High-Dimensional Bayesian Optimization via Additive Models with Overlapping Groups

Paul Rolland[1], **Jonathan Scarlett**[2], **Ilija Bogunovic**[1], **Volkan Cevher**[1]
[1] Laboratory for Information and Inference Systems (LIONS), EPFL
[2] Department of Computer Science & Department of Mathematics, National University of Singapore

{paul.rolland, ilija.bogunovic, volkan.cevher}@epfl.ch, scarlett@comp.nus.edu.sg

Abstract

Bayesian optimization (BO) is a popular technique for sequential black-box function optimization, with applications including parameter tuning, robotics, environmental monitoring, and more. One of the most important challenges in BO is the development of algorithms that scale to high dimensions, which remains a key open problem despite recent progress. In this paper, we consider the approach of Kandasamy *et al.* (2015), in which the high-dimensional function decomposes as a sum of lower-dimensional functions on subsets of the underlying variables. In particular, we significantly generalize this approach by lifting the assumption that the subsets are disjoint, and consider additive models with *arbitrary* overlap among the subsets. By representing the dependencies via a graph, we deduce an efficient message passing algorithm for optimizing the acquisition function. In addition, we provide an algorithm for learning the graph from samples based on Gibbs sampling. We empirically demonstrate the effectiveness of our methods on both synthetic and real-world data.

1 Introduction

Bayesian optimization (BO) is a powerful method for sequentially optimizing an unknown function f that is costly to evaluate, for which noisy point evaluations are available. Since its introduction, BO has successfully been applied to a variety of applications, including algorithm parameter tuning (e.g., deep neural networks) [1], [2], [3] and robotics [4], [5]. However, most successful applications of BO have involved low-dimensional input spaces. Efficiently scaling to high dimensions remains a key open challenge, and is crucial in applications such as computer vision [6], biology [7], and larger-scale parameter tuning.

High-dimensional BO comes with two key inter-related challenges [8]: Identifying "low-dimensional structure" in the high-dimensional function, and choosing an acquisition function that can efficiently be optimized. There is an inherent tension between these goals, with richer forms of structure often leading to acquisition functions that are harder to optimize.

1.1 Related Work

A recent overview of BO can be found in [9]. Most BO algorithms can be posed as choosing the next point to maximize an *acquisition function*, which in turn depends on the current posterior of the function. Popular choices include upper confidence bound (GP-UCB) [10], probability of improvement (PI) [11], expected improvement (EI) [12], [13], and (predictive) entropy search (ES) [14, 15]. In this paper, we are particularly interested in high-dimensional extensions of GP-UCB.

The earliest works on high-dimensional BO considered functions that only vary along a low-dimensional subspace [16], [17], [18]. While such approaches can be effective, this assumption on the function is rather strong, and achieving the full potential high-dimensional BO requires moving to richer classes. A promising alternative approach was recently proposed by Kandasamy *et al.* [19], who modeled the function by a sum of independent low-dimensional functions, each defined on a fixed subset of the underlying variables.

Crucially, the work of [19] assumed that these subsets are *disjoint*. This constraint considerably simplifies the optimization of the acquisition function, but restricts the space of functions that can be modeled. A generalization of this approach was proposed in [20] that allows for possible rotations within each term of

Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84. Copyright 2018 by the author(s).

the decomposition.

Within the framework of [19], a crucial challenge is learning the underlying additive decomposition from samples (typically while simultaneously performing optimization). In [19], this was done based on randomly sampling the decompositions and choosing one to maximize the likelihood. More efficient approaches were proposed by Wang *et al.* [21] based on Gibbs sampling, and by Gardner *et al.* [22] based on a simple Markov Chain Monte Carlo (MCMC) method.

Additive models, not necessarily making use of Gaussian processes, have also found extensive use in other contexts such as function learning, e.g., see [23, 24] and the references therein.

1.2 Contributions

The main contributions of this paper are as follows:

- We generalize the additive model of [19] by allowing the additive model to consist of functions defined on *general* subsets of the underlying variables that need not be disjoint.
- By representing the interactions between the groups (i.e., the subsets on which the low-dimensional functions are defined) using a graph, we deduce an efficient high-dimensional variant of GP-UCB based on message passing.
- We present a Gibbs sampling algorithm for learning the structure of the graph from data, having a similar flavor to that of [21] while addressing new challenges for the case of overlapping groups.
- We demonstrate the effectiveness of our approach on both synthetic and real data sets, including improved versatility compared to the work of [19].
- In the supplementary material, we generalize certain aspects of the mathematical analysis from [19], and discuss their possible implications towards providing regret bounds.

We very recently learned of a closely related independent parallel work [25] adopting a similar model, but using an *approximate* decentralized approach to optimize the acquisition function, and learning the graph via an alternative approach building on [22].

2 Generalized Additive GP Model

We consider the optimization of a D-dimensional function $f(x)$, where $x = (x_1, \dots, x_D)$ is the input vector. We focus on discrete domains, where each variable x_i takes values in some finite set $\mathcal{X}_i$ (though this set could represent the quantization of a real interval such as $[0, 1]$). Hence, the high-dimensional domain is $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_D$. Each time we query the function f at some point $x \in \chi$, we obtain a noisy observation $y = f(x) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \eta^2)$. We aim at maximizing this function over the domain $\mathcal{X}$, i.e. finding $x_{\text{opt}} = \arg\max_{x\in\mathcal{X}} f(x)$.

