Derivative-Free Global Optimization Algorithms: Bayesian Method and Lipschitzian Approaches

Jiawei Zhang JIAWEI@IFMLAB.ORG
Founder and Director
Information Fusion and Mining Laboratory
(First Version: March 2019; Revision: April 2019.)

Abstract

In this paper, we will provide an introduction to the derivative-free optimization algorithms which can be potentially applied to train deep learning models. Existing deep learning model training is mostly based on the back propagation algorithm, which updates the model variables layers by layers with the gradient descent algorithm or its variants. However, the objective functions of deep learning models to be optimized are usually non-convex and the gradient descent algorithms based on the first-order derivative can get stuck into the local optima very easily. To resolve such a problem, various local or global optimization algorithms have been proposed, which can help improve the training of deep learning models greatly. The representative examples include the Bayesian methods, Shubert-Piyavskii algorithm, Direct, LIPO, MCS, GA, SCE, DE, PSO, ES, CMA-ES, hill climbing and simulated annealing, etc. One part of these algorithms will be introduced in this paper (including the Bayesian method and Lipschitzian approaches, e.g., Shubert-Piyavskii algorithm, Direct, LIPO and MCS), and the remaining algorithms (including the population based optimization algorithms, e.g., GA, SCE, DE, PSO, ES and CMA-ES, and random search algorithms, e.g., hill climbing and simulated annealing) will be introduced in the follow-up paper [18] in detail.

Keywords: Derivative-Free; Global Optimization; Bayesian Method; Lipschitzian Approach; Deep Learning

Contents

1. Introduction

As introduced in the previous IFM Lab tutorial article [19], existing deep learning model training is mostly based on the back propagation algorithm, which updates the model variables layers by layers with the gradient descent algorithms. Gradient descent together with its many variants have been shown to be effective for optimizing a large group of problems. However, for the non-convex objective functions of deep learning models, gradient descent algorithms cannot guarantee to identify the globally optimal solutions, whose iterative updating process may inevitably get stuck in the local optima. The performance of gradient descent algorithms are not very robust, which will be greatly degraded for the objective functions with non-smooth shape or learning scenarios polluted by noisy data. Furthermore, the distributed computation process of gradient descent requires heavy synchronization, which may hinder its adoption in large-cluster based distributed computational platforms.

Besides these optimization algorithms based on gradient descent, there also exist a group of optimization algorithms which can be potentially applied to learn the optimal variables for the deep learning models. These optimization algorithms don't need to compute the derivatives of the objective function regarding the model variables, thus they are called the derivative-free optimization algorithms in this paper. The representative examples include the Bayesian methods, Shubert-Piyavskii algorithm, DIRECT, LIPO, MCS, GA, SCE, DE, PSO, ES, CMA-ES, hill climbing and simulated annealing, etc. Many of these derivative-free optimization algorithms can even effectively compute the global optimal variables for the non-convex objective functions. These algorithms along can work well for training the deep learning models. Meanwhile, they can also work with the gradient descent algorithms in a hybrid manner similar to Gadam [20] as we have introduced in [19], which can integrate the advantages of both the gradient descent optimization algorithms together with these derivative-free optimization algorithms.

Here, we will briefly introduce the learning settings as follows. The training set for optimizing the deep learning models can be represented as $\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \cdots, (\mathbf{x}_n, \mathbf{y}_n)\}$, which involves n pairs of feature-label instances. Formally, for each data instance, its feature vector $\mathbf{x}_i \in \mathbb{R}^{d_x}, \forall i \in \{1, 2, \cdots, n\}$ and label vector $\mathbf{y}_i \in \mathbb{R}^{d_y}, \forall i \in \{1, 2, \cdots, n\}$ are of dimensions d_x and d_y respectively. The deep learning models define a mapping $F(\cdot; \boldsymbol{\theta}) : \mathcal{X} \to \mathcal{Y}$, which projects the data instances from the feature space $\mathcal{X}$ to the label space $\mathcal{Y}$. In the above representation of function $F(\cdot, \boldsymbol{\theta})$, vector $\boldsymbol{\theta} \in \Theta$ contains the variables involved in the deep learning model and Θ denotes the variable inference space. Formally, we can denote the dimension of variable vector $\boldsymbol{\theta}$ as d_θ, which will be used when introducing the algorithms later. Given one data instance featured by vector $\mathbf{x}_i \in \mathcal{X}$, we can denote its prediction label vector by the deep learning model as $\hat{\mathbf{y}}_i = F(\mathbf{x}_i; \boldsymbol{\theta})$. Compared against its true label vector $\mathbf{y}_i$, we can denote the introduced loss for instance $\mathbf{x}_i$ as $\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i)$. Several frequently used loss representations have been introduced in [19], and we will not redefine them here again. For all the data instances in the training set, we can represent the total loss term as

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{T}} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i). \tag{1}$$

And the deep model learning can be formally denoted as the following function:

$$\min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta}), \tag{2}$$

which is also the main objective function to be studied in this paper.

We need to add a remark here, the above objective function defines a minimization problem. Meanwhile, when introducing some of the optimization algorithms in the following sections, we may assume the objective function to be a maximization function instead for simplicity. The above objective can be transformed into a maximization problem easily by introducing a new term $\mathcal{L}'(\boldsymbol{\theta}) = -\mathcal{L}(\boldsymbol{\theta})$. We will clearly indicate it when the algorithm is introduced for a maximization problem.

In the following part of this paper, we will introduce the derivative-free optimization algorithms that can be potentially used to resolve the above objective function. To be more specific, this paper covers the introduction to the Bayesian method as well as the Lipschitzian approaches (including Shubert-Piyavskii algorithm, Direct, LIPO and MCS) for global optimization. The population based algorithms (e.g., GA, SCE, DE, PSO, ES and CMA-ES) and the random search based optimization algorithms (e.g., hill climbing and simulated annealing) will be introduced in the follow-up article [18] in detail.

2. Bayesian Method for Global Optimization

Bayesian methods [11] assume the objective function is a black-box. Evaluating the objective function is expensive or even impossible, whose derivatives and convexity properties are also unknown. Such an assumption holds for some deep learning models involving complicated structures with multiple layers of non-linear projections, since its analytical expression and derivatives will be too complex to analyze. Instead of optimizing the objective function directly, Bayesian methods propose to approximate the objective function based on a set of sampled points in the variable domain, and tries to select the optimal variable based on the approximated objective function. Such a process involves two basic functions in the Bayesian methods: *surrogate function* and *acquisition function*, respectively. To approximate the objective function, Bayesian method adopts a *surrogate function* to update the posterior distribution of the function based on both the prior distribution and the likelihood computed with the sampled points. Meanwhile, to sample efficiently, Bayesian methods use an *acquisition function* to determine the next optimal variable to sample from the variable domain.

In Algorithm 1, we provide the pseudo-code of the Bayesian Method, which involves 3 main steps: (1) optimal variable point computing via maximizing the acquisition function; (2) objective function value sampling subject to noise; and (3) Gaussian Process updating with the surrogate function to update $\mu(\boldsymbol{\theta})$ and $\sigma^2(\boldsymbol{\theta})$.

Algorithm 1 Bayesian Optimization Methods

```
Require: Input variable space $\Theta$; GP prior $\mu$, $\sigma$.
Ensure: Model Parameter $\boldsymbol{\theta}$
1: for $\tau = 1, 2, \cdots$ do
2:     Find $\boldsymbol{\theta}_\tau = \arg\max_{\boldsymbol{\theta} \in \Theta} \alpha(\boldsymbol{\theta}; \mathcal{S}_{\tau-1})$ by optimizing the acquisition function.
3:     Sample the objective function value $z_\tau = \mathcal{L}(\boldsymbol{\theta}_\tau) + \epsilon_\tau$.
4:     Update $\mathcal{S}_\tau = \mathcal{S}_{\tau-1} \cup \{(\boldsymbol{\theta}_\tau, z_\tau)\}$ and recompute $\mu(\boldsymbol{\theta})$, $\sigma^2(\boldsymbol{\theta})$ based on the surrogate function.
5: end for
6: Return model variable $\boldsymbol{\theta}$
```

Most of the steps in Algorithm 1 will be covered in the following subsections, except for the noisy function value sampling part. Considering noise-free observations are very hard to achieve in the real-world. In the algorithm, term $\epsilon_\tau \sim \mathcal{N}(0, \sigma^2_{\text{NOISE}})$ is a random noise sampled from the normal distribution, which performs a noisy transformation on function $\mathcal{L}(\boldsymbol{\theta})$. A tutorial on Bayesian methods is available at [1].

2.1 Surrogate Function for Posterior Distribution Updating

In Bayesian methods, the sample sequence achieved prior to iteration $\tau \geq 1$ can be denoted as $\mathcal{S}_{\tau-1} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_{\tau-1}\}$ (or $\mathcal{S}_{\tau-1} = \{(\boldsymbol{\theta}_1, z_1), (\boldsymbol{\theta}_2, z_2), \cdots, (\boldsymbol{\theta}_{\tau-1}, z_{\tau-1})\}$ if we pair the sampled variables with their introduced loss terms together), where $z_i = \mathcal{L}(\boldsymbol{\theta}_i)$ denotes the loss value introduced by variable $\boldsymbol{\theta}_i$. Formally, let $P(\mathcal{L})$ denote the prior distribution of the loss function, and $P(\mathcal{S}_{\tau-1}|\mathcal{L})$ denote the likelihood function of achieve the sampled variables based on the loss function $\mathcal{L}$. Based on these two terms, the posterior distribution of the objective loss function based on the sampled points can be denoted as

$$P(\mathcal{L}|\mathcal{S}_{\tau-1}) \propto P(\mathcal{S}_{\tau-1}|\mathcal{L}) P(\mathcal{L}). \tag{3}$$

The step of updating the posterior distribution of the loss function based on the sampled variable points is also interpreted as estimating the objective function with a *surrogate function* (also called a *response surface*).

