High-Dimensional Bayesian Optimization via Tree-Structured Additive Models

Eric Han,[1] **Ishank Arora,**[2] **Jonathan Scarlett**[1,3]
[1]School of Computing, National University of Singapore
[2]Indian Institute of Technology (BHU) Varanasi
[3]Department of Mathematics & Institute of Data Science, National University of Singapore
eric_han@nus.edu.sg, ishank.arora.cse14@iitbhu.ac.in, scarlett@comp.nus.edu.sg

Abstract

Bayesian Optimization (BO) has shown significant success in tackling expensive low-dimensional black-box optimization problems. Many optimization problems of interest are high-dimensional, and scaling BO to such settings remains an important challenge. In this paper, we consider generalized additive models in which low-dimensional functions with overlapping subsets of variables are composed to model a high-dimensional target function. Our goal is to lower the computational resources required and facilitate faster model learning by *reducing the model complexity* while retaining the *sample-efficiency* of existing methods. Specifically, we constrain the underlying dependency graphs to tree structures in order to facilitate both the structure learning and optimization of the acquisition function. For the former, we propose a hybrid graph learning algorithm based on Gibbs sampling and mutation. In addition, we propose a novel zooming-based algorithm that permits generalized additive models to be employed more efficiently in the case of continuous domains. We demonstrate and discuss the efficacy of our approach via a range of experiments on synthetic functions and real-world datasets.

1 Introduction

Bayesian Optimization (BO) is a widespread method for sequential global optimization (Snoek, Larochelle, and Adams 2012), and is suited to scenarios in which the target function f is unknown and expensive to evaluate. BO was traditionally used in model selection (Močkus 1975) and hyperparameter tuning (Snoek, Larochelle, and Adams 2012; Swersky, Snoek, and Adams 2013). Recently, BO has also found success in black-box adversarial attack (Ru et al. 2020), robotics (Jaquier et al. 2020), finance (Gonzalvez et al. 2019), pharmaceutical product development (Sano et al. 2019), natural language processing (Yogatama, Kong, and Smith 2015), and more. Two critical ingredients of BO include a model that captures prior beliefs about the objective function, and an acquisition function that can be optimized efficiently.

BO has been most successful in low dimensions (i.e. 10 or less) (Wang et al. 2013; Nayebi, Munteanu, and Poloczek 2019), whereas many applications require optimization in higher-dimensional spaces; this remains a critical problem in BO (Wang 2016; Rolland et al. 2018; Frazier 2018).

A key difficulty associated with high-dimensional BO is the curse of dimensionality (Spruyt 2014), namely, exponentially many observations are needed to find the global optimum in the absence of structural assumptions. Accordingly, two significant opposing challenges include the incorporation of suitable structural assumptions, and computationally efficient acquisition function optimization.

1.1 Related Work

In the literature, there are at least two approaches to high-dimensional BO with differing assumptions:

- Under *low effective dimensionality*, only few dimensions significantly affect f. (Chen, Castro, and Krause 2012) performed joint variable selection and optimization using GP-UCB. (Djolonga, Krause, and Cevher 2013) applied low-rank matrix recovery techniques to learn the underlying effective subspace, and (Zhang, Li, and Su 2019) proposed a related approach based on sliced inverse regression. (Wang et al. 2013) proposed REMBO, tackling the problem through random embedding. More recently, (Kirschner et al. 2019) proposed LineBO, decomposing the problem into a sequence of one-dimensional sub-problems. The use of *non-linear* low-dimensional embeddings has also recently been proposed (Lu et al. 2018; Moriconi, Kumar, and Deisenroth 2019).
- Under *additive structure*, small subsets of variables interact with each other. Specifically, additive models assume that f can be decomposed into sums of lower-dimensional functions. (Kandasamy, Schneider, and Póczos 2015) assumed that the variables constructing a particular lower-dimensional function are not present in the other decomposed functions (i.e., the variables of each function are pairwise disjoint), which we refer to as Graph No-Overlap. (Rolland et al. 2018) generalized the additive model to allow for an arbitrary dependency graph, removing the restriction of pairwise disjointness, which we refer to as Graph Overlap. Also considering overlapping groups, (Hoang et al. 2018) assumed that f can be decomposed into several sparse factor functions, allowing for distributed acquisition function approximation. (Li et al. 2016) generalized to a projected-additive assumption; the model proposed by (Kandasamy, Schneider, and Póczos 2015) is a special case when there is no projection. Ensemble BO (Wang et al. 2018) seeks to not only exploit addi-

tive structures, but also use an ensemble of GP models through a divide and conquer strategy. (Mutny and Krause 2018) combined additive GPs with approximations based on Fourier features, with the notable advantage of also establishing rigorous regret bounds.

In addition to the methods described above, other approaches have been taken to tackle high dimensionality. (Li et al. 2017) proposed a dropout strategy to optimize on a smaller subset of variables for every iteration. (Oh, Gavves, and Welling 2018) proposed BOCK, which tackles high-dimensionality via a cylindrical transformation of the search space. (Eriksson et al. 2019) proposed an approach based on running several local search procedures in parallel, and giving more samples to the most promising ones. Other methods use deep neural networks combined with BO, such as (Snoek et al. 2015; Cui, Yang, and Hu 2019).

The assumptions of low effective dimension vs. additive structure are complementary. The performance of the optimization is dependent on the structure of the high-dimensional function, and trade-offs exist between computation time and accuracy. Methods that assume low effective dimensionality are often computationally faster than additive methods; for example, due to scalability concerns, (Eriksson et al. 2019) omitted methods that attempt to learn an additive decomposition from their experiments. To the best of our knowledge, none of the existing works have scaled Graph Overlap past 20 to 30 dimension.

In this paper, our focus is on additive structures; in particular, we seek to build on Graph Overlap. Graph Overlap maintains computational tractability by using a message passing algorithm to optimize the acquisition function efficiently. However, the message passing algorithm runs exponentially in the size of the maximum clique of the triangulated dependency graph (Rolland et al. 2018), impeding its scalability.

We see in the above-outlined works (Rolland et al. 2018; Hoang et al. 2018; Li et al. 2016) that the trend in the study of additive models has been to increase the model expressiveness. An important caveat to such approaches is that a suitable model may be *much harder to find* given limited samples. Since one of the main premises of BO is optimizing with few samples, we contend that *simpler* models should also be sought to facilitate model learning with fewer samples, as well as reduced computation.

