A Details of variable selection and variable selection with momentum mechanism

Algorithm 2 Variable Selection (line 8 in Algorithm 1)

```
Input: $\mathcal{D} = \{(\mathbf{x}^i, y^i)\}_{i=1}^t$
Output: $\mathbf{x}_{ipt}$
 1: Fit a GP to $\mathcal{D}$ and calculate important scores $IS$, where $IS[i]$ is the important score of the i-th variable
 2: Sort variables according to their important scores from the most important to the least, $[\mathbf{x}_{s(1)}, \dots, \mathbf{x}_{s(D)}]$
 3: for $m = 1, 2, \dots D$ do
 4:     Fit a GP to $\mathcal{D}_m := \{(\mathbf{x}^i_{s(1):s(m)}, y^i)\}_{i=1}^{t-1}$ where $\mathbf{x}^i_{s(1):s(m)}$ is the $i$-th input with only the first $m$ important variables, let $L_m$ to be the value of final negative marginal log likelihood
 5:     if $m < 3$ then
 6:         continue
 7:     else if $L_{m-1} - L_m \le 0$ or $L_{m-1} - L_m < \frac{L_{m-2} - L_{m-1}}{10}$ then
 8:         break
 9:     end if
10: end for
11: return $\mathbf{x}_{ipt} = \{\mathbf{x}_{s(1)}, \dots, \mathbf{x}_{s(m-1)}\}$
```

Here, we describe the details of VS-momentum and the pseudo-codes. We say that the variable selection at iteration $t + N_{vs}$ is in an accurate case when $\max_{k \in \{t+1,\dots,t+N_{vs}\}} y^k > \max_{k \in \{1,\dots,t\}} y^k$, otherwise it is in an inaccurate case. In the accurate case, VS-BO first uses recursive feature elimination (RFE) based algorithm to remove redundant variables in $\mathbf{x}_{ipt}$ that is selected at t, then it adds new variables into the remaining only if the loss decreases evidently (Figure 3a). In the inaccurate case, variables selected at t will not be considered at $t + N_{vs}$ unless they still obtain very high important scores at $t + N_{vs}$ (marked by the blue box in Figure 3b). New variables are added via stepwise-forward algorithm. Algorithm 3 provides the pseudo-code of VS-momentum. Note that Algorithm 4 (momentum in the inaccurate case) is similar to Algorithm 2, except lines 4-6.

Algorithm 3 Variable Selection (VS) with Momentum

```
Input: iteration index $t$, $\mathcal{D} = \{(\mathbf{x}^i, y^i)\}_{i=1}^t$, $N_{init}$, $N_{vs}$, set of important variables chosen at iteration $t - N_{vs}$, denote as $\hat{\mathbf{x}}_{ipt}$
Output: Set of important variables chosen at iteration $t$, denote as $\mathbf{x}_{ipt}$
1: if $t = N_{init} + N_{vs}$ or $\hat{\mathbf{x}}_{ipt} = \mathbf{x}$ then
2:     return Algorithm 2
3: end if
4: if $\max_{k \in \{t-N_{vs}+1,\dots,t\}} y^k \le \max_{k \in \{1,\dots,t-N_{vs}\}} y^k$ then
5:     return Algorithm 4
6: else
7:     return Algorithm 5
8: end if
```

Algorithm 4 Momentum in the inaccurate case

```
Input: $\mathcal{D} = \{(\mathbf{x}^i, y^i)\}_{i=1}^t$, $N_{vs}$, set of important variables chosen at iteration $t - N_{vs}$, denote as $\hat{\mathbf{x}}_{ipt}$
Output: Set of important variables chosen at iteration $t$, denote as $\mathbf{x}_{ipt}$
 1: Fit a GP to $\mathcal{D}$ and calculate important scores $IS$ where $IS[i]$ is the important score of the i-th variable
 2: Sort variables according to their important scores from the most important to the least, $[\mathbf{x}_{s(1)}, \dots, \mathbf{x}_{s(D)}]$
 3: for $n = 1, \dots D$ do
 4:     if $\mathbf{x}_{s(n)} \notin \hat{\mathbf{x}}_{ipt}$ then
 5:         break
 6:     end if
 7: end for
 8: for $m = n, n+1, \dots D$ do
 9:     Fit a GP to $\mathcal{D}_m := \{(\mathbf{x}^i_{s(1):s(m)}, y^i)\}_{i=1}^{t-1}$ where $\mathbf{x}^i_{s(1):s(m)}$ is the $i$-th input with only the first $m$ important variables, let $L_m$ to be the value of final negative marginal log likelihood
10:     if $m - n < 2$ then
11:         continue
12:     else if $L_{m-1} - L_m \le 0$ or $L_{m-1} - L_m < \frac{L_{m-2} - L(m-1)}{10}$ then
13:         break
14:     end if
15: end for
16: return $\mathbf{x}_{ipt} = \{\mathbf{x}_{s(1)}, \dots, \mathbf{x}_{s(m-1)}\}$
```