As pointed out in [11], such a process can be modeled effectively with the Gaussian Process (GP), where the prior distribution of the loss function is subject to a Gaussion distribution. Here, let's misuse the loss function notation $\mathcal{L}$ as a variable, and it follows

$$\mathcal{L}(\boldsymbol{\theta}) \sim \mathcal{N}(m_{\boldsymbol{\theta}}, k_{\boldsymbol{\theta}}), \tag{4}$$

where $m_{\boldsymbol{\theta}}$ and $k_{\boldsymbol{\theta}}$ denote the mean and co-variance functions of the normal distribution respectively. For convenience, the prior mean is usually assumed to be zeros, and the variance function $k_{\boldsymbol{\theta}}$ has several different choices (we will introduce later).

Following such a distribution, let's assume we have sampled $\tau - 1$ variable points already from objective variable domain Θ, whose loss terms can be denoted as a vector

$[\mathcal{L}(\boldsymbol{\theta}_1), \mathcal{L}(\boldsymbol{\theta}_2), \cdots, \mathcal{L}(\boldsymbol{\theta}_{\tau-1})]^\top$. According to the above descriptions, they should follow the following multivariate normal distribution

$$\begin{bmatrix} \mathcal{L}(\boldsymbol{\theta}_1) \\ \mathcal{L}(\boldsymbol{\theta}_2) \\ \vdots \\ \mathcal{L}(\boldsymbol{\theta}_{\tau-1}) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{K}\right), \tag{5}$$

where the covariance matrix

$$\mathbf{K} = \begin{bmatrix} k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_1), & \cdots, & k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_{\tau-1}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{\theta}_{\tau-1}, \boldsymbol{\theta}_1), & \cdots, & k(\boldsymbol{\theta}_{\tau-1}, \boldsymbol{\theta}_{\tau-1}) \end{bmatrix} \tag{6}$$

By further sampling one more variable point $\boldsymbol{\theta}_\tau$, in a similar way, we can achieve the distribution of the sampled τ variable points as

$$\begin{bmatrix} \mathcal{L}(\boldsymbol{\theta}_1) \\ \mathcal{L}(\boldsymbol{\theta}_2) \\ \vdots \\ \mathcal{L}(\boldsymbol{\theta}_{\tau-1}) \\ \mathcal{L}(\boldsymbol{\theta}_\tau) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^\top & k(\boldsymbol{\theta}_\tau, \boldsymbol{\theta}_\tau) \end{bmatrix}\right), \tag{7}$$

where vector $\mathbf{k} = [k(\boldsymbol{\theta}_1, \boldsymbol{\theta}_\tau), k(\boldsymbol{\theta}_2, \boldsymbol{\theta}_\tau), \cdots, k(\boldsymbol{\theta}_{\tau-1}, \boldsymbol{\theta}_\tau)]^\top$.

Based on the Sherman-Morrison-Woodbury formula [14], we can represent the posterior distribution of the loss function for the sampled point $\boldsymbol{\theta}_\tau$ as follows

$$P(\mathcal{L}(\boldsymbol{\theta}_\tau) | \mathcal{S}_{\tau-1}, \boldsymbol{\theta}_\tau) \sim \mathcal{N}(\mu_\tau(\boldsymbol{\theta}_\tau), \sigma^2_\tau(\boldsymbol{\theta}_\tau)), \tag{8}$$

where

$$\mu_\tau(\boldsymbol{\theta}_\tau) = \mathbf{k}^\top \mathbf{K}^{-1} \begin{bmatrix} \mathcal{L}(\boldsymbol{\theta}_1) \\ \mathcal{L}(\boldsymbol{\theta}_2) \\ \vdots \\ \mathcal{L}(\boldsymbol{\theta}_{\tau-1}) \end{bmatrix}, \tag{9}$$

$$\sigma^2_\tau(\boldsymbol{\theta}_\tau) = k(\boldsymbol{\theta}_\tau, \boldsymbol{\theta}_\tau) - \mathbf{k}^\top \mathbf{K}^{-1} \mathbf{k}. \tag{10}$$

As we mentioned before, the co-variance function $k(\cdot, \cdot)$ may be defined in different ways, and some representative examples include:

- *squared exponential function*:

$$k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \exp(-\frac{1}{2} \left\| \boldsymbol{\theta}_i - \boldsymbol{\theta}_j \right\|^2). \tag{11}$$

- *squared exponential function with hyperparameters*:

$$k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \exp(-\frac{1}{2\beta^2} \left\| \boldsymbol{\theta}_i - \boldsymbol{\theta}_j \right\|^2), \tag{12}$$

where the hyperparameter β controls the width of the variance.

- *squared exponential function with automatic relevance determination hyperparameters*:

$$k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \exp(-\frac{1}{2}(\boldsymbol{\theta}_i - \boldsymbol{\theta}_j)^\top diag(\boldsymbol{\beta})^{-2}(\boldsymbol{\theta}_i - \boldsymbol{\theta}_j)), \tag{13}$$

  where $\boldsymbol{\beta}$ is a hyperparameter vector and notation $diag(\boldsymbol{\beta})$ denotes a diagonal matrix with entries $\boldsymbol{\beta}$ on its diagonal.

- *Matérn function*:

$$k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) = \frac{1}{2^{\varsigma-1}\Gamma(\varsigma)}(2\sqrt{\varsigma}\,\|\boldsymbol{\theta}_i - \boldsymbol{\theta}_j\|)^{\varsigma} H_{\varsigma}(2\sqrt{\varsigma}\,\|\boldsymbol{\theta}_i - \boldsymbol{\theta}_j\|), \tag{14}$$

  where $\Gamma(\cdot)$ and $H_{\varsigma}(\cdot)$ are the Gamma function and Bessel function of order ς respectively. As the order hyperparameter $\varsigma \to \infty$, the *Matérn function* will be reduced to the *squared exponential function*.

2.2 Acquisition Function for Optimal Variable Sampling

Based on the posterior distribution updated via the Gaussian Process iteratively, here, we will introduce the *acquisition function*, which guides the search for the optimal variables in Bayesian methods. Typically, acquisition functions are defined to ensure high acquisition function values correspond to potentially higher values of the objective function. Therefore, the optimal variable points will be selected iteratively which can maximize the acquisition function. Considering the objective function studied in deep learning model training is a minimization problem, which can be transformed into a maximization problem easily by defining a pseudo-loss function $\mathcal{L}'(\boldsymbol{\theta}) = -\mathcal{L}(\boldsymbol{\theta})$. The following contents are introduced based on the maximization objective function on the loss function $\mathcal{L}'(\boldsymbol{\theta})$ (to simplify the notations, we will use $\mathcal{L}(\boldsymbol{\theta})$ directly to denote the function to be maximized).

Formally, based on the sampling records prior to iteration τ, i.e., $\mathcal{S}_{\tau-1} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_{\tau-1}\}$, Bayesian methods will select the optimal variable for iteration τ with the following function:

$$\boldsymbol{\theta}_\tau = \arg\max_{\boldsymbol{\theta} \in \Theta} \alpha(\boldsymbol{\theta}; \mathcal{S}_{\tau-1}), \tag{15}$$

where $\alpha(\boldsymbol{\theta}; \mathcal{S}_{\tau-1})$ is called the *acquisition function*. Acquisition function helps the Bayesian methods to compute the next sampling point of the variables, and it has various representations. Popular acquisition functions include the *maximum probability of improvement* (MPI) [7], *expected improvement* (EI) [10] and *upper confidence bound* (UCB) [3], which will be introduced as follows respectively. Meanwhile, the above objective acquisition function can be optimized with the Direct algorithm as introduced in Section 3.2.

2.2.1 MPI

Formally, let $\boldsymbol{\theta}^+ = \arg\max_{\boldsymbol{\theta} \in \mathcal{S}_{\tau-1}} \mathcal{L}(\boldsymbol{\theta})$ denote the best variable among all the variables we have sampled in the previous $\tau - 1$ iterations. For the MPI acquisition function [7], its main objective will be to select the variable points which can lead to the improvement with the maximum probability. In other words, the MPI based acquisition function can be formally represented as

$$\alpha_{\text{MPI}}(\boldsymbol{\theta}) = P\big(\mathcal{L}\left(\boldsymbol{\theta}\right) \geq \mathcal{L}(\boldsymbol{\theta}^{+})\big) \tag{16}$$

$$= \Phi\left(\frac{\mu(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^{+})}{\sigma(\boldsymbol{\theta})}\right), \tag{17}$$

where function $\Phi(\cdot)$ denote the cumulative distribution function (CDF) of the standard Gaussian distribution and the iteration index subscript (i.e., τ as we used in Equations 9) on the mean and standard deviation functions $\mu(\boldsymbol{\theta})$ and $\sigma(\boldsymbol{\theta})$ are omitted for simplicity reasons.

According to the above function, the MPI based acquisition function mainly care about the probability value instead of the real advantages of the loss term $\mathcal{L}(\boldsymbol{\theta})$ against $\mathcal{L}(\boldsymbol{\theta}^{+})$. It is possible that maximizing the above acquisition function may lead to a variable point $\boldsymbol{\theta}$ which is slightly greater than $\boldsymbol{\theta}^{+}$ (i.e., $\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^{+} \to 0^{+})$ simply because the probability value $P\big(\mathcal{L}\left(\boldsymbol{\theta}\right) \geq \mathcal{L}(\boldsymbol{\theta}^{+})\big)$ is large. To resolve such a problem, some research works also propose to use the following MPI based acquisition function instead:

$$\alpha_{\text{MPI}}(\boldsymbol{\theta}) = P\big(\mathcal{L}\left(\boldsymbol{\theta}\right) \geq \mathcal{L}(\boldsymbol{\theta}^{+}) + \xi\big) \tag{18}$$

$$= \Phi\left(\frac{\mu(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^{+}) - \xi}{\sigma(\boldsymbol{\theta})}\right). \tag{19}$$

The parameter ξ defines the desired gap in the loss function in the variable selection, which is usually a hyperparameter inputted into the algorithm.

