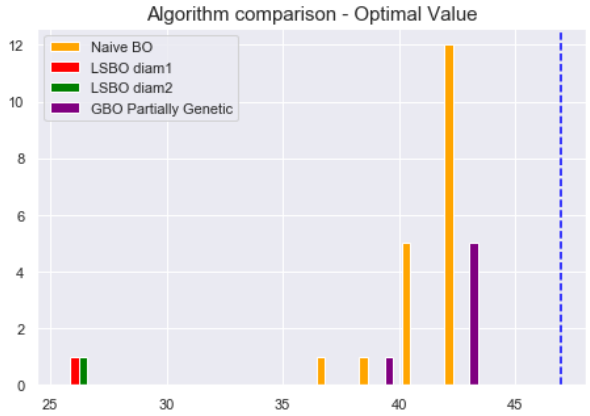


Figure 5.3: Best optimal value reached by the algorithms when the real optimum is not found. The dashed blue line indicated the real problem optimum.

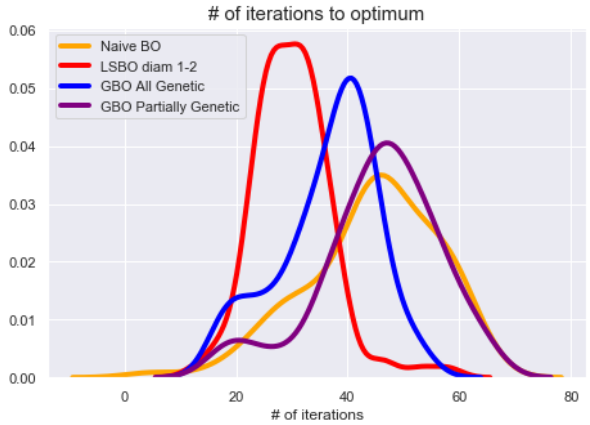


Figure 5.4: Number of iterations, i.e. function evaluations, to reach the optimum.

Chapter 6

Case Study: Mobile Networks

Mobile networks provide nowadays much more sophisticated services and are becoming more and more complex. Their environment changes constantly and dynamically due to different reasons [7] like exponential growth of traffic over the years, variable traffic load, dynamical radio propagation conditions, malfunction/insertion/deletion of stations, errors made in planning phase, temporal and spatial users distribution. Networks must be able to be adapted to all these circumstances in an efficient way. The traditional approach is based on manual intervention led by specialists on the network which is a highly complex and time and cost consuming task. The aim of our analysis is to provide an automated optimization solution that, relying on real-time measures and key performance indicators, proposes the optimal network configuration.

6.1 General Mobile Network Structure

The mobile network is modelled dividing the territory of interest into elemental areas called cells which are served by a particular base station [Fig.6.1]. For simplicity reasons, each cell can be geometrically approximated to a hexagon, even if it is not always like that due to inconsistent radio propagation characteristics and uneven environmental landscape. In general not all the cells cover the same area: smaller cells are necessary to cover areas with high density of users, while in areas with lower density of users cells with larger radius could be used.
As shown in [Fig. 6.1] each base station is equipped with three directional

antennas and so it is able to cover a sector made by three cells. This strategy, called sectorization, allows to reduce the number of needed base stations in the network resulting in a reduction of costs.

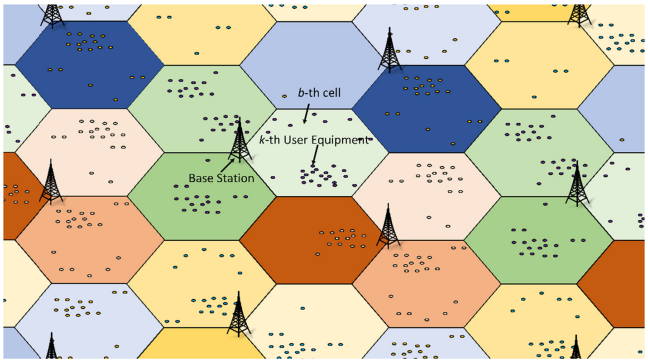


Figure 6.1: Mobile network structure.[7]

The design of each antenna of the base station is important because it is responsible for transforming the energy from the circuit into radiation energy. The tilt of the antenna influences the direction of the radiated electromagnetic energy and therefore it can be used to optimize coverage and capacity of the network depending on its needs in a particular moment. The antenna tilt, represented in [Fig. 6.2] is defined as the angle between the direction of the main beam of the antenna pattern and the horizon. Another important factor is represented by the half-power beam width(HPBW). As highlighted in the figure it is the angular separation in which the magnitude of the radiation pattern decreases by 50% (or -3dB) from the peak of the main beam. In other words, the beam width is the area where most of the power is radiated.

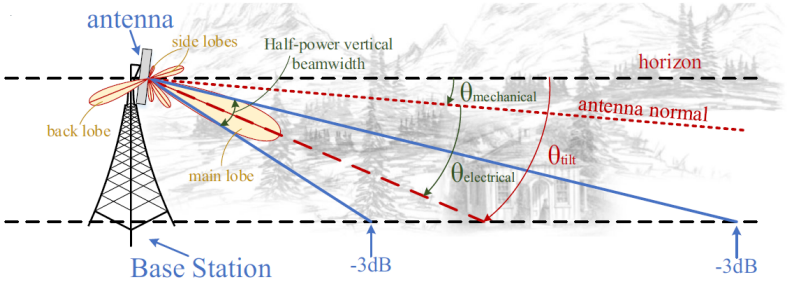


Figure 6.2: Graphical representation of a base station and one of its antenna. The main lobe of the antenna radiation pattern defines the antenna tilt. [7]

There are two possible types of tilts simultaneously present: the **mechanical tilt** and the **electrical tilt**. The total tilt angle is defined as the combination of the two:

$$\Theta_{tilt} = \Theta_{mechanical} + \Theta_{electrical}$$

For the mechanical tilt antenna elements are physically directed towards the ground. Instead, for the electrical tilt the modification is obtained changing the signal phase of each element of the antenna.

