Additive Tree-Structured Covariance Function for Conditional Parameter Spaces in Bayesian Optimization

Xingchen Ma
ESAT-PSI, KU Leuven, Belgium

Matthew B. Blaschko
ESAT-PSI, KU Leuven, Belgium

Abstract

Bayesian optimization (BO) is a sample-efficient global optimization algorithm for black-box functions which are expensive to evaluate. Existing literature on model based optimization in conditional parameter spaces are usually built on trees. In this work, we generalize the additive assumption to tree-structured functions and propose an additive tree-structured covariance function, showing improved sample-efficiency, wider applicability and greater flexibility. Furthermore, by incorporating the structure information of parameter spaces and the additive assumption in the BO loop, we develop a parallel algorithm to optimize the acquisition function and this optimization can be performed in a low dimensional space. We demonstrate our method on an optimization benchmark function, as well as on a neural network model compression problem, and experimental results show our approach significantly outperforms the current state of the art for conditional parameter optimization including SMAC, TPE and Jenatton et al. (2017).

1 INTRODUCTION

In many applications, we are faced with the problem of optimizing an expensive black-box function and we wish to find its optimum using as few evaluations as possible. *Bayesian Optimization* (BO) (Jones et al., 1998) is a global optimization technique, which is specially suited for these problems. BO has gained increasing attention in recent years (Srinivas et al., 2010; Brochu et al., 2010; Hutter et al., 2011; Shahriari et al., 2016; Frazier, 2018) and has been successfully applied to sensor location (Srinivas et al., 2010), hierarchical reinforcement learning (Brochu et al., 2010), and automatic machine learning (Klein et al., 2017).

In the general BO setting, we aim to solve the following problem:

$$\min_{\boldsymbol{x}\in\mathcal{X}\subset\mathbb{R}^d} f(\boldsymbol{x}),$$

where $\mathcal{X}$ is the parameter space and f is a black-box function which is expensive to evaluate. Typically, the parameter space $\mathcal{X}$ is treated as structureless, however, for many practical applications, there exists a conditional structure in $\mathcal{X}$:

$$f(\boldsymbol{x} \mid \boldsymbol{x}_{\mathcal{I}_A}) = f(\boldsymbol{x}_{\mathcal{I}_B} \mid \boldsymbol{x}_{\mathcal{I}_A}), \tag{1}$$

where the index sets $\mathcal{I}_A = \{a_1, \ldots, a_k\}$, $\mathcal{I}_B = \{b_1, \ldots, b_m\}$ and $\mathcal{I}_A \cup \mathcal{I}_B$ are subsets of $\mathcal{I}_D = \{1, \ldots, d\}$. Intuitively, Equation (1) means given the value of $\boldsymbol{x}_{\mathcal{I}_A}$, the value of $f(\boldsymbol{x})$ remains unchanged after removing $\boldsymbol{x}_{\mathcal{I}_D \setminus (\mathcal{I}_A \cup \mathcal{I}_B)}$. Here we use set based subscripts to denote the restriction of $\boldsymbol{x}$ to the corresponding indices.

This paper investigates optimization problems where the parameter space exhibits such a conditional structure. In particular, we focus on a specific instantiation of the general conditional structure in Equation (1): Tree-structured parameter spaces, which are also studied in Jenatton et al. (2017). Many problems fall into this category, for example, when fitting *Gaussian Processes* (GPs), we need to choose from several covariance functions and subsequently set their continuous hyper-parameters. Different covariance functions may share some hyper-parameters, such as the signal variance and the noise variance (Rasmussen and Williams, 2006).

By exploring the properties of this tree structure, we design an *additive tree-structured* (Add-Tree) covariance function, which enables information sharing between different data points under the *additive assumption*, and allows GP to model f in a sample-efficient way. Furthermore, by including the tree structure and the additive assumption in the BO loop, we develop a

Proceedings of the 23rdInternational Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

parallel algorithm to optimize the acquisition function, making the overall execution faster. Our proposed method also helps to alleviate the curse of dimensionality through two advantages: (i) we avoid modeling the response surface directly in a high-dimensional space, and (ii) the acquisition optimization is also operated in a lower-dimensional space.

In the next section, we will briefly review BO together with the literature related to optimization in a conditional parameter space. In Section 2, we formalize the family of objective functions that can be solved using our approach. We then present our Add-Tree covariance function in Section 3. In Section 4, we give the inference procedure and BO algorithm using our covariance function. We then report a range of experiments in Section 5. Finally, we conclude in Section 6.

1.1 RELATED WORK

1.1.1 Bayesian Optimization

BO has two major components. The first one is a probabilistic regression model used to fit the response surface of f. Popular choices include GPs (Brochu et al., 2010), random forests (Hutter et al., 2011) and adaptive Parzen estimators (Bergstra et al., 2011). We refer the reader to Rasmussen and Williams (2006) for the foundations of Gaussian Processes. The second one is an acquisition function u_{t-1} which is constructed from this regression model and is used to propose the next evaluation point. Popular acquisition functions include the *expected improvement* (EI) (Jones et al., 1998), *knowledge gradient* (KG) (Frazier et al., 2009), *entropy search* (ES) (Hennig and Schuler, 2012) and *Gaussian process upper confidence bound* (GP-UCB) (Srinivas et al., 2010).

One issue that often occurs in BO is, in high-dimensional parameter spaces, its performance may be no better than random search (Wang et al., 2013; Li et al., 2016). This deterioration is due to high uncertainty in fitting a regression model due to the curse of dimensionality (Györfi et al., 2002, Ch. 2), which in turn leads to pure-explorational behavior of BO. This will further cause inefficiency in the acquisition function, making the proposal of the next data point behave like random selection. Standard GP-based BO ignores the structure in a parameter space, and fits a regression model in $\mathbb{R}^d$. By leveraging this structure information, we can work in a low-dimensional space $\mathbb{R}^m$ (recall Equation (1)) instead of $\mathbb{R}^d$.