2.2.2 EI

Since $\mathcal{L}(\boldsymbol{\theta})$ is the objective function to optimize in our studied problem, the main focus in variable selection should be picking the ones which can lead the maximum improvement against the best historical variable, i.e., $\boldsymbol{\theta}^{+} = \arg\max_{\boldsymbol{\theta} \in \mathcal{S}_{\tau-1}} \mathcal{L}(\boldsymbol{\theta})$. In other words, the EI [10] based acquisition function can be defined as follows:

$$\alpha_{\text{EI}}(\boldsymbol{\theta}) = \mathbb{E}\left(I(\boldsymbol{\theta}; \mathcal{S}_{\tau-1})\right) \tag{20}$$

$$= \mathbb{E}\left(\max\{0, \mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^{+})\}\right), \tag{21}$$

where $I(\mathcal{L}(\boldsymbol{\theta}); \mathcal{S}_{\tau-1}) = \max\{0, \mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^{+})\}$ is also called the *improvement function*.

Meanwhile, the normal density function of I (here, we treat function $I(\boldsymbol{\theta}; \mathcal{S}_{\tau-1})$ as a variable) can be computed based on the posterior distribution parameterized by mean $\mu(\boldsymbol{\theta})$ and co-variance $\sigma^2(\boldsymbol{\theta})$ as follows:

$$P(I) = \frac{1}{\sqrt{2\pi}\sigma(\boldsymbol{\theta})} \exp\left(-\frac{\big(\mu(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^{+}) - I\big)^2}{2\sigma^2(\boldsymbol{\theta})}\right). \tag{22}$$

Therefore, the expected improvement can be represented as

$$\mathbb{E}(I) = \int_{I=0}^{I=\infty} I P(I) \mathrm{d}I \tag{23}$$

$$= \sigma(\boldsymbol{\theta}) \left[\frac{\mu(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^{+})}{\sigma(\boldsymbol{\theta})} \cdot \Phi\left(\frac{\mu(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^{+})}{\sigma(\boldsymbol{\theta})}\right) + \phi\left(\frac{\mu(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^{+})}{\sigma(\boldsymbol{\theta})}\right)\right], \tag{24}$$

where $\Phi(\cdot)$ and $\phi(\cdot)$ denote the CDF and PDF of the standard Gaussian distribution respectively.

Based on the above analysis, we can formally represent the EI based acquisition function as follows:

$$\alpha_{\text{EI}}(\boldsymbol{\theta}) = \begin{cases} \left(\mu(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^+)\right) \Phi(Z) + \sigma(\boldsymbol{\theta})\phi(Z) & \text{if } \sigma(\boldsymbol{\theta}) > 0; \\ 0 & \text{if } \sigma(\boldsymbol{\theta}) = 0, \end{cases} \tag{25}$$

where term $Z = \frac{\mu(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^+)}{\sigma(\boldsymbol{\theta})}$.

2.2.3 UCB

As introduced in [3], given a random function on variable $\boldsymbol{\theta}$, we can compute its maximum value by maximizing its *upper confidence bound* (UCB) as follows:

$$UCB(\boldsymbol{\theta}) = \mu(\boldsymbol{\theta}) + \kappa\sigma(\boldsymbol{\theta}), \tag{26}$$

where $\kappa \geq 0$ is a hyperparameter.

In recent years, some works propose to adopt the UCB function to define the acquisition function for optimizing the posterior distribution function. Formally, let $\mathcal{L}(\boldsymbol{\theta}^*)$ denote the globally optimal value. Therefore, for any variable $\boldsymbol{\theta}$ we choose, we can represent its distance compared with the optimal value as the following regret function [16]:

$$r(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}^*) - \mathcal{L}(\boldsymbol{\theta}). \tag{27}$$

In sampling the variable points, our main objective will be to minimize the regret function, which is equivalent to the maximization of function values at the sampled variable points, i.e.,

$$\min \sum_{i=1}^{i=\tau} r(\boldsymbol{\theta}_i) \propto \max \sum_{i=1}^{i=\tau} \mathcal{L}(\boldsymbol{\theta}_i). \tag{28}$$

By using the *upper confidence bound* as the selection criterion, we can define the UCB based acquisition function as follows:

$$\alpha_{\text{UCB}}(\boldsymbol{\theta}) = \mu(\boldsymbol{\theta}) + \sqrt{\nu\gamma_\tau}\sigma(\boldsymbol{\theta}), \tag{29}$$

where $\gamma_\tau = 2\log(d_\theta \tau^2 \pi^2 / 6\delta)$ is a function about τ, while $\delta \in (0,1)$ and $\nu > 0$ are the hyperparameters in the algorithm. By maximizing the UCB based acquisition function, we will be able to select promising variable points iteratively.

3. Lipschitzian Approaches for Global Optimization

Lipschitzian approach is the name for a group of optimization approaches based on the Lipschitz functions. Lipschitzian approaches usually require the prior knowledge about the Lipschitz constant, which is a bound on the change rate of the objective function. Meanwhile, in the case where the Lipschitz constant is unknown, some other variant approaches have been proposed, like Direct, LIPO and MCS, which will be introduced in this section as well. Lipschitzian approaches have many outstanding advantages compared against other optimization algorithms. First of all, Lipschitzian approaches have very few parameters to

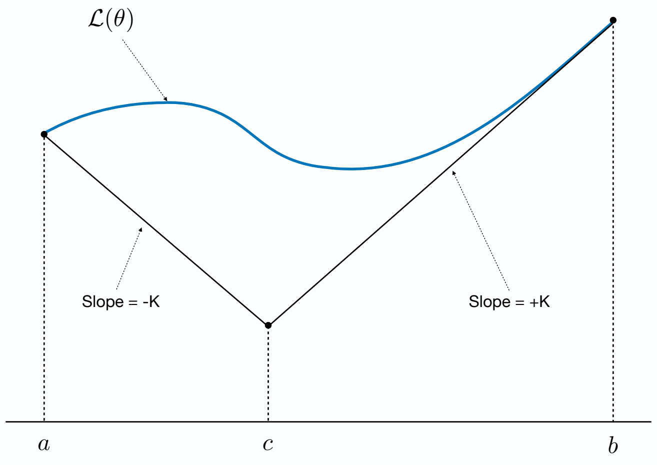


Figure 1: An Example to Illustrate the Shubert-Piyavskii Algorithm.

tune, which is a desired strength compared with other algorithms. On the other hand, since they are the deterministic methods, Lipschitzian approaches don't require multiple runs in computing the optimum.

3.1 Shubert-Piyavskii Algorithm

In this part, we will introduce the classic Shubert-Piyavskii algorithm [15, 13] based on an objective function with one single variable. For the case where the objective function is multivariate, several extension works have been proposed, and we will briefly mention some of them at the end of this subsection. Formally, let's assume the objective function to be minimized here is $\mathcal{L}(\theta)$, where the variable θ is within a closed interval $[a, b]$ (i.e., variable θ is constrained and a, b denote the upper bound and lower bound respectively).

Standard Lipschitzian approaches usually assume the objective function $\mathcal{L}(\theta)$ is subject to the Lipschitz function, i.e., there exist a finite bound on the rate of function changes. Formally, assuming there exist a positive constant K, the following Lipschitz function should hold:

$$\left|\mathcal{L}(\theta_1) - \mathcal{L}(\theta_2)\right| \leq K \cdot \left|\theta_1 - \theta_2\right|, \forall \theta_1, \theta_2 \in [a, b], \tag{30}$$

where constant K is also called the Lipschitz constant.

The above inequality actually creates a lower bound for the objective function, and we will illustrate it with the following example.

Example 1 *Based on the Lipschitz function provided in formula (30), we can replace θ_1 with θ and θ_2 with a, which will lead to*

$$\left|\mathcal{L}(\theta) - \mathcal{L}(a)\right| \leq K \cdot \left|\theta - a\right| \tag{31}$$

$$\Rightarrow \mathcal{L}(a) - \mathcal{L}(\theta) \leq \left|\mathcal{L}(\theta) - \mathcal{L}(a)\right| \leq K \cdot (\theta - a) \tag{32}$$

$$\Rightarrow \mathcal{L}(\theta) \geq \mathcal{L}(a) - K \cdot (\theta - a), \forall \theta \in [a, b]. \tag{33}$$

Similarly, by substituting θ_1 with θ and θ_2 with b, we can get

$$\mathcal{L}(\theta) \geq \mathcal{L}(b) + K \cdot (\theta - b), \forall \theta \in [a, b]. \tag{34}$$

These two inequalities introduce two lines with slopes $-K$ and $+K$ as shown in Figure 1 respectively, which forms a "V" structure. According to the plot, the curve of function $\mathcal{L}(\theta)$ is above these two lines, i.e., function $\mathcal{L}(\theta)$ is lower-bounded by these two lines. The intersection of these two lines is the lowest point for these two lines, whose coordinate can be denoted as c. With some simple derivations, we can get the representations of c as well as its corresponding function value to be

$$c = \frac{(a+b)}{2} + \frac{\mathcal{L}(a) - \mathcal{L}(b)}{2K}, \tag{35}$$

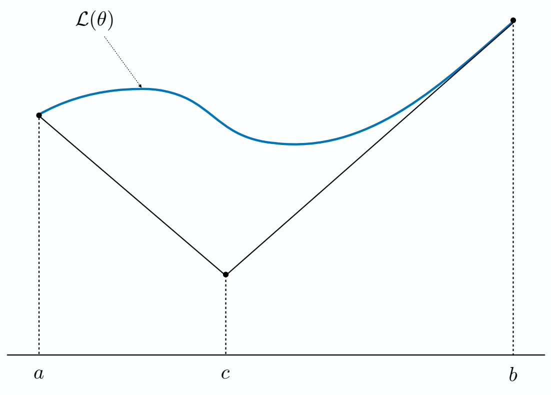
$$\mathcal{L}(c) = \frac{\mathcal{L}(a) + \mathcal{L}(b)}{2} - K \cdot (b - a). \tag{36}$$

The above two equations will serve as the core component in the Shubert-Piyavskii algorithm, whose detailed descriptions will be provided as follows. We can still use the curve in Figure 1 as an example.