Additive structure: Following the work of Kandasamy *et al.* [19], we assume that the target function can be decomposed into a sum of low-dimensional components as follows:

$$f(x) = f^{(1)}(x^{(1)}) + f^{(2)}(x^{(2)}) + \dots + f^{(M)}(x^{(M)}), \quad (1)$$

where each $x^{(i)} \in \mathcal{X}^{(i)} \subseteq \mathcal{X}$ is a low-dimensional component, with $\mathcal{X}^{(i)}$ being the product of a small number of $\mathcal{X}_k$. The variables involved in this product are referred to as the *i-th group*, and this set of variables is denoted by $\mathcal{G}^{(i)} \subseteq \{1, \dots, D\}$.

In [19], it was assumed that the different variable sets $\mathcal{G}^{(i)}$ do not overlap, i.e. $\mathcal{G}^{(i)} \cap \mathcal{G}^{(j)} = \emptyset$ for all (i, j). In our setting, we allow for arbitrary overlaps between these variable sets, thereby permitting a significantly richer model class that is suited to *interacting* groups.

Prior and posterior: We assume that each term $f^{(i)}$ is an independent sample from a Gaussian process $\text{GP}(\mu^{(i)}, \kappa^{(i)})$. As a result, the overall target function is also a sample from a *GP*: $f \sim \text{GP}(\mu, \kappa)$ where

$$\mu(x) = \sum_{i=1}^{M} \mu^{(i)}(x^{(i)}) \quad (2)$$

$$\kappa(x, x') = \sum_{i=1}^{M} \kappa^{(i)}(x^{(i)}, x'^{(i)}) \quad (3)$$

Let $\mathcal{D}_t = \{(x_i, y_i)\}_{i=1}^{t}$ be the data observed from the target function f where $\mathbf{y} = (y_1, \dots, y_n)$ are the noisy observations corresponding to $\mathbf{x} = (x_1, \dots, x_n)$, i.e. $y_i \sim \mathcal{N}(f(x_i), \eta^2), i = 1, \dots, n$. Conditioned on these observations $\mathcal{D}_t$, we can infer the posterior mean and variance for each term $f^{(i)}$ at an arbitrary point x_*. We show in the supplementary material that $(f_*^{(j)}|\mathbf{y}) \sim \mathcal{N}(\mu_{t-1}^{(i)}, (\sigma_{t-1}^{(i)})^2)$, where

$$\begin{aligned} \mu_{t-1}^{(j)} &= \kappa^{(j)}(x_*^{(j)}, \mathbf{x}^{(j)})\Delta^{-1}\mathbf{y} \\ (\sigma_{t-1}^{(j)})^2 &= \kappa^{(j)}(x_*^{(j)}, x_*^{(j)}) \\ &\quad - \kappa^{(j)}(x_*^{(j)}, \mathbf{x}^{(j)})\Delta^{-1}\kappa^{(j)}(\mathbf{x}^{(j)}, x_*^{(j)}) \end{aligned} \quad (4)$$

with $\Delta = \kappa(\mathbf{x}, \mathbf{x}) + \eta^2 I_n \in \mathbb{R}^{n\times n}$. Here $\kappa^{(j)}(\mathbf{x}^{(j)}, x_*^{(j)})$ is a column vector of size n whose i-th entry is $\kappa^{(j)}(x_i^{(j)}, x_*^{(j)})$, and $\kappa(\mathbf{x}, \mathbf{x})$ is a matrix of size $n \times n$ whose (i, i')-th entry is $\kappa(x_i, x_{i'})$.

Dependency graph: An additive decomposition can be represented by a *dependency graph*. The dependency graph is built by joining variables i and j

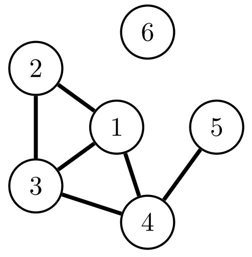

Figure 1: Example dependency graph. There are 4 maximal cliques: (1,2,3), (1,3,4), (4,5), and (6), each of which are associated to a term of the decomposition.

with an edge whenever they appear together within some set $x^{(k)}$. For example, the graph associated with the decomposition of $f(x) = f^{(1)}(x_1, x_2, x_3) + f^{(2)}(x_1, x_3, x_4) + f^{(3)}(x_4, x_5) + f^{(4)}(x_6)$ is shown in Figure 1. In the special case of [19], the dependency graph consists of disjoint fully connected components.

While different decompositions may lead to the same graph (e.g., consider the case of three functions on (x_1, x_2), (x_2, x_3), and (x_1, x_3) vs. a single function on (x_1, x_2, x_3)), one can always adopt the more general decomposition for a given graph to avoid any loss of generality (e.g., take the single function on (x_1, x_2, x_3)). To this end, one can let each low-dimensional function correspond to a single *maximal clique* (i.e., fully-connected component) of the graph.

3 Generalized Additive GP-UCB

3.1 Description of algorithm

Acquisition function: We assign to each term $f^{(i)}$ a low-dimensional UCB acquisition function [26]:

$$\phi_t^{(i)}(x^{(i)}) = \mu_{t-1}(x^{(i)}) + \beta_t^{\frac{1}{2}} \sigma_{t-1}(x^{(i)}), \tag{5}$$

where β_t is an exploration parameter. The global acquisition function is the sum of low-dimensional ones:

$$\phi_t(x) = \sum_{i=1}^{M} \phi_t^{(i)}(x^{(i)}). \tag{6}$$

The bulk of this section is devoted to methods for efficiently maximizing this function. While this is straightforward for disjoint groups [19], it is non-trivial in our generalized model.

Full algorithm: Our algorithm, which we call *generalized additive GP-UCB* (G-Add-GP-UCB), is given in Algorithm 1. As stated, the algorithm is suited to the case that the kernel (and hence the dependency graph) is known; this assumption is dropped in Section 4.