1.1.2 Conditional Parameter Spaces

Sequential Model-based Algorithm Conguration (SMAC) (Hutter et al., 2011) and Tree-structured Parzen Estimator Approach (TPE) (Bergstra et al., 2011) are two popular non-GP based optimization algorithms that are aware of the conditional structure in $\mathcal{X}$, however, they lack favorable properties of GPs: uncertainty estimation in SMAC is non-trivial and the dependencies between dimensions are ignored in TPE. Additionally, neither of these methods have a particular sharing mechanism, which is valuable in the low-data regime.

In the category of GP-based BO, which is our focus in this paper, Hutter and Osborne (2013) proposed a covariance function that can explicitly employ the tree structure and share information at those categorical nodes. However, their specification for the parameter space is too restrictive and they require the shared node to be a categorical variable. By contrast, we allow shared variables to be continuous (see Section 3). Swersky et al. (2014) applied the idea of Hutter and Osborne (2013) in a BO setting, but their method still inherits the limitations of Hutter and Osborne (2013). Another covariance function to handle tree-structured dependencies is presented in Lévesque et al. (2017). In that case, they force the similarity of two samples from different condition branches to be zero and the resulting model can be transformed into several independent GPs. We perform a comparison to an independent GP baseline in Section 5.1. In contrast to Add-Tree, the above approaches either have very limited applications, or lack a sharing mechanism. Jenatton et al. (2017) presented another GP-based BO approach, where they handle tree-structured dependencies by introducing a weight vector linking all sub-GPs, and this introduces an explicit sharing mechanism. Although Jenatton et al. (2017) overcame the above limitations, the enforced linear relationships between different paths make their semi-parametric approach less flexible compared with our method. We observe in our experiments that this can lead to a substantial difference in performance.

2 PROBLEM FORMULATION

We begin by summarizing notation used in this paper. Let $\mathcal{T} = (V, E)$ be a tree, in which V is the set of vertices, E is the set of edges, $P = \{p_i\}_{1 \le i \le |P|}$ be the set of leaves and r be the root of $\mathcal{T}$ respectively, $\{l_i\}_{1 \le i \le |P|}$ be the ordered set of vertices on the path from r to the i-th leaf p_i, and h_i be the number of vertices along l_i (including r and p_i). To distinguish an objective function defined on a tree-structured parameter space from a general objective function, we use $f_{\mathcal{T}}$ to indicate our objective function. In what follows, we will call $f_{\mathcal{T}}$ a *tree-structured function*.

To formalize the family of problems that can be solved

with our method, we start with some definitions.

Definition 1 (Tree-structured parameter space)**.** A *tree-structured parameter space* $\mathcal{X}$ is associated with a tree $\mathcal{T} = (V, E)$. For any $v \in V$, v is associated with a bounded continuous variable of $\mathcal{X}$; the set of outgoing edges E_v of v represent one categorical variable of $\mathcal{X}$ and each element of E_v represents a specific setting of the corresponding categorical variable.

Definition 2 (Tree-structured function)**.** A *tree-structured function* $f_\mathcal{T} : \mathcal{X} \to \mathbb{R}$ is defined on a d-dimensional tree-structured parameter space $\mathcal{X}$. The i-th leaf p_i is associated with a function $f_{p_i,\mathcal{T}}$ of the variables associated with the vertices along l_i. $f_\mathcal{T}$ is called *tree-structured* if for every leaf of the tree-structured parameter space

$$f_\mathcal{T}(\boldsymbol{x}) := f_{p_j,\mathcal{T}}(\boldsymbol{x}|_{l_j}), \tag{2}$$

where p_j is selected by the categorical values of $\boldsymbol{x}$ and $\boldsymbol{x}|_{l_j}$ is the restriction of $\boldsymbol{x}$ to l_j.

To aid in the understanding of a tree-structured function (and subsequently our proposed Add-Tree covariance function), we depict a simple tree-structured function in Figure 1. The outgoing edges of r represent the categorical variable $t \in \{1, 2\}$ and the settings of t are shown around these two edges. Vertices r, p_1, p_2 are associated with bounded variables $\boldsymbol{v}_r \in [-1, 1]^2, \boldsymbol{v}_{p_1} \in [-1, 1]^2, \boldsymbol{v}_{p_2} \in [-1, 1]^3$ respectively and leaves p_1, p_2 are associated with two functions shown in Figure 1. In Definition 2, the restriction of one input to a path means we collect variables associated with the vertices along that path and concatenate them using a fixed ordering. For example, in Figure 1, let $\boldsymbol{x} \in \mathcal{X}$ be an 8-dimensional input, then the restriction of $\boldsymbol{x}$ to path l_1 is a 4-dimensional vector. The function illustrated in Figure 1 can be compactly written down as:

$$f_\mathcal{T}(\boldsymbol{x}) = \mathbb{1}_{t=1} f_{p_1,\mathcal{T}}(\boldsymbol{x}|_{l_1}) + \mathbb{1}_{t=2} f_{p_2,\mathcal{T}}(\boldsymbol{x}|_{l_2}), \tag{3}$$

where $\boldsymbol{x}$ is the concatenation of $(\boldsymbol{v}_r, \boldsymbol{v}_{p_1}, \boldsymbol{v}_{p_2}, t)$, $\mathbb{1}$ denotes the indicator function, $f_{p_1,\mathcal{T}}(\boldsymbol{x}|_{l_1}) = \|\boldsymbol{v}_r\|^2 + \|\boldsymbol{v}_{p_1}\|^2$ and $f_{p_2,\mathcal{T}}(\boldsymbol{x}|_{l_2}) = \|\boldsymbol{v}_r\|^2 + \|\boldsymbol{v}_{p_2}\|^2$.

A tree-structured function $f_\mathcal{T}$ is actually composed of several smaller functions $\{f_{p_i,\mathcal{T}}\}_{1 \le i \le |P|}$, given a specific setting of the categorical variables, $f_\mathcal{T}$ will return the associated function at the i-th leaf. To facilitate our description in the following text, we define the *effective dimension* in Definition 3.

Definition 3 (Effective dimension)**.** The *effective dimension* of a tree-structured function $f_\mathcal{T}$ at the i-th leaf p_i is the sum of dimensions of the variables associated with the vertices along l_i.