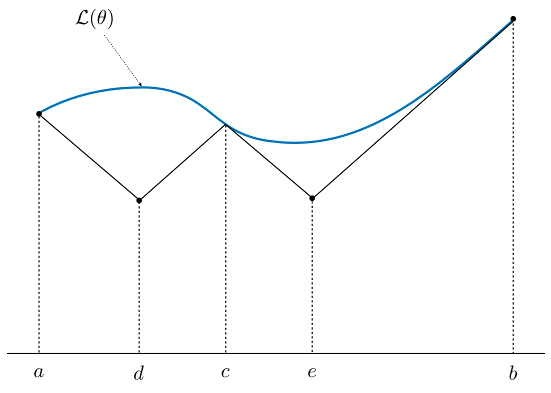
1. As shown by Figure 2(a), the algorithm starts by the evaluation of the objective function at the endpoints a and b to compute terms $\mathcal{L}(a)$ and $\mathcal{L}(b)$, which help determine the division point c together with its evaluation term $\mathcal{L}(c)$.
2. Point c divides the search space $[a, b]$ into two intervals $[a, c]$ and $[c, b]$, and we are interested in which interval may have a lower objective function value. As shown in Figure 2(b), the evaluation at the endpoints of them will bring about two new division points d and e together with their evaluations $\mathcal{L}(d)$ and $\mathcal{L}(e)$ respectively. In this example, we happen to have $\mathcal{L}(d) = \mathcal{L}(e)$, and the algorithm will randomly choose one interval to divide, e.g., interval $[a, c]$, which will lead to 3 intervals $[a, d]$, $[d, c]$ and $[c, b]$ respectively.
3. Next, as shown in Figure 2(c), the algorithm will compare the evaluations at the division points of these three intervals, i.e., f, g and e, and chose the next ones to explore. Among these three intervals, $[c, b]$ will lead to a smaller objective function value at the division point e and should be explored next.

Such an algorithm continues until the minimum of the approximation is within some pre-specified tolerance of the current best solution. The pseudo-code of the Shubert-Piyavskii algorithm is provided in Algorithm 2.

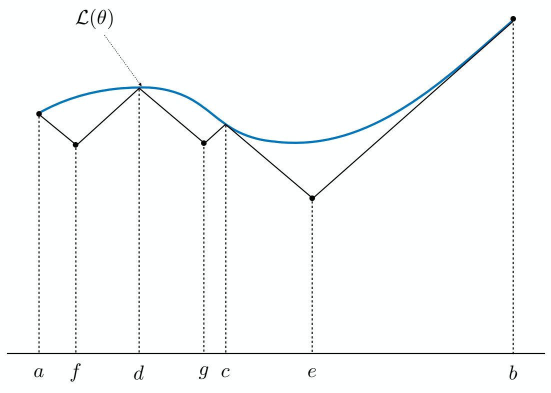
In the algorithm, the interval selection criteria is the objective function evaluation at the corresponding division point, as illustrated in Equation 35. Among all the existing intervals, the one with the minimal $\mathcal{L}(c) = \frac{\mathcal{L}(a)+\mathcal{L}(b)}{2} - K \cdot (b - a)$ will be picked. By checking the representation of $\mathcal{L}(c)$, we observe that $\mathcal{L}(c)$ can achieve a smaller value iff (1) $\frac{\mathcal{L}(a)+\mathcal{L}(b)}{2}$ is small, and (2) $K \cdot (b-a)$ is large. In other words, for the intervals whose endpoints correspond to smaller objective function values (i.e., the interval contains points with relatively smaller values), and the interval is wide (i.e., there exists more space to explore in the interval), they will be picked for exploration in the Shubert-Piyavskii algorithm.



(a) Step 1



(b) Step 2



(c) Step 3

Figure 2: Shubert-Piyavskii Algorithm Optimization Process.

The Shubert-Piyavskii algorithm has two main disadvantages in applications in the real world:

Algorithm 2 Shubert Piyavskii Algorithm

```
Require: Variable range [a, b].
Ensure: Model Parameter $\theta$
 1: Evaluate $\mathcal{L}(a)$ and $\mathcal{L}(b)$ for endpoints $a$ and $b$.
 2: Compute division point $c$ and evaluate $\mathcal{L}(c)$ according to Equation (35).
 3: Initialize set $\mathcal{S} = \{[\mathcal{L}(c) : (a, b; c)]\}$.
 4: while Stop criteria is not met. do
 5:     Select interval $[a_i, b_i]$ whose corresponding $\mathcal{L}(c_i)$ is the minimum.
 6:     $\mathcal{S} = \mathcal{S} \setminus \{[\mathcal{L}(c_i) : (a_i, b_i; c_i)]\}$.
 7:     Divide interval $[a_i, b_i]$ into two intervals $[a_i, c_i]$ and $[c_i, b_i]$.
 8:     Compute division point $d_i$ and evaluate $\mathcal{L}(d_i)$ for interval $[a_i, c_i]$ according to Equation (35).
 9:     Compute division point $e_i$ and evaluate $\mathcal{L}(e_i)$ for interval $[c_i, b_i]$ according to Equation (35).
10:     Update $\mathcal{S} = \mathcal{S} \cup \{[\mathcal{L}(d_i) : (a_i, c_i; d_i)], [\mathcal{L}(e_i) : (c_i, b_i; e_i)]\}$.
11: end while
```

- **Global Exploration**: The Shubert-Piyavskii algorithm is highly dependent on the hyperparameter K. To ensure the Lipschitz function can hold, the parameter K is usually a very large value. It also brings about a problem as the interval selection criteria $\mathcal{L}(c) = \frac{\mathcal{L}(a)+\mathcal{L}(b)}{2} - K \cdot (b-a)$ will be mainly determined by the interval width $b-a$. In other words, the Shubert-Piyavskii algorithm will focus on global exploration in applications on real-world problems.

- **Computational Complexity**: The Shubert-Piyavskii algorithm introduced here works well for the objective function with one single variable. When it comes to the scenario with d_θ variables, the exploration of the Shubert-Piyavskii algorithm will involve the evaluations at 2^{d_θ} endpoints, which will become very inefficient. To resolve such a problem, several variant version of the Shubert-Piyavskii algorithm have been proposed for the multivariate objective functions, including [4, 12, 9]. However, these extension algorithms may still be very computationally complex for some other reasons, which also hinders its application in many real-world optimization problems.

3.2 Direct Algorithm

The Shubert-Piyavskii algorithm requires the prior knowledge of the Lipschitz constant K for the objective function in advance, which may be impractical in real-world applications. In this part, we will introduce the Direct (DIviding RECTangles) algorithm [6], which can also effectively resolve the two disadvantages mentioned above for the Shubert-Piyavskii algorithm. We will first introduce the Direct algorithm for the one-dimensional variable first, and then talk about its extensions to the multi-dimensional variable objective function.

3.2.1 One-dimensional Direct Algorithm

According to the analysis provided before, one of the great challenges that hinder the application of the Shubert-Piyavskii algorithm in multivariate objective functions is due to its huge evaluation costs, which grows exponentially with the variable dimensions. Instead of evaluating the objective function at the endpoints of an interval, the Direct algorithm evaluates the function at the center point instead. In other words, when it deals with the

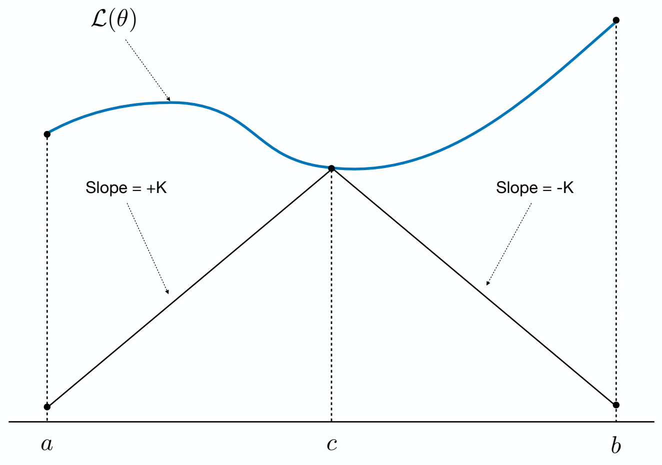


Figure 3: An Example to Illustrate the DIRECT Algorithm.

multivariate objective function (e.g., with n variable), the function evaluation will need to be done just once, instead of the 2^n as required by the Shubert-Piyavskii algorithm.

Based on the Lipschitz function (i.e., the inequality in Formula 30), by replacing θ_1 with θ and θ_2 with c, we can easily get the following two inequalities:

$$\mathcal{L}(\theta) \geq \mathcal{L}(c) + K(\theta - c), \forall \theta \leq c;$$
$$\mathcal{L}(\theta) \geq \mathcal{L}(c) - K(\theta - c), \forall \theta \geq c,$$

where $c = \frac{a+b}{2}$ denotes the center point of interval $[a, b]$.

Here, we assume the Lipschitz constant K is still known, and we will introduce the approach for unknown Lipschitz constant scenarios later. As illustrated in Figure 3, these two inequalities actually correspond to the two lines of slops $-K$ and $+K$ forming a "Λ" structure below the function curve. Values of these two lines at points a and b will define the lowest value of them within the interval $[a, b]$, i.e.,

$$\text{lower bound} = \mathcal{L}(c) - K\frac{b-a}{2}. \tag{37}$$

The interval division for the DIRECT algorithm is slightly different from the Shubert-Piyavskii algorithm, which partitions the interval into 3 sub-intervals instead.