Algorithm 1 G-Add-GP-UCB

```
1: Pick $(x_t)_{i=1}^{N_{\text{init}}}$ at random; evaluate $f$ at these points
   to get $(y_t)_{i=1}^{N_{\text{init}}}$ and add these pairs to $D_0$.
2: for $t = 1, ..., N_{\text{iter}}$ do
3:     Perform Bayesian posterior updates conditioned
       on $D_{t-1}$ to obtain $\mu_{t-1}^{(j)}$, $\sigma_{t-1}^{(j)}$ for $j = 1, ..., M$
4:     Optimize the acquisition function (cf., Section
       3.2) to obtain $x_t \leftarrow \arg\max_{x \in \mathcal{X}} \tilde{\phi}_t(x)$
5:     Evaluate $x_t$ and observe $y_t \leftarrow f(x_t) + \epsilon$
6:     Augment the data set : $D_t = D_{t-1} \cup (x_t, y_t)$
7: end for
```

3.2 Maximizing the acquisition function

The key to maximizing the acquisition function (when the dependency graph is known) is to connect the optimization with the problem of maximizing probability in Markov random fields [27]. In particular, in the same way as the latter setting, we can make use of message passing for efficient maximization. We first explain how this is done when the dependency graph is triangulated (i.e., when the graph has no chordless cycles of length greater than 3), and then extend the algorithm to general graphs.

Triangulated dependency graphs: We start by constructing a junction tree [27] of the dependency graph. The nodes of the junction tree are then the different maximal cliques of the dependency graph. By the construction of the dependency graph in the previous section, there is a low-dimensional acquisition function $\phi^{(C)}$ of the form (5) associated to each of these maximal cliques C.

Starting from the leaves of the junction tree, we sequentially maximize, for each clique C, the associated low-dimensional acquisition functions $\phi^{(C)}$, and pass "messages" $m_{C_p \leftarrow C}$ to the parent node C_p. Each message is a function of the variables of the parent node, with the variables of the previous nodes already optimized. This process propagates up to the root, with a given node adding the messages from its children (*cf.*, Algorithm 2). The running intersection property of the junction tree ensures that there is no conflict between the maximizations performed, and that this algorithm returns the maximum of the global acquisition function [27].

Arbitrary dependency graphs: When the dependency graph is not triangulated, there is no junction tree that can be constructed from the original dependency graph. We thus first need to triangulate it, and construct a junction tree for the processed dependency graph. However, since the dependency graph has been modified, we no longer directly have a unique low-dimensional acquisition function for each clique of the

Algorithm 2 Message Passing algorithm on a triangulated dependency graph

```
1: Root the junction tree at a random node $C_R$.
2: $d \leftarrow$ depth of rooted tree G.
3: while $d \geq 1$ do
4:   for each node $C$ of the junction tree at distance $d$ from the root do
5:     $C_p \leftarrow$ parent(C), $I \leftarrow C \cap C_p$, $J \leftarrow C \setminus C_p$
6:     Compute messages to be passed to the parent node:
       $m_{C_p \leftarrow C}(x^{(I)}) = \max_{x^{(J)}} \phi^C(x^{(C)}) + \sum_{C_c \in \text{Children}(C)} m_{C \leftarrow C_c}(x^{(C \cap C_c)})$
       for each $x^{(I)} \in \chi^{(C)}$
7:   end for
8:   $d \leftarrow d - 1$.
9: end while
10: Return $\max_{x^{(C_R)}} \phi^{(C_R)}(x^{(C_R)}) + \sum_{C_c \in \text{Children}(C_R)} m_{C_R \leftarrow C_c}(x^{(C_R \cap C_c)})$.
```

junction tree. In order to ensure that each function $\phi^{(i)}$ is maximized once and only once during the optimization process, we assign to each clique C the following "acquisition function":

$$\phi^{(C)}(x^{(C)}) = \sum_{\substack{\text{clique } c \text{ of } G,\ c \subset C, \\ c \notin C_c\ \forall C_c \in \text{Children}(C)}} \phi^{(c)}(x^{(c)}), \qquad (7)$$

where G corresponds to the original dependency graph. By doing so, we can then apply Algorithm 2 on the triangulated graph.

Complexity: The complexity of running the message-passing algorithm on a junction tree J is exponential in the size of the maximum clique of the triangulated graph associated with junction tree J. This quantity depends on the chosen triangulation, so it would be desirable to compute a triangulation that yields a small maximal clique [28, 29].

4 Learning the Dependency Graph

One of the main important practical challenges of BO is choosing a suitable kernel. In the high-dimensional setting, this challenge is even more difficult, as we need to learn not only the kernel parameters (e.g., length scales), but also the structure associated with the high-dimensional function (i.e., the dependency graph). In this section, we present a Gibbs sampling procedure for this purpose, building on the approach of [21] for the case of non-overlapping groups.

Preliminaries: As discussed previously, any decomposition can be represented by an undirected graph, in which each low-dimensional kernel $\kappa^{(j)}$ is associated to a *maximal clique*. For convenience, here we represent this decomposition by an adjacency matrix $Z \in \{0,1\}^{D \times D}$, where $Z_{ij} = 1$ if variable i is connected to j in the graph, and 0 otherwise. This matrix is symmetric and has zeros on the diagonal, so the number of free parameters is $\frac{D(D-1)}{2}$.

The dependency graph defines the kernel decomposition, and thus influences the number of low-dimensional kernels and their dimensions. Therefore, in general, the kernel parameters can be defined only once the decomposition is known. In order to learn these parameters simultaneously with the additive structure, we assume that these kernels only depend on a constant number n_{param} of parameters *independent of the decomposition* (see Section 5 for a concrete example). We then group them into the set $L = \{L_i\}_{i=1,\dots,n_{\text{param}}}$. Overall, the parameters that must be optimized are $\theta = \{\{Z_{ij}\}_{1 \leq i < j \leq D}, \{L_j\}_{1 \leq j \leq n_{\text{param}}}\}$, which results in $\frac{D(D-1)}{2} + n_{\text{param}}$ parameters.