1.2 Contributions

The main contributions of this paper are as follows:

1. We trade-off expressiveness for scalability and ease of learning by reducing the complexity of the additive model, constraining the dependency structure to tree structures. As the function class is simpler, it reduces overfitting of the GP kernel, and we are also able to reap computational efficiencies in both acquisition function optimization and dependency structure learning.
2. We propose a zooming technique for extending the message passing algorithm of (Rolland et al. 2018) to continuous domains, thus benefiting additive methods in general, and in particular our tree-based approach.
3. We propose a hybrid method to learn the additive tree structures, composed of the following two techniques:
   (a) a tree structure growing algorithm that efficiently discovers edges that do not form cycles via Gibbs sampling;
   (b) an edge mutation algorithm that obtains a new generation of trees from the current tree efficiently.
4. Although limiting to tree structures may seem potentially risky due to the reduced expressivity, we show this approach to be highly effective in a wide range of experiments, indicating a highly competitive trade-off between expressive power and ease of model learning.

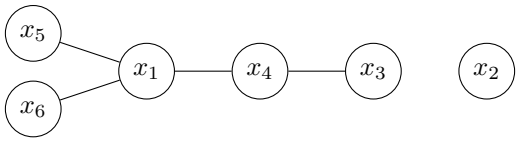


Figure 1: Dependency tree structure, $h(x) = h^A(x_1, x_6) + h^B(x_1, x_5) + h^C(x_1, x_4) + h^D(x_3, x_4) + h^E(x_2)$.

We briefly mention that the use of tree structures in BO appeared in prior works (Jenatton et al. 2017; Ma and Blaschko 2020), but with a very different type of model and motivation. These works aim to handle structured domains instead of real-valued domains, and in contrast with our work, the tree represents binary decisions with only the leaves corresponding to Gaussian Processes.

2 Additive GP-UCB using Tree Structures

We consider the sequential global optimization problem, seeking $x_{\max} = \arg\max_{x \in \mathcal{X}} f(x)$ for a D-dimensional black-box function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} = \times_{i=1}^{D} \mathcal{X}_i$ with each $\mathcal{X}_i$ being an interval in $\mathbb{R}$. At the t-th observation, the algorithm selects x_t and observes a noisy observation $y_t = f(x_t) + \epsilon_t$, with $\epsilon_t \sim \mathcal{N}(0, \eta^2)$.

2.1 Additive Dependency Tree Structures

We use a Gaussian Process (GP) model to reason about the target function f. Following (Rolland et al. 2018), we model f as a sum of several lower-dimensional components:

$$f(x) = \sum_{G \in \mathcal{G}} f^G\left(x^G\right), \tag{1}$$

where $G \subseteq \{1, \ldots, D\}$ denotes one subset of variables, and $\mathcal{G}$ represents the additive structure (see Fig. 1 for an example). The additive dependency structure is assumed to be tree-structured, possibly including forests. In contrast with (Rolland et al. 2018), in our setting the additive structure associated with any given graph is unique: Each lower-dimensional component $f^G : \mathcal{X}^G \to \mathbb{R}$ is either a 1 or 2-dimensional function defined on the variables in G, where $\mathcal{X}^G = \times_{v \in G} \mathcal{X}_v$. Each edge represents a 2-dimensional function, and each disconnected vertex represents a 1-dimensional function.

2.2 Prior and Posterior

We model $f \sim \mathcal{GP}(\mu, \kappa)$, with each f^G being an independent sample from a Gaussian Process $\mathcal{GP}\left(\mu^G, \kappa^G\right)$, and

$$\begin{aligned} \mu(x) &= \sum_{G \in \mathcal{G}} \mu^G\left(x^G\right), \\ \kappa\left(x, x'\right) &= \sum_{G \in \mathcal{G}} \kappa^G\left(x^G, x'^G\right). \end{aligned} \tag{2}$$

We know from (Rolland et al. 2018) that the posterior can be inferred via $\left(f_*^G \mid \mathbf{y}\right) \sim \mathcal{N}\left(\mu_{t-1}^G, \left(\sigma_{t-1}^G\right)^2\right)$ for each f_*^G at an arbitrary point x_* given $\mathcal{D}_t = \{(x_i, y_i)\}_{i=1}^t$, where $\mathbf{y} = (y_1, \ldots, y_t)$ correspond to $\mathbf{x} = (x_1, \ldots, x_t)$, and the posterior mean and variance are given by

$$\begin{aligned} \mu_{t-1}^G &= \kappa^G\left(x_*^G, \mathbf{x}^G\right) \Delta^{-1} \mathbf{y}, \\ \left(\sigma_{t-1}^G\right)^2 &= \kappa^G\left(x_*^G, x_*^G\right) \\ &\quad - \kappa^G\left(x_*^G, \mathbf{x}^G\right) \Delta^{-1} \kappa^G\left(\mathbf{x}^G, x_*^G\right). \end{aligned} \tag{3}$$

Here we define the matrix $\Delta = \kappa(\mathbf{x}, \mathbf{x}) + \eta^2 I_t \in \mathbb{R}^{t \times t}$, $\kappa(x_i, x_j)$ is the (i, j)-th entry of $\kappa(\mathbf{x}, \mathbf{x})$, and $\kappa^G\left(\mathbf{x}^G, x_*^G\right)$ is of length t, with i-th entry $\kappa^G\left(x_i^G, x_*^G\right)$.

3 Additive GP-UCB on Tree Structures

Algorithm 1: TREE-GP-UCB

```
Initialize D_0 ← {(x_t, y_t)}_{x_t ∈ X_init}
for t = N_init + 1, ..., N_iter do
    if t mod C = 0 then
        Learn G ← TREE-LEARNING (Alg. 3)
    Update μ_t^G, σ_t^G : ∀G ∈ G (3)
    Optimize x_t ← arg max_{x∈X} φ_t(x) (Alg. 2)
    Observe y_t ← f(x_t) + ε
    Augment D_t ← D_{t-1} ∪ {(x_t, y_t)}
return arg max_{(x,y)∈D} y
```

In Alg. 1, we present Tree-GP-UCB (Tree for short). Here, the total number of observations is $N = N_{\text{init}} + N_{\text{iter}}$, where N_{init} is the number of initial random samples X_{init} drawn uniformly from $\mathcal{X}$ and N_{iter} is the number of iterations. For efficiency, $\mathcal{G}$ and its hyperparameters are learned every C iterations, for some $C > 0$.