Example 2 *For instance, given the interval before division in Figure 5, the* DIRECT *algorithm partitions it into 3 segments as illustrated in the plot after division. Among all the potential intervals, the* DIRECT *algorithm will pick the interval with a smaller lower-bound (as defined in Equ 37) to explore. In many cases the strict Lipschitz constant K is hard to compute in advance, and the* DIRECT *algorithm allows us to use a rate-of-change constant $\tilde{K}$ to replace it instead.*

Suppose we have partition the original interval $[a, b]$ into a group of sub-intervals $\{[a_i, b_i]\}_{i=1,2,\cdots,m}$ with the corresponding midpoints $\{c_i\}_{i=1,2,\cdots,m}$. Let $\mathcal{L}_{min}$ denote the current best function

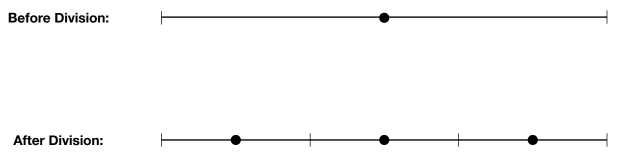


Figure 4: A Example for Division in the Direct Algorithm.

Algorithm 3 One-Dimensional Direct Algorithm

```
Require: Variable range [a, b]; Rate-of-change parameter $\tilde{K}$; Constant $\epsilon$.
Ensure: Model Parameter $\theta$
 1: Evaluate $\mathcal{L}(c)$ at the midpoint of intervan $[a, b]$.
 2: Initialize set $\mathcal{S} = \{[\mathcal{L}(c) : (a, b; c)]\}$.
 3: Initialize $\mathcal{L}_{min} = \mathcal{L}(c)$.
 4: while Stop criteria is not met. do
 5:     Select the optimal $[a_i, b_i]$ according to Inequalities (38).
 6:     Compute $\mathcal{S}$ size $m = |\mathcal{S}|$.
 7:     $\mathcal{S} = \mathcal{S} \setminus \{[\mathcal{L}(c_i) : (a_i, b_i; c_i)]\}$.
 8:     Compute $\delta = (b_i - a_i)/3$.
 9:     Update the interval $[a_i, b_i] = [a_i + \delta, a_i + 2\delta]$.
10:     Create two new intervals $[a_{m+1}, b_{m+1}] = [a_i, a_i + \delta]$ and $[a_{m+2}, b_{m+2}] = [a_i + 2\delta, b_i]$.
11:     Evaluate $\mathcal{L}(c_{m+1})$ and $\mathcal{L}(c_{m+2})$ at the midpoints $c_{m+1}$ and $c_{m+2}$.
12:     Update $\mathcal{L}_{min} = \min(\mathcal{L}_{min}, \mathcal{L}(c_{m+1}), \mathcal{L}(c_{m+2}))$.
13:     Update set with the updated/new intervals $\mathcal{S} = \mathcal{S} \cup \{[\mathcal{L}(c_i) : (a_i, b_i; c_i)], [\mathcal{L}(c_{m+1}) : (a_{m+1}, b_{m+1}; c_{m+1})], [\mathcal{L}(c_{m+2}) : (a_{m+2}, b_{m+2}; c_{m+2})]\}$.
14: end while
```

value and ϵ denote a positive constant. Interval $[a_j, b_j], j \in \{1, 2, \cdots, m\}$ will be selected selected to explore next iff there exists some rate-of-change constant $\tilde{K}$ such that

$$\mathcal{L}(c_j) - \tilde{K}\frac{b_j - a_j}{2} \leq \mathcal{L}(c_i) - \tilde{K}\frac{b_i - a_i}{2}, \forall i = 1, 2, \cdots, m; \tag{38}$$

$$\mathcal{L}(c_j) - \tilde{K}\frac{b_j - a_j}{2} \leq \mathcal{L}_{min} - \epsilon|\mathcal{L}_{min}|. \tag{39}$$

For the above inequalities, the first one indicates that interval $[a_j, b_j]$ to be on the lower right of the convex hull of the dots, and the second condition requires the selected interval to be better than the current best solution by a nontrivial amount. The pseudo-code of the one-dimensional Direct algorithm is provided in Algorithm 3.

3.2.2 Multi-dimensional Direct Algorithm

In this part, we will introduce the multi-dimensional Direct algorithm for computing the solution to the multivariate objective function. To simplify the problem settings, we assume

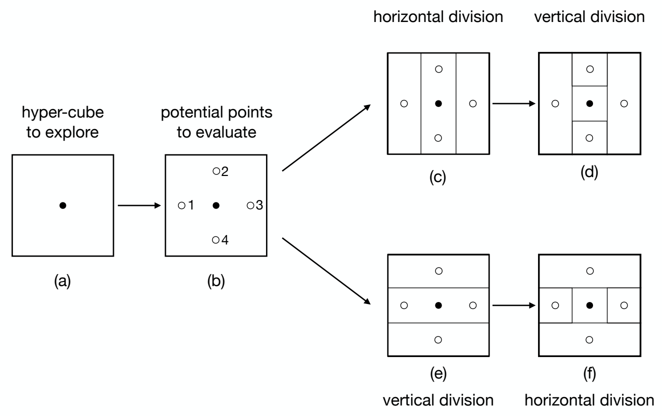


Figure 5: A Example of DIRECT in 2D Division.

every variable is within a pre-specified bound $[0, 1]$, which can be achieved by variable rescaling easily. In other words, the search space of the problem with be a n-dimensional unit hyper-cube instead, which will be partitioned into hyper-rectangles by the DIRECT algorithm in the learning process with a sample point in each of them. The main challenge to be studied here will be how to partition the hyper-cube for the multi-dimensional DIRECT algorithm.

The division process in the multi-dimensional DIRECT Algorithm is different from the division process in the uni-dimensional DIRECT Algorithm. Formally, let $\mathbf{c}$ denote the center point of the current hyper-cube to be explored, whose dimension length can be denoted as 3δ (i.e., one-third of the dimension length will be δ). Therefore, the potential points to be evaluated here can be denoted as $\{\mathbf{c} \pm \delta \cdot \mathbf{e}_i\}_{i=1}^{n}$, where vector $\mathbf{e}_i$ contains 1 at the i_{th} entry and 0s at the remaining entries.

Example 3 *For instance, we provide an example in Figure 5, where plot (a) is the hyper-cube to be explored. Based on the dimension length as well as the central solid point, we can represent the set of points to be evaluated as set $\{1, 2, 3, 4\}$, i.e., the 4 hollow dots in plot (b). Dots 1 and 3 are the ones sampled from the horizontal dimension, while dots 2 and 4 are sampled from the vertical dimension instead.*

Figure 5 also illustrate two different ways to partition the hyper-cube into smaller regions:

1. Plots (c) and (d) illustrate the partition of the hyper-cube in the horizontal dimension first, and then partition it along the vertical dimension.

2. Plots (e) and (f) illustrate the partition of the hyper-cube along the vertical dimension first, and then along the horizontal dimension.

To determine which division is more promising, the strategy adopted in the DIRECT algorithm is to make the biggest rectangles contain the best function values. Among all the

Algorithm 4 Multi-Dimensional Direct Algorithm

```
Require: Variable range hyper-rectangle; Rate-of-change parameter $\tilde{K}$; Constant $\epsilon$.
Ensure: Model Parameter $\boldsymbol{\theta}$
 1: Normalize the search space to be a unit hyper-cube.
 2: Evaluate $\mathcal{L}(\mathbf{c})$ at the center point of the hyper-cube.
 3: Initialize set $\mathcal{S} = \{[\mathcal{L}(\mathbf{c}) : \mathbf{c}]\}$.
 4: Initialize $\mathcal{L}_{min} = \mathcal{L}(\mathbf{c})$.
 5: Initialize hyper-rectangle number $m = 1$.
 6: while stop criteria is not met. do
 7:     Identify the set of optimal hyper-rectangles $\mathcal{S}' \subset \mathcal{S}$.
 8:     for hyper-rectangle centered at $\mathbf{c}_j$ in $\mathcal{S}'$ do
 9:         Identify the set $\mathcal{I}$ of dimensions of the maximum side length.
10:         Evaluate the function at points $\{\mathbf{c} \pm \mathbf{e}_i\}_{i \in \mathcal{I}}$.
11:         Sort the dimensions in $\mathcal{I}$ according to the weights $w_i = \min\{\mathcal{L}(\mathbf{c} + \delta\mathbf{e}_i), \mathcal{L}(\mathbf{c} - \delta\mathbf{e}_i)\}$ for
            dimension $i$ computed according to Equation 40.
12:         Divide the hyper-rectangle containing $\mathbf{c}$ into thirds along the dimensions in $\mathcal{I}$, starting with
            the dimension with the lowest $w_i$ and the so forth.
13:     end for
14:     Update $\mathcal{L}_{min}$ and $m$ according to the points in the newly sampled hyper-rectangles.
15: end while
```

potential dimensions $i = \{1, 2, \cdots, n\}$, the Direct algorithm will compute a weight w_i for each dimension as follows:

$$w_i = \min\{\mathcal{L}(\mathbf{c} + \delta\mathbf{e}_i), \mathcal{L}(\mathbf{c} - \delta\mathbf{e}_i)\}, \tag{40}$$

which denotes the best function value along dimension i.

The optimal dimension i^* will be selected to divide based on the weight values $\{w_i\}_{i=1,2,\cdots,n}$, i.e.,

$$i^* = \arg \min_{i \in \{1,2,\cdots,n\}} w_i. \tag{41}$$

Meanwhile, the selection of the optimal hyper-rectangles for the multivariate Direct algorithm is very similar to the one-dimensional version. Formally, let's assume we have partition the original hyper-cube into m hyper-rectangles and we can denote all the current hyper-rectangles available with their centers, i.e., $\{\mathbf{c}_i\}_{i=1,2,\cdots,m}$. A hyper-rectangle centered at $\mathbf{c}_j$ is said to be optimal if there exists some $\tilde{K}$ such that

$$\mathcal{L}(c_j) - \tilde{K} d_j \leq \mathcal{L}(c_i) - \tilde{K} d_i, \forall i = 1, 2, \cdots, m; \tag{42}$$

$$\mathcal{L}(c_j) - \tilde{K} d_j \leq \mathcal{L}_{min} - \epsilon |\mathcal{L}_{min}|, \tag{43}$$

where d_j denotes the distance from the center $\mathbf{c}_j$ to the vertices of the j_{th} hyper-rectangle.