B Proof of Theorem 1

Proof. Given query-output pairs $\mathcal{D} = \{(\mathbf{x}^i, y^i)\}_{i=1}^n$, the marginal log likelihood (MLL) that needs to be maximized at the step of fitting a GP has the following explicit form:

$$\log p(\Theta = \{\rho^2_{1:D}, \alpha_0^2\}, \sigma_0 \mid \mathcal{D}) = -\frac{1}{2}\mathbf{y}M^{-1}\mathbf{y}^\top - \frac{1}{2}\log|M| - \frac{n \log 2\pi}{2} \tag{7}$$

where $\mathbf{y} = [y^1, \dots y^n]$ is an n-dimensional vector and $M = \big(K(\mathbf{x}^{1:n}, \Theta) + \sigma_0^2\mathbf{I}\big)$. When the quasi-Newton method is used for maximizing MLL, the gradient should be calculated

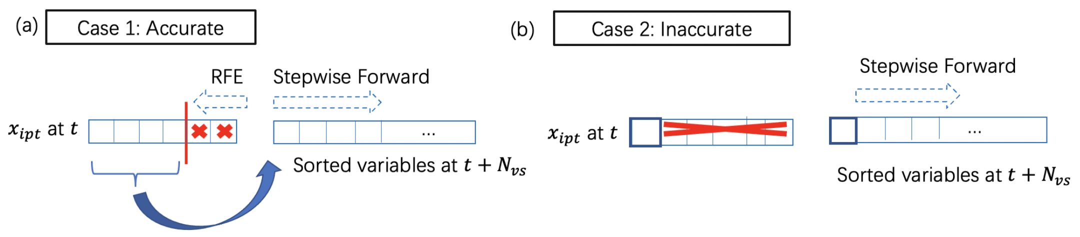

Figure 3: Momentum mechanism in VS-BO. (a) Accurate case, RFE is first used to remove redundant variables, and then new variables are added. (b) Inaccurate case, most variables are removed except those that are considered very important in both variable selection steps (blue box). New variables are then added.

for each iteration:

$$\nabla_{\Theta,\sigma_0} \log p(\Theta,\sigma_0 \mid \mathcal{D}) = -\frac{1}{2}\mathbf{y}M^{-1}\left(\nabla_{\Theta,\sigma_0}M\right)M^{-1}\mathbf{y}^\top - \frac{1}{2}\mathbf{tr}\left(M^{-1}\left(\nabla_{\Theta,\sigma_0}M\right)\right) \tag{8}$$

When only variables in $\mathbf{x}_{ipt}$ are used, we define the distance between two queries $\mathbf{x}^i$ and $\mathbf{x}^j$ as:

$$d(\mathbf{x}^i,\mathbf{x}^j) = \sqrt{\sum_{m:m\in\mathbf{x}_{ipt}} \rho_m^2(\mathbf{x}_m^i - \mathbf{x}_m^j)^2}$$

and all the other inverse squared length scales corresponding to unimportant variables are fixed to 0. Commonly chosen kernel functions are functions of the distance defined above; for example the squared exponential (SE) kernel is as the following:

$$k_{SE}(\mathbf{x}^i,\mathbf{x}^j,\Theta) = \alpha_0^2 \exp\left(-\frac{1}{2}d^2(\mathbf{x}^i,\mathbf{x}^j)\right)$$

and the Matern-5/2 kernel is as the following:

$$k_{Mt}(\mathbf{x}^i,\mathbf{x}^j,\Theta) = \alpha_0^2\left(1+\sqrt{5}d(\mathbf{x}^i,\mathbf{x}^j)+\frac{5}{3}d^2(\mathbf{x}^i,\mathbf{x}^j)\right) \cdot \exp\left(-\sqrt{5}d(\mathbf{x}^i,\mathbf{x}^j)\right). \tag{9}$$

Since the cardinality of $\mathbf{x}_{ipt}$ is p, the cardinality of parameters in the kernel function that are not fixed to 0 is $p+1$, hence the complexity of calculating the gradient of the distance is $\mathcal{O}(p)$, therefore whatever using SE kernel or Matern-5/2 kernel, the complexity of calculating $\nabla_\Theta k(\mathbf{x}^i,\mathbf{x}^j,\Theta)$ is $\mathcal{O}(p)$.