Maximum likelihood: In order to infer the structure of the dependency graph and the kernels' parameters, we seek to maximize data likelihood:

$$\log p(D_n|Z,L) = -\frac{1}{2} y^T \left(K_n^{G(Z),L} + \sigma^2 I\right)^{-1} y - \frac{1}{2} \log\left|K_n^{G(Z),L} + \sigma^2 I\right| - \frac{n}{2} \log 2\pi,$$

where $G(Z)$ is the graph corresponding to the adjacency matrix Z, y is the vector containing the current observations, and $K_n^{G,L} \in \mathbb{R}^{n \times n}$ is the kernel matrix of the observed data points, supposing a dependency graph G and kernel parameters $\{L_i\}_{1 \leq i \leq n_{param}}$, i.e., $(K_n^{G,L})_{ij} = \kappa^{G,L}(x_i, x_j)$. Here $\kappa^{G,L}$ represents the high-dimensional kernel determined by G and L.

Gibbs sampling: We adopt a Bayesian approach, in which we place a prior distribution on the parameters $\{\theta_i\}_{1 \leq i \leq N}$, and seek to sample from the posterior distribution $p(\theta_1, ..., \theta_N | D)$. Since we cannot directly sample from this high-dimensional probability distribution, we use the Gibbs sampling method (*cf.*, Algorithm 3). This provides a means for approximately sampling from the joint distribution, as long as we can sample from the 1-dimensional conditional distributions $p(\theta_i | \theta_{-i}, D)$ with $\theta_{-i} = \{\theta_j | j \neq i\}$.

The algorithm starts from a set of parameters $\theta^{(0)}$, and iteratively samples new sets of parameters $\theta^{(j)}$ by modifying one coordinate at a time based on the posterior (Algorithm 3). Once the sampling is performed, we choose the set of parameters that achieves the highest data likelihood. It can be shown [30] that under soft assumptions on the probability distribution $p(\theta|D)$, the Gibbs sampler tends to sample the same way as if we were to sample directly from $p(\theta|D)$.

Prior distributions: We model the variables

Algorithm 3 Structure learning via Gibbs Sampling

```
1: $\Theta \leftarrow \{\theta^{(0)}\}$, $\theta \leftarrow \theta^{(0)}$, $j \leftarrow 0$
2: for $j = 1, 2, \ldots$ do
3:     for $i = 1, ..., N$ do
4:         $\theta^{(j+1)} \leftarrow \theta^{(j)}$.
5:         Sample $\theta_i^{\text{new}}$ from $p(\theta_i | \theta_{-i}^{(j)}, D)$
6:         $\theta_i^{(j+1)} \leftarrow \theta_i^{\text{new}}$
7:         Augment the data set : $\Theta \leftarrow \Theta \cup \{\theta^{(j+1)}\}$
8:     end for
9: end for
```

$\{Z_{ij}\}_{1 \le i < j \le D}$ as Bernoulli random variables with parameter p: $Z_{ij} \sim \text{Bernoulli}(p)$ where p gives the probability of an edge joining variables i and j. This parameter p can be used to control the sparsity of the graph, or set to $\frac{1}{2}$ if no prior information is available.

Using this model, the posterior distribution for Z_{ij} is

$$\begin{aligned}
&p(Z_{ij} = 1 | Z_{-(ij)}, L, D_n; p) \\
&\propto p(D_n | Z_{-(ij)}, Z_{ij} = 1, L) \cdot p(Z_{ij} = 1 | Z_{-(ij)}, L; p) \\
&= p(D_n | Z_{-(ij)}, Z_{ij} = 1, L) \cdot p \\
&\propto p \cdot e^{\phi(Z_{-(ij)} \cup Z_{ij} = 1, L)}
\end{aligned}$$

with $\phi(Z, L) = -\frac{1}{2} y^T (K_n^{G(Z),L} + \sigma^2 I)^{-1} y - \frac{1}{2} \log \left| K_n^{G(Z),L} + \sigma^2 I \right|$. Concerning the kernels' parameters, we simply model them as uniform variables over pre-defined sets of possible values: $L_i \sim \text{Uniform}(\mathcal{L}_i)$ so that for each $l \in \mathcal{L}_i$, we have

$$p(L_i = l | Z, L_{-i}, D_n; p) \propto e^{\phi(Z, L_{-i} \cup \{L_i = l\})} \quad (8)$$

Using these posterior distributions, we can apply Gibbs sampling to the parameter set $\theta = \{\{Z_{ij}\}_{1 \le i < j \le D}, \{L_i\}_{1 \le i \le n_{param}}\}$, and select the set which produces the highest data likelihood. For the binary variables Z_{ij}, we simply compute $p_0 = (1 - p) \cdot e^{\phi(Z_{-(ij)} \cup \{Z_{ij} = 0\}, L)}$ and $p_1 = p \cdot e^{\phi(Z_{-(ij)} \cup \{Z_{ij} = 0\}, L)}$ and then sample from the binary distribution Bernoulli$(\frac{p_1}{p_0 + p_1})$. For the kernels' parameters, we proceed the same way by computing $e^{\phi(Z, L_{-i} \cup \{L_i = l\})}$ for values l in the set $\mathcal{L}_i$ and then sample from the normalized probability distribution.

Stopping criterion: We stop the process after some number N_{Gibbs} of data likelihoods have been computed. The more data we have, the better the algorithm will perform to find the true dependency graph, or a closely related one. Therefore, this learning process is repeated throughout the Bayesian optimization algorithm every N_{cyc} iterations, for some choice of N_{cyc}. Each time we learn new parameters (i.e., graph structure and kernel parameters), we start the sampling from the previously learned parameters.