(a) Mechanical Tilt (b) Electrical Tilt

Figure 6.3: Mechanical and electrical tilt modification. In the mechanical tilt the antenna elements are physically modified, while in electrical tilt we act only on the signal phase of the different elements. [Image taken from telecomHall]

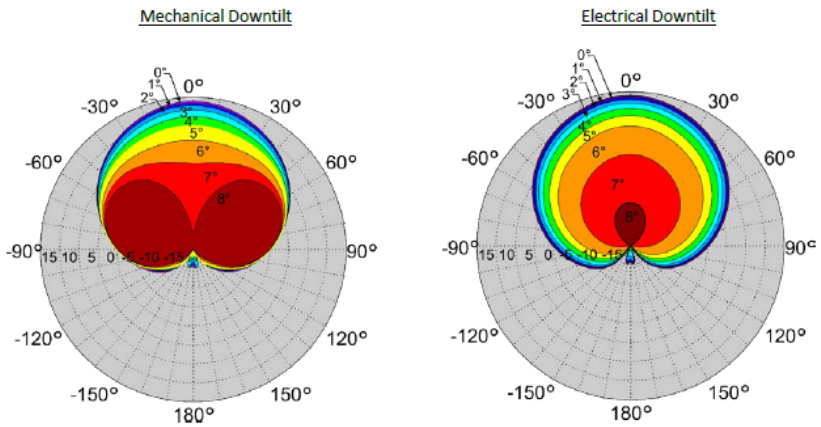


Figure 6.4: Radiation pattern for mechanical and electrical tilt. In particular, the picture shows the pattern in the horizontal (or azimuthal) plane of an antenna with an azimuth HPBW of 65° and an elevation HPBW of 7°. [14]

We can apply two kind of actions on the antenna: it can be up-tilted, which

means that the change in the angle is negative, or down-tilted towards the ground, which correspond to a positive change in the angle. Adjusting the mechanical tilt requires a visit to the site, which is inefficient both in terms of cost and time. In addition, another drawback of the mechanical down-tilt is that coverage isn't reduced uniformly, but it is reduced more at bore sight and less towards angles, as explained in the radiation pattern. For these reasons usually the mechanical tilt is set during installation and setup phase. On the contrary the electrical tilt could be adjusted periodically through remote operation and therefore we focus the attention on it.

6.2 Case Study Problem Description

In the thesis case study we focus the attention on a Mobile Network system located in an area of interest proposed by TIM group which involves a city of medium dimensions and its surrounding suburbs. Here are located 63 cells but only 12 of them are adjustable, due to some peculiarities of the case study. Each cell is associated to an antenna which can take 5 different values related to the 5 admissible values of its electrical tilt. Thus our problem consists of 12 variables and the domain is a discrete set of 5^{12} points.
The goal of the optimization problem is to find the best configuration for the adjustable cells in terms of tilt values. The quality of a configuration is computed considering a set of parameters identified by a group of telecommunications experts. These parameters values are provided by a simulator of the model, supplied by the TIM group, that we will consider as an expensive black-box function. In the following work we will refer to this simulator as **Oracle**.
All those parameters are then properly combined to define the **target** or **reward function** of the problem.
A more detailed analysis should take into account also some supplementary information related to the **interference between cells** which has not been taken into account in the reward function definition. From the theory on antennas we know that their working principle is based on electromagnetic radiations. If there is an interaction between two cells their radio-electrical coverage could be mutually influenced by a mutation of one of the two tilt configurations. An abstract of the summary of electromagnetic adjacencies provided by the firm is shown in [Fig. 6.5].
The goal of our work is to provide optimal coverage and capacity for the radio network of interest. As described in Section 6.1, the configuration of antennas may become obsolete over time due to different reasons, so it is necessary to

Reference	Adjacent	Distance (Km)	NumPixelsBS [#]	NumPixelsAdjacent [#]	PercPixelsAdjacent [%]
AA01H1	AA01H2	0	579	574	99.136
XX44H2	XY44H1	0	897	882	98.328
AZ23H1	SW23H2	0	292	283	96.918
AZ90R3	HU91Y2	0	5822	5201	89.334
SW23H2	JU12E1	0	1299	1145	88.145
XC76A1	JF64K3	0	4738	2510	52.976
AA01H2	AA01H1	0	7249	3825	52.766

Figure 6.5: Short abstract of the summary of electromagnetic adjacencies between cells of the area of interest.

adjust it. In particular, for each antenna we have to choose among the admissible values of the electrical tilt the one which provides the best improvement in the reward function. To evaluate the quality of a solution we have to question the Oracle about the configuration of interest. Since this is an expensive to evaluate function we have to minimize the number of algorithm calls, ensuring a satisfiable algorithm performance. Given these conditions, a proper choice would be to solve the problem using a Bayesian Optimization strategy for discrete variable problems. Furthermore we are in the right setting to apply our proposed methods.

6.3 Results

We apply on the case study problem both a state-of-the-art method, used as benchmark, and our proposed ones (LSBO and GBO).
As known it has not been provided a dataset on which training our model, so we fix 100 iterations for training, we train the model using them and then start optimization steps. The stopping criterion used in all the algorithms tested is the achievement of the maximum number of steps n_next, setted to 400.
We highlight that the black-box function is represented by the call of the Oracle. In the simulations we work with an artificial Oracle simulated through Random Forest Regressor by a given dataset. Despite that in real life application each Oracle call requires about 1 minute to give the result for the evaluation of the proposed configuration and so it represents our bottleneck.
Even if we work with a black-box function our simulations are created so that the optimum is given a priori in order to be able to analyse the goodness of our algorithm and make comparisons. To do this we consider in particular:

- the best reward (objective function value) obtained among the algorithm iterations;

- the configuration which provide the best reward and the number of tilts which differ from the optimal configuration given;
- the effective number of Oracle calls which provides a measure of expensiveness of the algorithm;
- the number of iterations needed to reach the optimum, from which we deduce if the algorithm got stuck on a local optimum.

In order to make some proper statistical analysis we will consider 200 iterations of the same algorithm each time. It is used the same randomly generated dataset of 200 iterations of 100 training configurations each for all the algorithms tested to make comparisons possible.

6.3.1 Naive BO Algorithm

The state-of-the-art method we focus on is the Naive BO Algorithm, the simplest approach used to adapt BO Algorithms to discrete variables. We recall that it consists in rounding the real value obtained from the acquisition function maximization to the closest admissible integer before evaluation. Despite the simplicity of this algorithm its main drawback is linked to the fact that generally it leads to sub-optimal solutions and to repetitions of already evaluated points. In the definition of the Bayesian Optimization method for Naive BO Algorithm we build:

- a **Gaussian Process** referred in the code as *CustomGP*. It is made by an ExactGP from GPyTorch models with constant mean and squared exponential/Matern kernel.
- the **acquisition function** imported by BoTorch library, notably we will adopt the Expected Improvement (EI). To select the next point to evaluate we use the *optimize_acqf* function with bounds $[0, 4]$ for all the 12 variables. The new point is then rounded to the nearest integer before model evaluation.