Since M is a $n\times n$ matrix and each entry M_{ij} equals to $k(\mathbf{x}^i,\mathbf{x}^j,\Theta)+\sigma_0^2\mathbb{1}(i=j)$, the complexity of calculating $\nabla_{\Theta,\sigma_0}M$ is $\mathcal{O}(pn^2)$. The complexity of calculating the inverse matrix M^{-1} is $\mathcal{O}(n^3)$ in general, and the following matrix multiplication and trace calculation need $\mathcal{O}(pn^2)$, therefore the complexity of calculating the gradient of MLL is $\mathcal{O}(pn^2+n^3)$. Once the gradient is obtained, each quasi-Newton step needs additional $\mathcal{O}(p^2)$. Therefore the complexity of one step of quasi-Newton method when fitting a GP is $\mathcal{O}(p^2+pn^2+n^3)$.

As described in section 2, the acquisition function is a function that depends on the posterior mean μ and the posterior standard deviation σ, hence the gradients of μ and σ should be calculated when the gradient of the acquisition function is needed.

When only variables in $\mathbf{x}_{ipt}$ are used, the gradient of μ with respect to $\mathbf{x}_{ipt}$ has the following form:

$$\nabla_{\mathbf{x}_{ipt}}\mu(\mathbf{x}_{ipt}\mid\mathcal{D}) = \left(\nabla_{\mathbf{x}_{ipt}}K(\mathbf{x}_{ipt},\mathbf{x}_{ipt}^{1:n})\right) \cdot \left(K(\mathbf{x}_{ipt}^{1:n},\Theta)+\sigma_0^2\mathbf{I}\right)^{-1}\mathbf{y}^\top \tag{10}$$

Here $\left(K(\mathbf{x}_{ipt}^{1:n},\Theta)+\sigma_0^2\mathbf{I}\right)^{-1}\mathbf{y}^\top$ is fixed so that its value can be calculated in advance and stored as a n-dimensional vector. $K(\mathbf{x}_{ipt},\mathbf{x}_{ipt}^{1:n})$ is a n-dimensional vector of which each element is a kernel value between $\mathbf{x}_{ipt}$ and $\mathbf{x}_{ipt}^i$, hence the complexity of calculating the gradient of each element in $K(\mathbf{x}_{ipt},\mathbf{x}_{ipt}^{1:n})$ is $\mathcal{O}(p)$. Therefore, the complexity is $\mathcal{O}(pn)$ to calculate $\nabla_{\mathbf{x}_{ipt}}K(\mathbf{x}_{ipt},\mathbf{x}_{ipt}^{1:n})$ and $\mathcal{O}(pn)$ for additional matrix manipulation, hence the total complexity for calculating $\nabla_{\mathbf{x}_{ipt}}\mu(\mathbf{x}_{ipt}\mid\mathcal{D})$ is $\mathcal{O}(pn)$.

The gradient of σ has the following form:

$$\nabla_{\mathbf{x}_{ipt}}\sigma(\mathbf{x}_{ipt}\mid\mathcal{D}) = \nabla_{\mathbf{x}_{ipt}}\sqrt{\mathcal{M}} = -\frac{\left(\nabla_{\mathbf{x}_{ipt}}K(\mathbf{x}_{ipt},\mathbf{x}_{ipt}^{1:n})\right)\left(K(\mathbf{x}_{ipt}^{1:n},\Theta)+\sigma_0^2\mathbf{I}\right)^{-1}}{\sqrt{\mathcal{M}}} \tag{11}$$

where

$$\mathcal{M} = -K(\mathbf{x}_{ipt},\mathbf{x}_{ipt}^{1:n})[K(\mathbf{x}_{ipt}^{1:n},\Theta)+\sigma_0^2\mathbf{I}]^{-1}K(\mathbf{x}_{ipt},\mathbf{x}_{ipt}^{1:n})^\top + k(\mathbf{x}_{ipt},\mathbf{x}_{ipt},\Theta) \tag{12}$$

Once $\nabla_{\mathbf{x}_{ipt}}K(\mathbf{x}_{ipt},\mathbf{x}_{ipt}^{1:n})$ is calculated, $\mathcal{O}(pn+n^2)$ is needed for additional matrix manipulation, hence the total complexity for calculating $\nabla_{\mathbf{x}_{ipt}}\sigma(\mathbf{x}_{ipt}\mid\mathcal{D})$ is $\mathcal{O}(pn+n^2)$.