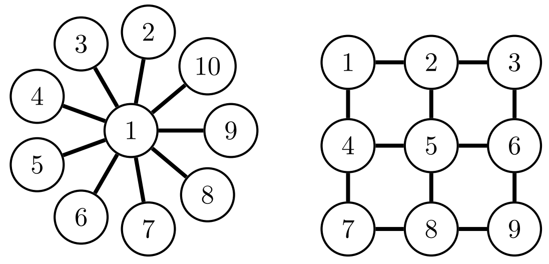

Figure 2: True dependency graphs for synthetic experiments. For both graphs, each edge forms a maximal clique, and thus corresponds to a term in the additive decomposition of the target function f. These graphs are not composed of disjoint cliques, and thus cannot be modeled by the constrained model without overlap.

5 Experiments

In this section, we experimentally compare our generalized algorithm against the one of [19]. We focus on squared exponential kernels for modeling the low-dimensional components of the target function:

$$\begin{aligned}
&\kappa^{(i), L^{(i)}}(x^{(i)}, x'^{(i)}) \\
&\qquad = \sigma^{(i)} \exp\left(-\frac{1}{2} (x^{(i)} - x'^{(i)})^T L^{(i)} (x^{(i)} - x'^{(i)}) \right)
\end{aligned}$$

The scales $\sigma^{(i)}$ are set to $\sigma^{(i)} = \frac{d_i}{\sum_j d_j}$, so that $\kappa(x, x) = 1$. The matrices $L^{(i)}$ are all diagonal and generated from one D-dimensional lengthscale vector l: $(L^{(i)})^{-1} = \text{diag}(l_{\chi^{(i)}})^2$ where $l_{\chi^{(i)}}$ is the vector containing the lengthscales of variables within $\chi^{(i)}$. This vector is then learned from the data via Gibbs sampling together with the dependency graph.

5.1 Experiments on synthetic data

We first test our algorithm on synthetic data by sampling functions from Gaussian processes, according to the dependency graphs shown in Figure 2. We use a squared exponential kernel with lengthscale matrix $L^{(i)} = 0.2 * I_2$. When applying Gibbs sampling use $p = \frac{1}{2}$ (i.e., no prior knowledge) for the sampling of the dependency graphs.

We apply our algorithm when considering 3 different cases, described as follows.

Overlap: In this case, we learn both the lengthscales and the dependency graph, and allow for overlaps between the groups in the function decomposition. Thus, any dependency graph is allowed, and junction trees are used when optimizing the acquisition function.

No Overlap: This case is similar to the previous one, except that we do not allow for overlaps between the groups in the function decomposition. This algorithm is thus the same as the one designed by Kandasamy *et al.* [19]. We place no "hard constraints" on the group sizes (i.e., restrictions on the maximum number of groups M and maximum group size $\mathrm{d}_{\max}$ therein), but we apply the Gibbs sampling procedure of [21], which only samples graphs that are sufficiently likely according to the samples.

Oracle: in the last situation, we assume that we know the true dependency graph and lengthscales, and hence we do not need to learn them throughout the algorithm. We expect this case to perform the best.

We also compare the results with a random algorithm that simply evaluates points at random. We perform 10 runs, starting from different initial situations where 10 points are chosen at random (for each run, these randomly chosen points are the same for each model), and then run the simulation for a certain number of iterations. All the parameters are the same for each model, and are summarized in Table 1.

$\beta_t^{(i)}$	$\mathrm{N}_{\mathrm{cyc}}$	$\mathrm{N}_{\mathrm{Gibbs}}$	$\max_{\mathrm{eval}}$
$0.5\log 2t$	30	200	1000

Table 1: Parameters for the synthetic BO examples.

Optimization performance: For each run and each iteration t, we compute the simple regret $S_t = \min_{i\leq t} r_i$ as well as the average cumulative regret $\frac{R_t}{t} = \frac{1}{t}\sum_{i=1}^t r_i$. We average these quantities over all trials (Figures 3 and 4).

We observe that the knowledge of the variable dependencies indeed greatly improves the performance of the algorithm. Around the beginning of the algorithm, the learning process does not perform well as it uses too few data. Therefore, there is no significant difference between the models "Overlap" and "No Overlap", while the "Oracle" is much more efficient as it uses the true dependency graph. As we get more data, the learning process improves and the "Overlap" model becomes more efficient compared to "No Overlap".

Graph learning accuracy: It is also interesting to assess the learning of the dependency graph throughout the algorithm. To do so, we define two quantities in order to evaluate how close a graph G is from a reference graph G_{true}. We first define the Correct Connections quantity:

$$\mathrm{CC}(G,G_{\text{true}}) = \frac{\#\text{ edges in both G and } G_{\text{true}}}{\#\text{ edges in } G_{\text{true}}},$$

which describes how well the variables connections in the true graph are represented by the learned graph. This quantity is between 0 and 1, and is equal to 1 if and only if all edges of the true graph are part of the edges of the learned graph. The second quantity that we define is the Correct Separation quantity:

$$\mathrm{CS}(G,G_{\text{true}}) = \frac{\#\text{ non-edges in both G and } G_{\text{true}}}{\#\text{ non-edges in } G_{true}}$$

which describes how well the variables separations in the true graph are represented by the learned graph. This quantity is also between 0 and 1, and is equal to 1 if and only if all edges of the learned graph are part of the edges of the true graph. It follows that $G_1 = G_2 \Leftrightarrow CC(G_1,G_2) = CS(G_1,G_2) = 1$.

In Figures 5 and 6, we plot these quantities as we obtain more data, and observe that the learning process steadily improves over time. However, in the "No Overlap" case, the graph constraint induces a saturation in the learning, while in the "Overlap" case, we observe converges to the true graph. We note that the high number of correct separations at early iterations is not an indicator of good performance, as the number of correct connections is very low.