3.1 Acquisition Function

We focus on upper confidence bound (UCB) based algorithms (Auer 2002; Srinivas et al. 2010). Specifically, following (Kandasamy, Schneider, and Póczos 2015) and (Rolland et al. 2018), we let the global acquisition function $\phi_t(x)$ be the sum of the individual acquisition functions with respect to the dependency structure $\mathcal{G}$:

$$\begin{aligned} \phi_t(x) &= \sum_{G \in \mathcal{G}} \phi_t^G\left(x^G\right), \\ \phi_t^G\left(x^G\right) &= \mu_{t-1}^G\left(x^G\right) + \beta_t^{1/2} \sigma_{t-1}^G\left(x^G\right). \end{aligned} \tag{4}$$

Maximization over Continuous Domains. The message passing approach proposed by (Rolland et al. 2018) works on discrete domains. A naive approach to handle continuous domains would be to discretize the continuous domain uniformly (i.e., a grid with equal spacing). However, this may require large amounts of computation, especially when the discretization is performed using a small spacing. Here, we present a refined message passing algorithm specifically designed for continuous domains.

Algorithm 2: MSG-PASSING-CONTINUOUS

```
Initialize (a, b) with the bounds of X
for l = 1, ..., L do
    for d = 1, ..., D do
        Discretize X_d ← [[a_d, b_d]]_R    // |X_d| = R
    X ← ×_{d=1}^D X_d
    (x, y) ← MSG-PASSING-DISCRETE (X)
    Select (a, b) ← ZOOM-STRATEGY (x)
return (x, y)
```

The optimization of the acquisition function over continuous domains is presented in Alg. 2; it starts with the full continuous domain $\mathcal{X} = \times_{d=1}^D \mathcal{X}_d$, where $\mathcal{X}_d \in [a_d, b_d] \subseteq \mathbb{R}$. Firstly, we discretize each variable's domain to a finite subset, and let R denote the size of the subset. Thereafter, we use a simplified version of the message passing algorithm MSG-PASSING-DISCRETE of (Rolland et al. 2018)—Alg. S1 in the appendix—to perform optimization over the discretized domain. As the dependency graph is a tree, the complexity of message passing is quadratic in R. The bounds $(\mathbf{a}, \mathbf{b})$ for the next level are picked by ZOOM-STRATEGY (see below) given the selected point. We perform the steps iteratively for some number L of levels.

Zoom Strategy. Different strategies can be employed in choosing the bounds and their representative points for the next level. We adopt a simple randomized strategy exemplified in Fig. 2: At each level, we partition each current interval $[a_i, b_i]$ uniformly onto a grid of size R, and choose a uniformly random point within that interval as its representative. We refer to this discretization of the domain as $[[a_i, b_i]]_R$. We use MSG-PASSING-DISCRETE restricted to these representatives, and for the one chosen, we recursively zoom into the corresponding sub-domain. Henceforth, we use MSG-PASSING-CONTINUOUS with this zoom strategy.

3.2 Additive Components

The choice of an appropriate kernel and the learning of its parameters are critical to the success of BO. In high-dimensional additive BO, the problem compounds, as we need to learn the dependency structure along with kernel parameters for every kernel in the additive model.

As mentioned previously, an additive decomposition $\mathcal{G}$ corresponds to a dependency graph; the additive function $f(x) = \sum_{G \in \mathcal{G}} f^G\left(x^G\right)$ is the sum over its additive components in $\mathcal{G}$. It will be convenient to work with the equiva-

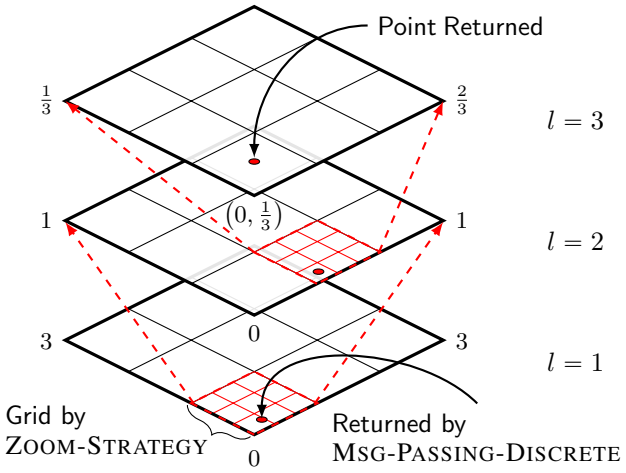


Figure 2: Example with two variables, grid size $R = 3$, and domain $[0, 3]$. Firstly, we partition each axis evenly into 3 partitions. Next, we draw a uniformly random point from each partition. The points from each axis form a discretized domain, and we run MSG-PASSING-DISCRETE on this discrete domain. Finally, we zoom into the square, representative of the selected point. In this manner, we recursively sub-divide the grid for all $L = 3$ levels.

lent representation of an adjacency matrix $Z \in \{0, 1\}^{D \times D}$, where $Z_{ij} = 1$ if variables i and j are connected on the (tree-structured) graph. Assuming that each function's kernel κ^G is parameterized by some kernel parameters θ^G (e.g., lengthscale etc.), the overall collection of parameters is $\Theta_{\mathcal{G}} = \left\{\theta^G\right\}_{G \in \mathcal{G}}$ given a decomposition $\mathcal{G}$. We note that learning the kernel parameters along with the decomposition $\mathcal{G}$ is difficult, as the search space is large and we may encounter problems with overfitting. We tackle this problem by defining a fixed set of dimensional kernel parameters Θ that are independent of the decomposition and defining the kernel parameters over them; see Sec. 4.1 for details.

Maximum likelihood. For model learning, we make use of the maximum log-likelihood score, given by

$$\rho(Z, \theta) = -\frac{1}{2}\mathbf{y}^T \left(K + \eta^2 I\right)^{-1} \mathbf{y} - \frac{1}{2}\log|K + \eta^2 I| - \frac{n}{2}\log 2\pi, \tag{5}$$

where $K \in \mathbb{R}^{n \times n}$ is the kernel matrix of the observed data points n, assuming a dependency graph $\mathcal{G}$ with an equivalent adjacency matrix Z and parameters θ.