Remark. The effective dimension of $f_\mathcal{T}$ can be much smaller than the dimension of $\mathcal{X}$.

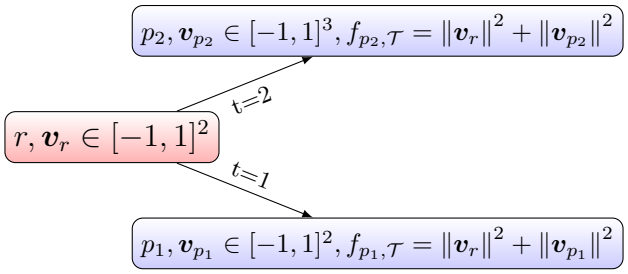

Figure 1: A Simple Tree-Structured Function

For the tree-structured function depicted in Figure 1, the dimension of $\mathcal{X}$ is $d = 8$ and the effective dimension at p_1 and p_2 are 4 and 5 respectively. Particularly, if $\mathcal{T}$ is a perfect binary tree, in which each vertex is associated with a 1-dimensional continuous variable, the effective dimension of $f_\mathcal{T}$ at every leaf is the depth h of $\mathcal{T}$, while the dimension of $\mathcal{X}$ is $3 \cdot 2^{h-1} - 2$. If the tree structure information is thrown away, we have to work in a much higher dimensional parameter space. It is known that in high dimensions, BO behaves like random search, which violates the entire purpose of model based optimization.

Now we have associated the parameter space $\mathcal{X}$ with a tree structure, which enables us to work in the low-dimensional effective space. How can we leverage this tree structure to optimize $f_\mathcal{T}$? Recalling $f_\mathcal{T}$ is a collection of $|P|$ functions $\{f_{p_i,\mathcal{T}}\}_{1 \le i \le |P|}$, a trivial solution is using $|P|$ independent GPs to model each $f_{p_i,\mathcal{T}}$ separately. In BO settings, we are almost always in low data regime because black-box calls to $f_\mathcal{T}$ are expensive (e.g. the cost of training and evaluating a machine learning model). Modeling $f_\mathcal{T}$ using a collection of GPs is obviously not an optimal way because we discard the correlation between $f_{p_i,\mathcal{T}}$ and $f_{p_j,\mathcal{T}}$ when $i \neq j$. How to make the most of the observed data, especially how to share information between data points coming from different leaves remains a crucial question. In this paper, we assume additive structure within each $f_{p_i,\mathcal{T}}$ for $i = 1, \cdots, |P|$. More formally, $f_{p_i,\mathcal{T}}$ can be decomposed in the following form:

$$f_{p_i,\mathcal{T}}(\boldsymbol{x}) = \sum_{j=1}^{h_i} f_{ij}(v_{ij}) \tag{4}$$

where v_{ij} is the associated variable on the j-th vertex along l_i. Additive assumption has been extensively studied in GP literature (Duvenaud et al., 2011; Kandasamy et al., 2015; Gardner et al., 2017; Rolland et al., 2018) and is a popular way for dimension reduction(Györfi et al., 2002). We note the tree-structured function discussed in this paper is a generalization of the objective function presented in these publications and our additive assumption in Equation (4) is also

a generalization of the additive structure considered previously. For example, the additive function discussed in Kandasamy et al. (2015) can be viewed as a tree-structured function the associated tree of which has a branching factor of 1, i.e. $|P| = 1$. Our generalized additive assumption will enable an efficient sharing mechanism as we develop in Section 3.

3 THE ADD-TREE COVARIANCE FUNCTION

In this section, we describe how we use the tree structure and the additive assumption to design a covariance function, which is sample-efficient in low-data regime. We start with the definition of the Add-Tree covariance function (Definition 4), then we show the intuition (Equation (9)) behind this definition and present an algorithm (Algorithm 1) to automatically construct an Add-Tree covariance function from the specification of a tree-structured parameter space, finally a proof of the validity of this covariance function is given.

Definition 4 (Add-Tree covariance function)**.** For a tree-structured function $f_{\mathcal{T}}$, let $\boldsymbol{x}_{i'}$ and $\boldsymbol{x}_{j'}$ be two inputs of $f_{\mathcal{T}}$, p_i and p_j be the corresponding leaves, a_{ij} be the lowest common ancestor (LCA) of p_i and p_j, l_{ij} be the path from r to a_{ij}. A covariance function $k_{f_{\mathcal{T}}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is said to be an *Add-Tree covariance function* if for each $\boldsymbol{x}_{i'}$ and $\boldsymbol{x}_{j'}$

$$\begin{aligned} k_{\mathcal{T}}(\boldsymbol{x}_{i'}, \boldsymbol{x}_{j'}) :=& k_{l_{ij}}(\boldsymbol{x}_{i'}|_{l_{ij}}, \boldsymbol{x}_{j'}|_{l_{ij}}) \\ =& \sum_{v \in l_{ij}} k_v(\boldsymbol{x}_{i'}|_v, \boldsymbol{x}_{j'}|_v) \end{aligned} \tag{5}$$

where $\boldsymbol{x}_{i'}|_{l_{ij}}$ is the restriction of $\boldsymbol{x}_{i'}$ to the variables along the path l_{ij}, and k_v is any positive semi-definite covariance function on the continuous variables appearing at a vertex v on the path l_{ij}. We note the notation l_{ij} introduced here is different from the notation l_i introduced in the beginning of Section 2.