We can observe that the time at which the "Overlap" model starts to be more efficient than the "No Overlap" one corresponds to the time where the dependency graph starts to be closer to the true graph.

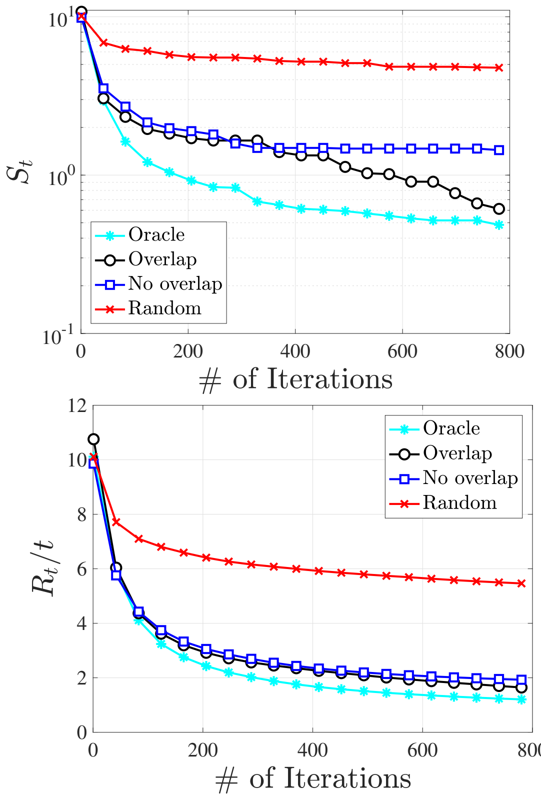

Figure 3: Star dependency graph: Simple and average cumulative regret averaged over 10 runs.

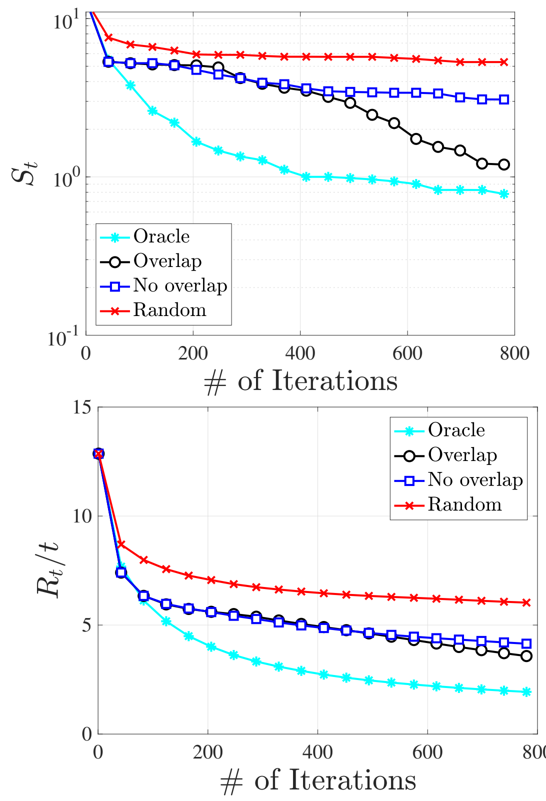

Figure 4: Grid dependency graph: Simple and average cumulative regret averaged over 10 runs.

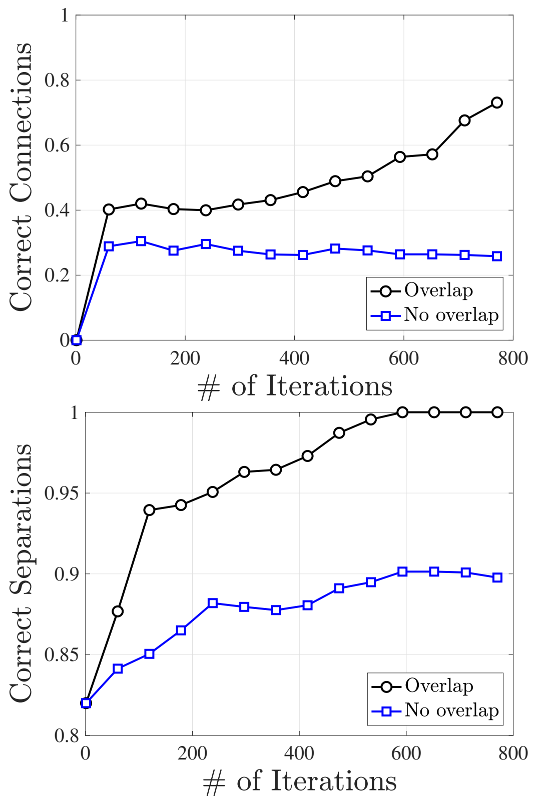

Figure 5: Star dependency graph: Correct Connections and Separations as a function of the number of data used for learning the graph.

5.2 Experiments on real data

Face recognition: We consider the optimization of parameters for the Viola and Jones (VJ) Cascade Classifier [31]. This algorithm aims at recognizing faces in images. It is based on a cascade of increasingly complex classifiers, which all detect particular features on the image to classify. At each stage, the classifier searches on the image for some features. If it does not find it, the image is rejected, directly classified and no more processing is performed. Otherwise, it continues to the next stage, and so on. It is thus necessary that the first stages have a low rate of false negatives.