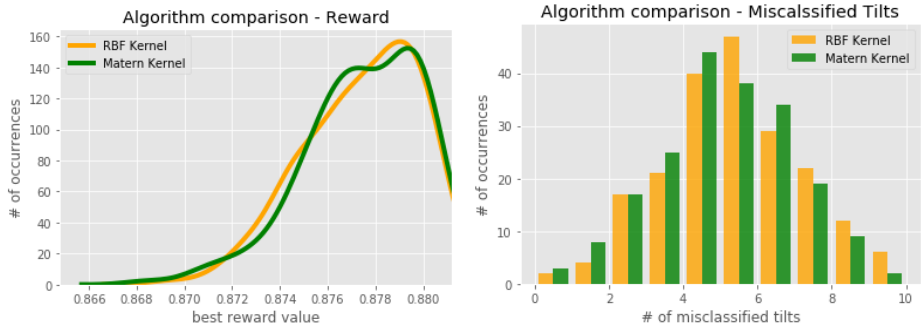
We simulate the Naive BO algorithm on the case study both with RBF (squared exponential) kernel

$$k(x, x') = exp\left(-\frac{\|x - x'\|_2^2}{2l^2}\right)$$

where $\|\cdot\|_2^2$ is the Euclidean distance and l is an hyperparameter known as characteristic length scale, and Matern kernel with $\nu = \frac{5}{2}$

$$k_{\nu=5/2}(x, x') = \left(1 + \frac{\|x - x'\|\sqrt{5}}{l} + \frac{(x - x')^2}{3l^2}\right) e^{-\frac{\|x-x'\|\sqrt{5}}{l}}.$$

The results comparison is displayed in [Fig.6.6] which represents respectively the density distribution of the best reward reached and the number of misclassified tilts. The values displayed are obtained taking the values associated to best result reached at each algorithm iteration.



(a) (b)

Figure 6.6: Comparison between the results obtained for the NaiveBo Algorithm with RBF and Matern Kernel. Figure (a) and (b) are referred respectively to best reward distribution and to the number of misclassified tilts.

From these plots we observe that RBF tends to perform a bit better in terms of best reward reached but in the meanwhile it performs worst than Matern Kernel in terms of number of misclassified tilts. This can be justified by the topological structure of the search space.
We can summarized these data using quantiles as described in Table 6.1 and 6.2, respectively for RBF and Matern kernel.

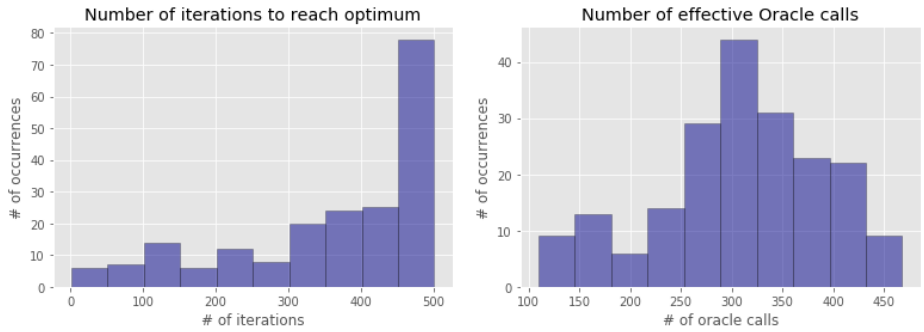
	0.25 quantile	**0.5 quantile**	**0.75 quantile**
best reward	0.87591	0.87778	0.87945
# of misclassified tilts	4	5	6

Table 6.1: Summary of the quantiles associated to best reward distribution and number of misclassified tilts in Naive BO Algorithm with RBF kernel.

	0.25 quantile	**0.5 quantile**	**0.75 quantile**
best reward	0.87589	0.87770	0.87951
# of misclassified tilts	3	5	6

Table 6.2: Summary of the quantiles associated to best reward distribution and number of misclassified tilts in Naive BO Algorithm with Matern kernel.

Recalling that the best optimum reward value is assumed to be equal to 0.88132, the average error rate reached is equal to $avg(err) = 0.00391$ ($min = 0.00013$, $max = 0.01207$) for the RBF kernel and to $avg(err) = 0.00387$ ($min = 0.00013$, $max = 0.01041$) for the Matern kernel.
In [Fig.6.7] are displayed the results related to the number of iterations needed to reach the optimum and the number of effective Oracle calls for the Naive BO with Matern kernel. These two measure should help us in the evaluation of the algorithm. We deduce that in general NaiveBO doesn't stuck in a local optimum at early stages but despite that it doesn't succeed in reaching the global optimum.



(a) (b)

Figure 6.7: Results of the simulation for 200 iteration of the Naive Bo Algorithm with Matern kernel. In (a) it is shown the number of iterations needed to reach optimum, in (b) the number of effective Oracle calls.

6.3.2 Other Existing Algorithms

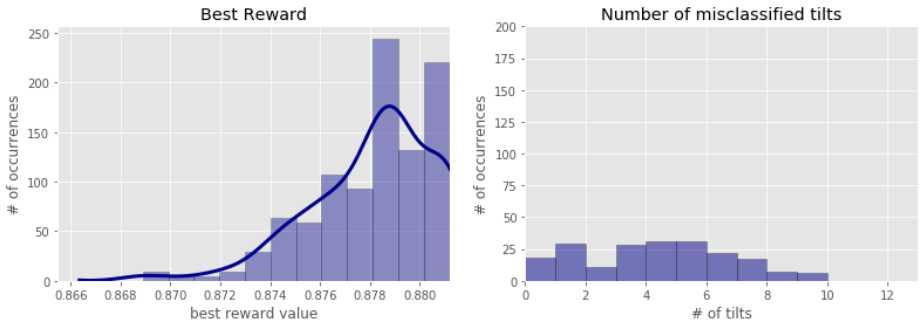
In chapter 3.2 many other methods have been enunciated. Some of them solve the problem of working with discrete variable changing the surrogate model of the method and working with Tree-based models. This is the case of the TPE, which works with a tree-based structure, and of SMAC, based on Random

Forests, i.e. a proper combination of trees. Despite that in our case study we don't apply any of them. The main reasons are related to their most relevant drawbacks, described in [4], [5].

- Although they need less time to be evaluated, they require many iterations to converge to an optimal solution. Given that the expensive black-box function represent our bottleneck, this is a non negligible limit.
- In addition they are generally less efficient in search with respect to GP. Their lack of accuracy is due to the fact that they don't model interactions between variables. In particular TPE ignores the recently proposed points and relied on the stochasticity of underling functions. In our proposed methods, LSBO and GBO, the correlation among variables cover a central role and so the comparison with those methods would be inefficient.