To give the ideas behind the Add-Tree family of covariance functions, we take the tree-structured function illustrated in Figure 1 (Equation (3)) as an example.[1] Let $X_1 \in \mathbb{R}^{n_1 \times d_1}$ and $X_2 \in \mathbb{R}^{n_2 \times d_2}$ be the inputs from l_1 and l_2, where $d_1 = 2 + 2$ and $d_2 = 2 + 3$ are the effective dimensions of $f_{\mathcal{T}}$ at p_1 and p_2 respectively. Denote the latent variables associated to the decomposed functions[2] at r, p_1 and p_2 by $\boldsymbol{f}_r \in \mathbb{R}^{n_1+n_2}$, $\boldsymbol{f}_1 \in \mathbb{R}^{n_1}$ and $\boldsymbol{f}_2 \in \mathbb{R}^{n_2}$, respectively. Reordering and partition $\boldsymbol{f}_r$ into two parts corresponding to p_1 and p_2, so that

$$\boldsymbol{f}_r = \begin{bmatrix} \boldsymbol{f}_r^{(1)} \\ \boldsymbol{f}_r^{(2)} \end{bmatrix}, \boldsymbol{f}_r^{(1)} \in \mathbb{R}^{n_1}, \boldsymbol{f}_r^{(2)} \in \mathbb{R}^{n_2}.$$

Let the gram matrix corresponding to $\boldsymbol{f}_r, \boldsymbol{f}_1, \boldsymbol{f}_2$ be $K_r \in \mathbb{R}^{(n_1+n_2)\times(n_1+n_2)}, K_1 \in \mathbb{R}^{n_1 \times n_1}, K_2 \in \mathbb{R}^{n_2 \times n_2}$. W.l.o.g, let the means of $\boldsymbol{f}_r, \boldsymbol{f}_1, \boldsymbol{f}_2$ be $\mathbf{0}$. By the additive assumption in Equation (4), the latent variables corresponding to the associated functions at p_1 and p_2 are $\boldsymbol{f}_r^{(1)} + \boldsymbol{f}_1$ and $\boldsymbol{f}_r^{(2)} + \boldsymbol{f}_2$, we have:

$$\begin{bmatrix} \boldsymbol{f}_r^{(1)} + \boldsymbol{f}_1 \\ \boldsymbol{f}_r^{(2)} + \boldsymbol{f}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_r^{(1)} \\ \boldsymbol{f}_r^{(2)} \end{bmatrix} + \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{f}_r^{(1)} \\ \boldsymbol{f}_r^{(2)} \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, K_r). \tag{6}$$

Due to $\boldsymbol{f}_1 \perp\!\!\!\perp \boldsymbol{f}_2$, where $\perp\!\!\!\perp$ denotes $\boldsymbol{f}_1$ is independent of $\boldsymbol{f}_2$, we have:

$$\begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_1 & \mathbf{0} \\ \mathbf{0} & K_2 \end{bmatrix}\right), \tag{7}$$

furthermore, because of the additive assumption in Equation (4),

$$\begin{bmatrix} \boldsymbol{f}_r^{(1)} \\ \boldsymbol{f}_r^{(2)} \end{bmatrix} \perp\!\!\!\perp \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \end{bmatrix}. \tag{8}$$

Combine Equations (6) to (8), we arrive at our key conclusion:

$$\begin{bmatrix} \boldsymbol{f}_r^{(1)} + \boldsymbol{f}_1 \\ \boldsymbol{f}_r^{(2)} + \boldsymbol{f}_2 \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_r^{(11)} + K_1 & K_r^{(12)} \\ K_r^{(21)} & K_r^{(22)} + K_2 \end{bmatrix}\right), \tag{9}$$

where K_r is decomposed as follows:

$$K_r = \begin{bmatrix} K_r^{(11)} & K_r^{(12)} \\ K_r^{(21)} & K_r^{(22)} \end{bmatrix}$$

in which $K_r^{(11)} \in \mathbb{R}^{n_1 \times n_1}, K_r^{(12)} \in \mathbb{R}^{n_1 \times n_2}, K_r^{(21)} \in \mathbb{R}^{n_2 \times n_1}, K_r^{(22)} \in \mathbb{R}^{n_2 \times n_2}$. The observation in Equation (9) is crucial in two aspects: firstly, we can use a single covariance function and a global GP to model our objective, secondly and more importantly, this covariance function allows an efficient sharing mechanism between data points coming from different paths, although we cannot observe the decomposed function values at the shared vertex r, we can directly read out this sharing information from $K_r^{(12)}$.

We summarize the construction of an Add-Tree covariance function in Algorithm 1, where the value of `Index` comes from applying BFS to the associated tree structure $\mathcal{T}$ of $f_{\mathcal{T}}$ and `Dim` at v is the dimension of the variable associated with vertex v. We provide implementation details in Appendix A. In Appendix E, we discuss the case when additive assumption is not enough to model the objective function.

[1] To simplify the presentation, we use a two-level tree structure in this example. The covariance function, however, generalizes to tree-structured functions of arbitrary depth (Algorithm 1).

[2] On functions and latent variables, one can refer to Rasmussen and Williams (2006, chap. 2)

Algorithm 1: Add-Tree Covariance Function

Input : The associated tree $\mathcal{T} = (V, E)$ of $f_{\mathcal{T}}$
Output: Add-Tree covariance function $k_{\mathcal{T}}$

$k_{\mathcal{T}} \leftarrow 0$
for $v \leftarrow V$ **do**
  $vi \leftarrow \texttt{Index}(v)$
  $k_v^d \leftarrow k_v^\delta(\boldsymbol{x}, \boldsymbol{x}')$ `/* k_v^δ is 1 iff x and x' both have vertex v in their paths */`
  $si \leftarrow vi + 1$ `/* start index of v */`
  $ei \leftarrow vi + 1 + \texttt{Dim}(v)$ `/* end index of v */`
  $k_v^c \leftarrow k_c(\boldsymbol{x}_{si \le i \le ei}, \boldsymbol{x}'_{si \le i \le ei})$ `/* k_c is any p.s.d. covariance function */`
  $k_v \leftarrow k_v^d \times k_v^c$
  $k_{\mathcal{T}} \leftarrow k_{\mathcal{T}} + k_v$

Proposition 1. The Add-Tree covariance function defined by Definition 4 is positive semi-definite for all tree-structured functions defined in Definition 2 with the additive assumption satisfied.