For commonly used acquisition functions such as upper confidence bound (UCB) [Auer, 2002]:

$$UCB(\mathbf{x}_{ipt}\mid\mathcal{D}) = \mu(\mathbf{x}_{ipt}\mid\mathcal{D})+\sqrt{\beta_n}\sigma(\mathbf{x}_{ipt}\mid\mathcal{D}) \tag{13}$$

and expected improvement (EI) [Močkus, 1975]:

$$EI(\mathbf{x}_{ipt}\mid\mathcal{D}) = \left(\mu(\mathbf{x}_{ipt}\mid\mathcal{D})-y_n^*\right)\Phi\left(\frac{\mu(\mathbf{x}_{ipt}\mid\mathcal{D})-y_n^*}{\sigma(\mathbf{x}_{ipt}\mid\mathcal{D})}\right) + \sigma(\mathbf{x}_{ipt}\mid\mathcal{D})\varphi\left(\frac{\mu(\mathbf{x}_{ipt}\mid\mathcal{D})-y_n^*}{\sigma(\mathbf{x}_{ipt}\mid\mathcal{D})}\right) \tag{14}$$

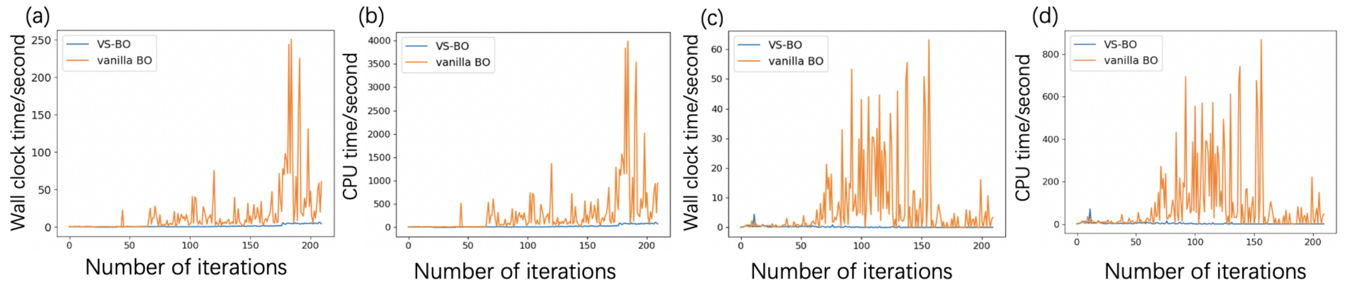

Figure 4: The wall clock time or CPU time comparison between VS-BO and vanilla BO for each iteration. The Branin test function with $d_e = [2, 2, 2]$ and $D = 50$ is run here. (a) Wall clock time comparison at the GP fitting step (b) CPU time comparison at the GP fitting step (c) Wall clock time comparison at the acquisition function optimization step (d) CPU time comparison at the acquisition function optimization step

where $y_n^* = \max_{i \le n} y^i$, $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution, and $\varphi(\cdot)$ is the probability density function, once the gradients of μ and σ are derived, only additional $\mathcal{O}(p)$ is needed for vector calculation, hence the total complexity of calculating the gradient of the acquisition function is $\mathcal{O}(pn + n^2)$. Again, once the gradient is obtained, each quasi-Newton step needs additional $\mathcal{O}(p^2)$, therefore the complexity of one step of quasi-Newton method for maximising the acquisition function is $\mathcal{O}(p^2 + pn + n^2)$.