Each classifier has a threshold parameter that controls this rejection. The problem is then to optimize these thresholds in order to obtain the highest possible classification accuracy. For each set of parameters, we evaluate the classification accuracy on 1500 images. Among these images, 1000 of them contain exactly one face, and the 500 others do not contain any face. The algorithm correctly classifies an image containing a face if it detects exactly one face. The classification accuracy is defined as the proportion of correctly classified images. We use the OpenCV implementation of the Cascade VJ algorithm [32], which uses a 22-stages cascade algorithm. OpenCV also provides an optimized set of thresholds, which gives a classification accuracy of 92.6% on our data set. When applying our Bayesian optimization algorithm to this problem, we choose as the domain a neighborhood of the optimized parameter set given by OpenCV, and set the target function as the classification accuracy.

Methods: As in the synthetic data example, we want to compare our "Overlap" model to the "No Overlap" one. In their paper [19], Kandasamy *et al.* applied their model to this same data set, and observed that the optimal extra parameters for this optimization problem were $M = 4$ and $\mathrm{d}_{\max} = 6$. We thus use these parameters for this model.

As discussed above, consecutive stages will have more similar rejection thresholds than stages that are far from each other. Therefore, we may suppose that two consecutive stages have more similar thresholds than stages that are far apart. This assumption is enforced by the fact that the set of parameters provided by OpenCV is increasing with the number of stages, justifying this possible correlation between close stages. Therefore, for our "Overlap" model, we set a dependency graph where each variable i is connected to variables $i-1$ and $i+1$ $\forall i = 2, ..., 21$, and do not apply the graph structure learning throughout the BO.

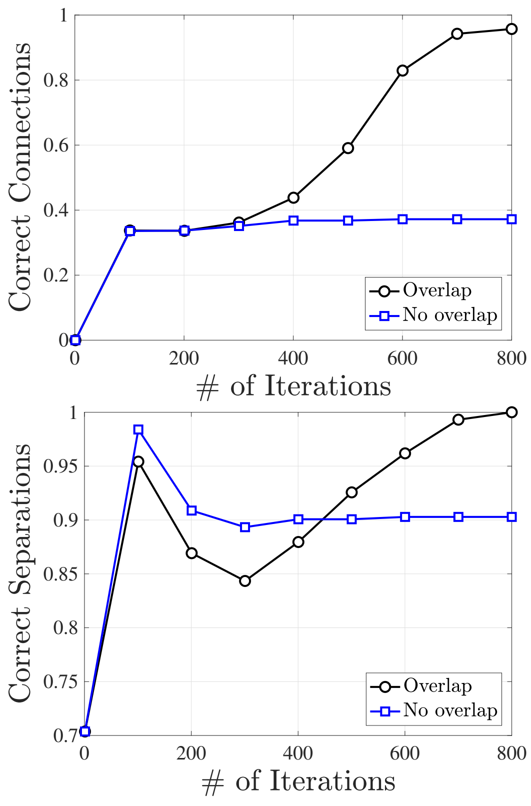

Figure 6: Grid dependency graph: Correct Connections and Separations as a function of the number of data used for learning the graph.

The Oracle method cannot be applied here since we do not know the true structure of the target function.

N_{iter}	$\beta_t^{(i)}$	N_{cyc}	N_{Gibbs}	$\max_{\text{eval}}$
200	$0.5 \log 2t$	50	300	2000

Table 2: BO parameters for face recognition data.

Optimization performance: We apply the two algorithms using the same set of parameters, described in Table 2. We also compare them to a random algorithm which queries random points on the same domain. The results are given in Figure 7, where we average over 15 runs.

We observe that the "Overlap" model performs slightly better than the "No Overlap" algorithm both in terms of convergence speed and optimal value. Both algorithms reach better classification accuracy than with the parameters provided by OpenCV. We can also observe that the "Overlap" model is much better in terms of the average cumulative regret.

Astrophysical data: In the supplementary material, we provide a second example for real-world data based on maximizing likelihood in an astrophysical model.

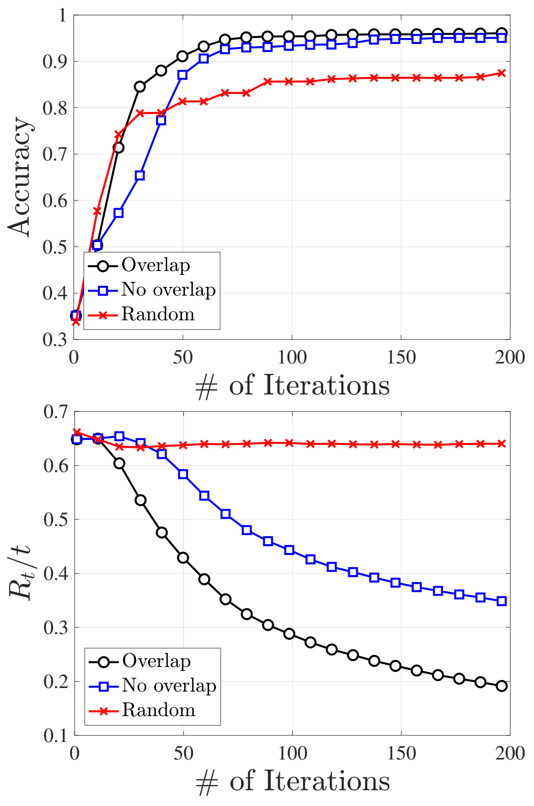

Figure 7: Face detection using VJ algorithm: classification accuracy and average cumulative regret.

6 Conclusion

We introduced a novel model and algorithm for high-dimensional Bayesian optimization, in which the target function to optimize can be represented by a sum of possibly overlapping low-dimensional components. By defining an acquisition function with the same additive structure, we can efficiently maximize it using message passing. In addition, we proposed a Gibbs sampling method for learning the structure from samples. By expanding the space of functions that can be modeled, we observed through experiments that the efficiency of Bayesian optimization is improved.