Proof. We will consider each term in Equation (5) and demonstrate that it results in a positive semi-definite covariance function over the whole set of data points, not just the data points that follow the given path. In particular, consider the p.s.d. covariance function $k_v^\delta(\boldsymbol{x}_{i'}, \boldsymbol{x}_{j'}) = \begin{cases} 1 & \text{if } v \in l_i \wedge v \in l_j \\ 0 & \text{otherwise} \end{cases}$, for some vertex v. We see that the product $k_v^c \times k_v^\delta$ defines a p.s.d. covariance function over the entire space of observations (since the product of two p.s.d. covariance functions is itself p.s.d.), and not just those sharing vertex v. In this way, we may interpret Equation (5) as a summation over only the non-zero terms of a set of covariance functions defined over all vertices in the tree. As the resulting covariance function sums over p.s.d. covariance functions, and positive semi-definiteness is closed over positively weighted sums, the result is p.s.d. □

4 BO FOR TREE-STRUCTURED FUNCTIONS

In this section, we first describe how to perform the inference with our proposed Add-Tree covariance function, then we present a parallel algorithm for the optimization of the acquisition function.

4.1 Inference with Add-Tree

Given noisy observations $\mathcal{D}_n = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^n$, we would like to obtain the predictive distribution for the latent variable $f_{*\mathcal{T}}$ at a new input $\boldsymbol{x}_*$. We begin with some notation. Let p_* be the leaf selected by the categorical values of $\boldsymbol{x}_*$, l_* be the path from the root r to p_*. All n inputs are collected in the design matrix $X \in \mathbb{R}^{n \times d}$, where the i-th row represents $\boldsymbol{x}_i \in \mathbb{R}^d$, and the targets and observation noise are aggregated in vectors $\boldsymbol{y} \in \mathbb{R}^n$ and $\boldsymbol{\sigma} \in \mathbb{R}^n$ respectively. Let $\Sigma = \text{diag}(\boldsymbol{\sigma})$ be the noise matrix, where $\text{diag}(\boldsymbol{\sigma})$ denotes a diagonal matrix containing the elements of vector $\boldsymbol{\sigma}$, $S = \{i \mid l_* \bigcap l_i \neq \varnothing\}$, $I \in \mathbb{R}^n$ be the identity matrix, $M \in \mathbb{R}^{|S| \times n}$ be a selection matrix, which is constructed by removing the j-th row of I if $j \notin S$, $X' = MX \in \mathbb{R}^{|S| \times d}$, $\boldsymbol{y'} = M\boldsymbol{y} \in \mathbb{R}^{|S|}$, $\Sigma' = M\Sigma M^T \in \mathbb{R}^{|S| \times |S|}$. We can then write down the joint distribution of $f_{*\mathcal{T}}$ and $\boldsymbol{y'}$ as:

$$\begin{bmatrix} f_{*\mathcal{T}} \\ \boldsymbol{y'} \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} k_{\mathcal{T}}(\boldsymbol{x}_*, \boldsymbol{x}_*) & k_{\mathcal{T}}(\boldsymbol{x}_*, X') \\ k_{\mathcal{T}}(X', \boldsymbol{x}_*) & k_{\mathcal{T}}(X', X') + \Sigma' \end{bmatrix}\right).$$

We note that this joint distribution has the same standard form (Rasmussen and Williams, 2006) as in all GP-based BO, but that it is made more efficient by the selection of X' based on the tree structure.

The predictive distribution for $f_{*\mathcal{T}}$ is:

$$f_{*\mathcal{T}} \mid X', \boldsymbol{y'}, \boldsymbol{x}_* \sim \mathcal{N}(\bar{f}_{*\mathcal{T}}, \text{Var}(f_{*\mathcal{T}})) \tag{10}$$

where

$$\begin{aligned} \bar{f}_{*\mathcal{T}} &= k_{\mathcal{T}}(\boldsymbol{x}_*, X') \left[k_{\mathcal{T}}(X', X') + \Sigma'\right]^{-1} \boldsymbol{y'}, \\ \text{Var}(f_{*\mathcal{T}}) &= k_{\mathcal{T}}(\boldsymbol{x}_*, \boldsymbol{x}_*) \\ &\quad - k_{\mathcal{T}}(\boldsymbol{x}_*, X')[k_{\mathcal{T}}(X', X') + \Sigma']^{-1} k_{\mathcal{T}}(X', \boldsymbol{x}_*). \end{aligned}$$

Black-box calls to the objective function usually dominate the running time of BO, and the time complexity of fitting GP is of less importance. In Appendix B, we provide details on time complexity for our Add-Tree along with other related methods for completeness.

4.2 Acquisition Function Optimization

In BO, the acquisition function $u_{t-1}(\boldsymbol{x}|\mathcal{D})$, where t is the current step of optimization, is used to guide the search for the optimum of our objective function. By trading off the exploitation and exploration, it is expected we can find the optimum using as few calls as possible to the objective. To get the next point at which we evaluate our objective function, we solve $\boldsymbol{x}_t = \arg\max_{\boldsymbol{x} \in \mathcal{X}} u_{t-1}(\boldsymbol{x}|\mathcal{D})$. For noisy observations, GP-UCB (Srinivas et al., 2010) has nice theoretical proprieties and explicit regret bounds for many commonly used covariance functions, and in this paper, we will use GP-UCB, which is defined as:

$$u_{t-1}(\boldsymbol{x}|\mathcal{D}) = \mu_{t-1}(\boldsymbol{x}) + \beta_t^{1/2} \sigma_{t-1}(\boldsymbol{x}),$$

where β_t are suitable constants, $\mu_{t-1}(\boldsymbol{x})$ and $\sigma_{t-1}(\boldsymbol{x})$ are the predictive posterior mean and variance at $\boldsymbol{x}$ from Equation (10). Throughout the experiments in

this paper, following Kandasamy et al. (2015), we set $\beta_t = 0.2\tilde{d}\log(2t)$, in which $\tilde{d}$ denotes the dimension of the space where we optimize GP-UCB and is usually smaller than d for a tree-structured function. We note the Add-Tree covariance function developed here can be combined with any other acquisition function. Appendix D contains more details on combining other acquisition functions with Add-Tree.