□

C Detailed experimental settings and extended discussion of experimental results

We use the framework of BoTorch to implement VS-BO. We compare VS-BO to the following existing BO methods: vanilla BO, which is implemented by the standard BoTorch framework[1]; REMBO and its variant REMBO Interleave [Wang *et al.*, 2016], of which the implementations are based on Metzen [2016][2]; HeSBO [Nayebi *et al.*, 2019] which has already been implemented in Adaptive Experimentation Platform (Ax)[3]; ALEBO[4] [Letham *et al.*, 2020]; DescentLineBO[5] [Kirschner *et al.*, 2019]; and SAASBO[6] [Eriksson and Jankowiak, 2021]. At the time when we write this manuscript, source code of the approach in Spagnol [2019] have not been released, so we cannot compare VS-BO with it. Both VS-BO and vanilla BO use Matern 5/2 as the kernel function and expected improvement as the acquisition function, and use limited-memory BFGS (L-BFGS) to fit GP and optimize the acquisition function. The number of initialized samples N_{init} is set to 5 for all methods, and N_{vs} in VS-BO is set to 20, N_{is} is set to 10000 for all experiments. The number of the interleaved cycle for REMBO Interleave is set to 4. Since our algorithm aims to maximize the black-box function, all the test functions that have minimum points are converted to the corresponding negative forms. As described in section 5, in order to decide the direction of the one-dimensional line, for each iteration DescentLineBO needs to evaluate the black box function multiple times with multiple queries (Algorithm 4 in Kirschner [2019]), making the comparison between this method and other BO methods unfair. Because of this property, LineBO is actually not suitable for optimizing a function that is expensive to evaluate. In our experiments, DescentLineBO will evaluate the black box function with 10 different queries for each iteration while all the other methods evaluate once.

[1]https://botorch.org

[2]https://github.com/jmetzen/bayesian_optimization

[3]https://github.com/facebook/Ax/tree/master/ax/modelbridge/strategies

[4]https://github.com/facebookresearch/alebo

[5]https://github.com/kirschnj/LineBO

[6]https://github.com/martinjankowiak/saasbo

In synthetic experiments, as described in section 5.1, for each test function we add some unimportant variables as well as unrelated variables to make it high-dimensional. The standard Branin function f_{Branin} has two dimensions with the input domain $\mathcal{X}_{Branin} = [-5, 10] \times [0, 10]$, and we construct a new Branin function F_{branin} as the following:

$$F_{branin}(\mathbf{x}) = f_{Branin}(\mathbf{x}_{[1:2]}) + 0.1 f_{Branin}(\mathbf{x}_{[3:4]}) + 0.01 f_{Branin}(\mathbf{x}_{[5:6]}),$$
$$\mathbf{x} \in \left(\bigotimes_{i=1}^{3} \mathcal{X}_{Branin} \right) \bigotimes_{i=1}^{44} [0, 1] \quad (15)$$

where $\bigotimes$ represents the direct product. We use $d_e = [2, 2, 2]$ to represent the dimension of the effective subspace of F_{branin}, the total effective dimension is 6, however, the number of important variables is only 2.

Likewise, for the standard Hartmann6 function $f_{Hartmann6}$ that has six dimensions with the input domain $[0, 1]^6$, we construct F_{hm6} as:

$$F_{hm6}(\mathbf{x}) = f_{Hartmann6}(\mathbf{x}_{[1:6]}) + 0.1 f_{Hartmann6}(\mathbf{x}_{[7:12]}) + 0.01 f_{Hartmann6}(\mathbf{x}_{[13:18]}) \quad \mathbf{x} \in [0, 1]^{50} \quad (16)$$

and use $d_e = [6, 6, 6]$ to represent the dimension of the effective subspace. For the Styblinski-Tang4 function f_{ST4} that has four dimensions with the input domain $[-5, 5]^4$, we construct F_{ST4} as:

$$F_{ST4}(\mathbf{x}) = f_{ST4}(\mathbf{x}_{[1:4]}) + 0.1 f_{ST4}(\mathbf{x}_{[5:8]}) + 0.01 f_{ST4}(\mathbf{x}_{[9:12]}) \quad \mathbf{x} \in [-5, 5]^{50} \quad (17)$$