Based on the above descriptions, we can provide the pseudo-code of multivariate Direct in Algorithm 4. As indicated in [6], the Direct algorithm is guaranteed to converge to the global optimal if the objective function is continuous, or at least continuous in the neighborhood of a global optimum.

3.3 LIPO Algorithm

The global optimization algorithms we have introduced in this section, including both the Shubert-Piyavskii algorithm and the Direct algorithm, is shown to converge subject to the local smoothness assumption. However, by this context so far, such convergence properties of these algorithms have not been considered in the scenarios where only the global smoothness assumption on the function can be specified. In this part, we will introduce another Lipschitz function based global optimization algorithm, namely LIPO [8]. LIPO effectively exploits the global smoothness of the unknown function, and is shown to converge faster on globally smooth problems than the other existing Lipschitz function based global optimization algorithms.

3.3.1 LIPO with Known Lipschitz Constant

Similar to the Shubert-Piyavskii algorithm and the Direct algorithm, LIPO is also a sequential algorithm, whose optimization process involves a sequence of sampled variable points and the function evaluations at these points. Formally, given the variable search space Θ, we can represent these sampled points as a sequence $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_n)$, where $\boldsymbol{\theta}_i$ denotes the point uniformly sampled from Θ at iteration $i \in \{1, 2, \cdots, n\}$. The LIPO algorithm will decide whether or not to evaluate the function at the point subject to certain criteria.

Before we talk about the evaluation criteria, we need to introduce several important concepts as follows, including *consistent function* and *potential maximizer*. Formally, give the Lipschitz constant $K \geq 0$, we can represent the set of functions subject to the corresponding Lipschitz function inequality (i.e., Formula 30) as $Lip(K)$, namely the K-Lipschitz functions. Based on the sequence of sampled points together with the function evaluations, a subset of the functions in $Lip(K)$ can be identified as the *consistent functions*.

Definition 1 *(Consistent Functions): The active subset of the K-Lipschitz functions consistent with the unknown function $\mathcal{L}(\cdot)$ over a sequence of n sampled points and function evaluations $[(\boldsymbol{\theta}_1, \mathcal{L}(\boldsymbol{\theta}_1)), (\boldsymbol{\theta}_2, \mathcal{L}(\boldsymbol{\theta}_2)), \cdots, (\boldsymbol{\theta}_n, \mathcal{L}(\boldsymbol{\theta}_n))]$ ($n \geq 1$) is defined as follows*

$$\mathcal{F}_{K,n} = \{g \in Lip(K) : \forall i \in \{1, 2, \cdots, n\}, g(\boldsymbol{\theta}_i) = \mathcal{L}(\boldsymbol{\theta}_i)\}. \tag{44}$$

Based on the above definition, the functions in set $\mathcal{F}_{K,n}$ can achieve consistent values with the evaluation of $\mathcal{L}(\cdot)$ at these sampled points in the search space. Considering that function $\mathcal{L}(\cdot)$ is unknown, its optimal variable can be potentially identified with the function candidates in set $\mathcal{F}_{K,n}$.

Definition 2 *(Potential Maximizers): Given the sampled points and function evaluation sequence $[(\boldsymbol{\theta}_1, \mathcal{L}(\boldsymbol{\theta}_1)), (\boldsymbol{\theta}_2, \mathcal{L}(\boldsymbol{\theta}_2)), \cdots, (\boldsymbol{\theta}_n, \mathcal{L}(\boldsymbol{\theta}_n))]$ together with the set of consistent functions $\mathcal{F}_{K,n}$, we can represent the set of potential maximizers as*

$$\Theta_{K,n} = \left\{ \boldsymbol{\theta} \in \Theta : \exists g \in \mathcal{F}_{K,n} \text{ such that } \boldsymbol{\theta} \in \arg\max_{\boldsymbol{\theta}' \in \Theta} g(\boldsymbol{\theta}') \right\}. \tag{45}$$

The set $\Theta_{K,n}$ defines the space that function $\mathcal{L}(\cdot)$ can achieve the maximum values. Given any variable $\boldsymbol{\theta}$, the notation "$\boldsymbol{\theta} \in \Theta_{K,n}$" has an equivalent representation according to the following Lemma 3.

Algorithm 5 LIPO Algorithm

```
Require: Variable search space $\Theta$; Lipschitz constant $K$; Constant $n$.
Ensure: Model Parameter $\boldsymbol{\theta}$
 1: Initialization: Sample $\boldsymbol{\theta}_1 \sim \mathcal{U}(\Theta)$ from $\Theta$ uniformly.
 2: Evaluate function value $\mathcal{L}(\boldsymbol{\theta}_1)$.
 3: Add $(\boldsymbol{\theta}_1, \mathcal{L}(\boldsymbol{\theta}_1))$ to the sampling sequence.
 4: Initialize index tag $t = 2$
 5: while $t \leq n$ do
 6:     Sample $\boldsymbol{\theta} \sim \mathcal{U}(\Theta)$ from $\Theta$ uniformly.
 7:     if $\min_{i \in \{1,2,\cdots,t\}} (\mathcal{L}(\boldsymbol{\theta}_i) + K \cdot \|\boldsymbol{\theta} - \boldsymbol{\theta}_i\|_2) \geq \max_{i' \in \{1,2,\cdots,t\}} \mathcal{L}(\boldsymbol{\theta}_{i'})$ then
 8:         Assign $\boldsymbol{\theta}_t = \boldsymbol{\theta}$.
 9:         Evaluate function $\mathcal{L}(\boldsymbol{\theta}_t)$.
10:         Add $(\boldsymbol{\theta}_t, \mathcal{L}(\boldsymbol{\theta}_t))$ to the sampling sequence.
11:         $t = t + 1$
12:     end if
13: end while
14: Return $\boldsymbol{\theta}^* = \arg\max_{i \in \{1,2,\cdots,n\}} \mathcal{L}(\boldsymbol{\theta}_i)$
```

Lemma 3 *Let $\Theta_{K,n}$ denote the set of potential maximizer defined above, we can achieve the following equivalent representation for any variable $\boldsymbol{\theta} \in \Theta_{K,n}$:*

$$\boldsymbol{\theta} \in \Theta_{K,n} \Leftrightarrow \min_{i \in \{1,2,\cdots,n\}} (\mathcal{L}(\boldsymbol{\theta}_i) + K \cdot \|\boldsymbol{\theta} - \boldsymbol{\theta}_i\|_2) \geq \max_{i' \in \{1,2,\cdots,n\}} \mathcal{L}(\boldsymbol{\theta}_{i'}). \tag{46}$$

Proof We will prove the above Lemma from two directions:

- *Direction 1*: $\boldsymbol{\theta} \in \Theta_{K,n} \Rightarrow \min_{i \in \{1,2,\cdots,n\}} (\mathcal{L}(\boldsymbol{\theta}_i) + K \cdot \|\boldsymbol{\theta} - \boldsymbol{\theta}_i\|_2) \geq \max_{i' \in \{1,2,\cdots,n\}} \mathcal{L}(\boldsymbol{\theta}_{i'})$.

  Given that variable $\boldsymbol{\theta} \in \Theta_{K,n}$, we can have $\boldsymbol{\theta} \in \arg\max_{\boldsymbol{\theta}' \in \Theta} g(\boldsymbol{\theta}')$ for some function $g \in \mathcal{F}_{K,n}$. Considering that g is a K-Lipschitz function, we can have

  $$|g(\boldsymbol{\theta}_1) - g(\boldsymbol{\theta}_2)| \leq K \cdot \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_2, \forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \Theta. \tag{47}$$

  Here, let's assign $\boldsymbol{\theta}_1 = \arg\max_{\boldsymbol{\theta}' \in \Theta} g(\boldsymbol{\theta}')$ and $\boldsymbol{\theta}_2 = \arg\min_{i \in \{1,2,\cdots,n\}} (g(\boldsymbol{\theta}_i) + K \cdot \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_i\|_2)$, we have

  $$g(\boldsymbol{\theta}_1) \geq g(\boldsymbol{\theta}_2) \tag{48}$$

  and according to Inequality 47, we can get

  $$\max_{i' \in \{1,2,\cdots,n\}} g(\boldsymbol{\theta}_{i'}) \leq g(\boldsymbol{\theta}_1) \leq g(\boldsymbol{\theta}_2) + K \cdot \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_2. \tag{49}$$

  Considering that function $g \in \mathcal{F}_{K,n}$, function evaluation $g(\boldsymbol{\theta}_i) = \mathcal{L}(\boldsymbol{\theta}_i), \forall i \in \{1, 2, \cdots, n\}$. In other words, we have

  $$\max_{i' \in \{1,2,\cdots,n\}} \mathcal{L}(\boldsymbol{\theta}_{i'}) \leq \min_{i \in \{1,2,\cdots,n\}} \mathcal{L}(\boldsymbol{\theta}_i) + K \cdot \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_i\|_2, \tag{50}$$

  where $\boldsymbol{\theta}_1 \in \Theta_{K,n}$.