6.3.3 LSBO Algorithm

We consider now the proposed LSBO Algorithm. The following simulations take into account different parameters settings involving both the parameter *diameter*, characteristic of the algorithm designed, and the components of the BO structure (GP model and acquisition function).
Firstly we consider a neighbourhood with diameter equal to 1, i.e. neighbours could differ at most for 1 tilt from the current solution, and we compare the results obtained with the two acquisition function selected, EI and UCB.



(a) (b)

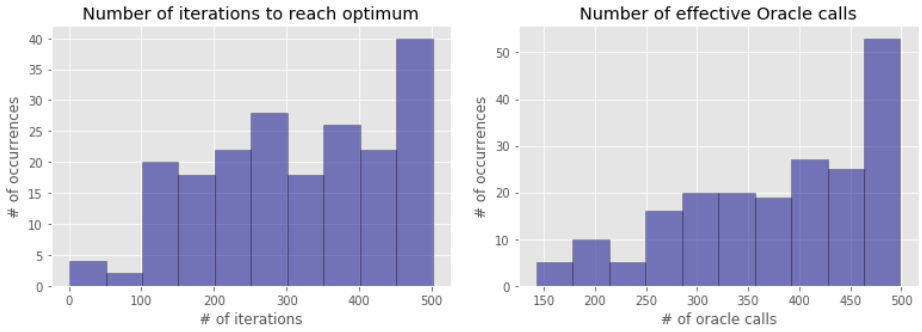
Figure 6.8: Results of the simulation for the LSBO Algorithm with $diam = 1$. The acquisition function used is the Expected Improvement. In (a) it is shown the best reward distribution, in (b) the number of misclassified tilts with respect to the optimum configuration known a priori.

In [Fig. 6.8] are represented respectively the best reward distribution and the histogram of the number of misclassified tilts. The results can be summarized evaluating the quantiles like displayed in Table 6.3.

	0.25 quantile	**0.5 quantile**	**0.75 quantile**
best reward	0.87663	0.87852	0.87975
# of misclassified tilts	2	4	6

Table 6.3: Summary of the quantiles associated to best reward distribution and number of miscalssified tilts in LSBO Algorithm with $diam = 1$ and EI acquisition function.

Recalling that the maximum reward is equal to 0.88132, in this case we obtain an average error equal to $avg(err) = 0.00319$ ($min = 0.00015$, $max = 0.01242$). Compared to the results obtained for the Naive BO algorithm, summarized in tables 6.1 and 6.2, we highlight that we reach an improvement in terms of quantiles and average error rate, whereas the performance seems to be quite worst on the queues (max error rate: Naive BO = 0.01207 (RBF) / 0.01041 (Matern) vs Single Step = 0.01242).



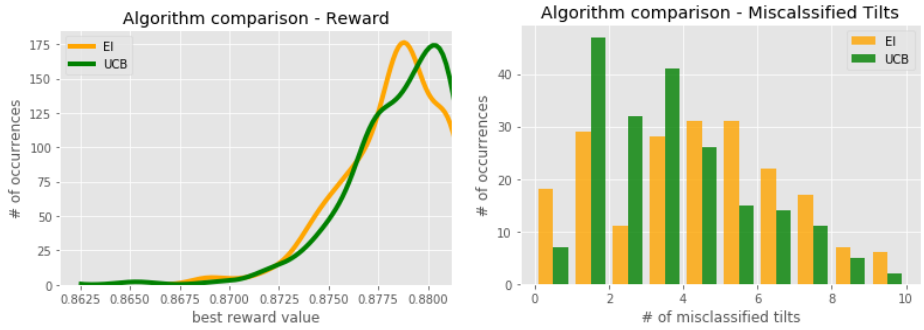
(a) (b)

Figure 6.9: Results of the simulation for the LSBO Algorithm with $diam = 1$. The acquisition function used is the Expected Improvement. In (a) it is shown the number of iterations needed to reach optimum, in (b) the number of effective Oracle calls.

If we consider now the results shown in [Fig. 6.9], looking to the number of effective Oracle calls displayed in sub-figure (b), we deduce that we have a wider space exploration, but in the meanwhile sub-figure (a) highlights that there is a non-negligible probability to reach the optimum[1] in fewer iterations.

[1] We refer here to the optimum reached by the algorithm, not to the real global optimum.

We try to modify the chosen acquisition function and compare the results obtained for the Expected Improvement and the Upper Confidence Bound.

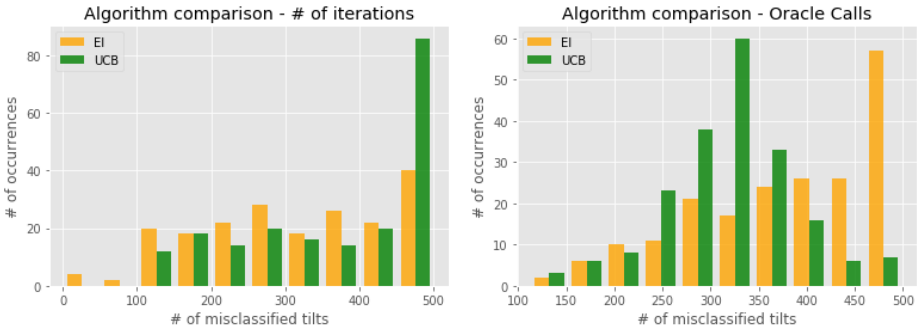


(a) (b)

Figure 6.10: Comparison between the results obtained for the LSBO Algorithm with $diam = 1$ and EI(*orange*) / UCB(*green*) acquisition function. Figure (a) and (b) are referred respectively to the best reward distribution and to the number of misclassified tilts.

The UCB performs better both in terms of best reward reached and number of misclassified tilts, as highlighted in [Fig. 6.10]. Even if the EI finds the exact optimal configuration with a higher frequency than the UCB, this is not reflected in terms of best reward distribution. This could be due to multimodality[2] of the black-box function.
Looking to [Fig. 6.11] we observe that UCB acquisition function (setted with parameter $\beta = 1$) leads to a limited space exploration compared to EI, while the number of iterations needed to reach optimum is condensed on higher values. This means that during algorithm iterations the UCB tries to evaluate an already evaluated point many times, but despite that the results reached are better as seen in [Fig. 6.10].
Another parameter that we could try to work on is the *diameter* which influences the exploration rate. From theory we know that use an higher diameter value is better in terms of exploration because it avoids to stuck in local optima but it also implies a higher evaluation time. At first we run the algorithm with EI acquisition function. The quantile results are summarized in Table 6.4.
The quantile results outperform the previous ones reaching an average error $avg(err) = 0.00205$ ($min = 0.00014$, $max = 0.00843$).
As before we try to change the acquisition function and compare the results

[2] A function is said *multimodal* if it has multiple local optima.