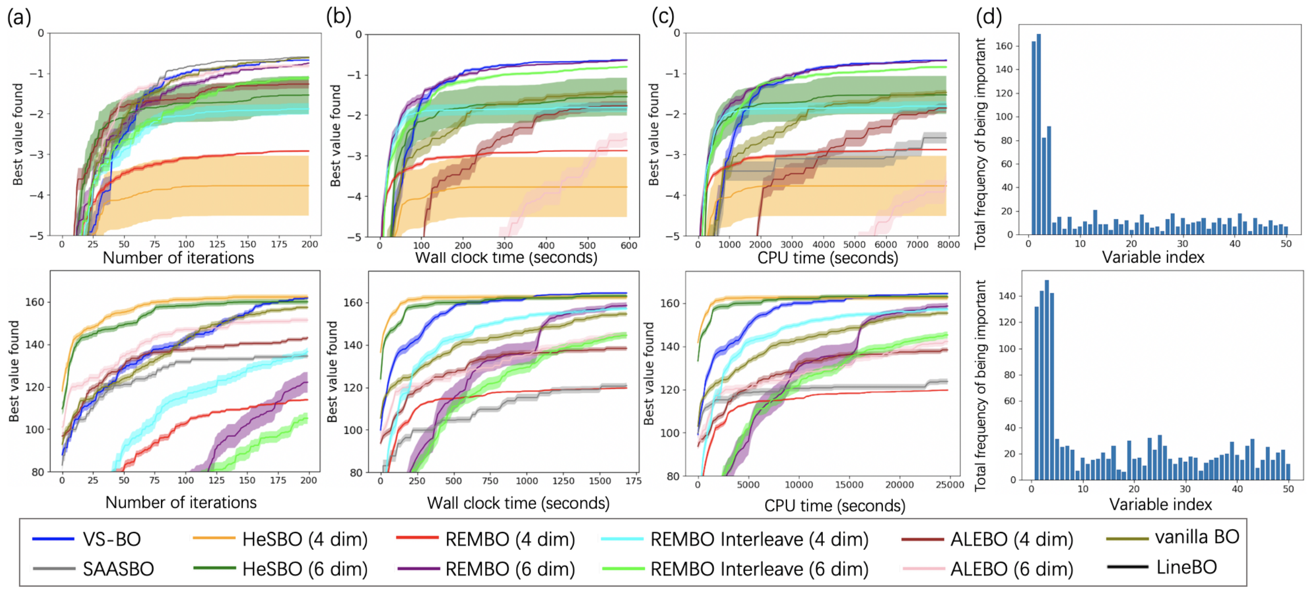

Figure 5: (a,b,c) Performance of BO methods on the Branin (first row) and Styblinski-Tang4 (second row) test functions. For each test function, we perform 20 independent runs for each method except SAASBO which is very time consuming, we perform 10 runs instead. We plot the mean and 1/8 standard deviation of the best maximum value found by (a) iterations, (b) wall clock time or (c) CPU time. (d) The total frequency of being chosen as important for each variable on Branin case and Styblinski-Tang4 case. Note that on Branin case the first 2 variables are the most important in reality, and the second 2 variables are the secondary important, and on Styblinski-Tang4 case the first 4 variables are the most important.

and use $d_e = [4, 4, 4]$ to represent the dimension of the effective subspace. All synthetic experiments are run on the same Linux cluster that has 40 3.0 GHz 10-Core Intel Xeon E5-2690 v2 CPUs.

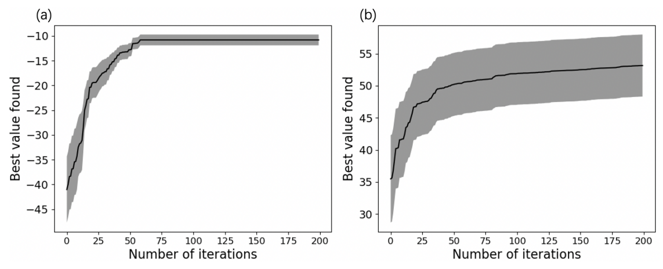

Figure 6: Performance of LineBO on the Branin (a) and Styblinski-Tang4 (b) function. We do 20 independent runs. We plot the mean and 1/8 standard deviation of the best maximum value found by iterations. Compared to Figures 1 and 5, we can see that the performance of LineBO is significantly worse than any other method on these two cases.