- *Direction 2*: $\min_{i \in \{1,2,\cdots,n\}} \left(\mathcal{L}(\boldsymbol{\theta}_i) + K \cdot \|\boldsymbol{\theta} - \boldsymbol{\theta}_i\|_2\right) \geq \max_{i' \in \{1,2,\cdots,n\}} \mathcal{L}(\boldsymbol{\theta}_{i'}) \Rightarrow \boldsymbol{\theta} \in \Theta_{K,n}$. According to the previous analysis, we know that for any $\boldsymbol{\theta} \in \Theta$, we have

$$\mathcal{L}(\boldsymbol{\theta}) \leq \min_{i \in \{1,2,\cdots,n\}} \left(\mathcal{L}(\boldsymbol{\theta}_i) + K \cdot \|\boldsymbol{\theta} - \boldsymbol{\theta}_i\|_2\right). \tag{51}$$

In other words, the terms on the right-hand-side actually defines the upper bound of potential values that function $\mathcal{L}(\cdot)$ can achieve at point $\boldsymbol{\theta}$, which can be formally defined as follows:

$$UB(\boldsymbol{\theta}) = \min_{i \in \{1,2,\cdots,n\}} \left(\mathcal{L}(\boldsymbol{\theta}_i) + K \cdot \|\boldsymbol{\theta} - \boldsymbol{\theta}_i\|_2\right). \tag{52}$$

Meanwhile, the space denoted by the following inequality actually defines the set of potential variable $\boldsymbol{\theta}$ which can achieve the maximum value for the function $\mathcal{L}(\cdot)$:

$$\Theta_{\text{UB}} = \left\{\boldsymbol{\theta} : \boldsymbol{\theta} \in \Theta, UB(\boldsymbol{\theta}) \geq \max_{i' \in \{1,2,\cdots,n\}} \mathcal{L}(\boldsymbol{\theta}_{i'})\right\}. \tag{53}$$

Therefore, $\forall \boldsymbol{\theta} \notin \Theta_{K,n}$, we can have $\boldsymbol{\theta} \notin \Theta_{\text{UB}}$ (i.e., $\forall \boldsymbol{\theta} \in \Theta_{\text{UB}} \Rightarrow \boldsymbol{\theta} \in \Theta_{K,n}$), which proves the Direction 2.

■

The inequality $\min_{i \in \{1,2,\cdots,n\}} \left(\mathcal{L}(\boldsymbol{\theta}_i) + K \cdot \|\boldsymbol{\theta} - \boldsymbol{\theta}_i\|_2\right) \geq \max_{i' \in \{1,2,\cdots,n\}} \mathcal{L}(\boldsymbol{\theta}_{i'})$ mentioned in the above Lemma 3 can serve as the criteria for function evaluation in the sampling process of LIPO, and the pseudo-code of LIPO in Algorithm 5.

3.3.2 AdaLIPO with Unknown Lipschitz Constant

In many cases, the Lipschitz constant is hard to know in advance. In this part we will introduce the AdaLipo (Adaptive LIPO) algorithm for optimizing function $\mathcal{L} \in \cup_{k \geq 0} Lip(k)$, whose specific Lipschitz constant is unknown.

The pseudo-code of AdaLipo is provided in Algorithm 6. Compared with LIPO, AdaLipo don't need unknown Lipschitz constant K, but will take two more parameters: (1) Bernoulli distribution parameter p, and (2) Lipschitz constant meshgrid $K_{i \in \mathbb{Z}}$. Here, the notation $K_{i \in \mathbb{Z}}$ defines a meshgrid of $\mathbb{R}^+$ such that $\forall x > 0, \exists i \in \mathbb{Z}$ with $K_i \leq x \leq K_{i+1}$. The AdaLipo algorithm starts with an initialized Lipschitz constant $\hat{K} = 0$, and proceeds with the exploration by sampling the variables from either Θ or $\Theta_{\hat{K}_t, t}$, which is controlled via a Bernoulli distribution variable $B_t \sim \mathcal{B}(p)$. According to the algorithm pseudo-code, AdaLipo extends LIPO by adding a step to infer the potential Lipschitz constant $\hat{K}_t$ (i.e., Line 16 in Algorithm 6) in each step iteratively.

Besides the inference technique introduced in Algorithm 6, some other Lipschitz constant inference approaches proposed in [2, 17] can be used as well to implement the algorithm. To be consistent with [8], we will not introduce them here and the readers may refer to the cited papers for more information.

Algorithm 6 AdaLipo Algorithm

```
Require: Variable search space $\Theta$; Constant $n$; Bernoulli distribution parameter $p$, Lipschitz constant meshgrid $K_{i \in \mathbb{Z}}$.
Ensure: Model Parameter $\boldsymbol{\theta}$
 1: Initialization: Sample $\boldsymbol{\theta}_1 \sim \mathcal{U}(\Theta)$ from $\Theta$ uniformly.
 2: Evaluate function value $\mathcal{L}(\boldsymbol{\theta}_1)$.
 3: Add $(\boldsymbol{\theta}_1, \mathcal{L}(\boldsymbol{\theta}_1))$ to the sampling sequence.
 4: Initialize inferred Lipschitz constant $\hat{K}_2 = 0$.
 5: Initialize index tag $t = 2$
 6: while $t \leq n$ do
 7:     Sample variable $B_t \sim \mathcal{B}(p)$ from the Bernoulli distribution.
 8:     if $B_t = 1$ then
 9:         Sample $\boldsymbol{\theta}_t \sim \mathcal{U}(\Theta)$ from $\Theta$.
10:     else
11:         Sample $\boldsymbol{\theta}_t \sim \mathcal{U}(\Theta_{\hat{K}_t, t})$ from $\Theta_{\hat{K}_t, t}$ (which denotes the set of potential optimizers computed with $\hat{K}_t$ and $t$).
12:     end if
13:     Evaluate function $\mathcal{L}(\boldsymbol{\theta}_t)$.
14:     Add $(\boldsymbol{\theta}_t, \mathcal{L}(\boldsymbol{\theta}_t))$ to the sampling sequence.
15:     $t = t + 1$
16:     Update $\hat{K}_i = \inf \left\{ K_{i \in \mathbb{Z}} : \max_{i \neq j} \frac{|\mathcal{L}(\boldsymbol{\theta}_i) - \mathcal{L}(\boldsymbol{\theta}_j)|}{\|\boldsymbol{\theta}_i - \boldsymbol{\theta}_j\|_2} \leq K_i \right\}$.
17: end while
18: Return $\boldsymbol{\theta}^* = \arg\max_{i \in \{1,2,\cdots,n\}} \mathcal{L}(\boldsymbol{\theta}_i)$
```

3.4 MCS Algorithm

The MCS algorithm [5] to be introduced here is a heuristic-based optimization algorithm, and it has a close relationship with the Direct algorithm introduced before. In Direct, the search space will be normalized to a hyper-cube $[0,1]^{d_\theta}$, which will be partitioned into smaller hyper-rectangles in the searching process. Each hyper-rectangle is characterized by its center-point, and the dimension length of the hyper-rectangles will be always $3^{-k}, k \in \mathbb{N}$. The Direct algorithm may suffer from several disadvantages: (1) Direct cannot handle the search space with infinite bounds; and (2) Direct cannot reach the boundary and may lead to slow learning process if the minimum lies at the boundaries. The MCS algorithm to be introduced in this part can resolve such these problems.

3.4.1 Overview of MCS

Here, we will provide the overview of MCS, whose detailed information will be provided as follows. Similar to the Shubert-Piyavskii algorithm and Direct, MCS also combines the global search with local search, where "global search" denotes the division of hyper-rectangles with large unexplored territory and "local search" denotes the division of hyper-rectangles with the optimal function values (corresponding to the two terms indicated in Equations 42 and 37). MCS adopts a multi-level search strategy to balance between global search and local search, where each hyper-rectangle will be assigned with a level index $s \in \{0, 1, \cdots, s_{max}\}$. Here, level $s = 0$ denotes the hyper-rectangle has been divided and will be ignored for further division, and level $s = s_{max}$ denotes the hyper-rectangle is too small for further division. Meanwhile, for the hyper-rectangles with a level $0 < s < s_{max}$, after division, their level will be set to be $s = 0$ but their descendants will get level $s+1$ or $\min(s+2, s_{max})$. Viewed in such a perspective, the hyper-rectangle level s will denote its size effectively, and the ones with a small level will have a larger size (except level 0).

After the initialization step, the MCS algorithm will *sweep* through all the available hyper-rectangles with levels $0 < s < s_{max}$ from the lowest levels to the higher ones (which corresponds to the global part in MCS). For the hyper-rectangles belonging to the same level, MCS will select the ones with the lowest function evaluation values, which corresponds to the local part in MCS. For the finally selected hyper-rectangles, MCS will divide them along one single coordinate (i.e., one single variable) in one step, and the coordinate as well as the division point are determined by the information from the sampled points. Different from Direct, the division points lies at the boundaries of the hyper-rectangle in MCS, which allows MCS to resolve the second disadvantage of Direct as mentioned before. With multiple times of division, the hyper-rectangles will become arbitrarily small and the hyper-rectangles will shrink sufficiently fast.

MCS without local search will put the sampled points together with their function evaluations of hyper-rectangles of level s_{max} into a basket (containing the "useful" points). Meanwhile, the MCS with local search will start a local search from these points before putting them into the basket so as to accelerate the convergence of the algorithm. MCS with local search will first check whether the point in the current hyper-rectangle is in the basin of attraction of a local minimizer or not. If it is not the case, MCS will build a quadratic model by triple searches, and identify minimum of the quadratic function with a line search along a promising direction.

For the terms and techniques mention above, we will provide a much more detailed description in the following subsections in great detail.

3.4.2 Initialization

In MCS, the sampled points lie at the boundaries of the hyper-rectangles, which may belong to multiple hyper-rectangles (which share the same boundary or vertices). These sampled points are also called the *base points*. In MCS, besides these base points, each hyper-rectangle is also assigned with an *opposite point*, which together with the corresponding *base point* can precisely identify the hyper-rectangle. Let the initial point in the hyper-rectangle as $\boldsymbol{\theta}^0$. MCS starts the initialization step by selecting a set of initial base points (at least three points) along some coordinate, e.g., coordinate $i \in \{1, 2, \cdots, d_\theta\}$ corresponding to variable $\boldsymbol{\theta}_i$, which can be denoted as $\{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \cdots, \boldsymbol{\theta}^{l_i}\}$. These points share the same coordinates with the initial point $\boldsymbol{\theta}^0$ for all the other coordinates except the i_{th} coordinate, and their i_{th} coordinate takes the following values respectively

$$a_i \leq \boldsymbol{\theta}_i^1 < \boldsymbol{\theta}_i^2 < \cdots < \boldsymbol{\theta}_i^{l_i} \leq b_i, \tag{54}$$

where $[a_i, b_i]$ denote the lower/upper bounds along the i_{th} dimension in the search space. The objective function will be evaluated at these selected points, which can form the pairs $\left\{\left(\boldsymbol{\theta}^1, \mathcal{L}(\boldsymbol{\theta}^1)\right), \left(\boldsymbol{\theta}^2, \mathcal{L}(\boldsymbol{\theta}^2)\right), \cdots, \left(\boldsymbol{\theta}^{l_i}, \mathcal{L}(\boldsymbol{\theta}^{l_i})\right)\right\}$. Among these points, the one achieving the optimal function evaluation will be denoted as $\boldsymbol{\theta}^*$.