A naïve way to obtain the next evaluation point for a tree-structured function is to independently find $|P|$ optima, each one corresponding to the optimum of the associated function at a leaf, and then choose the best candidate across these optima. This approach is presented in Jenatton et al. (2017) and the authors there already pointed out this is too costly. Here we develop a much more efficient algorithm, which is dubbed as Add-Tree-GP-UCB, to find the next point and we summarise it in Algorithm 2. By explicitly utilizing the associated tree structure $\mathcal{T}$ of $\boldsymbol{f}_{\mathcal{T}}$ and the additive assumption, the first two nested `for` loops can be performed in parallel. Furthermore, as a by-product, each acquisition function optimization routine is now performed in a low dimensional space whose dimension is even smaller than the effective dimension. Time complexity analysis of Algorithm 2 is given in Appendix C.

Algorithm 2: Add-Tree-GP-UCB

Input : The associated tree $\mathcal{T} = (V, E)$ of $f_{\mathcal{T}}$, Add-Tree covariance function $k_{\mathcal{T}}$ from Algorithm 1, paths $\{l_i\}_{1 \le i \le |P|}$ of $\mathcal{T}$

$D_0 \leftarrow \varnothing$
for $t \leftarrow 1, \dots$ **do**
  for $v \leftarrow V$ **do**
    $\boldsymbol{x}_t^v \leftarrow \arg\max_x \mu_{t-1}^v(\boldsymbol{x}) + \sqrt{\beta_t}\sigma_{t-1}^v(\boldsymbol{x})$
    $u_t^v \leftarrow \mu_{t-1}^v(\boldsymbol{x}_t^v) + \sqrt{\beta_t}\sigma_{t-1}^v(\boldsymbol{x}_t^v)$
  for $i \leftarrow 1, \dots, |\mathcal{P}|$ **do**
    $U_t^{l_i} \leftarrow \sum_{v \in l_i} u_t^v$ `/* additive assumption from Equation (4) */`
  $j \leftarrow \arg\max_i \{U_t^{l_i} \mid i = 1, \dots, |\mathcal{P}|\}$
  $\boldsymbol{x}_t \leftarrow \bigcup_{v \in l_j} \{\boldsymbol{x}_v\}$
  $y_t \leftarrow \boldsymbol{f}_{\mathcal{T}}(\boldsymbol{x}_t)$
  $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \bigcup \{(\boldsymbol{x}_t, y_t)\}$
  Fitting GP using $\mathcal{D}_t$ to get $\{(\mu_t^v, \sigma_t^v)\}_{v \in V}$ using maximum likelihood

5 EXPERIMENTS

In this section, we present results for two sets of experiments. To demonstrate the efficiency of our Add-Tree-GP-UCB, we first optimize the synthetic functions presented in Jenatton et al. (2017), comparing to SMAC (Hutter et al., 2011), TPE (Bergstra et al., 2011), random search (Bergstra and Bengio, 2012), standard GP-based BO from GPyOpt (The GPyOpt authors, 2016), and the semi-parametric approach proposed in Jenatton et al. (2017). To facilitate our following description, we refer to the above competing algorithms as **smac**, **tpe**, **random**, **gpyopt**, and **tree** respectively. We refer to our approach as **add-tree**. To verify our Add-Tree covariance function indeed enables sharing between different paths, we compare Add-Tree with independent GPs in the regression setting showing greater sample efficiency for our method. We then apply our method to the application of model compression for a three-layer fully connected neural network, outperforming competing methods.

For all GP-based BO, including **gpyopt**, **tree** and **add-tree**, we use the squared exponential (SE) covariance function: $k_{\text{SE}}(r) = \sigma \exp(-r^2/2l^2)$. To optimize the parameters of Add-Tree, we maximize the marginal log-likelihood function of the corresponding GP. As for the numerical routine used in fitting the GPs and optimizing the acquisition functions, we use multi-started L-BFGS-B, as suggested by Kim and Choi (2019). For all results in this section, we display the mean and twice the standard deviation of 10 independent runs.

We note the original code for Jenatton et al. (2017) is unavailable,[3] thus we have implemented their framework from scratch to obtain the results presented here. There are several hyper-parameters in their algorithm which are not specified in the publication. To compare fairly with their method, we tune these hyper-parameters such that our implementation has a similar performance on the synthetic functions to that reported by Jenatton et al. (2017), and subsequently fix the hyper-parameter settings in the model compression task.

5.1 Synthetic Experiments

In our first experiment, we optimize the synthetic tree-structured function depicted in Figure 2 and originally presented in Jenatton et al. (2017). Non-shared variables including x_4, x_5, x_6, x_7 are defined in $[-1, 1]$, all shared variables including r_8, r_9 are bounded in $[0, 1]$, and all categorical variables including x_1, x_2, x_3 are binary. The dimension of $\mathcal{X}$ is $d = 9$ and the effective dimension at any leaf is 2.

Figure 3 shows the optimization results for the different competing methods. The x-axis shows the iteration number and the y-axis shows the log_{10} distance between the best minimum achieved so far and the known minimum value, which in this case is 0.1. It

[3] A request for the code was denied due to IP restrictions.

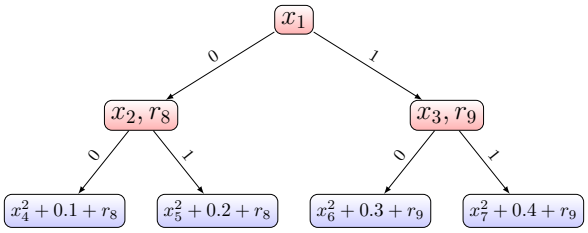

Figure 2: Synthetic Function From Jenatton et al.

is clear from Figure 3 that our method has a substantial improvement in performance compared with other algorithms. After 60 iterations, the $\log_{10}$ distance of **tree** is still higher than -4, while our method can quickly obtain a much better performance in less than 20 iterations. Interestingly, our method performs substantially better than independent GPs, which will be shown later, while in Jenatton et al. (2017), their algorithm is inferior to independent GPs. We note **gpyopt**[4] performs worst, and this is expected (recall Section 1.1.1). **gpyopt** encodes categorical variables using a one-hot representation, thus it actually works in a space whose dimension is $d' = d + c = 12$, which is relatively high considering we have less than 100 data points. In this case, **gpyopt** behaves like random search, but in a 12-dimensional space instead of the 9-dimensional space of a naïve random exploration.