For real-world problems, the rover trajectory problem is a high-dimensional optimization problem with input domain $[0, 1]^{60}$. The problem setting in our experiment is the same as that in Wang [2017], and the source code of this problem can be found in https://github.com/zi-w/Ensemble-Bayesian-Optimization. MOPTA08 is another high-dimensional optimization problem with input domain $[0, 1]^{124}$. It has one objective function $f_{mopta}(\mathbf{x})$ that needs to be minimized and 68 constraints $c_i(\mathbf{x}), i \in \{1, 2, \ldots 68\}$. Similar to Eriksson [2021], we convert these constraints to soft penalties and convert the minimization problem to the maximization problem by adding a minus at the front of the objective function, i.e., we construct the following new function F_{mopta}:

$$F_{mopta}(\mathbf{x}) = -f_{mopta}(\mathbf{x}) - 10\sum_{i=1}^{68} \max(0, c_i(\mathbf{x})) \tag{18}$$

The Fortran codes of MOPTA can be found in https://www.miguelanjos.com/jones-benchmark and we further use codes in https://gist.github.com/denis-bz/c951e3e59fb4d70fd1a52c41c3675187 to wrap it in python. All experiments for these two real-world problems are run on the same Linux cluster that has 80 2.40 GHz 20-Core Intel Xeon 6148 CPUs.

As described in section 5.2, we design a sampling experiment to test the accuracy of the variable selection real world problems. The indices of the first 5 variables that have been chosen most frequently are $\{1, 2, 3, 59, 60\}$ on the rover trajectory problem and $\{30, 37, 42, 79, 112\}$ on MOPTA08, and the indices of the first 5 variables that have been chosen least frequently are $\{15, 18, 29, 38, 51\}$ and $\{59, 77, 91, 105, 114\}$ respectively (Figure 2c and 8c). The total number of samples in each set is 800000. Figure 2d and 8d show the empirical distributions of function values from two sets of samples. The significant difference between two distributions in each panel tells us that changing the values of variables that have been chosen more frequently can alter the function value more significantly, indicating that these variables are more important.

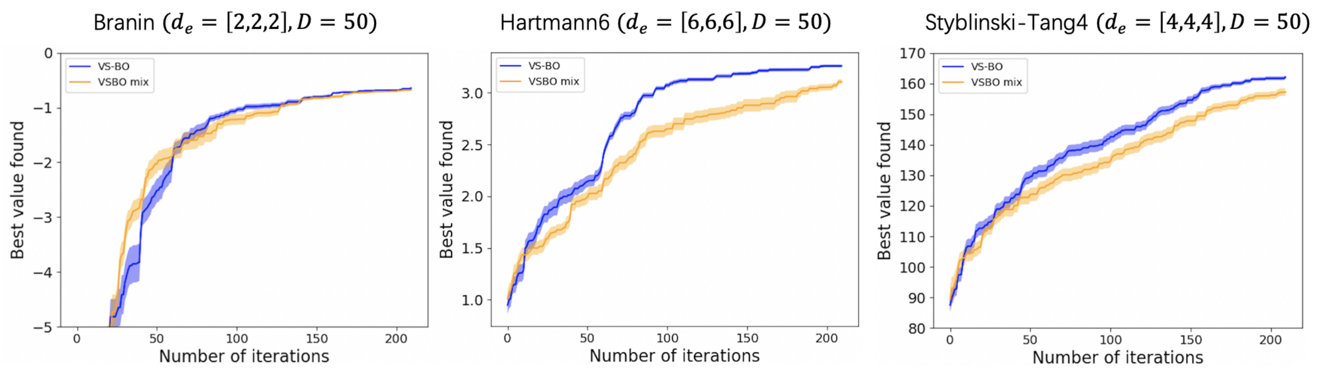

Figure 7: Performance of VS-BO and VSBO mix on Branin, Hartmann6 and Styblinski-Tang4 test functions. For each test function, we do 20 independent runs for each method. We plot the mean and 1/8 standard deviation of the best maximum value found by iterations.