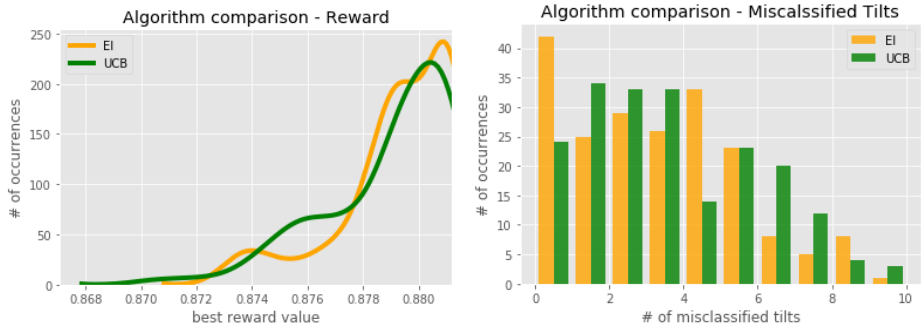
(a) (b)

Figure 6.11: Comparison between the results obtained for the LSBO Algorithm with $diam = 1$ and EI(*orange*) / UCB(*green*) acquisition function. Figure (a) and (b) are referred respectively to the number of iterations needed to reach optimum and to the effective number of Oracle calls.

	0.25 quantile	**0.5 quantile**	**0.75 quantile**
best reward	0.87859	0.87955	0.88112
# of misclassified tilts	1	3	4

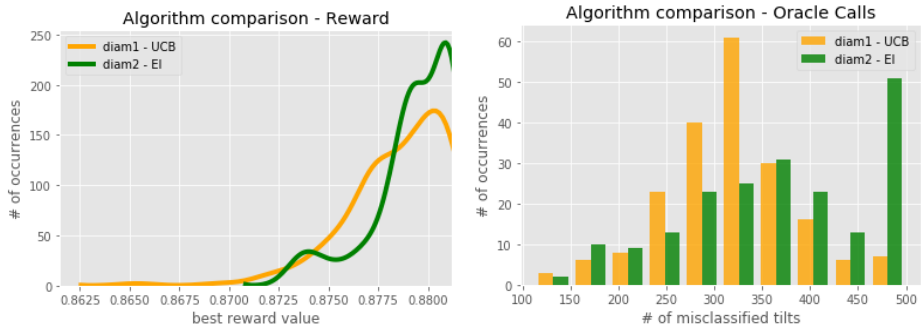
Table 6.4: Summary of the quantiles associated to best reward distribution and number of misclassified tilts in LSBO Algorithm with $diam = 2$ and EI acquisition function.

obtained in order to select the better parameters setting. In [Fig. 6.12] are displayed the results for the LSBO Algorithm with diameter 2 and acquisition function EI (orange) / UCB (green). In this case both in terms of distribution of reward and number of misclassified tilts EI could be evaluated as better than its opponent. Given the results obtained we choose as acquisition function the UBC in case of $diam = 1$ and the EI in case of $diam = 2$. In the comparison we focus especially on the distribution of best reward reached and on the number of Oracle call, that we expect to be higher in case of $diam = 2$ due to the wider space exploration guaranteed. The outcomes are shown in [Fig. 6.13]. The results highlight that parameter $diam = 2$ outdoes the performances obtain by the same algorithm with $diam = 1$. Even so, the main cons is related to the computational complexity of the algorithms: in terms of time per step the LSBO Algorithm with diameter $diam = 2$ is 2.83 times more expensive than the one with $diam = 1$ on average, as we can see in [Fig. 6.14].



(a) (b)

Figure 6.12: Comparison between the results obtained for the LSBO Algorithm with $diam = 2$ and EI(*orange*) / UCB(*green*) acquisition function. Figure (a) and (b) are referred respectively to the best reward distribution and to the number of misclassified tilts.



(a) (b)

Figure 6.13: Comparison between the results obtained for the LSBO Algorithm with $diam = 1$ and UCB acquisition function(*orange*) vs $diam = 2$ and EI acquisition function(*green*). Figure (a) and (b) are referred respectively to the best reward distribution and to the number of effective Oracle calls.

6.3.4 GBO Algorithm

We focus now on the second proposed algorithm, GBO Algorithm, and we make simulations for both its variants, the All Genetic and the Partially Genetic.

We choose as acquisition function the Expected Improvement which seems to achieve better results in case of wider space exploration.

Firstly we explain the results obtained through the All Genetic alternative. We

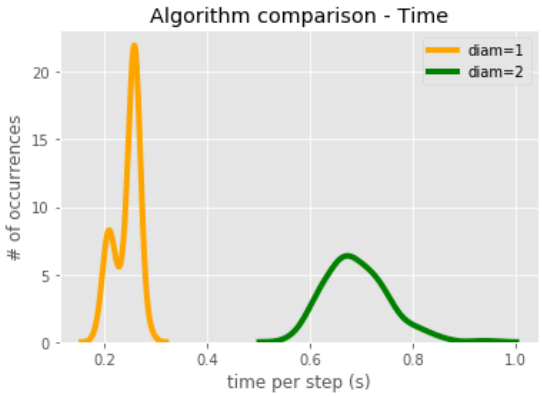


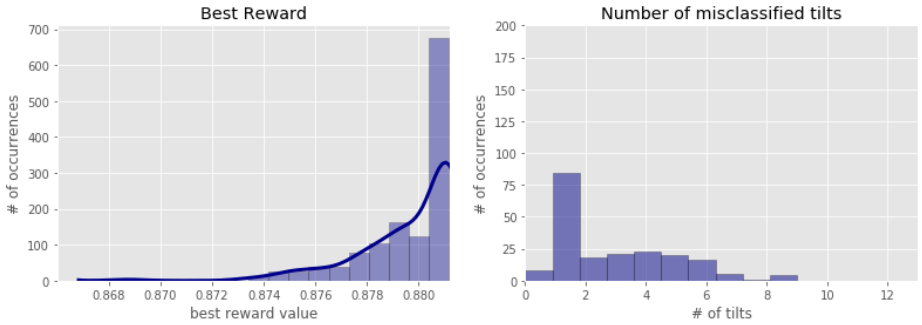
Figure 6.14: Comparison between the average time spent for each step by the LSBO Algorithm with $diam = 1$ (*orange*) and $diam = 2$ (*green*).

recall that is that case at each iteration the neighbourhood is generated applying mutations and recombinations on the selected parents. The weighted matrix W, used by the recombination operator to identify the cut sites, is referred in this case to the percentage of adjacency between cells and so to the electromagnetic mutual influence. In other words subsets of cells that highly influence each other are considered as building blocks in recombinations (i.e. supernodes of the graph associated to variables). The reason to use this strategy is linked to the fact that we would avoid to lose through wide exploration the progresses obtained in local exploitation on properly chosen subsets of variables. The results in terms of best reward distribution reached and number of misclassified tilts are displayed in [Fig. 6.15]. In Table 6.5 we summarize the corresponding quantiles.