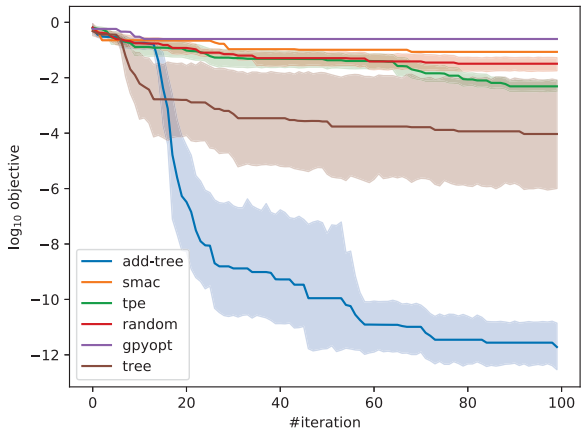

Figure 3: Optimization Performance Comparison Of The Synthetic Function

To show that Add-Tree allows efficient information sharing across different paths, we compare it with independent GPs and **tree** in a regression setting and results are shown in Figure 4. The training data is generated from the synthetic function in Figure 2: categorical values are generated from a Bernoulli distribution with $p = 0.5$, continuous values are uniformly generated from their domains. The x-axis shows the number of generated training samples and the y-axis shows the $\log_{10}$ of Mean Squared Error (MSE) on a separate randomly generated data set with 50 test samples. From Figure 4, we see that Add-Tree models the response surface better even though independent GPs have more parameters. For example, to obtain a test performance of 10^{-4}, Add-Tree needs only 24 observations, while independent GPs require 44 data points. If we just look at the case when we have 20 training samples, the absolute MSE of independent GPs is 10^{-1}, while for Add-Tree, it is 10^{-3}. The reason for such a huge difference is when training data are rare, some paths will have few data points, and Add-Tree can use the shared information from other paths to improve the regression model. This property of Add-Tree is valuable in BO settings.

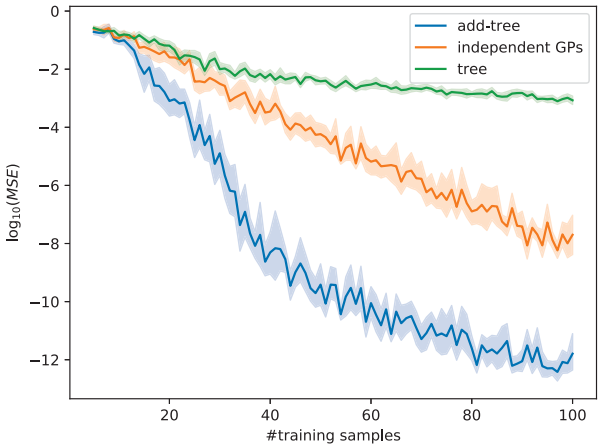

Figure 4: Regression Performance Comparison Of The Synthetic Function

5.2 Model Compression

Neural network compression is essential when it comes to deploying models on devices with limited memory and low computational resources. For parametric compression methods, like Singular Value Decomposition (SVD) and weight pruning (WP), it is necessary to tune their parameters in order to obtain the desired trade-off between model size and performance. Existing publications on model compression usually determine parameters for a single compression method, and do not have an automated selection mechanism over different methods. By encoding this problem using a tree-structured function, different compression methods can now be applied to different layers and this formulation is more flexible than the current literature.

In this experiment, we apply our method to compress

[4]GPyOpt (The GPyOpt authors, 2016) is a state-of-the-art open source Bayesian optimization software package with support for categorical variables.

a three-layer fully connected network *FC3* originally defined in Ma et al. (2019). FC3 has 784 input nodes, 10 output nodes, 1000 nodes for each hidden layer and is pre-trained on the MNIST dataset. For each layer, we find a compression method between SVD and WP, then optimize either the rank of the SVD or the pruning threshold of WP. We only compress the first two layers, because the last layer occupies 0.56% of total weights. The rank parameters are constrained to be in $[10, 500]$ and the pruning threshold parameters are bounded in $[0, 1]$. Following Ma et al. (2019), the objective function used in compressing FC3 is:

$$\gamma \mathcal{L}(\tilde{f}_\theta, f^*) + R(\tilde{f}_\theta, f^*), \tag{11}$$

where f^* is the original FC3, $\tilde{f}_\theta$ is the compressed model using parameter θ, $R(\tilde{f}_\theta, f^*)$ is the compression ratio and is defined to be the number of weights in the compressed network divided by the number of weights in the original network. $\mathcal{L}(\tilde{f}_\theta, f^*) := \mathbb{E}_{x\sim P}(\|\tilde{f}_\theta(x) - f^*(x)\|_2^2)$, where P is an empirical estimate of the data distribution. Intuitively, the R term in Equation (11) prefers a smaller compressed network, the $\mathcal{L}$ term prefers a more accurate compressed network and γ is used to trade off these two terms. In this experiment, γ is fixed to be 0.01, and the number of samples used to estimate $\mathcal{L}$ is 50 following Ma et al. (2019). Figure 5 shows the results of our method (Add-Tree) compared with other methods. For this experiment, although **smac**, **tpe** and **tree** all choose SVD for both layers at the end, our method converges significantly faster, once again demonstrating our method is more sample-efficient than other competing methods. We note **gpyopt** also has the worst performance among all other competing methods in this experiment.