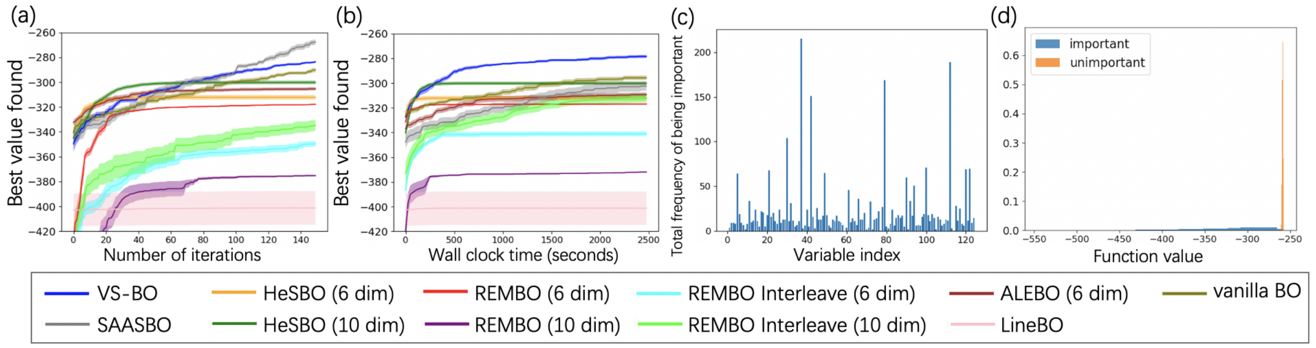

Figure 8: (a,b) Performance of BO methods on the MOPTA08 problem. For each test function, we perform 15 independent runs for each method except SAASBO which is very time consuming, we perform 10 runs instead. We plot the mean and 1/4 standard deviation of the best maximum value found by (a) iterations and (b) wall clock time. (c) The total frequency of being chosen as important for each variable. (d) The distribution of function values when sampling the first 5 variables that have been chosen most frequently (important) or the first 5 variables that have been chosen least frequently (unimportant) with all the other variables fixed.

Algorithm 5 Momentum in accurate case

```
Input: $\mathcal{D} = \{(\mathbf{x}^i, y^i)\}_{i=1}^t$, $N_{vs}$, set of important variables chosen at iteration $t - N_{vs}$, denoted as $\hat{\mathbf{x}}_{ipt}$, with cardinality $w = |\hat{\mathbf{x}}_{ipt}|$
Output: Set of important variables chosen at iteration $t$, denote as $\mathbf{x}_{ipt}$
 1: Fit a GP to $\mathcal{D}$ and calculate important scores $IS$ where $IS[i]$ is the important score of the i-th variable
 2: Sort variables according to $IS$, $[\mathbf{x}_{s(1)}, \dots, \mathbf{x}_{s(D)}]$, from the most important to the least
 3: Fit a GP by using variables in $\hat{\mathbf{x}}_{ipt}$, i.e. fit a GP to $\{(\hat{\mathbf{x}}_{ipt}^i, y^i)\}_{i=1}^t$, and calculate important scores of these variables $\widehat{IS}$. Let $\hat{L}_w$ be the value of final negative marginal log likelihood
 4: Sort variables in $\hat{\mathbf{x}}_{ipt}$ according to $\widehat{IS}$, $[\mathbf{x}_{s'(1)}, \dots, \mathbf{x}_{s'(w)}]$, from the most important to the least.
 5: for $m = w - 1, w - 2, \dots 0$ do
 6:     if $m = 0$ then
 7:         Set $\mathbf{x}_{ipt} = \{\mathbf{x}_{s'(1)}\}$
 8:         break
 9:     end if
10:     Fit a GP by only using the first $m$ important variables according to $\widehat{IS}$. Let $\hat{L}_m$ to be the value of final negative marginal log likelihood
11:     if $\hat{L}_m > \hat{L}_{m+1}$ then
12:         Set $\mathbf{x}_{ipt} = \{\mathbf{x}_{s'(1)}, \dots, \mathbf{x}_{s'(m+1)}\}$
13:         Set $L_0 = \hat{L}_{m+1}$
14:         break
15:     end if
16: end for
17: for $m = 1, 2, \dots D$ do
18:     if $\mathbf{x}_{s(m)} \in \mathbf{x}_{ipt}$ then
19:         Set $L_m = L_{m-1}$, $L_{m-1} = L_{m-2}$
20:         continue
21:     end if
22:     Fit a GP with variables in $\mathbf{x}_{ipt} \cup \{\mathbf{x}_{s(m)}\}$. Let $L_m$ to be the value of final negative marginal log likelihood
23:     if $m < 2$ then
24:         $\mathbf{x}_{ipt} = \mathbf{x}_{ipt} \cup \{\mathbf{x}_{s(m)}\}$
25:         continue
26:     else if $L_{m-1} - L_m \leq 0$ or $L_{m-1} - L_m < \frac{L_{m-2} - L(m-1)}{10}$ then
27:         break
28:     end if
29:     $\mathbf{x}_{ipt} = \mathbf{x}_{ipt} \cup \{\mathbf{x}_{s(m)}\}$
30: end for
31: return $\mathbf{x}_{ipt}$
```