	0.25 quantile	**0.5 quantile**	**0.75 quantile**
best reward	0.87881	0.88068	0.88117
# of misclassified tilts	1	2	4

Table 6.5: Summary of the quantiles associated to best reward distribution and number of misclassified tilts in GBO Algorithm with all genetic steps.

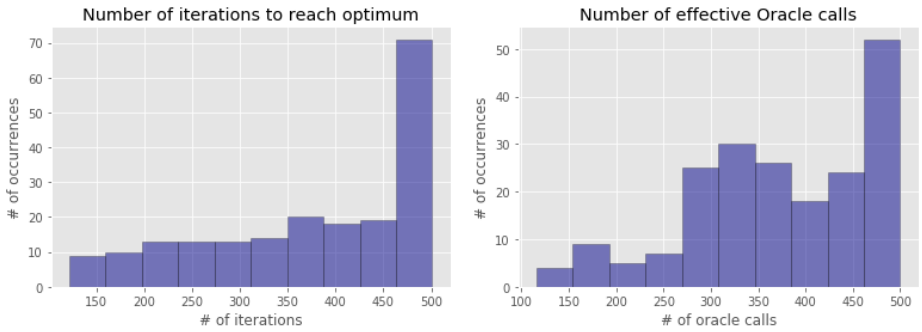
Recalling that the maximum reward is equal to 0.88132, in this case we obtain an average error equal to $avg(err) = 0.00157$ ($min = 0.00015$, $max = 0.00789$). Looking to the results associated to the number of iterations needed to reach the optimum and to the number of effective Oracle calls represented in [Fig. 6.16], we highlight that we have a non-negligible probability of reaching the optimum with fewer Oracle calls. Despite that the plot in 6.16a presents a pick corresponding



(a) (b)

Figure 6.15: Results of the simulation for the GBO Algorithm with all genetic steps. The acquisition function used is the Expected Improvement. In (a) it is shown the best reward distribution, in (b) the number of misclassified tilts with respect to the optimum configuration known a priori.

to the maximum number of available iterations. This indicates that with a fairly large frequency the candidate solution proposed by the algorithm at each step coincide with an already evaluated one, but this doesn't represent a real limit to space exploration and the optimum is reached making use of all the available iterations.



(a) (b)

Figure 6.16: Results of the simulation for the GBO Algorithm with all genetic steps. The acquisition function used is the Expected Improvement. In (a) it is shown the number of iterations needed to reach optimum, in (b) the number of effective Oracle calls.

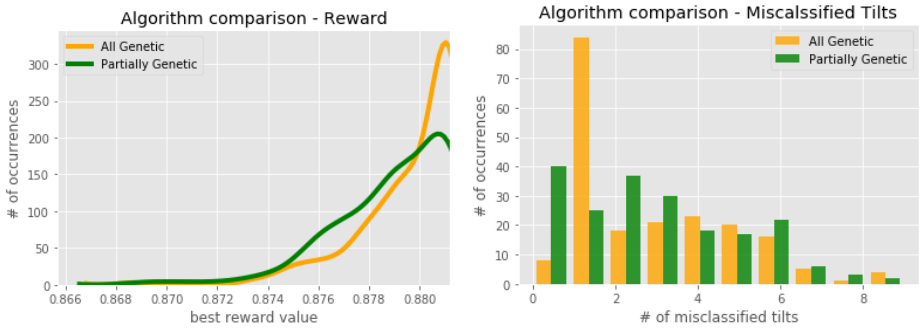
The second alternative tested is the GBO Algorithm with Partially Genetic strategy. In this case in the design of the neighbourhood we alternate the Local

Search and the Genetic metaheuristics, in particular we adopt the GA one out of ten times as widely described in chapter 5. The results in terms of quantile are summarized in Table 6.6.

	0.25 quantile	**0.5 quantile**	**0.75 quantile**
best reward	0.87770	0.87937	0.88099
# of misclassified tilts	1	2	4.25

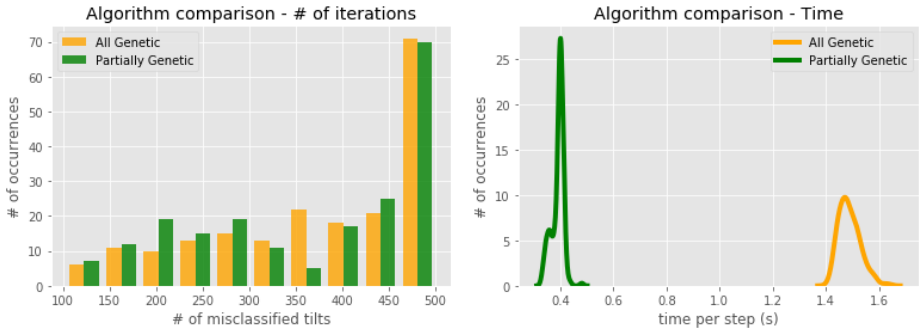
Table 6.6: Summary of the quantiles associated to best reward distribution and number of misclassified tilts in GBO Algorithm with partially genetic steps.

Related to the known optimum, we compute that the average error rate is in this case equal to $avg(err) = 0.00229(min = 0.00015, max = 0.01129)$. Comparing the results obtained for All Genetic and Partially Genetic alternatives, we observe that the performance of the Partially Genetic one is worst both in terms of distribution of best reward and of number of misclassified tilts [Fig. 6.17]. Figure 6.18 underlines no conspicuous difference in terms of number of iterations needed to reach the optimum, but the computational time spent for each algorithm iteration by All Genetic strategy is sensibly higher (3.80 times the one requested by the Partially Genetic alternative).



(a) (b)

Figure 6.17: Comparison between the results obtained for the GBO Algorithm with All Genetic steps(*orange*) vs Partially Genetic steps(*green*). Figure (a) and (b) are referred respectively to the best reward distribution and to the number of misclassified tilts.



(a) (b)

Figure 6.18: Comparison between the results obtained for the GBO Algorithm with All Genetic steps(*orange*) vs Partially Genetic steps(*green*). Figure (a) represents the number of iterations needed to reach optimum, while Figure (b) is associated to the computational time spent for each algorithm iteration.

6.3.5 Algorithms Comparisons

Given a problem of interest, choose the best algorithm is not so simple because it depends on multiple factors whose weight is submitted to business decisions. In the following section we report the main comparisons among the previously implemented algorithms. Their parameters are setted choosing for each the ones which guarantee the best performances. Thus we will analyse:

- Naive Bayesian Optimization with GP surrogate model, RBF squared exponential kernel and EI acquisition function;
- LSBO Algorithm with $diameter = 1$ and UCB acquisition function;
- LSBO Algorithm with $diameter = 2$ and EI acquisition function;
- GBO Algorithm with All Genetic strategy;
- GBO Algorithm with Partially Genetic strategy.

Firstly we focus on the **quality of the solutions** obtained, expressed by the distribution of best rewards reached by each algorithm [Fig. 6.19]. The GBO Algorithm with All Genetic strategy is the one which guarantee the best results in term of objective function. More in general all the implemented algorithms provide an improvement in the results with respect to the considered benchmark.

In [Fig. 6.20] are represented the boxplots associated to the best reached rewards

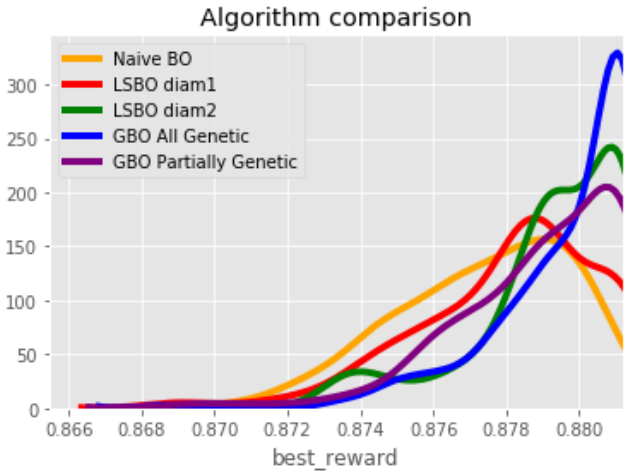


Figure 6.19: Comparison among the selected algorithms in terms of density distribution of best reached rewards.

distributions. From this we highlight that the best performing algorithms are the GBO with All Genetic strategy and the LSBO with diameter=2.
The simpler version of the LSBO (with diameter=1) has better performances then the standard Naive BO but in the meanwhile it may reach also worst values in the left queue.

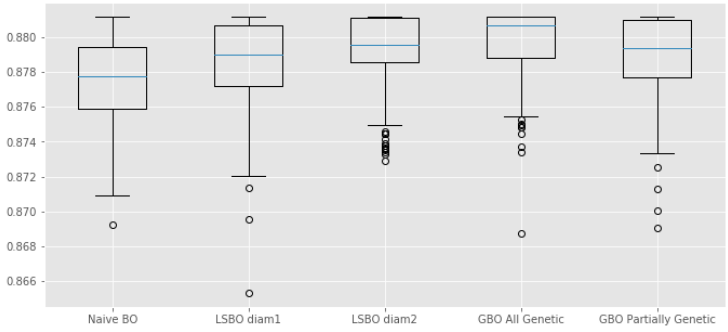


Figure 6.20: Boxplots associated to the best rewards reached by the selected algorithms.

In the analysis of solutions quality we focus only on the best reached rewards (i.e. the objective function value), whereas we ignore the results in terms of misclassified tilts given that, as previously observed, they are strongly influenced by the structure of the fitness space.

Furthermore, another parameter to take into account is the **execution time**, which potentially represents a bottleneck of the implemented algorithm. In the thesis case study is supposed that an Oracle call requires about 1 minute to be processed, so the average time per iteration can be considered negligible. In [Fig. 6.21] and in [Fig. 6.22] are represented respectively the distribution of the average time per iteration and the associated boxplots. The GBO with All Genetic steps, which is the one which guarantees the best performance, is the one with highest execution time. The other implemented algorithms instead outperform the standard Naive BO also in terms of time per iteration.

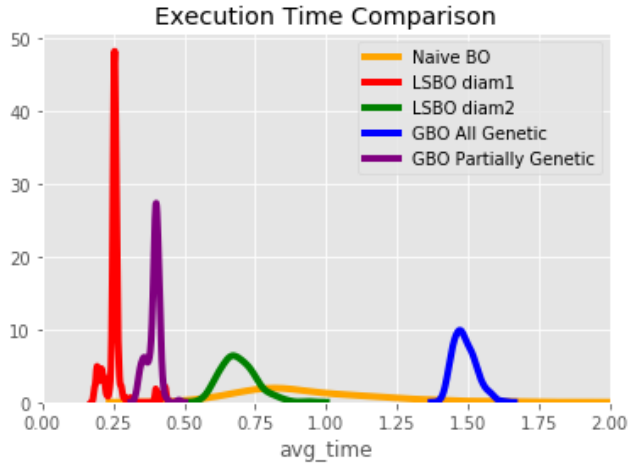


Figure 6.21: Comparisons among the selected algorithms in terms of execution time (average time per iteration).

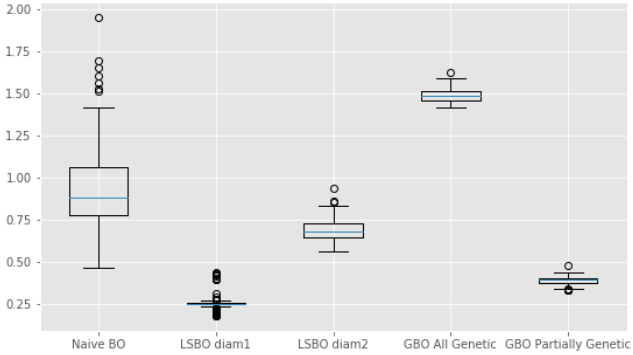


Figure 6.22: Boxplots associated to the average time needed per iteration for each selected algorithm.