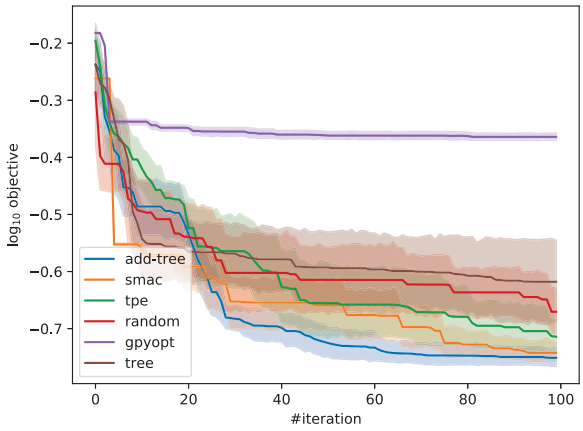

Figure 5: Optimization Performance Comparison Of FC3 Compression

Table 1 shows the results of pairwise Wilcoxon signed-rank tests[5] for the above two objective functions at different iterations. In Table 1, almost always performs significantly better than other competing methods (significance level $\alpha = 0.05$), while no method is significantly better than ours.

Table 1: Wilcoxon Signed-Rank Test

Experiment	Iter	smac	tpe	random	gpyopt	tree
synthetic function	40	0.003	0.030	0.005	0.003	0.018
	60	0.003	0.003	0.003	0.003	0.003
	80	0.003	0.003	0.003	0.003	0.003
model compression	40	0.101	0.023	0.101	0.003	0.014
	60	0.037	0.018	0.011	0.003	0.008
	80	0.166	0.005	0.003	0.003	0.006

6 CONCLUSION

In this work, we have designed a covariance function that can explicitly utilize the problem structure, and demonstrated its efficiency on a range of problems. In the low data regime, our proposed Add-Tree covariance function enables a powerful information sharing mechanism, which in turn makes BO more sample-efficient compared with other model based optimization methods. Contrary to other GP-based BO methods, we do not impose restrictions on the structure of a conditional parameter space, greatly increasing the applicability of our method. We also directly model the dependencies between different observations under the framework of Gaussian Processes, instead of placing parametric relationships between different paths, making our method more flexible. In addition, we incorporate this structure information and develop a parallel algorithm to optimize the acquisition function. For both components of BO, our proposed method allows us to work in a lower dimensional space compared with the dimension of the original parameter space.

Empirical results on an optimization benchmark function and on a neural network compression problem show our method significantly outperforms other competing model based optimization algorithms in conditional parameter spaces, including SMAC, TPE and Jenatton et al. (2017).

Acknowledgements

Xingchen Ma is supported by Onfido. This research received funding from the Flemish Government under the Onderzoeksprogramma Artificile Intelligentie (AI) Vlaanderen programme.

[5]We used the Wilcoxon signed rank implementation from scipy.stats.wilcoxon with option (alternative == 'greater').

References

Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-parameter Optimization. *J. Mach. Learn. Res.*, 13:281–305.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for Hyper-Parameter Optimization. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc.

Brochu, E., Cora, V. M., and de Freitas, N. (2010). A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *CoRR*.

Duvenaud, D., Nickisch, H., and Rasmussen, C. E. (2011). Additive gaussian processes. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 226–234, USA. Curran Associates Inc.

Frazier, P., Powell, W., and Dayanik, S. (2009). The Knowledge-Gradient Policy for Correlated Normal Beliefs. *INFORMS Journal on Computing*, 21(4):599–613.

Frazier, P. I. (2018). A Tutorial on Bayesian Optimization. *arXiv:1807.02811 [cs, math, stat]*.

Gardner, J., Guo, C., Weinberger, K., Garnett, R., and Grosse, R. (2017). Discovering and Exploiting Additive Structure for Bayesian Optimization. In *Artificial Intelligence and Statistics*, pages 1311–1319.

Györfi, L., Kohler, M., Krzyżak, A., and Walk, H. (2002). *A Distribution-Free Theory of Nonparametric Regression*. Springer Series in Statistics. Springer New York, New York, NY.

Hennig, P. and Schuler, C. J. (2012). Entropy Search for Information-efficient Global Optimization. *J. Mach. Learn. Res.*, 13(1):1809–1837.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential Model-Based Optimization for General Algorithm Configuration. In Coello, C. A. C., editor, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 507–523. Springer Berlin Heidelberg.

Hutter, F. and Osborne, M. A. (2013). A Kernel for Hierarchical Parameter Spaces. *arXiv:1310.5738 [cs, stat]*.

Jenatton, R., Archambeau, C., González, J., and Seeger, M. (2017). Bayesian Optimization with Tree-structured Dependencies. In *International Conference on Machine Learning*, pages 1655–1664.

Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492.

Kandasamy, K., Schneider, J., and Poczos, B. (2015). High Dimensional Bayesian Optimisation and Bandits via Additive Models. In *International Conference on Machine Learning*, pages 295–304.

Kim, J. and Choi, S. (2019). On Local Optimizers of Acquisition Functions in Bayesian Optimization. *arXiv:1901.08350 [cs, stat]*.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017). Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Artificial Intelligence and Statistics*, pages 528–536.

Lévesque, J., Durand, A., Gagné, C., and Sabourin, R. (2017). Bayesian optimization for conditional hyperparameter spaces. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 286–293.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2016). Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *arXiv:1603.06560 [cs, stat]*.

Ma, X., Rannen Triki, A., Berman, M., Sagonas, C., Cali, J., and Blaschko, M. B. (2019). A Bayesian optimization framework for neural network compression. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass.

Rolland, P., Scarlett, J., Bogunovic, I., and Cevher, V. (2018). High-dimensional bayesian optimization via additive models with overlapping groups. In Storkey, A. and Perez-Cruz, F., editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 298–307, Playa Blanca, Lanzarote, Canary Islands. PMLR.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2010). Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. *IEEE Transactions on Information Theory*, 58(5):3250–3265.

Swersky, K., Duvenaud, D., Snoek, J., Hutter, F., and Osborne, M. A. (2014). Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces. *arXiv:1409.4011 [stat]*.

The GPyOpt authors (2016). GPyOpt: A Bayesian optimization framework in python.

Wang, Z., Zoghi, M., Hutter, F., Matheson, D., and De Freitas, N. (2013). Bayesian Optimization in High Dimensions via Random Embeddings. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 1778–1784. AAAI Press.