Dependency Structure Learning. Following (Wang et al. 2017; Rolland et al. 2018), we adopt a Bayesian approach to structure learning, on which we place a prior distribution on Z and seek to sample from the posterior distribution. We use Gibbs sampling to sample approximately, avoiding the difficult task of sampling directly from the high-dimensional distribution over tree structures.

Specifically, we use such sampling to update the presence/absence of edges from variable i to j, but to maintain the tree structure, we discard edges that would create a cycle. We assume a prior with Bernoulli random variables with parameter γ, $Z_{ij} \sim \text{Bernoulli}(\gamma)$. We can use this model to formulate the posterior for Z_{ij}; letting $\mathcal{D}$ denote the data collected, and letting $Z_{-(ij)}$ be the adjacency variables excluding (i, j), we have the following (Rolland et al. 2018):

$$P\left(Z_{ij} = 1 \mid Z_{-(ij)}, \theta, \mathcal{D}; \gamma\right) \propto \gamma\, e^{\rho\left(Z_{ij}=1 \cup Z_{-(ij)}, \theta\right)}. \tag{6}$$

For each Z_{ij}, we compare the log of the posterior for two cases: $\log(\gamma) + \rho(Z_{ij} = 1 \cup Z_{-ij}, \theta)$ vs. $\log(1 - \gamma) + \rho(Z_{ij} = 0 \cup Z_{-ij}, \theta)$. The parameter γ can be set to $1/2$ if there is no prior information about Z. We use the log-likelihood in two ways, combining them to learn the structure in Alg. 3. First, we use Gibbs Sampling to build a connected tree from an empty graph iteratively. Once the dependency graph is a connected tree, we apply mutation in subsequent iterations. Thus, we grow the empty graph into a tree and then seek improvements via mutation.

Algorithm 3: TREE-LEARNING

```
Z ← {Z^current}
Z^(k) ← Z^current
while k < S do
    if NUMBER-OF-EDGES(Z^(k)) < D − 1 then
        Update (Z, k) via GIBBS-SAMPLING (Alg. 4)
    else
        Update (Z, k) via MUTATION (Alg. 5)
return Z ∈ Z with the highest likelihood score
```

Adding Edges. Alg. 4 samples from the marginal posteriors, while only adding edges that maintain that Z is still a tree. The Union-Find (UF) data structure tracks a set of disjoint sets, providing the operations *union* and *find*. Both operations can be performed in (amortized) time $O(\alpha(D))$ when implemented using weights with path compression (Cormen et al. 2009; Sedgewick and Wayne 2011), where $\alpha(D)$ is the inverse Ackermann function. In short, both operations can be performed in nearly constant time (amortized). In our algorithm, we use UF to track the connected components of $\mathcal{G}$, represented by disjoint subsets of variables. We use the find operation to check for cycles. After adding the edge, we update UF by performing the union operation.

Mutation. Alg. 5 describes the mutation operation that we perform when the dependency graph $\mathcal{G}$ is a connected tree. We borrow the idea of mutation from genetic algorithms; the mutation operation can maintain tree structure diversity from one generation to another. The purpose of the mutation operation is to preserve and introduce diversity, wherein genetic algorithms, a mutation helps to avoid getting stuck in local maxima by making minor changes to the previous generation. In our context, the population is a new generation of trees in

Algorithm 4: GIBBS-SAMPLING at k-th iteration

Initalize UF data structure
for $j = 1, \ldots, D$ **do**
  for $i = 1, \ldots, j-1$ **do**
    $Z^{(k+1)} \leftarrow Z^{(k)}$
    if *cycle not formed by* $Z_{ij}^{(k+1)} = 1$ **then**
      Sample $Z_{ij}^{(\text{new})}$ from posterior
      $Z^{(k+1)} \leftarrow Z_{ij}^{(\text{new})}$
      Update UF via union operation
      Add $\mathcal{Z} \leftarrow \mathcal{Z} \cup \left\{Z^{(k+1)}\right\}$
    $k \leftarrow k+1$

each iteration, and the fitness function is the log-likelihood. Using mutation, we can simultaneously avoid local maxima and efficiently maintain a tree structure.

We note that one could simply use the Gibbs sampling approach or the mutation approach separately rather than using the former followed by the latter, but we found this combined approach to be effective experimentally.

Algorithm 5: MUTATION at k-th iteration

$Z^{(k+1)} \leftarrow Z^{(k)}$
$i, j \leftarrow$ Sample random edge for which $Z_{ij}^{(k+1)} = 1$
Remove edge: $Z_{ij}^{(k+1)} = 0$
$i', j' \leftarrow$ Sample nodes from the disconnected sub-trees
Sample $Z_{i'j'}^{(\text{new})}$ using posterior
$Z^{(k+1)} \leftarrow Z_{i'j'}^{(\text{new})}$
Augment the dataset: $\mathcal{Z} \leftarrow \mathcal{Z} \cup \left\{Z^{(k+1)}\right\}$
$k \leftarrow k+1$

4 Experimental Results

For each of our experiments,[1] we compare our method, Tree, to several state-of-the-art black-box global optimization methods, particularly BO methods including Graph No-Overlap (Kandasamy, Schneider, and Póczos 2015), Graph Overlap (Rolland et al. 2018), LineBO (Kirschner et al. 2019), and REMBO (Wang et al. 2013).

To avoid clutter, we avoid including every algorithm and baseline in our charts. Instead, we make an effort to compare against the *best* algorithm for the function at hand, leveraging on prior works' experimental results to complete our discussion. For example, in cases where LineBO was already shown to outperform standard GPs and REMBO in (Kirschner et al. 2019), we omit these worse-performing methods.

We run all additive methods using the zooming-based message passing algorithm, analogous to Alg. 2. In addition, we compare to Random, which evaluates points at random. Where possible, we also compare our results to Oracle, which has access to the true dependency graph along with the true kernel parameters. The functions and data sets considered are summarized in Table S2 in the appendix.

4.1 Setup

Whenever possible, we used identical parameters across all competing algorithms and functions. However, we note that most algorithms have unique hyperparameters. We set those hyperparameters to reasonable values, discussed in the appendix. The competing algorithms and their unique hyperparameters are given in Table S1 in the appendix. We ran each algorithm 25 times for every function with varied conditions.[2] We ran all experiments with $N_{\text{init}} = 10$ initial points and $N_{\text{iter}} = 1000$ total points. The same conditions are used across all algorithms to ensure a fair comparison.

Kernel. We adopt the widely-used Radial Basis Function (RBF) kernel, more specifically using a variant known as the RBF-ARD kernel (Murphy 2012), which consists of a dimensional lengthscale ℓ_i for every dimension i. In addition, we decompose the scale parameter $\sigma^G = \sqrt{\sum_{i \in G} {\sigma_i}^2}$ into its dimensional components σ_i, so that we can learn the parameters tractably. Each low-dimensional kernel corresponds to a low-dimensional function with set of variables G:

$$\kappa_{\text{RBF}}^G\left(x, x'\right) = \sigma^G \exp\left(-\frac{1}{2}\sum_{i \in G}\frac{(x_i - x_i')^2}{\ell_i^2}\right). \quad (7)$$

In this manner, the kernel parameters $\Theta_{\mathcal{G}}$ are defined over the dimensional kernel parameters $\Theta = \{(\ell_i, \sigma_i)\}_{i=1}^D$. We adopt the established gradient-based approach to learning Θ; see Sec. S1 in the appendix. We initialize the dimensional lengthscale and scale parameters as $\sigma_i = 0.5$, and $l_i = 0.1$ for all i. We set $\eta = 0.1$ in (3) to account for noisy observations.

Additive Models. All additive models start with an empty graph of the appropriate size for the given function. Concerning the learning of the dependency structure, we assume no prior knowledge ($\gamma = 0.5$). We sample the structure for $S = 250$ times every $C = 15$ iterations. After learning the structure, we choose the best kernel parameters using the gradient approach mentioned above. We set the trade-off parameter in UCB to be $\beta(t) = 0.5\log(2t)$, as suggested in (Rolland et al. 2018). For *discrete* experiments, we discretize each dimension to 50 levels, with the maximum number of individual acquisition function evaluations capped at 1000. For *continuous* experiments, we let each level's grid size be $R = 4$ and the number of levels be $L = 4$ (see Fig. 2) with no maximum evaluation limits.

4.2 Metrics

Following (Wang et al. 2013), we plot the mean and $1/4$ standard deviation confidence intervals of the metrics over all

[1] The code is available at https://github.com/eric-vader/HD-BO-Additive-Models.

[2] Conditions include initial points, instances of the objective function, and random seeds used by the algorithm.

25 runs of the algorithm. For convenience, each plot's legend is ordered according to the curves' final y-value.

Graph Learning Performance. We measure how close the estimate G is from its target graph G_{opt} by calculating

$$\mathrm{F}_1\text{score}\,(G) = 2\frac{\text{Precision}\,(G) \times \text{Recall}\,(G)}{\text{Precision}\,(G) + \text{Recall}\,(G)}, \quad (8)$$

where $\text{Precision}\,(G) = \frac{|\text{Edges}\,(G) \cap \text{Edges}\,(G_{\text{opt}})|}{|\text{Edges}\,(G)|}$ and $\text{Recall}\,(G) = \frac{|\text{Edges}\,(G) \cap \text{Edges}\,(G_{\text{opt}})|}{|\text{Edges}\,(G_{\text{opt}})|}$, with $\text{Edges}\,(G)$ denoting the set of edges in graph G. A larger $\mathrm{F}_1\text{score}$ indicates better graph learning performance.

Optimization Performance. In accordance with the ultimate goal of BO, we compute the *best regret* to measure closeness to the best value $f_{\max}$ at iteration i:

$$R_t = f_{\max} - f_i^*, \quad (9)$$

where f_i^* denotes the best $f\,(x)$ value sampled up to iteration i. For functions where $f_{\max}$ is unknown, we instead consider f_i^*, i.e., the *best value* found.

Discussion on Computation Time. In general, it is difficult to compare the amount of computational resources by various algorithms, as it is very much affected by many factors such as implementation, hardware, underlying GP backends, etc. However, we provide a brief discussion of the general trends observed. We generally found LineBO to be one of the faster approaches due to the use of 1D subroutines, though we also found its optimization perform to be limited in several cases. A fairly similar discussion applies to REMBO. On the other hand, the computational requirements of the *additive* methods are somewhat easier to compare in a fair manner, as we now discuss.

Cost Efficiency. For the additive methods, we compute the *Message Passing Cost* counting the number of individual acquisition function (ϕ^G) evaluations; see (S4) in the appendix. This metric is a proxy for the computational resources used in the optimization of the acquisition function. While it may not always correspond exactly to the total computation time, we expect that each message passing operation for Tree is *at least* as fast as in Graph No-Overlap and Graph Overlap. This is because Tree only works with functions containing only one or two variables, whereas the others may contain a larger number of variables.

4.3 Experiments with Additive GP Functions

We first compare Tree to other additive methods for functions drawn from a GP with additive structure. We focus our discussion on understanding the additive methods' scaling ability and performance. Afterwards, we compare Tree to other methods using various non-GP functions. On all synthetic experiments, we add Gaussian $\mathcal{N}\left(0, 0.15^2\right)$ noise to simulate noise that occurs in real-world applications.

Similar to (Rolland et al. 2018), we test our algorithm on synthetic data by sampling functions from GPs with several different additive dependency structures. We use an RBF kernel with corresponding dimensional lengthscale and scale parameters set to $\sigma_i^{\text{opt}} = 1$ and $l_i^{\text{opt}} = 0.2$ for all i. We tested several dependency structures; three notable examples are illustrated in Fig. 4, and a full list is given in the appendix.

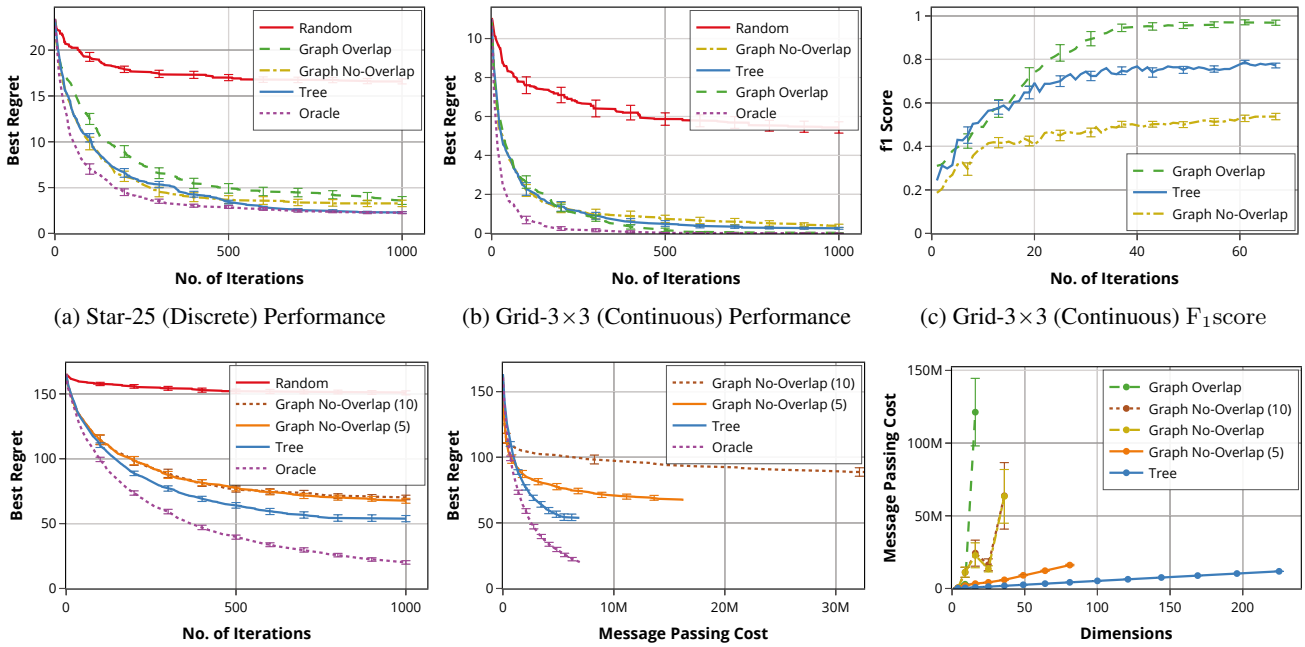
In Fig. 3a, it is unsurprising that Tree outperforms the other additive methods for Star-25. The dependency graph of Star-25 is indeed a tree, enabling our method to be effective in learning the dependency structure. From Fig. S3e in the appendix, by plotting $\mathrm{F}_1\text{score}$ over iterations, we observe that it is efficient in learning the dependency structure. The dependency structure learned by Tree is closest to the ground-truth throughout the experiment, when compared with other additive models. This efficiency is also reflected in Fig. S3f, where Tree achieves the best performance as a function of the message passing cost. We additionally demonstrate in the appendix that the reduction in cost becomes significantly higher in the case of continuous domains, achieving better performance return on cost than other additive methods.

Next, we turn to the case that the underlying graph is not a tree. Fig. 3b-3c corresponds to the Grid-3×3 structure, and we find that Graph Overlap performs the best in terms of learning the dependency structure. This is because, for the grid graph model, only Graph Overlap's underlying structural assumptions are correct. Both Tree and Graph No-Overlap face difficulty learning the graph accurately, albeit worse for Graph No-Overlap. Interestingly, Tree still remains competitive in terms of optimization performance despite poorer graph learning. That is, when Tree makes errant connections (or errant non-connections) in the dependency graph, the performance does not degrade significantly, and the algorithm can tweak other parameters (e.g., σ_i and l_i) to minimize the effect of any errant connections. From Fig. 3b, despite all additive algorithms being mutually competitive in terms of regret, both Graph No-Overlap and Graph Overlap needed more acquisition function evaluations to achieve the same performance as Tree (more than triple for Graph No-Overlap); see Fig. S4l in the appendix. In this instance, Graph No-Overlap's pairwise disjoint assumption not only results in worse graph learning, but also worse cost efficiency. Next, we compare Graph No-Overlap and Tree using an Ancestry-132 dependency structure (132D).[3] We found that Graph Overlap was unable to complete such high-dimensional experiments in a reasonable time. For Graph No-Overlap to work efficiently, we limited the maximum clique size, consider limits of both 5 and 10, represented by Graph No-Overlap (5) and Graph No-Overlap (10) respectively. In Fig. 3d-3e, we find Tree performing best in high-dimensions, and being the most cost efficient.

Scalability. Here, we test Tree's scalability to higher dimensions up to 225D, focusing on studying how the total cumulative message passing cost incurred scales as dimension increases. We used the same setup and parameters as Sec. 4.3, across additive grid structures of varying sizes – Grid-i×i for $i \in [2, 15]$. We again include Graph No-Overlap (5) and Graph No-Overlap (10) for this experiment.

From Fig. 3f, we can see that the amount of cost needed for Graph Overlap and Graph No-Overlap quickly increases as

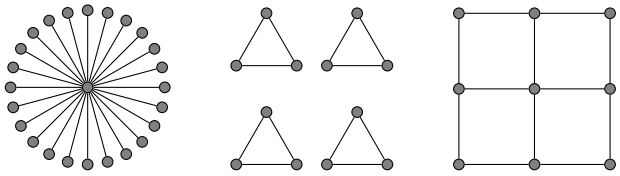
[3] See the appendix for more details on Ancestry-132.



(a) Star-25 (Discrete) Performance (b) Grid-3×3 (Continuous) Performance (c) Grid-3×3 (Continuous) F_1score

(d) Ancestry-132 (Continuous) Performance (e) Ancestry-132 (Continuous) Cost (f) Scalability of Tree over dimensions

Figure 3: Summarized comparison of various additive methods across various functions.



(a) Star-25 (b) Partition-12 (c) Grid-3×3

Figure 4: Synthetic Dependency Graphs Structures.