Besides these l_i points, an extra group of $l_i - 1$ division points will be selected, which can be denoted as $\{\mathbf{z}^2, \mathbf{z}^3, \cdots, \mathbf{z}^{l_1}\}$. These points also share the same coordinates with $\boldsymbol{\theta}^0$

except the i_{th} dimension, which take value

$$\mathbf{z}_i^l = \boldsymbol{\theta}_i^{l-1} + q^k \left(\boldsymbol{\theta}_i^l - \boldsymbol{\theta}_i^{l-1} \right), l \in \{2, 3, \cdots, l_i\}. \tag{55}$$

Here, in the above equation term $q = \frac{\sqrt{5}-1}{2}$ denotes the golden section ratio, and $k \in \{1, 2\}$ is chose to ensure the parts with smaller function evaluations can get larger intervals.

Depending how many points in $\{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \cdots, \boldsymbol{\theta}^{l_i}\}$ lie at the lower/upper boundaries, these points together with the above extra division points $\{\mathbf{z}^2, \mathbf{z}^3, \cdots, \mathbf{z}^{l_i}\}$ will divide the search space into $2l_i - 2$ (if $a_i = \boldsymbol{\theta}_i^1$ and $\boldsymbol{\theta}_i^{l_i} = b_i$), $2l_i - 1$ (if $a_i = \boldsymbol{\theta}_i^1$ or $\boldsymbol{\theta}_i^{l_i} = b_i$), and $2l_i$ (if $a_i < \boldsymbol{\theta}_i^1$ and $\boldsymbol{\theta}_i^{l_i} < b_i$) sub-hyper-rectangles. In addition, $\{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \cdots, \boldsymbol{\theta}^{l_i}\}$ will serve as the base points of these new hyper-rectangles, which will also be used to update $\boldsymbol{\theta}^*$. If $\boldsymbol{\theta}^*$ is not shared by multiple hyper-rectangles, then the one containing $\boldsymbol{\theta}^*$ will be picked as the "current" hyper-rectangle for the next round of division. Otherwise, MCS will select the sub-hyper-rectangle to divide for the next dimension in the initialization with a quadratic interpolation. By sampling 3 random consecutive points selected from $\{\boldsymbol{\theta}^1, \boldsymbol{\theta}^2, \cdots, \boldsymbol{\theta}^{l_i}\}$, MCS fits a quadratic model with these 3 points. For the hyper-rectangles sharing $\boldsymbol{\theta}^*$, the one containing the minimizer of the quadratic model will be picked as the "current" hyper-rectangle for the next dimension.

Such a process continues for $i = 1, 2, \cdots, d_\theta$ so as to finish the initialization step.

3.4.3 Sweep

After initialization, MCS continues with a sweep process across all the hyper-rectangles according to the following three steps:

- For all the hyper-rectangles with level $s \in (0, s_{max})$, MCS introduces a pointer p_s indicating the hyper-rectangle belonging to level s with the smallest function evaluation value. If there is no hyper-rectangle of level s, then MCS sets $p_s = 0$.

- According to the rules to be introduced in the following subsection, for level s, if the hyper-rectangle pointed by p_s is not divided, MCS will increase its level by 1 and update b_{s+1} if necessary. If the hyper-rectangle pointed by b_s is divided, MCS will mark it as divided and insert its children to higher levels. MCS will update the list and the pointers if any children hyper-rectangles introduce a smaller function evaluations.

- Increase s by 1. If $s = s_{max}$, start a new sweep; else if $s = 0$, go to Step 3; else go to step 2.

The sweep step outlines the general architecture of the MCS algorithm, and next we will introduce the division rules in the following subsection.

3.4.4 Hyper-rectangle Division

Given a candidate hyper-rectangle of level $s < s_{max}$ for division, the division rules include the following two categories:

1. *Division by Rank*: Let n_i denotes the times coordinate i has been divided in the past for the hyper-rectangle, we can denote $n_{min} = \min_{i \in \{1,2,\cdots,d_\theta\}} n_i$ as the minimum division count for all the coordinates in the search space. If

   $$s > 2d_\theta(n_{min} + 1), \tag{56}$$

   the hyper-rectangle will always be divided, and the division dimension index will be $i = \arg\min_{i \in \{1,2,\cdots,d_\theta\}} n_i$. In other words, for the hyper-rectangles which have been divided for many times (with a relatively large level s), there may exist some coordinates which haven't been divided very often and MCS will choose to divide these coordinates.

   For the selected coordinate i, (1) if $n_i = 0$ (i.e., the coordinate hasn't beed divided at all), the division rules will be very similar to the initialization division process (involving the selection of l_i sample points as well as the division points at the golden section). Meanwhile, (2) if $n_i > 0$, MCS will select two division points to divide the hyper-rectangle into three parts. In MCS, to store the hyper-rectangle information, besides the base point $\boldsymbol{\theta}$, another *opposite point* $\mathbf{o}$ will be stored as well, which denotes one of the corners of the hyper-rectangle farthest away from $\boldsymbol{\theta}$. Therefore, the range to be divided along the i_{th} dimension will be $[\boldsymbol{\theta}_i, \mathbf{o}_i]$ and the selected division points include: (1) $\mathbf{z}_i = \boldsymbol{\theta}_i + \frac{2}{3}(\xi'' - \boldsymbol{\theta}_i)$, and (2) the golden section division point. Here, term

   $$\xi'' = subint(\boldsymbol{\theta}_i, \mathbf{o}_i) = \begin{cases} sign(\mathbf{o}_i), & \text{if } 1000|\boldsymbol{\theta}_i| < 1, |\mathbf{o}_i| > 1000; \\ 10 sign(\mathbf{o}_i)|\boldsymbol{\theta}_i|, & \text{if } 1000|\boldsymbol{\theta}_i| < 1, |\mathbf{o}_i| > 1000|\boldsymbol{\theta}_i|; \\ \mathbf{o}_i, & \text{otherwise.} \end{cases} \tag{57}$$

   Therefore, with these two points, the hyper-rectangle will be divided into 3 parts with one more function evaluation at $\mathbf{z}$ (a point with identical coordinates with θ except coordinate i with $\mathbf{z}_i$), which will also serve as the base point for the second and third sub-hyper-rectangles. The smaller fraction of the golden section split gets level $\min(s + 2, s_{max})$ and the other two get level $s + 1$.

2. *Division by Expected Gain*: If

   $$s \leq 2d_\theta(n_{min} + 1), \tag{58}$$

   the hyper-rectangle may be divided along a coordinate where the maximum function value gain can be achieved according to a local quadratic model obtained by fitting $2d_\theta + 1$ function values. Meanwhile, if the function expected gain is not large enough, MCS will increase its level by 1 without any division.

   If in the history of the current hyper-rectangle, coordinate i has never been divided, i.e., $n_i = 0$, then MCS will divide the hyper-rectangle along the i_{th} dimension according to the initialization step as introduced before. Meanwhile, if $n_i > 0$, MCS will retrieve the closest two points to the base point in the current hyper-rectangle for each coordinate $i \in \{1, 2, \cdots, d_\theta\}$. By fitting a quadratic model with these 3 points, MCS builds the following function

   $$e_i(\xi) = \alpha(\xi - \boldsymbol{\theta}_i) + \beta(\xi - \boldsymbol{\theta}_i)^2. \tag{59}$$

Within interval $[\xi', \xi'']$ (ξ'' is defined in Equation 57, and $\xi' = \boldsymbol{\theta}_i + (\xi'' - \boldsymbol{\theta}_i)/10$), MCS will identify the point achieving the minimum as the potential division point, which can be denoted as

$$\mathbf{z}_i = \arg \min_{\xi \in [\min(\xi', \xi''), \max(\xi', \xi'')]} e_i(\xi). \tag{60}$$

MCS will divide the hyper-rectangle along dimension i at $\mathbf{z}_i$ if the following condition holds:

$$\mathcal{L}(\boldsymbol{\theta}) + \min_{i \in \{1, 2, \cdots, d_\theta\}} \mathcal{L}(\mathbf{z}) < \mathcal{L}^*, \tag{61}$$

where $\mathcal{L}^*$ denotes the current best function value (including the function values obtained by local optimization). Otherwise, MCS will increase the current hyper-rectangle level by 1 without division.

In addition to the three steps mentioned above, MCS also adopts a local search step, which provides powerful tools for the task of optimization of a smooth function when knowledge about the gradient or even the Hessian is known. Meanwhile, when there exists no information about the derivative information, successive line searches can be adopted. In this paper, we will not introduce the local search step adopted in MCS, and the readers may refer to [5] for more information.

4. Summary

In this paper, we have introduce two categories of derivative-free optimization algorithms, including the Bayesian methods and Lipschitzian approaches. Many of these introduced algorithms can be potentially applied to learn the deep neural network models. This tutorial paper will also be updated accordingly as we observe more new developments on this topic in the near future.

References

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

[2] Sébastien Bubeck, Gilles Stoltz, and Jia Yuan Yu. Lipschitz bandits without the lipschitz constant. In *Proceedings of the 22Nd International Conference on Algorithmic Learning Theory*, ALT'11, pages 144–158, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] D. D. Cox and S. John. A statistical method for global optimization. In *[Proceedings] 1992 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1241–1246 vol.2, Oct 1992.

[4] Efim A Galperin. The cubic algorithm. *Journal of Mathematical Analysis and Applications*, 112(2):635–640, 1985.

[5] Waltraud Huyer and Arnold Neumaier. Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14(4):331–355, