Monte Carlo Tree Search based Variable Selection for High Dimensional Bayesian Optimization

Lei Song[*], **Ke Xue**[*], **Xiaobin Huang, Chao Qian**[†]
State Key Laboratory for Novel Software Technology,
Nanjing University, Nanjing 210023, China
{songl, xuek, huangxb, qianc}@lamda.nju.edu.cn

Abstract

Bayesian optimization (BO) is a class of popular methods for expensive black-box optimization, and has been widely applied to many scenarios. However, BO suffers from the curse of dimensionality, and scaling it to high-dimensional problems is still a challenge. In this paper, we propose a variable selection method MCTS-VS based on Monte Carlo tree search (MCTS), to iteratively select and optimize a subset of variables. That is, MCTS-VS constructs a low-dimensional subspace via MCTS and optimizes in the subspace with any BO algorithm. We give a theoretical analysis of the general variable selection method to reveal how it can work. Experiments on high-dimensional synthetic functions and real-world problems (i.e., NAS-bench problems and MuJoCo locomotion tasks) show that MCTS-VS equipped with a proper BO optimizer can achieve state-of-the-art performance.

1 Introduction

In many real-world tasks such as neural architecture search (NAS) [39] and policy search in reinforcement learning (RL) [6], one often needs to solve the expensive black-box optimization problems. Bayesian optimization (BO) [2, 11, 23, 32] is a sample-efficient algorithm for solving such problems. It iteratively fits a surrogate model, typically Gaussian process (GP), and maximizes an acquisition function to obtain the next point to evaluate. While BO has been employed in a wide variety of settings, successful applications are often limited to low-dimensional problems.

Recently, scaling BO to high-dimensional problems has received a lot of interest. Decomposition-based methods [13, 15, 17, 26, 31] assume that the high-dimensional function to be optimized has a certain structure, typically the additive structure. By decomposing the original high-dimensional function into the sum of several low-dimensional functions, they optimize each low-dimensional function to obtain the point in the high-dimensional space. However, it is not easy to decide whether a decomposition exists as well as to learn the decomposition.

Other methods often assume that the original high-dimensional function with dimension D has a low-dimensional subspace with dimension $d \ll D$, and then perform the optimization in the low-dimensional subspace and project the low-dimensional point back for evaluation. For example, embedding-based methods [20, 27, 40] use a random matrix to embed the original space into the low-dimensional subspace. Another way is to select a subset of variables directly, which can even avoid the time-consuming matrix operations of embedding-based methods. For example, Dropout [21] selects d variables randomly in each iteration. Note that for both embedding and variable selection methods, the parameter d can have a large influence on the performance, which is, however, difficult to set in real-world problems.

[*]Equal Contribution
[†]Corresponding Author

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

In this paper, we propose a new Variable Selection method using Monte Carlo Tree Search (MCTS), called MCTS-VS. MCTS is employed to partition the variables into important and unimportant ones, and only those selected important variables are optimized via any black-box optimization algorithm, e.g., vanilla BO [32] or TuRBO [10]. The values of unimportant variables are sampled using historical information. Compared with Dropout-BO, MCTS-VS can select important variables automatically.

We also provide regret and computational complexity analyses of general variable selection methods, showing that variable selection can reduce the computational complexity while increasing the cumulative regret. Our regret bound generalizes that of GP-UCB [36] which always selects all variables, as well as that of Dropout [21] which selects d variables randomly in each iteration. The results suggest that a good variable selection method should select as important variables as possible.

Experiments on high-dimensional synthetic functions and real-world problems (i.e., NAS and RL problems) show that MCTS-VS is better than the previous variable selection method Dropout [21], and can also achieve the competitive performance to state-of-the-art BO algorithms. Furthermore, its running time is small due to the advantage of variable selection. We also observe that MCTS-VS can select important variables, explaining its good performance based on our theoretical analysis.

2 Background

2.1 Bayesian Optimization

We consider the problem $\max_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})$, where f is a black-box function and $\mathcal{X} \subseteq \mathbb{R}^D$ is the domain. The basic framework of BO contains two critical components: a surrogate model and an acquisition function. GP is the most popular surrogate model. Given the sampled data points $\{(\boldsymbol{x}^i, y^i)\}_{i=1}^{t-1}$, where $y^i = f(\boldsymbol{x}^i) + \epsilon^i$ and $\epsilon^i \sim \mathcal{N}(0, \eta^2)$ is the observation noise, GP at iteration t seeks to infer $f \sim \mathcal{GP}(\mu(\cdot), k(\cdot,\cdot) + \eta^2\mathbf{I})$, specified by the mean $\mu(\cdot)$ and covariance kernel $k(\cdot,\cdot)$, where $\mathbf{I}$ is the identity matrix of size D. After that, an acquisition function, e.g., Probability of Improvement (PI) [19], Expected Improvement (EI) [16] or Upper Confidence Bound (UCB) [36], is used to determine the next query point $\boldsymbol{x}^t$ while balancing exploitation and exploration.

2.2 High-dimensional Bayesian Optimization

Scaling BO to high-dimensional problems is a challenge due to the curse of dimensionality and the computation cost. As the dimension increases, the search space increases exponentially, requiring more samples, and thus more expensive evaluations, to find a good solution. Furthermore, the computation cost of updating the GP model and optimizing the acquisition function will be very time-consuming [30]. There have been a few common approaches to tackle high-dimensional BO with different assumptions.

Decomposition. Assuming that the function can be decomposed into the sum of low-dimensional functions with disjoint subspaces, Kandasamy et al. [17] proposed the Add-GP-UCB algorithm to optimize those low-dimensional functions separately, which was further generalized to overlapping subspaces [26, 31]. Wang et al. [41] proposed ensemble BO that uses an ensemble of additive GP models for scalability. Han et al. [13] constrained the dependency graphs of decomposition to tree structures to facilitate the decomposition learning and optimization. For most problems, however, the decomposition is unknown, and also difficult to learn.

Embedding. Assuming that only a few dimensions affect the high-dimensional function significantly, embedding-based methods embed the high-dimensional space into a low-dimensional subspace, and optimize in the subspace while projecting the point back for evaluation. REMBO and its variants use a random matrix to embed the search space into a low-dimensional subspace [3, 4, 40]. Nayebi et al. [27] used a hash-based method for embedding. Letham et al. [20] proposed ALEBO, focusing on several misconceptions in REMBO to improve the performance. The VAE-based approaches were also employed to project a structured input space (e.g., graphs and images) to a low-dimensional subspace [12, 22].

Variable Selection. Based on the same assumption as embedding, variable selection methods iteratively select a subset of variables to build a low-dimensional subspace and optimize through BO. The selected variables can be viewed as important variables that are valuable for exploitation, or having high uncertainty that are valuable for exploration. A classical method is Dropout [21], which

randomly chooses d variables in each iteration. When evaluating the sampled point, the values of those unselected variables are obtained by random sampling or using historical information. Note that variable selection can be faster than embedding, because the embedding cost (e.g., matrix inversion) is time-consuming for high-dimensional optimization.

Both embedding and variable selection methods need to specify the parameter d, i.e., the dimension of low-dimensional subspace, which will affect the performance significantly, but is not easy to set. There are also some methods to improve the basic components of BO directly for high-dimensional problems. For example, DNGO [35] uses the neural network as an alternative of GP to speed up inference; BO-PP [29] generates pseudo-points (i.e., data points whose objective values are not evaluated) to improve the GP model; SAASBO [9] uses sparsity-inducing prior to perform variable selection implicitly, making the coefficients of unimportant variables near to zero and thus restraining over-exploration on these variables. Note that different from Dropout and our proposed MCTS-VS, SAASBO still optimizes all variables, and also due to its high computational cost of inference, it is very time-consuming as reported in [9]. These methods can be combined with the above-mentioned dimensionality reduction methods, which may bring further improvement.

2.3 Monte Carlo Tree Search

MCTS [5] is a tree search algorithm based on random sampling, and has shown great success in high-dimensional tasks, such as Go [33, 34]. A tree node represents a state, describing the current situation, e.g., the position in path planning. Each tree node X stores a value v_X representing its goodness, and the number n_X that it has been visited. They are used to calculate UCB [1], i.e.,

$$v_X + 2C_p\sqrt{2(\log n_p)/n_X}, \tag{1}$$

where C_p is a hyper-parameter, and n_p is the number of visits of the parent of X. UCB considers both exploitation and exploration, and will be used for node selection.

MCTS iteratively selects a leaf node of the tree for expansion. Each iteration can be divided into four steps: *selection*, *expansion*, *simulation* and *back-propagation*. Starting from the root node, selection is to recursively select a node with larger UCB until a leaf node, denoted as X. Expansion is to execute a certain action in the state represented by X and transfer to the next state, e.g., move forward and arrive at a new position in path planning. We use the child node Y of X to represent the next state. Simulation is to obtain the value v_Y via random sampling. Back-propagation is to update the value and the number of visits of Y's ancestors.

To tackle high-dimensional optimization, Wang et al. [38] proposed LA-MCTS, which applies MCTS to iteratively partition the search space into small sub-regions, and optimizes only in the good sub-regions. That is, the root of the tree represents the entire search space Ω, and each tree node X represents a sub-region Ω_X. The value v_X is measured by the average objective value of the sampled points in the sub-region Ω_X. In each iteration, after selecting a leaf node X, LA-MCTS performs the optimization in Ω_X by vanilla BO [32] or TuRBO [10], and the sampled points are used for clustering and classification to bifurcate Ω_X into two disjoint sub-regions, which are "good" and "bad", respectively. Note that the sub-regions are generated by dividing the range of variables, and their dimensionality does not decrease, which is still the number of all variables. Wang et al. [38] have empirically shown the good performance of LA-MCTS. However, as the dimension increases, the search space increases exponentially, and more partitions and evaluations are required to find a good solution, making the application of LA-MCTS to high-dimensional optimization still limited.

3 MCTS-VS Method

In this section, we propose a Variable Selection method based on MCTS for high-dimensional BO, briefly called MCTS-VS. The main idea is to apply MCTS to iteratively partition all variables into important and unimportant ones, and perform BO only for those important variables. Let $[D] = \{1, 2, \ldots, D\}$ denote the indexes of all variables $\boldsymbol{x}$, and $\boldsymbol{x}_{\mathbb{M}}$ denote the subset of variables indexed by $\mathbb{M} \subseteq [D]$.

We first introduce a D-dimensional vector named *variable score*, which is a key component of MCTS-VS. Its i-th element represents the importance of the i-th variable x_i. During the running process of MCTS-VS, after optimizing a subset $\boldsymbol{x}_{\mathbb{M}}$ of variables where $\mathbb{M} \subseteq [D]$ denotes the indexes

of the variables, a set $\mathcal{D}$ of sampled points will be generated, and the pair $(\mathbb{M}, \mathcal{D})$ will be recorded into a set $\mathbb{D}$, called *information set*. The variable score vector is based on $\mathbb{D}$, and calculated as

$$\boldsymbol{s} = \left(\sum_{(\mathbb{M},\mathcal{D})\in\mathbb{D}} \sum_{(\boldsymbol{x}^i,y^i)\in\mathcal{D}} y^i \cdot g(\mathbb{M}) \right) \Big/ \left(\sum_{(\mathbb{M},\mathcal{D})\in\mathbb{D}} |\mathcal{D}| \cdot g(\mathbb{M}) \right), \tag{2}$$

where the function $g : 2^{[D]} \to \{0,1\}^D$ gives the Boolean vector representation of a variable index subset $\mathbb{M} \subseteq [D]$ (i.e., the i-th element of $g(\mathbb{M})$ is 1 if $i \in \mathbb{M}$, and 0 otherwise), and $/$ is the element-wise division. Each dimension of $\sum_{(\mathbb{M},\mathcal{D})\in\mathbb{D}} \sum_{(\boldsymbol{x}^i,y^i)\in\mathcal{D}} y^i \cdot g(\mathbb{M})$ is the sum of query evaluations using each variable, and each dimension of $\sum_{(\mathbb{M},\mathcal{D})\in\mathbb{D}} |\mathcal{D}| \cdot g(\mathbb{M})$ is the number of queries using each variable. Thus, the i-th element of variable score $\boldsymbol{s}$, representing the importance of the i-th variable x_i, is actually measured by the average goodness of all the sampled points that are generated by optimizing a subset of variables containing x_i. The variable score $\boldsymbol{s}$ will be used to define the value of each tree node of MCTS as well as for node expansion.

In MCTS-VS, the root of the tree represents all variables. A tree node X represents a subset of variables, whose index set is denoted by $\mathbb{A}_X \subseteq [D]$, and it stores the value v_X and the number n_X of visits, which are used to calculate the value of UCB as in Eq. (1). The value v_X is defined as the average score (i.e., importance) of the variables contained by X, which can be calculated by $\boldsymbol{s} \cdot g(\mathbb{A}_X)/|\mathbb{A}_X|$, where $g(\mathbb{A}_X)$ is the Boolean vector representation of $\mathbb{A}_X$ and $|\mathbb{A}_X|$ is the size of $\mathbb{A}_X$, i.e., the number of variables in node X.

At each iteration, MCTS-VS first recursively selects a node with larger UCB until a leaf node (denoted as X), which is regarded as containing important variables. Note that if we optimize the subset $\boldsymbol{x}_{\mathbb{A}_X}$ of variables represented by the leaf X directly, the variables in $\boldsymbol{x}_{\mathbb{A}_X}$ will have the same score (because they are optimized together), and their relative importance cannot be further distinguished. Thus, MCTS-VS uniformly selects a variable index subset $\mathbb{M}$ from $\mathbb{A}_X$ at random, and employs BO to optimize $\boldsymbol{x}_{\mathbb{M}}$ as well as $\boldsymbol{x}_{\mathbb{A}_X \setminus \mathbb{M}}$; this process is repeated for several times. After that, the information set $\mathbb{D}$ will be augmented by the pairs of the selected variable index subset $\mathbb{M}$ (or $\mathbb{A}_X \setminus \mathbb{M}$) and the corresponding sampled points generated by BO. The variable score vector $\boldsymbol{s}$ will be updated using this new $\mathbb{D}$. Based on $\boldsymbol{s}$, the variable index set $\mathbb{A}_X$ represented by the leaf X will be divided into two disjoint subsets, containing variables with larger and smaller scores (i.e., important and unimportant variables), respectively, and the leaf X will be bifurcated into two child nodes accordingly. Finally, the v values of these two children will be calculated using the variable score vector $\boldsymbol{s}$, and back-propagation will be performed to update the v value and the number of visits of the nodes along the current path of the tree.

MCTS-VS can be equipped with any specific BO optimizer, resulting in the concrete algorithm MCTS-VS-BO, where BO is used to optimize the selected subsets of variables during the running of MCTS-VS. Compared with LA-MCTS [38], MCTS-VS applies MCTS to partition the variables instead of the search space, and thus can be more scalable. Compared with the previous variable selection method Dropout [21], MCTS-VS can select important variables automatically instead of randomly selecting a fixed number of variables in each iteration. Next we introduce it in detail.

3.1 Details of MCTS-VS

The procedure of MCTS-VS is described in Algorithm 1. In line 1, it first initializes the information set $\mathbb{D}$. In particular, a variable index subset $\mathbb{M}_i$ is randomly sampled from $[D]$, and the Latin hypercube sampling [24] is used to generate two sets (denoted as $\mathcal{D}_i$ and $\mathcal{D}_{\bar{i}}$) of N_s points to form the two pairs of $(\mathbb{M}_i, \mathcal{D}_i)$ and $(\bar{\mathbb{M}}_i, \mathcal{D}_{\bar{i}})$, where $\bar{\mathbb{M}}_i = [D] \setminus \mathbb{M}_i$. This process will be repeated for N_v times, resulting in the initial $\mathbb{D} = \{(\mathbb{M}_i, \mathcal{D}_i), (\bar{\mathbb{M}}_i, \mathcal{D}_{\bar{i}})\}_{i=1}^{N_v}$. The variable score vector $\boldsymbol{s}$ is calculated using this initial $\mathbb{D}$ in line 3, and the Monte Carlo tree is initialized in line 4 by adding only a root node, whose v value is calculated according to $\boldsymbol{s}$ and number of visits is 0. MCTS-VS uses the variable t to record the number of evaluations it has performed, and thus t is set to $2 \times N_v \times N_s$ in line 5 as the initial $\mathbb{D}$ contains $2 \times N_v \times N_s$ sampled points in total.

In each iteration (i.e., lines 7–28) of MCTS-VS, it selects a leaf node X by UCB in line 10, and optimizes the variables (i.e., $\boldsymbol{x}_{\mathbb{A}_X}$) represented by X in lines 13–23. Note that to measure the relative importance of variables in $\boldsymbol{x}_{\mathbb{A}_X}$, MCTS-VS optimizes different subsets of variables of $\boldsymbol{x}_{\mathbb{A}_X}$ instead of $\boldsymbol{x}_{\mathbb{A}_X}$ directly. That is, a variable index subset $\mathbb{M}$ is randomly sampled from $\mathbb{A}_X$ in line 14,

Algorithm 1 MCTS-VS

Parameters: batch size N_v of variable index subset, sample batch size N_s, total number N_e of evaluations, threshold N_{bad} for re-initializing a tree and N_{split} for splitting a node, hyper-parameter k for the best-k strategy

Process:

1: Initialize the information set $\mathbb{D} = \{(\mathbb{M}_i, \mathcal{D}_i), (\bar{\mathbb{M}}_i, \mathcal{D}_{\bar{i}})\}_{i=1}^{N_v}$;
2: Store the best k sampled points in $\mathbb{D}$;
3: Calculate the variable score $\boldsymbol{s}$ using $\mathbb{D}$ as in Eq. (2);
4: Initialize the Monte Carlo tree;
5: Set $t = 2 \times N_v \times N_s$ and $n_{bad} = 0$;
6: **while** $t < N_e$ **do**
7: **if** $n_{bad} > N_{bad}$ **then**
8: Initialize the Monte Carlo tree and set $n_{bad} = 0$
9: **end if**
10: $X \leftarrow$ the leaf node selected by UCB;
11: Let $\mathbb{A}_X$ denote the indexes of the subset of variables represented by X;
12: Increase n_{bad} by 1 once visiting a right child node on the path from the root node to X;
13: **for** $j = 1 : N_v$ **do**
14: Sample a variable index subset $\mathbb{M}$ from $\mathbb{A}_X$ uniformly at random;
15: Fit a GP model using the points $\{(\boldsymbol{x}_{\mathbb{M}}^i, y^i)\}_{i=1}^t$ sampled-so-far, where only the variables indexed by $\mathbb{M}$ are used;
16: Generate $\{\boldsymbol{x}_{\mathbb{M}}^{t+i}\}_{i=1}^{N_s}$ by maximizing an acquisition function;
17: Determine $\{\boldsymbol{x}_{[D]\setminus\mathbb{M}}^{t+i}\}_{i=1}^{N_s}$ by the "fill-in" strategy;
18: Evaluate $\boldsymbol{x}^{t+i} = [\boldsymbol{x}_{\mathbb{M}}^{t+i}, \boldsymbol{x}_{[D]\setminus\mathbb{M}}^{t+i}]$ to obtain y^{t+i} for $i = 1, 2, \ldots, N_s$;
19: $\mathbb{D} = \mathbb{D} \cup \{(\mathbb{M}, \{(\boldsymbol{x}^{t+i}, y^{t+i})\}_{i=1}^{N_s})\}$;
20: Store the best k points sampled-so-far;
21: $t = t + N_s$;
22: Repeat lines 15–21 for $\bar{\mathbb{M}} = \mathbb{A}_X \setminus \mathbb{M}$
23: **end for**
24: Calculate the variable score $\boldsymbol{s}$ using $\mathbb{D}$ as in Eq. (2);
25: **if** $|\mathbb{A}_X| > N_{split}$ **then**
26: Bifurcate the leaf node X into two child nodes, whose v value and number of visits are calculated by $\boldsymbol{s}$ and set to 0, respectively
27: **end if**
28: Back-propagate to update the v value and number of visits of the nodes on the path from the root to X
29: **end while**

and the corresponding subset $\boldsymbol{x}_{\mathbb{M}}$ of variables is optimized by BO in lines 15–16. The data points $\{(\boldsymbol{x}_{\mathbb{M}}^i, y^i)\}_{i=1}^t$ sampled-so-far is used to fit a GP model, and N_s (called *sample batch size*) new points $\{\boldsymbol{x}_{\mathbb{M}}^{t+i}\}_{i=1}^{N_s}$ are generated by maximizing an acquisition function. Note that this is a standard BO procedure, which can be replaced by any other variant. To evaluate $\boldsymbol{x}_{\mathbb{M}}^{t+i}$, we need to fill in the values of the other variables $\boldsymbol{x}_{[D]\setminus\mathbb{M}}^{t+i}$, which will be explained later. After evaluating $\boldsymbol{x}^{t+i} = [\boldsymbol{x}_{\mathbb{M}}^{t+i}, \boldsymbol{x}_{[D]\setminus\mathbb{M}}^{t+i}]$ in line 18, the information set $\mathbb{D}$ is augmented with the new pair of $(\mathbb{M}, \{(\boldsymbol{x}^{t+i}, y^{t+i})\}_{i=1}^{N_s})$ in line 19, and t is increased by N_s accordingly in line 21. For fairness, the complement subset $\boldsymbol{x}_{\bar{\mathbb{M}}}$ of variables, where $\bar{\mathbb{M}} = \mathbb{A}_X \setminus \mathbb{M}$, is also optimized by the same way, i.e., lines 15–21 of Algorithm 1 is repeated for $\bar{\mathbb{M}}$. The whole process of optimizing $\boldsymbol{x}_{\mathbb{M}}$ and $\boldsymbol{x}_{\bar{\mathbb{M}}}$ in lines 14–22 will be repeated for N_v times, which is called *batch size of variable index subset*.

To fill in the values of the un-optimized variables in line 17, we employ the *best-k* strategy, which utilizes the best k data points sampled-so-far, denoted as $\{(\boldsymbol{x}^{*j}, y^{*j})\}_{j=1}^k$. That is, $\{y^{*j}\}_{j=1}^k$ are the k largest objective values observed-so-far. If the i-th variable is un-optimized, its value will be uniformly selected from $\{x_i^{*j}\}_{j=1}^k$ at random. Thus, MCTS-VS needs to store the best k data points in line 2 after initializing the information set $\mathbb{D}$, and update them in line 20 after augmenting $\mathbb{D}$. Other direct "fill-in" strategies include sampling the value randomly, or using the average variable

value of the best k data points. The superiority of the employed best-k strategy will be shown in the experiments in Appendix D.

After optimizing the variables $\boldsymbol{x}_{\mathbb{A}_X}$ represented by the selected leaf X, the variable score vector $\boldsymbol{s}$ measuring the importance of each variable will be updated using the augmented $\mathbb{D}$ in line 24. If the number $|\mathbb{A}_X|$ of variables in the leaf X is larger than a threshold N_{split} (i.e., line 25), $\mathbb{A}_X$ will be divided into two subsets. One contains those "important" variable indexes with score larger than the average score of $\boldsymbol{x}_{\mathbb{A}_X}$, and the other contains the remaining "unimportant" ones. The leaf X will be bifurcated into a left child Y and a right child Z in line 26, containing those important and unimportant variables, respectively. Meanwhile, v_Y and v_Z will be calculated according to $\boldsymbol{s}$, and the number of visits is 0, i.e., $n_Y = n_Z = 0$. Finally, MCTS-VS performs back-propagation in line 28 to re-calculate the v value and increase the number of visits by 1 for each ancestor of Y and Z.

MCTS-VS will run until the number t of performed evaluations reaches the budget N_e. Note that as the Monte Carlo tree may be built improperly, we use a variable n_{bad} to record the number of visiting a right child node (regarded as containing unimportant variables), measuring the goodness of the tree. In line 5 of Algorithm 1, n_{bad} is initialized as 0. During the procedure of selecting a leaf node by UCB in line 10, n_{bad} will be increased by 1 once visiting a right child node, which is updated in line 12. If n_{bad} is larger than a threshold N_{bad} (i.e., line 7), the current tree is regarded as bad, and will be re-initialized in line 8. Furthermore, the frequency of re-initialization can be used to indicate whether MCTS-VS can do a good variable selection for the current problem. For ease of understanding, we also provide an example illustration of MCTS-VS in Appendix A.

4 Theoretical Analysis

Although it is difficult to analyze the regret of MCTS-VS directly, we can theoretically analyze the influence of general variable selection by adopting the acquisition function GP-UCB. The considered general variable selection framework is as follows: after selecting a subset of variables at each iteration, the corresponding observation data (i.e., the data points sampled-so-far where only the selected variables are used) is used to build a GP model, and the next data point is sampled by maximizing GP-UCB. We use $\mathbb{M}_t$ to denote the sampled variable index subset at iteration t, and let $|\mathbb{M}_t| = d_t$.

Regret Analysis. Let $\boldsymbol{x}^*$ denote an optimal solution. We analyze the cumulative regret $R_T = \sum_{t=1}^{T}(f(\boldsymbol{x}^*) - f(\boldsymbol{x}^t))$, i.e., the sum of the gap between the optimum and the function values of the selected points by iteration T. To derive an upper bound on R_T, we pessimistically assume that the worst function value, i.e., $\min_{\boldsymbol{x}_{[D]\setminus\mathbb{M}_t}} f([\boldsymbol{x}_{\mathbb{M}_t}, \boldsymbol{x}_{[D]\setminus\mathbb{M}_t}])$, given $\boldsymbol{x}_{\mathbb{M}_t}$ is returned in evaluation. As in [21, 36], we assume that $\mathcal{X} \subset [0, r]^D$ is convex and compact, and f satisfies the following Lipschitz assumption.

Assumption 4.1. The function f is a GP sample path. For some $a, b > 0$, given $L > 0$, the partial derivatives of f satisfy that $\forall i \in [D]$, $\exists \alpha_i \geq 0$,

$$P\left(\sup_{\boldsymbol{x}\in\mathcal{X}} |\partial f/\partial x_i| < \alpha_i L\right) \geq 1 - ae^{-(L/b)^2}. \tag{3}$$

Based on Assumption 4.1, we define α_i^* to be the minimum value of α_i such that Eq. (3) holds, which characterizes the importance of the i-th variable x_i. The larger α_i^*, the greater influence of x_i on the function f. Let $\alpha_{\max} = \max_{i\in[D]} \alpha_i^*$.

Theorem 4.2 gives an upper bound on the cumulative regret R_T with high probability for general variable selection methods. The proof is inspired by that of GP-UCB without variable selection [36] and provided in Appendix B.1. If we select all variables each time (i.e., $\forall t : \mathbb{M}_t = [D]$) and assume $\forall i : \alpha_i^* \leq 1$, the regret bound Eq. (4) becomes $R_T \leq \sqrt{C_1 T \beta_T^* \gamma_T} + 2$, which is consistent with [36]. Note that $\forall t : |\mathbb{M}_t| = d_t = D$ in this case, which implies that β_t increases with t, leading to $\beta_T^* = \beta_T$. We can see that using variable selection will increase R_T by $2\sum_{t=1}^{T}\sum_{i\in[D]\setminus\mathbb{M}_t} \alpha_i^* Lr$, related to the importance (i.e., α_i^*) of unselected variables at each iteration. The more important variables unselected, the larger R_T. Meanwhile, the term $\sqrt{C_1 T \beta_T^* \gamma_T}$ will decrease as β_T^* relies on the number d_t of selected variables positively. Ideally, if the unselected variables at each iteration are always unrelated (i.e., $\alpha_i^* = 0$), the regret bound will be better than that of using all variables [36].

Theorem 4.2. *$\forall \delta \in (0, 1)$, let $\beta_t = 2\log(4\pi_t/\delta) + 2d_t \log(d_t t^2 br\sqrt{\log(4Da/\delta)})$ and $L = b\sqrt{\log(4Da/\delta)}$, where r is the upper bound on each variable, and $\{\pi_t\}_{t\geq 1}$ satisfies $\sum_{t\geq 1} \pi_t^{-1} = 1$*

and $\pi_t > 0$. Let $\beta_T^* = \max_{1\le i\le T} \beta_t$. At iteration T, the cumulative regret

$$R_T \le \sqrt{C_1 T \beta_T^* \gamma_T} + 2\alpha_{\max} + 2\sum_{t=1}^{T} \sum_{i\in[D]\setminus\mathbb{M}_t} \alpha_i^* L r \tag{4}$$

holds with probability at least $1-\delta$, where C_1 is a constant, $\gamma_T = \max_{|\mathcal{D}|=T} I(\boldsymbol{y}_{\mathcal{D}}, \boldsymbol{f}_{\mathcal{D}})$, $I(\cdot,\cdot)$ is the information gain, and $\boldsymbol{y}_{\mathcal{D}}$ and $\boldsymbol{f}_{\mathcal{D}}$ are the noisy and true observations of a set $\mathcal{D}$ of points, respectively.

By selecting d variables randomly at each iteration and assuming that $r = 1$ and $\forall i : \alpha_i^* \le 1$, it has been proved [21] that the cumulative regret of Dropout satisfies

$$R_T \le \sqrt{C_1 T \beta_T \gamma_T} + 2 + 2TL(D-d). \tag{5}$$

In this case, we have $d_t = |\mathbb{M}_t| = d$, $r = 1$ and $\forall i : \alpha_i^* \le 1$. Thus, Eq. (4) becomes

$$R_T \le \sqrt{C_1 T \beta_T^* \gamma_T} + 2 + 2TL(D-d). \tag{6}$$

Note that $\beta_T^* = \beta_T$ here, as β_t increases with t given $d_t = d$. This implies that our bound Eq. (4) for general variable selection is a generalization of Eq. (5) for Dropout [21].

Computational Complexity Analysis. The computational complexity of one iteration of BO depends on three critical components: fitting a GP surrogate model, maximizing an acquisition function and evaluating a sampled point. If using the squared exponential kernel, the computational complexity of fitting a GP model at iteration t is $\mathcal{O}(t^3 + t^2 d_t)$. Maximizing an acquisition function is related to the optimization algorithm. If we use the Quasi-Newton method to optimize GP-UCB, the computational complexity is $\mathcal{O}(m(t^2 + td_t + d_t^2))$ [28], where m denotes the Quasi-Newton's running rounds. The cost of evaluating a sampled point is fixed. Thus, by selecting only a subset of variables, instead of all variables, to optimize, the computational complexity can be decreased significantly. The detailed analysis is provided in Appendix B.2.

Insight. The above regret and computational complexity analyses have shown that variable selection can reduce the computational complexity while increasing the regret. Given the number d_t of variables to be selected, a good variable selection method should select as important variables as possible, i.e., variables with as large α_i^* as possible, which may help to design and evaluate variable selection methods. The experiments in Section 5.1 will show that MCTS-VS can select a good subset of variables while maintaining a small computational complexity.

5 Experiment

To examine the performance of MCTS-VS, we conduct experiments on different tasks, including synthetic functions, NAS-bench problems and MuJoCo locomotion tasks, to compare MCTS-VS with other black-box optimization methods. For MCTS-VS, we use the same hyper-parameters except C_p, which is used for calculating UCB in Eq. (1). For Dropout and embedding-based methods, we set the parameter d to the number of valid dimensions for synthetic functions, and a reasonable value for real-world problems. The hyper-parameters of the same components of different methods are set to the same. We use five identical random seeds (2021–2025) for all problems and methods. More details about the settings can be found in Appendix C. Our code is available at `https://github.com/lamda-bbo/MCTS-VS`.

5.1 Synthetic Functions

We use Hartmann ($d = 6$) and Levy ($d = 10$) as the synthetic benchmark functions, and extend them to high dimensions by adding unrelated variables as [20, 27, 40]. For example, Hartmann6_300 has the dimension $D = 300$, and is generated by appending 294 unrelated dimensions to Hartmann. The variables affecting the value of f are called *valid variables*.

Effectiveness of Variable Selection. Dropout [21] is the previous variable selection method which randomly selects d variables in each iteration, while our proposed MCTS-VS applies MCTS to automatically select important variables. We compare them against vanilla BO [32] without variable selection. The first two subfigures in Figure 1 show that Dropout-BO and MCTS-VS-BO are better than vanilla BO, implying the effectiveness of variable selection. We can also see that MCTS-VS-BO performs the best, implying the superiority of MCTS-based variable selection over random selection.

We also equip MCTS-VS and Dropout with the advanced BO algorithm TuRBO [10], resulting in MCTS-VS-TuRBO and Dropout-TuRBO. The last two subfigures in Figure 1 show the similar results except that MCTS-VS-TuRBO needs more evaluations to be better than Dropout-TuRBO. This is because TuRBO costs more evaluations than BO on the same selected variables, and thus needs more evaluations to generate sufficient samples for an accurate estimation of the variable score in Eq. (2).

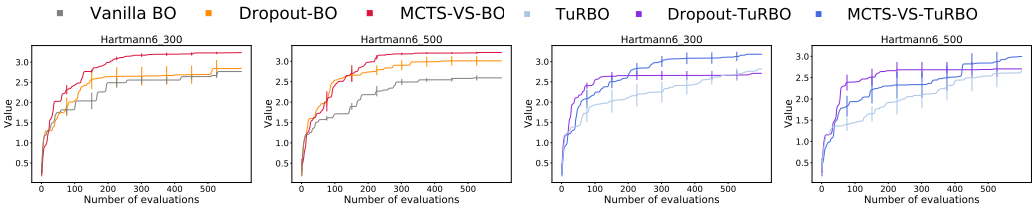


Figure 1: Performance comparison among the two variable selection methods (i.e., MCTS-VS and Dropout) and the BO methods (i.e., Vanilla BO and TuRBO) on two synthetic functions.

Comparison with State-of-The-Art Methods. We compare MCTS-VS with the state-of-the-art methods, including TuRBO [10], LA-MCTS-TuRBO [38], SAASBO [9], HeSBO [27], ALEBO [20] and CMA-ES [14]. TuRBO fits a collection of local models to optimize in the trust regions for overcoming the homogeneity of the global model and over-exploration. LA-MCTS-TuRBO applies MCTS to partition the search space and uses TuRBO to optimize in a small sub-region. SAASBO uses sparsity-inducing prior to select variables implicitly. HeSBO and ALEBO are state-of-the-art embedding methods. CMA-ES is a popular evolutionary algorithm. We also implement VAE-BO by combining VAE [18] with vanilla BO directly, as a baseline of learning-based embedding. For MCTS-VS, we implement the two versions of MCTS-VS-BO and MCTS-VS-TuRBO, i.e., MCTS-VS equipped with vanilla BO and TuRBO.

As shown in Figure 2, MCTS-VS can achieve the best performance except on Levy10_100, where it is a little worse than TuRBO. For low-dimensional functions (e.g., $D = 100$ for Levy10_100), TuRBO can adjust the trust region quickly while MCTS-VS needs samples to estimate the variable score. But as the dimension increases, the search space increases exponentially and it becomes difficult for TuRBO to adjust the trust region; while the number of variables only increases linearly, making MCTS-VS more scalable. SAASBO has similar performance to MCTS-VS due to the advantage of sparsity-inducing prior. HeSBO is not stable, which has a moderate performance on Hartmann but a relatively good performance on Levy. Note that we only run SAASBO and ALEBO for 200 evaluations on Hartmann functions because it has already taken more than hours to finish one iteration when the number of samples is large. More details about runtime are shown in Table 1. VAE-BO has the worst performance, suggesting that the learning algorithm in high-dimensional BO needs to be designed carefully. We also conduct experiments on extremely low and high dimensional variants of Hartmann (i.e., Hartmann6_100 and Hartmann6_1000), showing that MCTS-VS still performs well, and perform the significance test by running each method more times. Please see Appendix E.

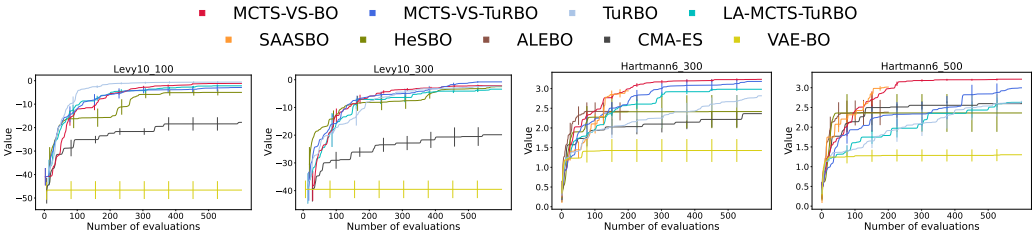


Figure 2: Comparison among MCTS-VS and state-of-the-art methods on synthetic functions.

Next, we compare the practical running overheads of these methods. We run each method for 100 evaluations independently using 30 different random seeds, and calculate the average wall clock time. The results are shown in Table 1. As expected, when using variable selection (i.e., Dropout and MCTS-VS), the time is less than that of Vanilla BO or TuRBO, because we only optimize a subset of variables. MCTS-VS is a little slower than Dropout, which is because MCTS-VS needs to build the search tree and calculate the variable score, while Dropout only randomly selects variables. MCTS-VS is much faster than LA-MCTS-TuRBO, showing the advantage of partitioning the variables to partitioning the search space. SAASBO optimizes all variables instead of only a subset of variables

Table 1: Wall clock time (in seconds) comparison among different methods.

METHOD	LEVY10_100	LEVY10_300	HARTMANN6_300	HARTMANN6_500
VANILLA BO	3.190	4.140	4.844	5.540
DROPOUT-BO	2.707	3.225	3.237	3.685
MCTS-VS-BO	2.683	3.753	3.711	4.590
TURBO	8.621	9.206	9.201	9.754
LA-MCTS-TURBO	14.431	22.165	25.853	34.381
MCTS-VS-TURBO	4.912	5.616	5.613	5.893
SAASBO	/	/	2185.678	4163.121
HESBO	220.459	185.092	51.678	55.699
ALEBO	/	/	470.714	512.641
CMA-ES	0.030	0.043	0.043	0.045

Table 2: Recall comparison between MCTS-VS and Dropout.

METHOD	LEVY10_100	LEVY10_300	HARTMANN6_300	HARTMANN6_500
DROPOUT	0.100	0.030	0.020	0.012
MCTS-VS	0.429	0.433	0.352	0.350

and uses No-U-Turn sampler (NUTS) to inference, consuming $\times 500 - \times 1000$ time. HeSBO and ALEBO consume $\times 10 - \times 500$ time compared with the variable selection methods. CMA-ES is very fast because it does not need to fit a GP model or optimize an acquisition function. The reasons for the small running overhead of MCTS-VS can be summarized as follows: 1) it only optimizes a selected subset of variables; 2) the depth of the search tree is shallow, i.e., $O(\log D)$ in expectation and less than D in the worse case; 3) the variable score vector in Eq. (2) is easy to calculate for bifurcating a tree node.

Why MCTS-VS Can Perform Well. The theoretical results have suggested that a good variable selection method should select as important variables as possible. Thus, we compare the quality of the variables selected by MCTS-VS and Dropout (i.e., random selection), measured by the recall d_t^*/d, where d is the number of valid variables, and d_t^* is the number of valid variables selected at iteration t. Dropout randomly selects d variables at each iteration, and thus, the recall is d/D in expectation. For MCTS-VS, we run MCTS-VS-BO for 600 evaluations on five different random seeds, and calculate the average recall. As shown in Table 2, the average recall of MCTS-VS is much larger than that of Dropout, implying that MCTS-VS can select better variables than random selection, and thus achieve a good performance as shown before. Meanwhile, the recall between 0.35 and 0.433 of MCTS-VS also implies that the variable selection method could be further improved.

5.2 Real-World Problems

We further compare MCTS-VS with the baselines on real-world problems, including NAS-Bench-101 [43], NAS-Bench-201 [7], Hopper and Walker2d. NAS-Bench problems are popular benchmarks in high-dimensional BO. Hopper and Walker2d are robot locomotion tasks in MuJoCo [37], which is a popular black-box optimization benchmark and much more difficult than NAS-Bench problems. The experimental results on more real-world problems can refer to Appendix E.

NAS-Bench Problems. NAS-Bench-101 is a tabular data set that maps convolutional neural network architectures to their trained and evaluated performance on CIFAR-10, and we create a constrained problem with $D = 36$ in the same way as [20]. NAS-Bench-201 is an extension to NAS-Bench-101, leading to a problem with $D = 30$ but without constraints. Figure 3 shows the results with the wall clock time as the x-axis, where the gray dashed line denotes the optimum. The results using the number of evaluations as the x-axis are provided in Appendix E, showing that the performance of BO-style methods is similar, as already observed in [20]. This may be because there are many structures whose objective values are close to the optimum. But when considering the actual runtime, MCTS-VS-BO is still clearly better as shown in Figure 3, due to the advantage of variable selection. We also provide results on more NAS-Bench problems, including NAS-Bench-1Shot1 [44], TransNAS-Bench-101 [8] and NAS-Bench-ASR [25] in Appendix E.

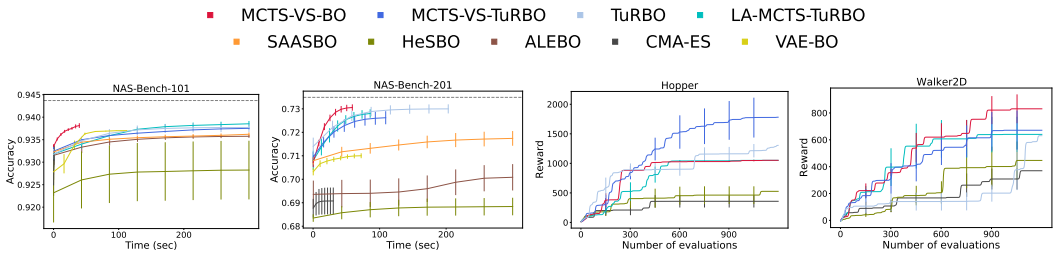


Figure 3: Comparison on NAS-Bench.

Figure 4: Comparison on MuJoCo.

MuJoCo Locomotion Tasks. Next we turn to the more difficult MuJoCo tasks in RL. The goal is to find the parameters of a linear policy maximizing the accumulative reward. Different from previous problems, the objective f (i.e., the accumulative reward) is highly stochastic here, making it difficult to solve. We use the mean of three independent evaluations to estimate f, and limit the evaluation budget to 1200 due to expensive evaluation. Note that we do not run SAASBO, ALEBO, and VAE-BO because SAASBO and ALEBO are extremely time-consuming, and VAE-BO behaves badly in previous experiments. The results are shown in Figure 4. TuRBO behaves well on Hopper with a low dimension $D = 33$, and MCTS-VS-TuRBO, combining the advantage of variable selection and TuRBO, achieves better performance, outperforming all the other baselines. On Walker2d with a high dimension $D = 102$, MCTS-VS-BO performs the best, because of the good scalability. Most methods have large variance due to the randomness of f. For HeSBO, we have little knowledge about the parameter d, and use 10 and 20 for Hopper and Walker2d, respectively. Its performance may be improved by choosing a better d, which, however, requires running the experiment many times, and is time-consuming. Note that on the two MuJoCo tasks, Hopper and Walker2d, each variable is valid. The good performance of MCTS-VS may be because optimizing only a subset of variables is sufficient for achieving the goal and MCTS-VS can select them. For example, the Walker2D robot consists of four main body parts: a torso, two thighs, two legs and two feet, where adjacent ones are connected by two hinges. The goal is to move forward by optimizing the hinges, each of which is valid. But even locking the hinges between legs and feet, the robot can still move forward by optimizing the other hinges. This is similar to that when the ankles are fixed, a person can still walk.

Further Studies. We further perform sensitivity analysis about the hyper-parameters of MCTS-VS, including the employed optimizer, "fill-in" strategy, C_p for calculating UCB in Eq. (1), number $2 \times N_v \times N_s$ of sampled data in each iteration, threshold N_{bad} for re-initializing a tree and N_{split} for splitting a tree node. Please see Appendix D. We conduct additional experiments in Appendix E, including experiments on synthetic functions depending on a subset of variables to various extent and with increasing ratio of valid variables, examination of combining MCTS-VS with SAASBO (which can be viewed as a hierarchical variable selection method), and comparison with other variable selection methods (e.g., LASSO).

6 Conclusion

In this paper, we propose the MCTS-VS method for variable selection in high-dimensional BO, which uses MCTS to recursively partition the variables into important and unimportant ones, and only optimizes those important variables. Theoretical results suggest selecting as important variables as possible, which may be of independent interest for variable selection. Comprehensive experiments on synthetic, NAS-bench and MuJoCo problems demonstrate the effectiveness of MCTS-VS.

However, MCTS-VS relies on the assumption of low effective dimensionality, and might not work well if the percentage of valid variables is high. The amount of hyper-parameters might be another limitation, though our sensitivity analysis has shown that the performance of MCTS-VS is not sensitive to most hyper-parameters. The current theoretical analysis is for general variable selection, while it will be very interesting to perform specific theoretical analysis for MCTS-VS.

Acknowledgement

The authors would like to thank reviewers for their helpful comments and suggestions. This work was supported by the NSFC (62022039, 62276124) and the Fundamental Research Funds for the Central Universities (0221-14380014).

References

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[2] M. Binois and N. Wycoff. A survey on high-dimensional Gaussian process modeling with application to Bayesian optimization. *ACM Transactions on Evolutionary Learning and Optimization*, 2(2):1–26, 2022.

[3] M. Binois, D. Ginsbourger, and O. Roustant. A warped kernel improving robustness in Bayesian optimization via random embeddings. In *Proceedings of the 9th International Conference on Learning and Intelligent Optimization (LION'15)*, pages 281–286, Lille, France, 2015.

[4] M. Binois, D. Ginsbourger, and O. Roustant. On the choice of the low-dimensional domain for global optimization via random embeddings. *Journal of Global Optimization*, 76(1):69–90, 2020.

[5] C. Browne, E. J. Powley, D. Whitehouse, S. M. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. P. Liebana, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[6] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1):5–23, 2015.

[7] X. Dong and Y. Yang. NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *Proceedings of the 8th International Conference on Learning Representations (ICLR'20)*, Addis Ababa, Ethiopia, 2020.

[8] Y. Duan, X. Chen, H. Xu, Z. Chen, X. Liang, T. Zhang, and Z. Li. TransNAS-Bench-101: Improving transferability and generalizability of cross-task neural architecture search. *CoRR abs/2105.11871*, 2021.

[9] D. Eriksson and M. Jankowiak. High-dimensional Bayesian optimization with sparse axis-aligned subspaces. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence (UAI'21)*, pages 493–503, Virtual, 2021.

[10] D. Eriksson, M. Pearce, J. R. Gardner, R. D. Turner, and M. Poloczek. Scalable global optimization via local Bayesian optimization. In *Advances in Neural Information Processing Systems 32 (NeurIPS'19)*, pages 5497–5508, Vancouver, Canada, 2019.

[11] P. I. Frazier. A tutorial on Bayesian optimization. *CoRR abs/1807.02811*, 2018.

[12] R. Gómez-Bombarelli, D. K. Duvenaud, J. M. Hernández-Lobato, J. Aguilera-Iparraguirre, T.D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268 – 276, 2018.

[13] E. Han, I. Arora, and J. Scarlett. High-dimensional Bayesian optimization via tree-structured additive models. In *Proceedings of the 35th Association for the Advancement of Artificial Intelligence (AAAI'21)*, pages 7630–7638, Virtual, 2021.

[14] N. Hansen. The CMA evolution strategy: A tutorial. *CoRR abs/1604.00772*, 2016.

[15] T. N. Hoang, Q. M. Hoang, R. Ouyang, and K. H. Low. Decentralized high-dimensional Bayesian optimization with factor graphs. In *Proceedings of the 32nd Association for the Advancement of Artificial Intelligence (AAAI'18)*, pages 3231–3239, New Orleans, LA, 2018.

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

[17] K. Kandasamy, J. G. Schneider, and B. Póczos. High dimensional Bayesian optimisation and bandits via additive models. In *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, pages 295–304, Lille, France, 2015.

[18] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *CoRR abs/1312.6114*, 2014.

[19] H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

[20] B. Letham, R. Calandra, A. Rai, and E. Bakshy. Re-examining linear embeddings for high-dimensional Bayesian optimization. In *Advances in Neural Information Processing Systems 33 (NeurIPS'20)*, pages 1546–1558, Vancouver, Canada, 2020.

[21] C. Li, S. Gupta, S. Rana, V. Nguyen, S. Venkatesh, and A. Shilton. High dimensional Bayesian optimization using dropout. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, pages 2096–2102, Melbourne, Australia, 2017.

[22] X. Lu, J. I. González, Z. Dai, and N. D. Lawrence. Structured variationally auto-encoded optimization. In *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, pages 3306–3314, Stockholm, Sweden, 2018.

[23] M. Malu, G. Dasarathy, and A. Spanias. Bayesian optimization in high-dimensional spaces: A brief survey. In *Proceedings of the 12th International Conference on Information, Intelligence, Systems & Applications (IISA'21)*, pages 1–8, Virtual, 2021.

[24] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2): 239–245, 1979.

[25] A. Mehrotra, A. G. C. P. Ramos, S. Bhattacharya, Ł. Dudziak, R. Vipperla, T. Chau, M. S. Abdelfattah, S. Ishtiaq, and N. D. Lane. NAS-Bench-ASR: Reproducible neural architecture search for speech recognition. In *Proceedings of the 9th International Conference on Learning Representations (ICLR'21)*, Virtual, 2021.

[26] M. Mutný and A. Krause. Efficient high dimensional Bayesian optimization with additivity and quadrature Fourier features. In *Advances in Neural Information Processing Systems 31 (NeurIPS'18)*, pages 9005–9016, Montreal, Canada, 2018.

[27] A. Nayebi, A. Munteanu, and M. Poloczek. A framework for Bayesian optimization in embedded subspaces. In *Proceedings of the 36th International Conference on Machine LearninG (ICML'19)*, pages 4752–4761, Long Beach, CA, 2019.

[28] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, NY, second edition edition, 2006.

[29] C. Qian, H. Xiong, and K. Xue. Bayesian optimization using pseudo-points. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI'20)*, pages 3044–3050, Yokohama, Japan, 2020.

[30] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, MA, 2006.

[31] P. Rolland, J. Scarlett, I. Bogunovic, and V. Cevher. High-dimensional Bayesian optimization via additive models with overlapping groups. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS'18)*, pages 298–307, Playa Blanca, Spain, 2018.

[32] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[33] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489, 2016.

[34] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676): 354–359, 2017.

[35] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams. Scalable Bayesian optimization using deep neural networks. In *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, pages 2171–2180, Lille, France, 2015.

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

[37] E. Todorov, E. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[38] L. Wang, R. Fonseca, and Y. Tian. Learning search space partition for black-box optimization using Monte Carlo tree search. In *Advances in Neural Information Processing Systems 33 (NeurIPS'20)*, pages 19511–19522, Vancouver, Canada, 2020.

[39] L. Wang, S. Xie, T. Li, R. Fonseca, and Y. Tian. Sample-efficient neural architecture search by learning actions for Monte Carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[40] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55(1):361–387, 2016.

[41] Z. Wang, C. Gehring, P. Kohli, and S. Jegelka. Batched large-scale Bayesian optimization in high-dimensional spaces. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS'18)*, pages 745–754, Playa Blanca, Spain, 2018.

[42] J. T. Wilson, R. Moriconi, F. Hutter, and M. P. Deisenroth. The reparameterization trick for acquisition functions. *CoRR abs/1712.00424*, 2017.

[43] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, pages 7105–7114, Long Beach, CA, 2019.

[44] A. Zela, J. Siems, and F. Hutter. NAS-Bench-1Shot1: Benchmarking and dissecting one-shot neural architecture search. In *Proceedings of the 8th International Conference on Learning Representations (ICLR'20)*, Addis Ababa, Ethiopia, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See the end of Section 5.1 and the last paragraph of the paper.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 4.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix B.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Appendix C, and the code is provided in GitHub.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix C.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We show error bars by the length of vertical bars in the figures.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [Yes]

   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Example Illustration of MCTS-VS

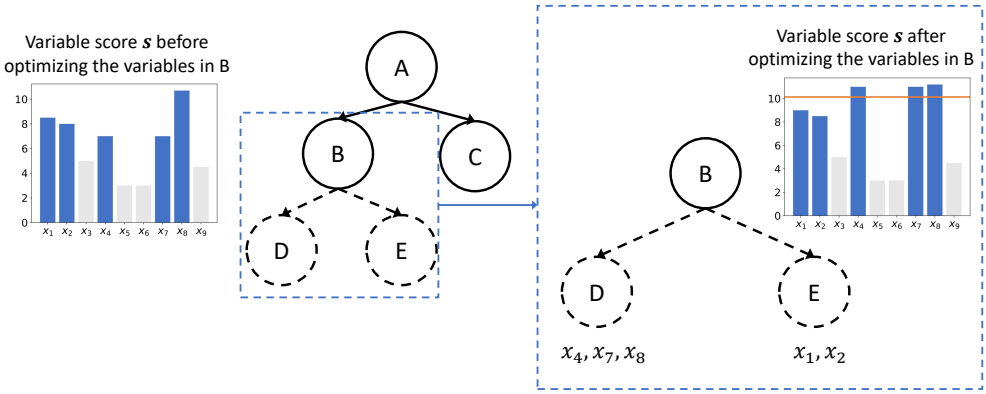


Figure 5: Example illustration of how MCTS-VS bifurcates a leaf node.

Figure 5 gives an example of how MCTS-VS bifurcates a leaf node. Assume that we are to optimize a problem with dimension $D = 9$, and the variables are denoted as $x_1, x_2, \ldots, x_9$. The Monte Carlo tree shown in the middle of Figure 5 now has three nodes, i.e., A, B and C, denoted as the solid circles. The root A contains all the nine variables. The current variable score vector $\boldsymbol{s} = [8.5, 8, 5, 7, 3, 3, 7, 10.7, 4.5]$, which is represented by the bar graph as shown in the left of Figure 5. For each i, the value of s_i represents the importance of the corresponding variable x_i. The blue and gray bars denote the important and unimportant variables, respectively, which are contained by the leaf nodes B and C, respectively. That is, the leaf B contains x_1, x_2, x_4, x_7, x_8, and C contains the remaining x_3, x_5, x_6, x_9. The current v values (i.e., the average scores of the contained variables) of the three nodes A, B and C are $v_A = (8.5 + 8 + 5 + 7 + 3 + 3 + 7 + 10.7 + 4.5)/9 = 6.3$, $v_B = (8.5 + 8 + 7 + 7 + 10.7)/5 = 8.24$ and $v_C = (5 + 3 + 3 + 4.5)/4 = 3.875$, respectively. For the number that they have been visited, we have $n_A = 1$, $n_B = 0$ and $n_C = 0$.

MCTS-VS starts from the root node A at one iteration and recursively selects a node with a larger UCB value until a leaf node. According to the way of calculating UCB in Eq. (1), the UCB values of the leaf nodes B and C are both ∞ as $n_B = n_C = 0$. In this case, MCTS-VS will select B and C randomly. Assume that B is selected. The variables (i.e., x_1, x_2, x_4, x_7 and x_8) contained by B will then be optimized by BO with $\mathbb{A}_B = \{1, 2, 4, 7, 8\}$, as in lines 13–23 in Algorithm 1. After that, the variable score vector $\boldsymbol{s}$ will be re-calculated, which is assumed to be $[9, 8.5, 5, 11, 3, 3, 11, 11.2, 4.5]$, as shown in the right of Figure 5. The average score of the five variables in B is denoted as the orange horizontal line, calculated by $(9 + 8.5 + 11 + 11 + 11.2)/5 = 10.14$. We can see that the variables x_4, x_7 and x_8 have score larger than the average value 10.14, which are regarded as more important variables in B. We use the left child D to represent these variables. The scores of variables x_1 and x_2 are smaller than the average, which are regarded as less important variables in B. We use the right child E to represent them. Thus, the node B has been partitioned into two children D and E, denoted as the dashed circles in Figure 5. The v values and the number of visits of these two new leaf nodes are then calculated. The v value of a node is the average score of the contained variables. Thus, v_D is the average score of x_4, x_7 and x_8, i.e., $(11 + 11 + 11.2)/3 = 11.067$, and v_E is the average score of x_1 and x_2, i.e., $(9 + 8.5)/2 = 8.75$. For the number of visits, obviously $n_D = n_E = 0$. Finally, back-propagation is performed to update the v value and the number of visits of the nodes along the path from the root A to the node B. v_A is the average score of all the variables, i.e., $(9 + 8.5 + 5 + 11 + 3 + 3 + 11 + 11.2 + 4.5)/9 = 7.356$. v_B is the average score of x_1, x_2, x_4, x_7 and x_8, i.e., $(9 + 8.5 + 11 + 11 + 11.2)/5 = 10.14$. Their number of visits will be increased by 1. That is, $n_A = 2$ and $n_B = 1$. By far, one iteration of MCTS-VS has been finished, and this process will be performed iteratively.

B Details of Theoretical Analysis

B.1 Detailed Proof of Theorem 4.2

The proof is inspired by [36]. To prove the upper bound on the cumulative regret R_T in Theorem 4.2, we analyze the instantaneous regret $r_t = f(\boldsymbol{x}^*) - f(\boldsymbol{x}^t_{\mathbb{M}_t})$, i.e., the gap between the function values of the optimal point $\boldsymbol{x}^*$ and the sampled point $\boldsymbol{x}^t_{\mathbb{M}_t}$ at iteration t. Note that $R_T = \sum_{t=1}^T r_t$. Let $\mu_{t-1}(\cdot)$ and $\sigma^2_{t-1}(\cdot)$ denote the posterior mean and variance after running $t-1$ iterations, respectively. Lemma B.1 gives a confidence bound on $f(\boldsymbol{x}^t_{\mathbb{M}_t})$, leading to a lower bound on $f(\boldsymbol{x}^t_{\mathbb{M}_t})$, i.e., $f(\boldsymbol{x}^t_{\mathbb{M}_t}) \geq \mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}) - \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t})$. Note that $\mathbb{M}_t$ denotes the sampled variable index subset at iteration t, and $|\mathbb{M}_t| = d_t$.

Lemma B.1. *$\forall \delta \in (0,1), \forall t \geq 1$, let $\beta_t = 2\log(\pi_t/\delta)$, where $\sum_{t\geq 1}\pi_t^{-1} = 1, \pi_t > 0$. Then, $\forall t \geq 1$,*

$$|f(\boldsymbol{x}^t_{\mathbb{M}_t}) - \mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t})| \leq \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t})$$

holds with probability at least $1-\delta$, where $\boldsymbol{x}^t_{\mathbb{M}_t}$ is the point obtained at iteration t.

Proof. At iteration t, $f(\boldsymbol{x}^t_{\mathbb{M}_t}) \sim \mathcal{N}(\mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}), \sigma^2_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}))$, and thus, $Y = \frac{f(\boldsymbol{x}^t_{\mathbb{M}_t}) - \mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t})}{\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t})} \sim \mathcal{N}(0,1)$. We have

$$\begin{aligned}
&P\left(|f(\boldsymbol{x}^t_{\mathbb{M}_t}) - \mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t})| > \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t})\right) \\
&= P\left(|Y| > \beta_t^{1/2}\right) = 2\int_{\beta_t^{1/2}}^{\infty}(2\pi)^{(-1/2)}\exp\left(-\frac{y^2}{2}\right)dy \\
&= 2\exp\left(-\frac{\beta_t}{2}\right)\int_{\beta_t^{1/2}}^{\infty}(2\pi)^{(-1/2)}\exp\left(-\frac{(y-\beta_t^{1/2})^2}{2}\right)\exp\left(-\beta_t^{1/2}(y-\beta_t^{1/2})\right)dy \\
&\leq 2\exp\left(-\frac{\beta_t}{2}\right)P(Y>0) \leq \exp\left(-\frac{\beta_t}{2}\right) = \frac{\delta}{\pi_t}.
\end{aligned}$$

Using the union bound for all $t \in \mathbb{N}$, we have

$$P\left(\forall t \geq 1 : |f(\boldsymbol{x}^t_{\mathbb{M}_t}) - \mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t})| \leq \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t})\right) \geq 1 - \sum_{t\geq 1}\frac{\delta}{\pi_t} = 1-\delta,$$

where the equality holds by $\sum_{t\geq 1}\pi_t^{-1} = 1$. Thus, the lemma holds. □

Next we are to analyze the upper bound on $f(\boldsymbol{x}^*)$, which can be represented as $(f(\boldsymbol{x}^*) - f(\boldsymbol{x}^*_{\mathbb{M}_t})) + f(\boldsymbol{x}^*_{\mathbb{M}_t})$, where $\boldsymbol{x}^*_{\mathbb{M}_t}$ denotes the point obtained by projecting $\boldsymbol{x}^*$ onto $\mathbb{M}_t$. The first term $f(\boldsymbol{x}^*) - f(\boldsymbol{x}^*_{\mathbb{M}_t})$ can be upper bounded by Assumption 4.1. To upper bound the second term $f(\boldsymbol{x}^*_{\mathbb{M}_t})$, we need to discretize the decision space $\mathcal{X}_{\mathbb{M}_t}$ at iteration t into $\tilde{\mathcal{X}}_{\mathbb{M}_t}$, where $|\tilde{\mathcal{X}}_{\mathbb{M}_t}| = (\tau_t)^{d_t}$, i.e., we divide each variable of $\mathcal{X}_{\mathbb{M}_t}$ into τ_t parts equally. Let $\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}$ denote the point closest to $\boldsymbol{x}^*_{\mathbb{M}_t}$ in the discretized space $\tilde{\mathcal{X}}_{\mathbb{M}_t}$. Then, we can write $f(\boldsymbol{x}^*_{\mathbb{M}_t})$ as $(f(\boldsymbol{x}^*_{\mathbb{M}_t}) - f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})) + f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})$. The first term $f(\boldsymbol{x}^*_{\mathbb{M}_t}) - f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})$ again can be upper bounded by Assumption 4.1. Lemma B.2 gives a confidence bound on $f(\tilde{\boldsymbol{x}}_{\mathbb{M}_t})$ for any discretized point $\tilde{\boldsymbol{x}}_{\mathbb{M}_t} \in \tilde{\mathcal{X}}_{\mathbb{M}_t}$, leading to an upper bound on $f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})$, i.e., $f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) \leq \mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) + \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})$.

Lemma B.2. *$\forall \delta \in (0,1), \forall t \geq 1$, let $\beta_t = 2\log(|\tilde{\mathcal{X}}_{\mathbb{M}_t}|\pi_t/\delta)$, where $\sum_{t\geq 1}\pi_t^{-1} = 1, \pi_t > 0$. Then, $\forall t \geq 1, \forall \tilde{\boldsymbol{x}}_{\mathbb{M}_t} \in \tilde{\mathcal{X}}_{\mathbb{M}_t}$,*

$$|f(\tilde{\boldsymbol{x}}_{\mathbb{M}_t}) - \mu_{t-1}(\tilde{\boldsymbol{x}}_{\mathbb{M}_t})| \leq \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}_{\mathbb{M}_t})$$

holds with probability at least $1-\delta$.

Proof. Similar to Lemma B.1, we can derive

$$P\left(|f(\tilde{\boldsymbol{x}}_{\mathbb{M}_t}) - \mu_{t-1}(\tilde{\boldsymbol{x}}_{\mathbb{M}_t})| > \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}_{\mathbb{M}_t})\right) \leq \exp\left(-\frac{\beta_t}{2}\right) = \frac{\delta}{|\tilde{\mathcal{X}}_{\mathbb{M}_t}|\pi_t}.$$

Using the union bound for all $t \in \mathbb{N}$ and $\tilde{\boldsymbol{x}}_{\mathbb{M}_t} \in \tilde{\mathcal{X}}_{\mathbb{M}_t}$, we have

$$\begin{aligned} &P\left(\forall t \geq 1, \forall \tilde{\boldsymbol{x}}_{\mathbb{M}_t} \in \tilde{\mathcal{X}}_{\mathbb{M}_t} : |f(\tilde{\boldsymbol{x}}_{\mathbb{M}_t}) - \mu_{t-1}(\tilde{\boldsymbol{x}}_{\mathbb{M}_t})| \leq \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}_{\mathbb{M}_t})\right) \\ &\geq 1 - \sum_{t\geq 1}\sum_{\tilde{\boldsymbol{x}}_{\mathbb{M}_t} \in \tilde{\mathcal{X}}_{\mathbb{M}_t}} \frac{\delta}{|\tilde{\mathcal{X}}_{\mathbb{M}_t}|\pi_t} = 1 - \delta. \end{aligned}$$

Thus, the lemma holds. □

Now, we can upper bound $f(\boldsymbol{x}^*_{\mathbb{M}_t})$ based on Assumption 4.1 and Lemma B.2, as shown in Lemma B.3. Note that $\boldsymbol{x}^*_{\mathbb{M}_t}$ denotes the point obtained by projecting $\boldsymbol{x}^*$ onto $\mathbb{M}_t$, and $\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}$ denotes the point closest to $\boldsymbol{x}^*_{\mathbb{M}_t}$ in $\tilde{\mathcal{X}}_{\mathbb{M}_t}$.

Lemma B.3. *$\forall \delta \in (0,1), t \geq 1$, let $\beta_t = 2\log(2\pi_t/\delta) + 2d_t \log\left(d_t t^2 br\sqrt{\log\frac{2Da}{\delta}}\right)$, where $\sum_{t\geq 1}\pi_t^{-1} = 1, \pi_t > 0$. Set $\tau_t = d_t t^2 br\sqrt{\log\frac{2Da}{\delta}}$ and $L = b\sqrt{\log\frac{2Da}{\delta}}$. Then, $\forall t \geq 1$,*

$$|f(\boldsymbol{x}^*_{\mathbb{M}_t}) - \mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})| \leq \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) + \frac{\alpha_{\max}}{t^2} + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr$$

holds with probability at least $1-\delta$.

Proof. First, we have

$$\begin{aligned} |f(\boldsymbol{x}^*_{\mathbb{M}_t}) - \mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})| &= |f(\boldsymbol{x}^*_{\mathbb{M}_t}) - f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) + f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) - \mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})| \\ &\leq |f(\boldsymbol{x}^*_{\mathbb{M}_t}) - f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})| + |f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) - \mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})|. \end{aligned} \tag{7}$$

By Assumption 4.1 with $L = b\sqrt{\log\frac{2Da}{\delta}}$, we have $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$, with probability at least $1 - D \cdot ae^{-(L/b)^2} = 1 - \delta/2$,

$$\begin{aligned} |f(\boldsymbol{x}) - f(\boldsymbol{y})| &\leq \sum_{i=1}^{D}\alpha_i^* L|x_i - y_i| \\ &\leq \sum_{i\in\mathbb{M}_t}\alpha_i^* L|x_i - y_i| + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr \\ &\leq \alpha_{\max} L\|\boldsymbol{x}_{\mathbb{M}_t} - \boldsymbol{y}_{\mathbb{M}_t}\|_1 + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr, \end{aligned} \tag{8}$$

where the second inequality holds by $\mathcal{X} \subset [0,r]^D$, and the last inequality holds by $\alpha_{\max} = \max_{i\in[D]}\alpha_i^*$. Thus, it holds with probability at least $1-\delta/2$ that

$$|f(\boldsymbol{x}^*_{\mathbb{M}_t}) - f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})| \leq \alpha_{\max} L\|\boldsymbol{x}^*_{\mathbb{M}_t} - \tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}\|_1 + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr. \tag{9}$$

By Lemma B.2 with $\beta_t = 2\log(2(\tau_t)^{d_t}\pi_t/\delta) = 2\log(2|\tilde{\mathcal{X}}_{\mathbb{M}_t}|\pi_t/\delta)$, we have, with probability at least $1-\delta/2$,

$$|f(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) - \mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})| \leq \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}). \tag{10}$$

Applying Eqs. (9) and (10) to Eq. (7), it holds with probability at least $1-\delta$ that

$$\begin{aligned} |f(\boldsymbol{x}^*_{\mathbb{M}_t}) - \mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t})| &\leq \alpha_{\max} L\|\boldsymbol{x}^*_{\mathbb{M}_t} - \tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}\|_1 + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr + \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) \\ &\leq \alpha_{\max} L\frac{d_t r}{\tau_t} + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr + \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) \\ &\leq \frac{\alpha_{\max}}{t^2} + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr + \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}), \end{aligned}$$

where the second inequality holds by $|\mathbb{M}_t| = d_t$ and the way of discretization (i.e., each variable is discretized into τ_t parts equally), and the last inequality holds by the definition of τ_t and L. Thus, the lemma holds. □

Lemma B.3 implies an upper bound on $f(\boldsymbol{x}^*_{\mathbb{M}_t})$, i.e., $f(\boldsymbol{x}^*_{\mathbb{M}_t}) \leq \mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) + \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) + \alpha_{\max}/t^2 + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr$. Combining this upper bound on $f(\boldsymbol{x}^*_{\mathbb{M}_t})$ with $f(\boldsymbol{x}^*) - f(\boldsymbol{x}^*_{\mathbb{M}_t})$ (which can be upper bounded by Assumption 4.1), we can derive an upper bound on $f(\boldsymbol{x}^*)$. Together with the lower bound on $f(\boldsymbol{x}^t_{\mathbb{M}_t})$ given by Lemma B.1, we can derive an upper bound on the instantaneous regret r_t. Thus, we are now ready to prove the upper bound on the cumulative regret R_T in Theorem 4.2, which is re-stated in Theorem B.4 for clearness.

Theorem B.4. *$\forall\delta \in (0,1)$, let $\beta_t = 2\log(4\pi_t/\delta) + 2d_t\log(d_t t^2 br\sqrt{\log(4Da/\delta)})$ and $L = b\sqrt{\log(4Da/\delta)}$, where $\{\pi_t\}_{t\geq 1}$ satisfies $\sum_{t\geq 1}\pi_t^{-1} = 1$ and $\pi_t > 0$. Let $\beta_T^* = \max_{1\leq i\leq T}\beta_t$. At iteration T, the cumulative regret*

$$R_T \leq \sqrt{C_1 T\beta_T^*\gamma_T} + 2\alpha_{\max} + 2\sum_{t=1}^{T}\sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr$$

holds with probability at least $1-\delta$, where $C_1 > 0$ is a constant, $\gamma_T = \max_{|\mathcal{D}|=T} I(\boldsymbol{y}_\mathcal{D}, \boldsymbol{f}_\mathcal{D})$, $I(\cdot,\cdot)$ denotes the information gain, and $\boldsymbol{y}_\mathcal{D}$ and $\boldsymbol{f}_\mathcal{D}$ are the noisy and true observations of a set $\mathcal{D}$ of points, respectively.

Proof. For all $t \geq 1$, we have

$$r_t = f(\boldsymbol{x}^*) - f(\boldsymbol{x}^t_{\mathbb{M}_t}) = f(\boldsymbol{x}^*) - f(\boldsymbol{x}^*_{\mathbb{M}_t}) + f(\boldsymbol{x}^*_{\mathbb{M}_t}) - f(\boldsymbol{x}^t_{\mathbb{M}_t}). \tag{11}$$

By Eq. (8), we have

$$f(\boldsymbol{x}^*) - f(\boldsymbol{x}^*_{\mathbb{M}_t}) \leq \alpha_{\max}L\|\boldsymbol{x}^*_{\mathbb{M}_t} - \boldsymbol{x}^*_{\mathbb{M}_t}\|_1 + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr = \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr. \tag{12}$$

Note that $L = b\sqrt{\log(4Da/\delta)}$ here, and thus Eq. (12) holds with probability at least $1-\delta/4$. By Lemma B.3 with $\beta_t = 2\log(4\pi_t/\delta) + 2d_t\log(d_t t^2 br\sqrt{\log(4Da/\delta)})$ and $L = b\sqrt{\log(4Da/\delta)}$, setting $\tau_t = d_t t^2 br\sqrt{\log(4Da/\delta)}$ leads to that

$$f(\boldsymbol{x}^*_{\mathbb{M}_t}) \leq \mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) + \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) + \frac{\alpha_{\max}}{t^2} + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr \tag{13}$$

holds with probability at least $1-\delta/2$. By Lemma B.1 with $\beta_t = 2\log(4\pi_t/\delta) + 2d_t\log(d_t t^2 br\sqrt{\log(4Da/\delta)}) \geq 2\log(4\pi_t/\delta)$, it holds with probability at least $1-\delta/4$ that

$$f(\boldsymbol{x}^t_{\mathbb{M}_t}) \geq \mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}) - \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}). \tag{14}$$

Applying Eqs. (12), (13) and (14) to Eq. (11), it holds with probability at least $1-\delta$ that $\forall t \geq 1$,

$$\begin{aligned}
r_t \leq& \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr + \mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) + \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) + \frac{\alpha_{\max}}{t^2} + \sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr \\
&- \mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}) \\
\leq& \mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}) + \frac{\alpha_{\max}}{t^2} + 2\sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr - \mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}) \\
=& 2\beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}) + \frac{\alpha_{\max}}{t^2} + 2\sum_{i\in[D]\setminus\mathbb{M}_t}\alpha_i^* Lr,
\end{aligned}$$

where the second inequality holds because $\boldsymbol{x}^t_{\mathbb{M}_t}$ is generated by maximizing GP-UCB, and thus $\mu_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) + \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}^*_{\mathbb{M}_t}) \leq \mu_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^t_{\mathbb{M}_t})$.

By summing up r_t from $t = 1$ to T, we have with probability at least $1 - \delta$ that, $\forall T \geq 1$,

$$R_T = \sum_{t=1}^{T} r_t \leq \sum_{t=1}^{T} 2\beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}_{\mathbb{M}_t}^t) + \sum_{t=1}^{T} \frac{\alpha_{\max}}{t^2} + \sum_{t=1}^{T} 2 \sum_{i\in[D]\setminus\mathbb{M}_t} \alpha_i^* Lr$$
$$\leq \sum_{t=1}^{T} 2\beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}_{\mathbb{M}_t}^t) + 2\alpha_{\max} + 2\sum_{t=1}^{T} \sum_{i\in[D]\setminus\mathbb{M}_t} \alpha_i^* Lr, \tag{15}$$

where the second inequality holds by $\sum_{t=1}^{T} 1/t^2 \leq \pi^2/6 \leq 2$. Furthermore, let $C_1 = 8/\log(1+\eta^{-2})$, and Lemma 5.4 in [36] has shown that $\sum_{t=1}^{T} 2\beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}_{\mathbb{M}_t}^t) \leq \sqrt{C_1 T \beta_T^* \sum_{t=1}^{T} \log(1+\eta^{-2}\sigma_{t-1}^2(\boldsymbol{x}_{\mathbb{M}_t}^t))/2} \leq \sqrt{C_1 T \beta_T^* \gamma_T}$. Finally, by applying this inequality to Eq. (15), the theorem holds. □

We also summarize the main idea of the above proof. The proof is inspired by [36], i.e., to derive the upper bound on the gap $r_t = f(\boldsymbol{x}^*) - f(\boldsymbol{x}_{\mathbb{M}_t}^t)$ between the function values of the optimal point $\boldsymbol{x}^*$ and the sampled point $\boldsymbol{x}_{\mathbb{M}_t}^t$ at iteration t. Let $\boldsymbol{x}_{\mathbb{M}_t}^*$ denote the point obtained by projecting $\boldsymbol{x}^*$ onto $\mathbb{M}_t$, and $\tilde{\boldsymbol{x}}_{\mathbb{M}_t}^*$ denote its closest discretized point. By utilizing the posterior mean $\mu_{t-1}(\cdot)$ and variance $\sigma_{t-1}^2(\cdot)$ of $f(\boldsymbol{x}_{\mathbb{M}_t}^t)$ and $f(\tilde{\boldsymbol{x}}_{\mathbb{M}_t}^*)$, we can have $f(\boldsymbol{x}_{\mathbb{M}_t}^t) \geq \mu_{t-1}(\boldsymbol{x}_{\mathbb{M}_t}^t) - \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}_{\mathbb{M}_t}^t)$ and $f(\boldsymbol{x}^*) = (f(\boldsymbol{x}^*) - f(\boldsymbol{x}_{\mathbb{M}_t}^*)) + (f(\boldsymbol{x}_{\mathbb{M}_t}^*) - f(\tilde{\boldsymbol{x}}_{\mathbb{M}_t}^*)) + f(\tilde{\boldsymbol{x}}_{\mathbb{M}_t}^*) \leq \sum_{i\in[D]\setminus\mathbb{M}_t} \alpha_i^* Lr + \alpha_{\max}/t^2 + \sum_{i\in[D]\setminus\mathbb{M}_t} \alpha_i^* Lr + \mu_{t-1}(\tilde{\boldsymbol{x}}_{\mathbb{M}_t}^*) + \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}_{\mathbb{M}_t}^*)$, where the terms $\sum_{i\in[D]\setminus\mathbb{M}_t} \alpha_i^* Lr$ and $\alpha_{\max}/t^2$ are led by variable selection and discretization, respectively. As $\boldsymbol{x}_{\mathbb{M}_t}^t$ is generated by maximizing GP-UCB, we have $\mu_{t-1}(\tilde{\boldsymbol{x}}_{\mathbb{M}_t}^*) + \beta_t^{1/2}\sigma_{t-1}(\tilde{\boldsymbol{x}}_{\mathbb{M}_t}^*) \leq \mu_{t-1}(\boldsymbol{x}_{\mathbb{M}_t}^t) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}_{\mathbb{M}_t}^t)$. Thus, $r_t \leq 2\beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}_{\mathbb{M}_t}^t) + \alpha_{\max}/t^2 + 2\sum_{i\in[D]\setminus\mathbb{M}_t} \alpha_i^* Lr$. Finally, summing up r_t from $t = 1$ to T and using Lemma 5.4 in [36] can lead to Theorem 4.2.

The main difference from the proof of GP-UCB [36] is that variable selection brings some uncertainty introduced by the unselected variables. Based on the Lipschitz condition in Assumption 4.1, the uncertainty by the i-th unselected variable can be upper bounded by $\alpha_i^* Lr$, leading to the additional regret $2\sum_{t=1}^{T}\sum_{i\in[D]\setminus\mathbb{M}_t} \alpha_i^* Lr$ in Eq. (4).

B.2 Details of Computational Complexity Analysis

The computational complexity of one iteration of BO depends on three critical components: fitting a GP surrogate model, maximizing an acquisition function and evaluating a sampled point. Assume that the kernel function is squared exponential kernel. At iteration t, the number of selected variables is d_t. When fitting a GP model, we calculate the marginal likelihood [30] and gradient as follows:

$$\log P(\boldsymbol{y}_t \mid \mathbf{X}_t, \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}_t^\top(\mathbf{K}_t + \eta^2\mathbf{I})^{-1}\boldsymbol{y}_t - \frac{1}{2}\log|\mathbf{K}_t + \eta^2\mathbf{I}| - \frac{t}{2}\log(2\pi)$$
$$\nabla_{\boldsymbol{\theta}} \log P(\boldsymbol{y}_t \mid \mathbf{X}_t, \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}_t^\top(\mathbf{K}_t + \eta^2\mathbf{I})^{-1}\nabla_{\boldsymbol{\theta}}(\mathbf{K}_t + \eta^2\mathbf{I})(\mathbf{K}_t + \eta^2\mathbf{I})^{-1}\boldsymbol{y}_t$$
$$- \frac{1}{2}\mathrm{tr}\left((\mathbf{K}_t + \eta^2\mathbf{I})^{-1}\nabla_{\boldsymbol{\theta}}(\mathbf{K}_t + \eta^2\mathbf{I})\right)$$

where $\boldsymbol{y}_t = [y^1, \ldots, y^t]^\top$, $\mathbf{X}_t = [\boldsymbol{x}^1, \ldots, \boldsymbol{x}^t]$, $\boldsymbol{\theta}$ are the kernel parameters, $\mathbf{K}_t$ is the covariance matrix, $|\cdot|$ and $\mathrm{tr}(\cdot)$ denote the determinant and trace of a matrix, respectively. Then, we can use the gradient-based methods to optimize the likelihood function. Therefore, the computational complexity of calculating the kernel parameters is $\mathcal{O}(t^3 + t^2 d_t)$. Note that $\boldsymbol{\theta}$ has been ignored, because its dimension is much smaller than d_t and t. When calculating the mean $\mu_t(\boldsymbol{x})$ and variance $\sigma_t^2(\boldsymbol{x})$, the computational complexity is $\mathcal{O}(t^3 + t^2 d_t)$, due to the calculation of the kernel matrix and its inverse. Thus, the total computational complexity of fitting the GP model is $\mathcal{O}(t^3 + t^2 d_t)$. Maximizing an acquisition function is related to the optimization algorithm. If we use the Quasi-Newton method to optimize GP-UCB, the computational complexity is $\mathcal{O}(m(t^2 + td_t + d_t^2))$ [28], where m denotes the Quasi-Newton's running rounds. We note that in BO setting, t will not grow very large. The running rounds m, however, will grow with d_t. Thus, the complexity of optimizing the acquisition

function can be much larger than the square of d_t. The cost of evaluating a sampled point is fixed. Thus, by selecting only a subset of variables, instead of all variables, to optimize, the computational complexity can decrease significantly.

C Method Implementation and Experimental Setting

We use the authors' reference implementations for TuRBO[1], LA-MCTS[2] and SAASBO.[3] For HeSBO and ALEBO, their implementations in Adaptive Experimentation Platform (Ax[4]) are used. We use the pycma library for CMA-ES.[5] Their hyper-parameters are summarized as follows.

- **Vanilla BO.** We use the GP model in Scikit-learn[6] and the qExpectedImprovement acquisition function [42]. For the optimization of acquisition function, we randomly generate numerous points and select some ones with the maximal expected improvements, which is similar to the implementation in TuRBO [10], LA-MCTS [38], and HeSBO [27].
- **MCTS-VS.** For the "fill-in" strategy, we use the best-k strategy with $k = 20$. The hyper-parameter C_p for calculating UCB in Eq. (1) varies on different problems, as shown in Table 3. We set all the other parameters to be *same* on different problems, where the batch size N_v of variable index set is 2, the sample batch size $N_s = 3$, the threshold N_{bad} for re-initializing a tree is 5, and the threshold N_{split} for splitting a node is 3. When using TuRBO as the optimizer, we limit the maximal number of evaluations in TuRBO to 50.

Table 3: Setting of the hyper-parameter C_p for calculating UCB on different problems.

	LEVY	HARTMANN	NAS-BENCH	MUJOCO
C_p	10	0.1	0.1	50

- **Dropout.** We set the parameter d to the number of valid dimensions for synthetic functions, and use the same "fill-in" strategy as MCTS-VS.
- **TuRBO.** We use the default parameter setting in the authors' reference implementation.
- **LA-MCTS-TuRBO.** We use the same TuRBO setting as MCTS-VS. The parameter C_p is recommended between 1% and 10% of the optimum in LA-MCTS [38]. Because all our selected values of C_p for MCTS-VS have belonged to the recommended range for LA-MCTS, we use them directly. The RBF kernel is used for SVM classification.
- **SAASBO.** We use the default parameter setting in the authors' reference implementation, but modify the acquisition function optimization to the same as other methods for fair comparison.
- **HeSBO and ALEBO.** We set the parameter d to the number of valid dimensions for synthetic functions. For real-world problems, we do not know the number of valid dimensions, and thus we just set a reasonable value, i.e., $d = 10$ for NAS-Bench, $d = 10$ for Hopper, and $d = 20$ for Walker2d.
- **CMA-ES.** We only adjust the step-size parameter σ for different problems, because the default setting $\sigma = 0.01$ leads to extremely poor performance. We set $\sigma = 0.8$ for Hartmann problems, $\sigma = 10$ for Levy problems, $\sigma = 0.1$ for NAS-Bench, and $\sigma = 0.01$ for MuJoCo tasks. We set the population size to 20 and maintain all the other parameters to default.
- **VAE-BO** uses VAE for embedding. That is, VAE-BO uses the encoder to embed the original high-dimensional space into a low-dimensional subspace, then optimizes via BO in the subspace and uses the decoder to project the new sampled point back for evaluation. We set the learning rate to 0.01 and the interval of updating VAE to 30.

[1] `https://github.com/uber-research/TuRBO`
[2] `https://github.com/facebookresearch/LaMCTS`
[3] `https://github.com/martinjankowiak/saasbo`
[4] `https://github.com/facebook/Ax`
[5] `https://github.com/CMA-ES/pycma`
[6] `https://github.com/scikit-learn/scikit-learn`

The experiments of comparing wall clock time are conducted on Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz and use single thread.

D Sensitivity Analysis of Hyper-parameters of MCTS-VS

We provide further studies to examine the influence of the hyper-parameters of MCTS-VS, including the employed optimization algorithm for optimizing the selected variables in each iteration, the "fill-in" strategy, the hyper-parameter k used in the best-k strategy, the hyper-parameter C_p for calculating UCB in Eq. (1), the number $2 \times N_v \times N_s$ of sampled data in each iteration, the threshold N_{bad} for re-initializing a tree, and the threshold N_{split} for splitting a tree node.

The optimization algorithm is employed by MCTS-VS to optimize the selected variables in each iteration. We compare three different optimization algorithms, i.e., random search (RS), BO and TuRBO. First, we conduct experiments similar to "Effectiveness of Variable Selection" in Section 5.1, to show the effectiveness of MCTS-VS even when equipped with RS. Figure 6 shows that MCTS-VS-RS is better than Dropout-RS and RS, revealing the advantage of MCTS-VS.

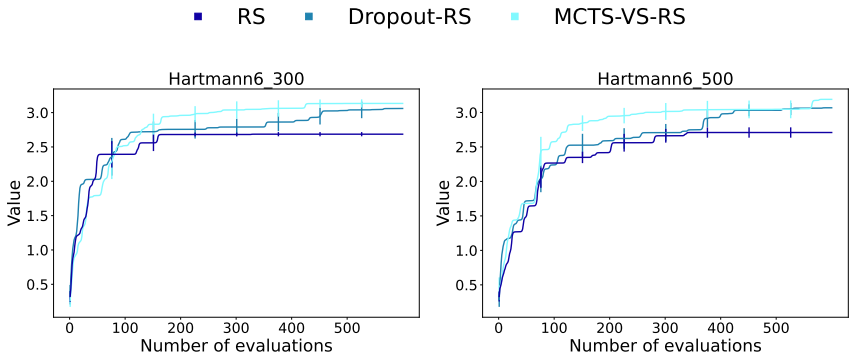


Figure 6: Effectiveness of MCTS-VS when equipped with RS.

Next we compare the performance of MCTS-VS equipped with RS, BO and TuRBO, by experiments on the Hartmann functions with increasing ratio of valid variables. Hartmann6_500 has 6 valid variables. Hartmann6_5_500 is generated by mixing 5 Hartmann6 functions as Hartmann6$(\boldsymbol{x}_{1:6})+$ Hartmann6$(\boldsymbol{x}_{7:12}) + \cdots +$ Hartmann6$(\boldsymbol{x}_{25:30})$, and appending 470 unrelated dimensions, where $\boldsymbol{x}_{i:j}$ denotes the i-th to j-th variables. Hartmann6_10_500 is generated alike. Thus, Hartmann6_5_500 and Hartmann6_10_500 have 30 and 60 valid variables, respectively. The results in Figure 7 show that as the ratio of valid variables increases, MCTS-VS-TuRBO gradually surpasses MCTS-VS-RS and MCTS-VS-BO, while MCTS-VS-RS becomes worse and worse. This is expected. If the ratio of valid variables is high, MCTS-VS is more likely to select the valid variables, so it is worth to use the expensive optimization algorithm, e.g., TuRBO, to optimize the selected variables. If the ratio is low, unrelated variables are more likely to be selected most of the time, so using a cheap optimization algorithm would be better. These observations also give us some guidance on selecting optimization algorithms in practice.

"Fill-in" strategy is a basic component of variable selection methods, which influences the quality of the value of unselected variables. We compare the employed best-k strategy ($k = 20$) with the average best-k strategy and the random strategy. The average best-k strategy uses the average of the best k data points for the unselected variables, and the random strategy samples the value of an unselected variable from its domain randomly. As shown in Figure 8(a), the random strategy leads to the poor performance of MCTS-VS-BO, which may be because it does not utilize the historical information and leads to over-exploration. The best-k strategy utilizes the historical points that have high objective values to fill in the unselected variables, thus behaving much better. The performance of the average strategy is between the best-k and random strategies. We recommend using the best-k strategy in practice.

The hyper-parameter k used in the best-k strategy controls the degree of exploitation for the unselected variables. As shown in Figure 8(b), a smaller k encourages exploitation, which results in better performance in the early stage, but easily leads to premature convergence. A larger k

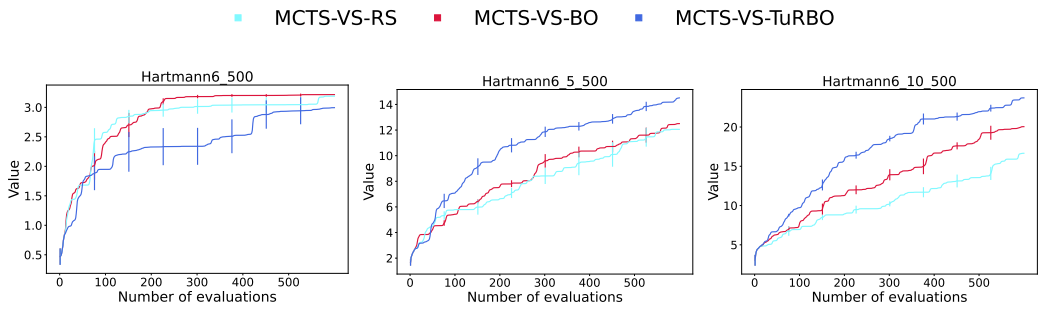
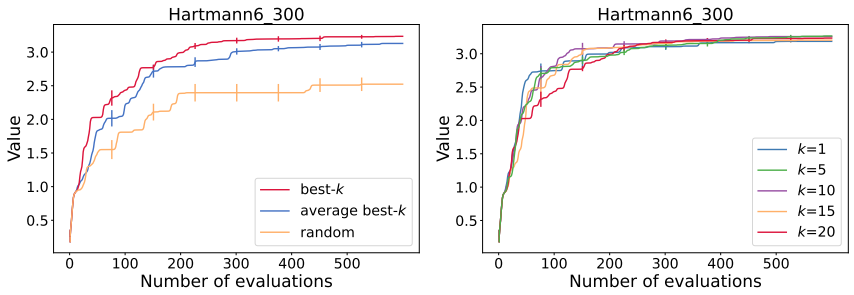


Figure 7: Sensitivity analysis of the optimization algorithm.

encourages exploration and behaves worse in the early stage, but may converge to a better value. We recommend using a larger k if allowing enough evaluations.



(a) "Fill-in" strategy (b) Hyper-parameter k of the best-k strategy

Figure 8: Sensitivity analysis of the "fill-in" strategy and the hyper-parameter k of the best-k strategy, using MCTS-VS-BO on Hartmann6_300.

The hyper-parameter C_p for calculating UCB in Eq. (1) balances the exploration and exploitation of MCTS. As shown in Figure 9, a too small C_p leads to relatively worse performance, highlighting the importance of exploration. A too large C_p may also lead to over-exploration. But overall MCTS-VS is not very sensitive to C_p. We recommend setting C_p between 1% and 10% of the optimum (i.e., $\max f(\boldsymbol{x})$), which is consistent with that for LA-MCTS [38].

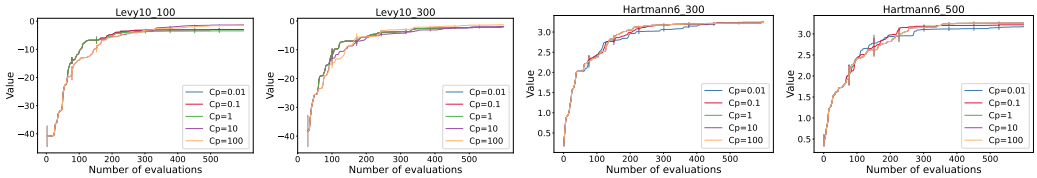


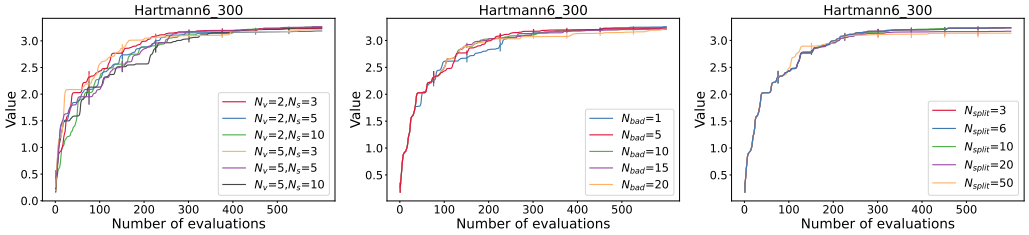
Figure 9: Sensitivity analysis of the hyper-parameter C_p for calculating UCB in Eq. (1), using MCTS-VS-BO on Levy and Hartmann.

The number $2 \times N_v \times N_s$ of sampled data in each iteration depends on the batch size N_v of variable index subset and the sample batch size N_s, and will influence the accuracy of estimating the variable score vector in Eq. (2). If we increase N_v and N_s, we can calculate the variable score more accurately, but also need more evaluations. Figure 10(a) shows that given the same number of evaluations, MCTS-VS-BO achieves the best performance when $N_v = 2$ and $N_s = 3$. Thus, this setting may be a good choice to balance the accuracy of variable score and the number of evaluations, which is also used throughout the experiments.

The threshold N_{bad} for re-initializing a tree controls the tolerance of selecting bad tree nodes (i.e., nodes containing unimportant variables). A smaller N_{bad} leads to frequent re-initialization, which

can adjust quickly but may cause under-exploitation of the tree. A larger N_{bad} can make full use of the tree, but may optimize too much on unimportant variables. Figure 10(b) shows that MCTS-VS achieves the best performance when $N_{bad} = 5$. Thus, we recommend to use this setting, to balance the re-initialization and exploitation of the tree.

The threshold N_{split} for splitting a node. If the number of variables in a node is larger than N_{split}, the node can be further partitioned. That is, the parameter N_{split} controls the least number of variables in a leaf node and thus affects the number of selected variables, which has a direct influence on the wall clock time. Note that MCTS-VS selects a leaf node and optimizes the variables contained by this node in each iteration. The smaller N_{split}, the shorter the time. Figure 10(c) shows that N_{split} has little influence on the performance of MCTS-VS-BO, and thus we recommend to set $N_{split} = 3$ to reduce the wall clock time.



(a) Number of samples (b) N_{bad} (c) N_{split}

Figure 10: Sensitivity analysis of the number $2 \times N_v \times N_s$ of sampled data in each iteration, the threshold N_{bad} for re-initializing a tree and the threshold N_{split} for splitting a node, using MCTS-VS-BO on Hartmann6_300.

Influence of the hyper-parameters on the runtime of MCTS-VS. We also provide some intuitive explanation about the influence of the hyper-parameters on the runtime. The threshold N_{split} for splitting a node has a direct impact on the runtime, because it controls the least number of variables to be optimized in a leaf node. That is, the runtime will increase with N_{split}. Other parameters may affect the depth of the tree and thus the runtime. For the threshold N_{bad} for re-initializing a tree, if it is set to a small value, MCTS-VS will re-build the tree frequently and the depth of the tree is small. The shallow nodes have more variables, leading to more runtime to optimize. For the hyper-parameter C_p for calculating UCB, if it is set to a large value, the exploration is preferred and MCTS-VS will tend to select the right node (regarded as containing unimportant variables). The tree thus will be re-built frequently, leading to more runtime. For the number $2 \times N_v \times N_s$ of sampled data at each iteration, if N_v and N_s are set to large values, the depth of the tree will be small given the total number of evaluations, and thus lead to more runtime.

E Additional Experiments

Detailed results on NAS-Bench-101 and NAS-Bench-201. Figure 11 shows the performance of the compared methods on the task of NAS-Bench-101 and NAS-Bench-201 when using the number of evaluations and wall clock time as the x-axis, respectively. Though most of their performance is similar in the left two subfigures, it can be clearly observed from the right two subfigures that MCTS-VS-BO uses the least time to achieve the best accuracy. Note that we only show the subfigures with the wall clock time as the x-axis in the main paper due to the space limitation. Besides, we also run a longer time here (i.e., in the right two subfigures) to provide a more complete observation.

Experiments on more NAS-Bench problems. We also conduct experiments on NAS-Bench-1Shot1 [44], TransNAS-Bench-101 [8] and NAS-Bench-ASR [25]. NAS-Bench-1Shot1 is a weight-sharing benchmark based on one-shot NAS methods, deriving from the large architecture space of NASBench-101. TransNAS-Bench-101 is a benchmark dataset containing network performance across seven vision tasks, e.g., object classification, scene classification and so on. We use the scene classification task with cell-level search space in our experiments. NAS-Bench-ASR is a benchmark for Automatic Speech Recognition (ASR) and trained on the TIMIT audio dataset. For NAS-Bench-ASR, we use Phoneme Error Rate (PER) on the validation dataset as the metric. In the same way as [20], we create problems with $D = 33$, $D = 24$ and $D = 30$ for NAS-Bench-

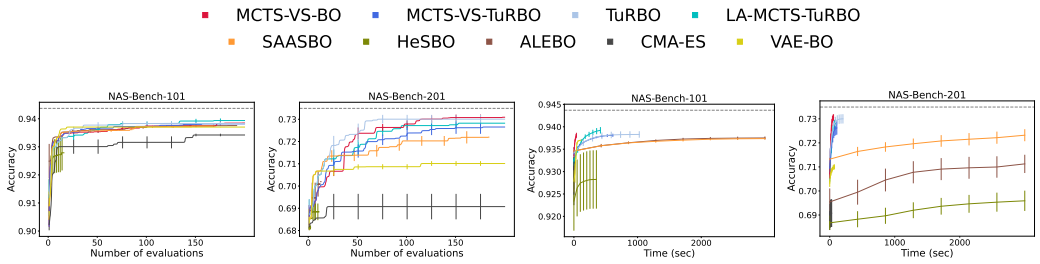


Figure 11: Performance comparison on NAS-Bench-101 and NAS-Bench-201, using the number of evaluations and wall clock time as the x-axis, respectively.

1Shot1, TransNAS-Bench-101 and NAS-Bench-ASR, respectively. The results in Figure 12 show that MCTS-VS-BO still uses the least time to achieve the best performance.

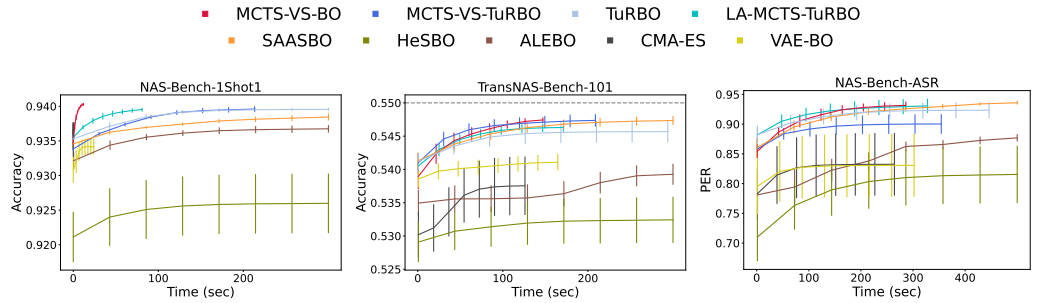
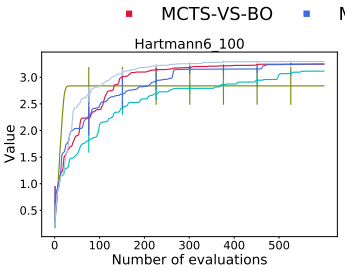


Figure 12: Performance comparison on more NAS-Bench problems.

Experiments on extremely low and high dimensional problems. We also evaluate the compared methods for extremely low and high dimensional problems by testing on Hartmann6_100 and Hartmann6_1000. We only run MCTS-VS, TuRBO, LA-MCTS-TuRBO and HeSBO here, because they behave well in the previous experiments. As expected, the right subfigure of Figure 13 shows that MCTS-VS-BO has a clear advantage over the rest methods on the extremely high dimensional function Hartmann6_1000. The left subfigure shows that on Hartmann6_100, TuRBO behaves the best and MCTS-VS is the runner-up, implying that MCTS-VS can also tackle low dimensional problems to some degree.

Experiments on synthetic functions depending on a subset of variables to various extent. In the experiments, the synthetic functions are generated by adding unrelated variables directly. For example, Hartmann6_500 has the dimension $D = 500$, and is generated by appending 494 unrelated dimensions to Hartmann with 6 variables. Here, we test the performance of MCTS-VS on a synthetic function whose dependence on a subset of variables is more various. For this purpose, we generate Hartmann6_5_500_v by mixing five Hartmann6 functions as $0.5^0\text{Hartmann6}(\boldsymbol{x}_{1:6}) + 0.5^1\times\text{Hartmann6}(\boldsymbol{x}_{7:12}) + \cdots + 0.5^4\text{Hartmann6}(\boldsymbol{x}_{25:30})$, and appending 470 unrelated dimensions, where $\boldsymbol{x}_{i:j}$ denotes the i-th to j-th variables, and different coefficients represent various degrees of dependence. The results in Figure 14 show that MCTS-VS-BO performs the best.

Experiments with increasing ratio of valid variables. We also examine the performance of MCTS-VS when the ratio of valid variables increases. We use the synthetic function Hartmann6_500, and generate the variants with more valid variables by mixing multiple Hartmann6 functions as in Appendix D. For example, Hartmann6_5_500 is generated by mixing five Hartmann6 functions as Hartmann6$(\boldsymbol{x}_{1:6})$+ Hartmann6$(\boldsymbol{x}_{7:12})$ + $\cdots$+ Hartmann6$(\boldsymbol{x}_{25:30})$, and appending 470 unrelated dimensions. We have compared MCTS-VS-TuRBO with LA-MCTS-TuRBO and TuRBO on Hartmann6_500, Hartmann6_5_500, Hartmann6_10_500, ..., Hartmann6_30_500, and Hartmann6_83_500, which has the largest number (i.e., $6 \times 83 = 498$) of valid variables. The results are shown in Figure 15. It can be observed that LA-MCTS-TuRBO performs the worst. As expected, when the percentage of valid variables is low (e.g., in Hartmann6_500, Hartmann6_5_500 and Hartmann6_10_500), MCTS-VS-TuRBO can be better than TuRBO; but as the percentage of



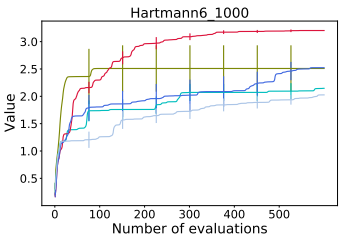


Figure 13: Performance comparison on extremely low and high dimensional problems.

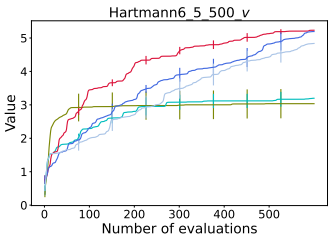


Figure 14: Performance comparison on synthetic functions depending on a subset of variables to various extent.

valid variables increases, TuRBO becomes better, because a leaf node of MCTS can contain only a small fraction of valid variables.

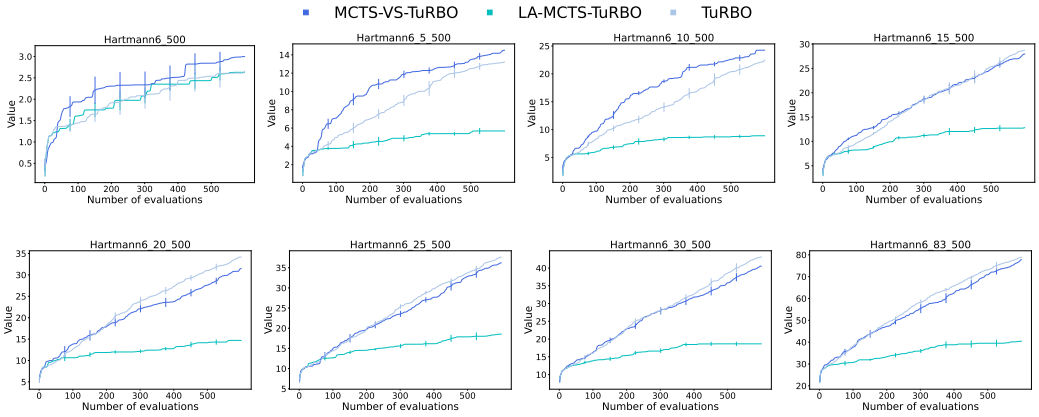


Figure 15: Performance comparison with increasing ratio of valid variables.

Hierarchical variable selection for extremely high-dimensional problems. We attempt to combine MCTS-VS and SAASBO (i.e., MCTS-VS-SAASBO) to handle extremely high-dimensional problems. MCTS-VS-SAASBO can be viewed as a hierarchical variable selection method, i.e., MCTS-VS first performs an efficient but rough variable selection to select some variables, and then SAASBO performs a time-consuming but precise variable selection under the relative low-dimensional space, to further select the important variables. We run MCTS-VS-SAASBO and SAASBO on Hartmann6_500. The results are shown in Figure 16. The performance of MCTS-VS-SAASBO and SAASBO is similar. But when considering the runtime, the time of 200 iterations of MCTS-VS-SAASBO is about 6000s, while the time of SAASBO is about 45000s. That is, MCTS-VS-SAASBO can achieve more than 7 times acceleration. The curves of using the wall clock time as the x-axis in the right sub-figure of Figure 16 clearly show the advantage of MCTS-VS-SAASBO over SAASBO. MCTS-VS-SAASBO selects the variables containing important ones by MCTS and then uses SAASBO to optimize the selected variables, which reduces the dimension and thus costs much less time than using SAASBO directly. The combination of MCTS-VS and SAASBO may be a potential solution for BO to handle extremely high-dimensional optimization problems, where it is difficult to select important variables directly.

Comparison with LASSO-VS. There are other variable selection methods (e.g., LASSO), which are not designed for high dimensional BO but can be used directly. We have implemented the LASSO-based variable selection method, named LASSO-VS. We compare MCTS-VS, LASSO-VS and Dropout on the synthetic function Hartmann6_300. When using LASSO-VS, the d variables with the largest absolute values of the regression coefficients are selected at each iteration. The results are shown in Figure 17. When equipped with either BO or TuRBO, the proposed MCTS-VS always performs the best. We can also observe that when equipped with BO, LASSO-VS can even

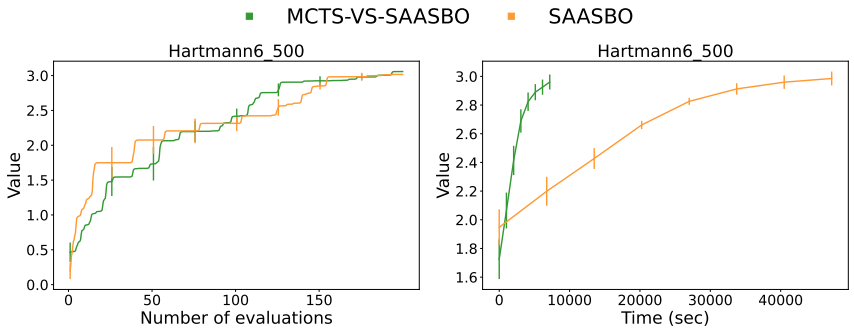


Figure 16: Performance comparison among MCTS-VS-SAASBO and SAASBO on the synthetic function Hartmann6_500.

be worse than Dropout. This may be because many of existing variable selection methods (e.g., LASSO) usually require a large number of samples to fit the linear regression model well, while in BO scenarios, only a limited number of samples can be evaluated.

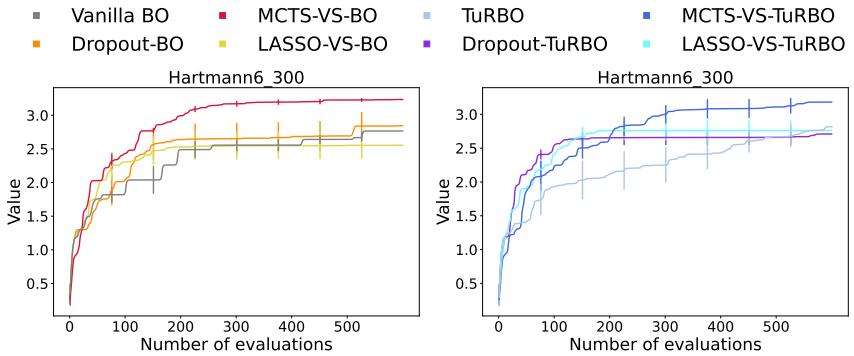


Figure 17: Performance comparison among MCTS-VS, LASSO-VS and Dropout on the synthetic function Hartmann6_300.

Statistical tests. As most of the previous works, we have conducted experiments using 5 random seeds (2021–2025). Here, we also conduct statistical tests on Hartmann and Levy functions by running the methods for 50 times (random seeds 2021–2070), to make a more confident comparison. Considering the performance and runtime of the methods we have observed in Figure 2, we only compare MCTS-VS with LA-MCTS-TuRBO and TuRBO, which achieve good performance in acceptable time. The results are shown in Table 4. MCTS-VS-BO achieves the best average objective value on all the synthetic functions except Levy10_100 where the dimension is relatively low and TuRBO performs the best. By the Wilcoxon signed-rank test with confidence level 0.05, MCTS-VS-TuRBO is significantly better than LA-MCTS-TuRBO on all the synthetic functions, showing the advantage of MCTS-VS over LA-MCTS for variable selection. Compared with TuRBO, MCTS-VS-TuRBO is only significantly better on Hartmann functions, which may be because the ratio of valid variables of Hartmann6_300 and Hartmann6_500 is lower than that of Levy10_100 and Levy10_300, and thus the advantage of performing variable selection by MCTS-VS is more clear. Note that the observations about the performance rank of the compared methods are consistent with that observed in Figure 2, which plot the results of the compared methods by running five times.

F Enlargement of Some Figures in the Main Paper

Due to space limitation, Figures 1 and 2 in the main paper are a little small. Here, we also provide their enlarged versions, i.e., Figures 18 and 19.

Table 4: Objective values obtained by MCTS-VS-BO, MCTS-VS-TuRBO, LA-MCTS-TuRBO and TuRBO on synthetic functions. Each result consists of the mean and standard deviation of 50 runs. The best mean value on each problem is bolded. The symbols '+', '−' and '≈' indicate that MCTS-VS-TuRBO is significantly superior to, inferior to, and almost equivalent to the corresponding method, respectively, according to the Wilcoxon signed-rank test with confidence level 0.05.

Problem	MCTS-VS-BO	MCTS-VS-TuRBO	LA-MCTS-TuRBO	TuRBO
Levy10_100	-2.620(1.757) +	-1.102(1.711)	-2.444(1.708) +	**-0.662**(1.049) −
Levy10_300	**-1.506**(0.854) ≈	-1.765(1.811)	-6.218(3.389) +	-1.855(2.038) ≈
Hartmann6_300	**3.223**(0.074) ≈	3.153(0.264)	2.892(1.147) +	2.857(0.475) +
Hartmann6_500	**3.200**(0.091) −	3.012(0.434)	2.619(0.672) +	2.629(0.672) +
+/−/≈	1/1/2	/	4/0/0	2/1/1

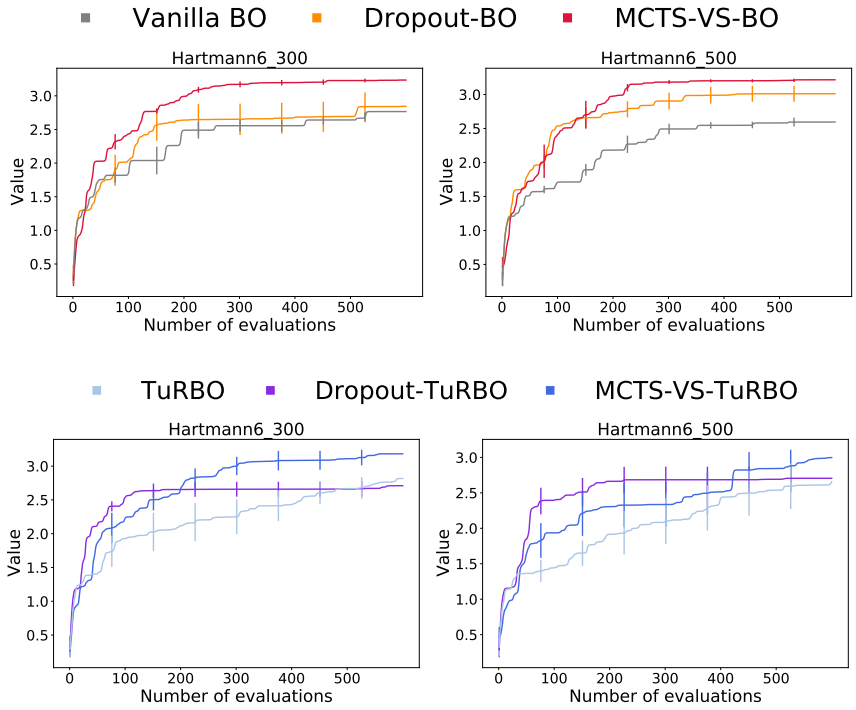


Figure 18: Performance comparison among the two variable selection methods (i.e., MCTS-VS and Dropout) and the BO methods (i.e., Vanilla BO and TuRBO) on two synthetic functions.

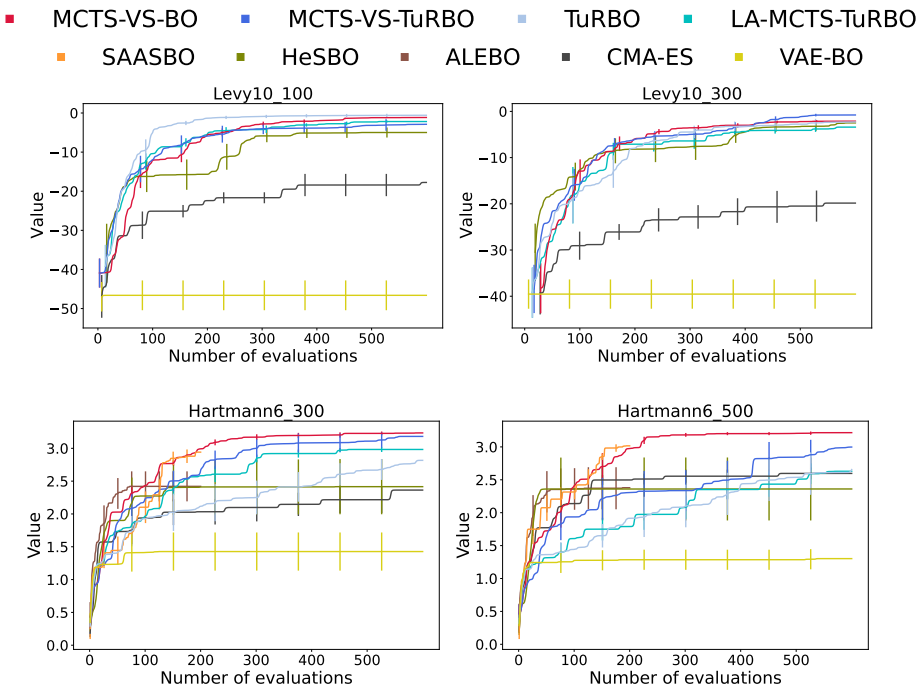


Figure 19: Comparison among MCTS-VS and state-of-the-art methods on synthetic functions.