Chapter 7

Conclusions

In this thesis we address the problem of optimization of expensive-to-evaluate black-box functions defined on discrete domains. We discuss about the main existing methods based on Bayesian Optimization and take the Naive BO as a benchmark for further analyses. Our contribution is to propose two different approaches to the problem, working on the way of search space exploration.
The literature research on the topic outlines how the main existing methods develop solutions starting from the main components of the BO and adapt them to a discrete domain. So, Garrido-Merchan and Hernandez-Lobato [9] propose to act a transformation on the kernel function of the GP surrogate model, or either Luong [15] works on the UCB acquisition function. We work in a different way, preserving the general structure of the Bayesian Optimization and modifying the manner in which new candidate solution is chosen. Our contribution here is twofold. On one hand we infer a topological structure, identified by the grid, on the configuration space. On the other hand we consider the variables as nodes of a directed weighted graph, where the arcs weights are associated to the correlations among them. This is distinctive compared to the previously recalled methods that don't point out the value of variable correlations.
The results highlight how our proposed methods represent, in the analysed case, a tangible improvement on Naive BO, both in terms of quality of the solution and average computational time per iteration. Looking just to the thesis case study we should deduce that the best performing method is represented by the GBO with All Genetic strategy. In fact, this obtains best results in terms of frequency of the achievement of the global optimum, queue distribution and reward average and first-quartile. The only downside is given by the execution time, significantly higher than all the other methods, however here it might be

considered negligible if related to the objective function evaluation time.
Generalizing, the choice of the most suitable method depends on the main characteristics of the problem or else on the particular business policy. If we have a limited amount of time available and the function evaluation doesn't represent a heavy bottleneck either in terms of time or costs, it might be convenient to apply the LSBO with diameter setted to 2, which allows increasing the number of algorithm iterations and so a better space exploration, ensuring higher performances. On the other hand, if the limit is the number of function evaluations and we are interested also in avoiding heavy queues, the GBO with All Genetic approach represents the best choice. In other words a global optimal method, that satisfies at the same time all the possible needs, cannot be identified, but it must be selected case by case evaluating all pros and cons.

Future possible improvements include at first the adaptation and test of the proposed methods on different case studies and simulations, and a detailed comparison with other existing approaches not yet examined, to provide a greater statistical relevance to our work. Furthermore it would be challenging to design and infer a proper graph structure on the entire problem topology, both on the configuration space and on the variables set.

Acknowledgements

I would like to acknowledge my supervisors, Professor Fabio Fagnani and Professor Giacomo Como, for their continuous support during my thesis research. Their valuable advices and suggestions have extensively contributed to the content of this dissertation.
I am also thankful to the PhD student Luca Damonte for his support and guidance during the whole research, implementation and drafting steps.
Moreover, I would express my gratitude to the TIM group who provided the research problem and the needed data allowing us to achieve these results. In particular my sincere thanks go to Eng. Roberta Giannantonio and Eng. Ennio Grasso who constantly followed and supported all the research work.
I also extend my thanks to the Department of Mathematical Sciences (DISMA) of the Politecnico di Torino, Department of excellence 2018-2022, that supported this work through its computational resources.

Bibliography

[1] Archetti F., Candelieri A., *Bayesian Optimization and Data Science*, Springer-Briefs in Optimization, 2019

[2] Bajpai P., Kumar M., *Genetic Algorithm – an Approach to Solve Global Optimization Problems*, Indian Journal of Computer Science and Engineering, Vol 1 No 3 199-206

[3] Baptista R., Poloczek M., *Bayesian Optimization of Combinatorial Structures*, International Conference on Machine Learning 2018

[4] Bergstra J., Bardenet R., Bengio Y., Kegl B., *Algorithms for Hyper-Parameter Optimization*, Advances in Neural Information Processing Systems, 2011

[5] Bissuel A., *Hyper-parameter optimization algorithms: a short review*, in Criteo Tech Blog - Medium, April 16 2019

[6] Brochu E., Cora V.M., De Freitas N., *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*, 2010, in arXiv:1012.2599

[7] Dandanov N., Al-Shatri H., Klein A., Poulkov V., *Dynamic Self-Optimization of the Antenna Tilt for Best Trade-off Between Coverage and Capacity in Mobile Neworks*, Springer Science+Business Media New York, 2016

[8] Frazier P.I., *A Tutorial on Bayesian Optimization*, 2018, in arXiv:1807.02811

[9] Garrido-Merchan E.C., Hernandez-Lobato D., *Dealing with Categorical and Integer-valued Variables in Bayesian Optimization with Gaussian Processes*, 2018, in arXiv:1805.03463v2

[10] Goldberg D.E., *Genetic Algorithms in Seatch, Optimizatio & Machine Learning*, Addison-Wesley Publishing Company, 1989

[11] Hutter F., Hoos H., Leyton-Brown K., *An Evaluation of Sequential Model-Based Optimization for Expensive Blackbox Functions*, GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherland

[12] Hutter F., Hoos H., Leyton-Brown K., *Sequential Model-Based Optimization for General Algorithm Configuration*, Coello C.A.C. (eds) Learning and Intelligent Optimization. LION 2011. Lecture Notes in Computer Science, vol 6683. Springer, Berlin, Heidelberg

[13] Jebari K. Madiafi M., *Selection Methods for Genetic Algorithms*, Int. J. Emerg. Sci., 3(4), 333-344, December 2013

[14] KP Performance Antennas, *Downtilt: How to set it*, kppperformance, 2017

[15] Luong P., Gupts S., Nguyen D., Rana S., Venkatesh S., *Bayesian Optimization with Discrete Variables*, AI 2019: Advances in Artificial Intelligence, 32nd Australasian Joint Conference, Adelaide, SA, Australia, December 2–5, 2019

[16] Lizotte D. J., Wang T., Bowling M. H. and Schuurman, D., *Automatic Gait Optimization with Gaussian Process Regression*, in IJCAI, Vol. 7, pp. 944-949, 2007

[17] Negoescu D. M., Frazier P. I., and Powell W. B., *The knowledge-gradient algorithm for sequencing experiments in drug discovery*, INFORMS Journal on Computing, Vol. 23, No. 3, pp. 346–363, 2011.

[18] Rasmussen C.E., Williams C.K.I., *Gaussian Processes for Machine Learning*, the MIT Press, 2006

[19] Shahriari B. et al., *Taking the Human Out of the Loop: A Review of Bayesian Optimization*, Vol. 104, No. 1, Jan 2016, Proceedings of the IEEE

[20] Snoek J., Larochelle H. and Adams R. P., *Practical Bayesian Optimization of Machine Learning Algorithms*, in Advances in Neural Information Processing Systems 25, pp. 2171-2180, 2012

[21] Srinivas N., Krause A., Kakade S. M., Seeger M. W., *Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting*, IEEE Transactions on Information Theory, vol. 58, no. 5, MAY 2012

[22] Tadei R., Della Croce F., Grosso A., *Fondamenti di Ottimizzazione*, Società Editrice Esculapio, 2013

[23] Talbi E., *Metaheuristics from design to implementation*, Wiley, 2009