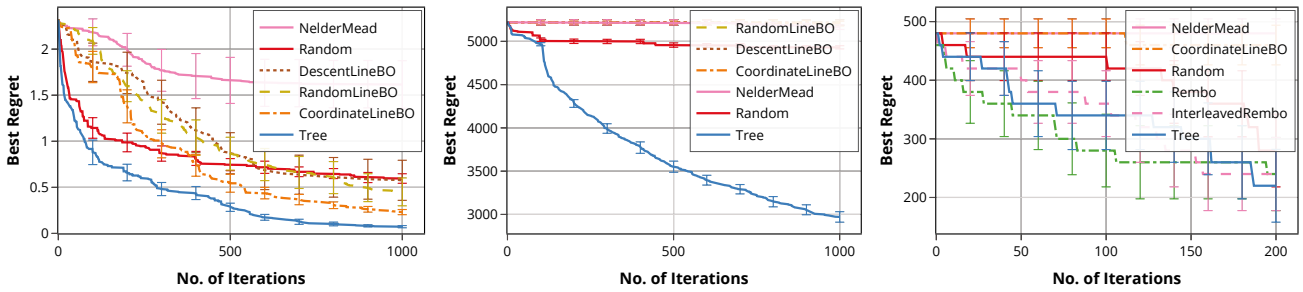
the dimensionality increases. In fact, we were unable to complete the experiment for larger grids in a reasonable amount of time. Recalling that Graph Overlap runs in time exponential in the size of the maximum clique of the triangulated dependency graph (Rolland et al. 2018), we note that even if that clique size does not grow large for the *true* graph, it may still tend to increase for the *estimated* graph. Similarly, Graph No-Overlap may be slow due to the consideration of large cliques, unless the clique size is explicitly limited. Even after imposing the limits, we found that Tree still incurs the lowest cost when compared with both Graph No-Overlap (5) and Graph No-Overlap (10). This is because, for tree structures, the message passing cost is quadratic in the number of discretization levels of a single dimension.

4.4 Experiments with Non-GP Functions

Non-GP Synthetic Functions. Here we test our algorithm against commonly used BO synthetic function benchmarks (Oh, Gavves, and Welling 2018; Kirschner et al. 2019), including Hartmann6 (6D) and Stybtang250 (250D). We also tested Tree on benchmarks with invariant subspaces; following the setup in (Kirschner et al. 2019), Hartmann6+14Aux (20D) was obtained by augmenting the synthetic functions with 14 auxiliary dimensions. In Fig. 5a, we see that the regret of Tree reduces rapidly compared to other methods, with variants of LineBO catching up in later iterations. In Fig. 5b, we see that Tree again manages to scale well in higher-dimensional synthetic functions. From additional synthetic experiments (Fig. S6a-S6f in the appendix), Tree is also competitive against LineBO variants across both lower and higher dimensional settings, even in cases with invariant subspaces.

Linear Programming Solver. We consider tuning the parameters of lpsolve, an open-source Mixed Integer Linear Programming (MILP) solver (Berkelaar, Eikland, and Notebaert 2004). The parameters within each algorithm typically have some relationship with each other; tweaking a parameter can potentially affect another. We consider a similar configuration problem as defined by (Hutter, Hoos, and Leyton-Brown 2010; Wang et al. 2013), focusing on tuning lpsolve's 74 parameters - 59 binary, 10 ordinal and 5 categorical. Our objective is to find the set of parameters of lpsolve that minimize the *optimality gap* it can achieve with a time limit of five seconds for the MIP encoding 'misc05inf' found in the benchmark MIPLIB (Gleixner et al. 2021).

From Fig. 5c, we observe that REMBO is competitive in performance for optimizing the linear programming solver, as parameter optimization problems often have low effective dimensionality (Wang et al. 2013; Hoos and Leyton-Brown 2014). Despite being based on a very different notion of



(a) Hartmann6+14Aux Performance (b) Stybtang250 Performance (c) Lpsolve-misc05inf Performance

Figure 5: Comparison of various optimization algorithms for both synthetic functions and Lpsolve functions.

structure, Tree attains better performance than REMBO in this example, with both clearly outperforming Random. In the appendix, we provide two additional lpsolve examples in which Tree outperforms REMBO.

Additional Experiments. Additional experiments on the NAS-Bench-101 (NAS) dataset (Ying et al. 2019; Klein and Hutter 2019) and BO-based adversarial attacks (BA) (Ru et al. 2020) can be found in the appendix.

5 Conclusion

For the problem of GP optimization with generalized additive models, we traded off expressivity for computational efficiency and ease of model learning by reducing the model complexity, constraining the dependency graph to tree structures. Our method efficiently learns the additive tree structure using Gibbs Sampling and edge mutation, suitable for resource-limited settings in line with the primary motivation of BO. Besides, we presented a zooming-based message passing approach that can benefit BO with generalized additive models in continuous domains, with or without tree structures. We demonstrated that Tree is competitive on both synthetic functions and real datasets, and that the computation can be significantly reduced compared to more complex graph structures, without sacrificing the optimization performance.

Acknowledgments

This work was supported by both the Singapore National Research Foundation (NRF) under grant number R-252-000-A74-281 and the AWS Cloud Credits for Research program.

References

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

Berkelaar, M.; Eikland, K.; and Notebaert, P. 2004. lp solve 5.5, open source (mixed-integer) linear programming system. Software. Available at ¡http://lpsolve.sourceforge.net/5.5/¿. Last accessed Dec, 18 2009.

Chen, B.; Castro, R. M.; and Krause, A. 2012. Joint optimization and variable selection of high-dimensional Gaussian processes. In *Int. Conf. Mach. Learn. (ICML)*, 1379–1386.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to algorithms*. MIT press.

Cui, J.; Yang, B.; and Hu, X. 2019. Deep Bayesian optimization on attributed graphs. In *AAAI Conf. on Art. Intel.*, volume 33, 1377–1384.

Djolonga, J.; Krause, A.; and Cevher, V. 2013. High-dimensional Gaussian process bandits. In *Conf. Neur. Inf. Proc. Sys. (NIPS)*, 1025–1033.

Eriksson, D.; Pearce, M.; Gardner, J.; Turner, R. D.; and Poloczek, M. 2019. Scalable Global Optimization via Local Bayesian Optimization. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32, 5496–5507. Curran Associates, Inc.

Frazier, P. I. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Gleixner, A.; Hendel, G.; Gamrath, G.; Achterberg, T.; Bastubbe, M.; Berthold, T.; Christophel, P. M.; Jarck, K.; Koch, T.; Linderoth, J.; Lübbecke, M.; Mittelmann, H. D.; Ozyurt, D.; Ralphs, T. K.; Salvagnin, D.; and Shinano, Y. 2021. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*.

Gonzalvez, J.; Lezmi, E.; Roncalli, T.; and Xu, J. 2019. Financial applications of Gaussian processes and Bayesian optimization. *arXiv preprint arXiv:1903.04841*.