We have not discussed the theoretical aspects of Bayesian optimization (e.g., see [33]). In the supplementary material, we provide some mathematical analysis in this direction, including the calculation of the fundamental *information gain*, and discuss difficulties in attaining regret bounds for G-Add-GP-UCB.

An interesting direction for future research is to develop variants of message passing that are targeted to continuous settings. Our techniques can readily be applied by discretizing each continuous variable, but it would potentially be more effective (albeit non-trivial) to combine the message passing idea with a continuous global optimization procedure such as DiRect [34].

References

[1] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959. 2012.

[2] James Bergstra, R. Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Conference on Neural Information Processing Systems (NIPS)*, volume 24, Granada, Spain, December 2011.

[3] Nimalan Mahendran, Ziyu Wang, Firas Hamze, and Nando de Freitas. Adaptive MCMC with Bayesian optimization. *Journal of Machine Learning Research - Proceedings Track for Artificial Intelligence and Statistics (AISTATS)*, 22:751–760, 2012.

[4] Daniel Lizotte, Tao Wang, Michael Bowling, and Dale Schuurmans. Automatic gait optimization with Gaussian process regression. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 944–949, 2007.

[5] R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[6] D. Yamins Bergstra, J. and D. D. Cox. Making a Science of Model Search: hyper-parameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning (ICML), Atlanta, Gerorgia, 2013)*, 2013.

[7] David C. James Neil D. Lawrence Javier Gonzlez, Joseph Longworth. Bayesian optimization for synthetic gene design. *arXiv:1505.01627*, 2015.

[8] Nando. de Freitas. Talk on current challenges and open problems in Bayesian optimization. 2015.

[9] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE*, 104(1):148–175, 2016.

[10] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, 3:397–422, March 2003.

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

[12] Jonas Mockus. Application of Bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4(4):347–365, 1994.

[13] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 2010.

[14] Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *J. Mach. Learn. Research*, 13(1):1809–1837, 2012.

[15] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Adv. Neur. Inf. Proc. Sys. (NIPS)*, pages 918–926, 2014.

[16] Andreas Krause Bo Chen, Rui Castro. Joint optimization and variable selection of high-dimensional gaussian processes. *arXiv:1206.6396*, 2012.

[17] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, N Freitas, et al. Bayesian optimization in high dimensions via random embeddings. International Joint Conferences on Artificial Intelligence (AAAI), 2013.

[18] Josip Djolonga, Andreas Krause, and Volkan Cevher. High-dimensional Gaussian process bandits. In *Conference on Neural Information Processing Systems (NIPS)*, pages 1025–1033. 2013.

[19] Kirthevasan Kandasamy, Jeff G. Schneider, and Barnabas Poczos. High dimensional Bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning*, volume 37, pages 295–304, 2015.

[20] Chun-Liang Li, Kirthevasan Kandasamy, Barnabas Poczos, and Jeff Schneider. High dimensional Bayesian optimization via restricted projection pursuit models. In *International Conference on Artificial Intelligence and Statistics*, Cadiz, Spain, 2016.

[21] Zi Wang, Chengtao Li, Stefanie Jegelka, and Pushmeet Kohli. Batched high-dimensional Bayesian optimization via structural kernel learning. *arXiv preprint arXiv:1703.01973*, 2017.

[22] Jacob Gardner, Chuan Guo, Kilian Weinberger, Roman Garnett, and Roger Grosse. Discovering

and exploiting additive structure for Bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, Fort Lauderdale, FL, USA, 20–22 Apr 2017.

[23] Pradeep Ravikumar, Han Liu, John Lafferty, and Larry Wasserman. Spam: Sparse additive models. In *Conf. Neur. Inf. Proc. Sys. (NIPS)*. Curran Associates Inc., 2007.

[24] Hemant Tyagi, Anastasios Kyrillidis, Bernd Gärtner, and Andreas Krause. Algorithms for learning sparse additive models with interactions in high dimensions. *Information and Inference*, 00:1–67, August 2017.

[25] Trong Nghia Hoang, Quang Minh Hoang, Ruofei Ouyang, and Kian Hsiang Low. Decentralized high-dimensional bayesian optimization with factor graphs. http://arxiv.org/abs/1711.07033v3, 2018.

[26] Niranjan Srinivas, Andreas Krause, Matthias Seeger, and Sham M. Kakade. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML)*, pages 1015–1022, 2010.

[27] Martin J. Wainwright. *Graphical Models and Message-Passing Algorithms: Some Introductory Lectures*, pages 51–108. Springer International Publishing, Cham, 2015.

[28] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[29] Andrés Cano and Serafín Moral. Heuristic algorithms for the triangulation of graphs. *Advances in Intelligent Computing (IPMU)*, pages 98–107, 1995.

[30] G.O. Roberts and A.F.M. Smith. Simple conditions for the convergence of the Gibbs sampler and Metropolis-Hastings algorithms. *Stochastic Processes and their Applications*, 49(2):207 – 216, 1994.

[31] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features, 2001.

[32] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning OpenCV, 1st Edition.* O'Reilly Media, Inc., first edition, 2008.

[33] N. Srinivas, A. Krause, S.M. Kakade, and M. Seeger. Information-theoretic regret bounds for Gaussian process optimization in the bandit setting. *IEEE Trans. Inf. Theory*, 58(5):3250–3265, May 2012.

[34] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

[35] Carl Edward Rasmussen. Gaussian processes for machine learning. MIT Press, 2006.

[36] Kirthevasan Kandasamy, Jeff G. Schneider, and Barnabs Pczos. High dimensional Bayesian optimisation and bandits via additive models. *arXiv:1503.01673 [v3]*, 2017.

[37] MW. Seeger, SM. Kakade, and DP. Foster. Information consistency of nonparametric gaussian process methods. *IEEE Transactions on Information Theory*, 54(5):2376–2382, May 2008.