Hoang, T. N.; Hoang, Q. M.; Ouyang, R.; and Low, K. H. 2018. Decentralized high-dimensional Bayesian optimization with factor graphs. In *AAAI Conf. on Art. Intel.*

Hoos, H., and Leyton-Brown, K. 2014. An efficient approach for assessing hyperparameter importance. In *Int. Conf. Mach. Learn. (ICML)*, 754–762.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2010. Automated configuration of mixed integer programming solvers. In *Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, 186–202. Springer.

Jaquier, N.; Rozo, L.; Calinon, S.; and Bürger, M. 2020. Bayesian Optimization meets Riemannian Manifolds in Robot Learning. In *Conference on Robot Learning*, 233–246. PMLR.

Jenatton, R.; Archambeau, C.; González, J.; and Seeger, M. 2017. Bayesian optimization with tree-structured dependencies. In *Int. Conf. Mach. Learn. (ICML)*.

Kandasamy, K.; Schneider, J.; and Póczos, B. 2015. High dimensional Bayesian optimisation and bandits via additive models. In *Int. Conf. Mach. Learn. (ICML)*, 295–304.

Kirschner, J.; Mutny, M.; Hiller, N.; Ischebeck, R.; and Krause, A.

2019. Adaptive and safe Bayesian optimization in high dimensions via one-dimensional subspaces. In *Int. Conf. Mach. Learn. (ICML)*, 3429–3438.

Klein, A., and Hutter, F. 2019. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*.

Li, C.-L.; Kandasamy, K.; Póczos, B.; and Schneider, J. 2016. High dimensional Bayesian optimization via restricted projection pursuit models. In *Int. Conf. Art. Intel. Stats. (AISTATS)*, 884–892.

Li, C.; Gupta, S.; Rana, S.; Nguyen, V.; Venkatesh, S.; and Shilton, A. 2017. High dimensional Bayesian optimization using dropout. In *Int. Joint Conf. on Art. Intel. (IJCAI)*, 2096–2102.

Lu, X.; Gonzalez, J.; Dai, Z.; and Lawrence, N. 2018. Structured variationally auto-encoded optimization. In *Int. Conf. Mach. Learn. (ICML)*, 3267–3275.

Ma, X., and Blaschko, M. 2020. Additive Tree-Structured Covariance Function for Conditional Parameter Spaces in Bayesian Optimization. *Int. Conf. Art. Intel. Stats. (AISTATS)*.

Močkus, J. 1975. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conf.*, 400–404. Springer.

Moriconi, R.; Kumar, K.; and Deisenroth, M. P. 2019. High-dimensional bayesian optimization with manifold gaussian processes. *arXiv preprint arXiv:1902.10675*.

Murphy, K. P. 2012. *Machine learning: A probabilistic perspective*. MIT press.

Mutny, M., and Krause, A. 2018. Efficient High Dimensional Bayesian Optimization with Additivity and Quadrature Fourier Features. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31, 9005–9016. Curran Associates, Inc.

Nayebi, A.; Munteanu, A.; and Poloczek, M. 2019. A framework for Bayesian optimization in embedded subspaces. In *Int. Conf. Mach. Learn. (ICML)*, 4752–4761.

Oh, C.; Gavves, E.; and Welling, M. 2018. BOCK: Bayesian optimization with cylindrical kernels. In *Int. Conf. Mach. Learn. (ICML)*, 3868–3877.

Rolland, P.; Scarlett, J.; Bogunovic, I.; and Cevher, V. 2018. High-dimensional Bayesian optimization via additive models with overlapping groups. In *Int. Conf. Art. Intel. Stats. (AISTATS)*, 298–307.

Ru, B.; Cobb, A.; Blaas, A.; and Gal, Y. 2020. BayesOpt Adversarial Attack. In *Proc. of the International Conference on Learning Representations*.

Sano, S.; Kadowaki, T.; Tsuda, K.; and Kimura, S. 2019. Application of Bayesian optimization for pharmaceutical product development. *Journal of Pharmaceutical Innovation* 1–11.

Sedgewick, R., and Wayne, K. 2011. *Algorithms*. Addison-wesley professional.

Snoek, J.; Rippel, O.; Swersky, K.; Kiros, R.; Satish, N.; Sundaram, N.; Patwary, M.; Prabhat, M.; and Adams, R. 2015. Scalable Bayesian optimization using deep neural networks. In *Int. Conf. Mach. Learn. (ICML)*, 2171–2180.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical Bayesian optimization of machine learning algorithms. In *Conf. Neur. Inf. Proc. Sys. (NIPS)*, 2951–2959.

Spruyt, V. 2014. The curse of dimensionality in classification. *Computer Vision for Dummies* 21(3):35–40.

Srinivas, N.; Krause, A.; Kakade, S.; and Seeger, M. 2010. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Int. Conf. Mach. Learn. (ICML)*, 1015–1022.

Swersky, K.; Snoek, J.; and Adams, R. P. 2013. Multi-task Bayesian optimization. In *Conf. Neur. Inf. Proc. Sys. (NIPS)*, 2004–2012.

Wang, Z.; Zoghi, M.; Hutter, F.; Matheson, D.; and De Freitas, N. 2013. Bayesian optimization in high dimensions via random embeddings. In *Int. Joint Conf. on Art. Intel. (IJCAI)*.

Wang, Z.; Li, C.; Jegelka, S.; and Kohli, P. 2017. Batched high-dimensional Bayesian optimization via structural kernel learning. In *Int. Conf. Mach. Learn. (ICML)*, 3656–3664. JMLR. org.

Wang, Z.; Gehring, C.; Kohli, P.; and Jegelka, S. 2018. Batched large-scale Bayesian optimization in high-dimensional spaces. In *Int. Conf. Art. Intel. Stats. (AISTATS)*, 745–754.

Wang, Z. 2016. *Practical and theoretical advances in Bayesian optimization*. Ph.D. Dissertation, University of Oxford.

Ying, C.; Klein, A.; Christiansen, E.; Real, E.; Murphy, K.; and Hutter, F. 2019. NAS-Bench-101: Towards reproducible neural architecture search. In *Int. Conf. Mach. Learn. (ICML)*, 7105–7114.

Yogatama, D.; Kong, L.; and Smith, N. A. 2015. Bayesian optimization of text representations. In *Conf. on Empirical Methods in NLP (EMNLP)*, 2100–2105.

Zhang, M.; Li, H.; and Su, S. 2019. High Dimensional Bayesian Optimization via Supervised Dimension Reduction. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 4292–4298. International Joint Conferences on Artificial Intelligence Organization.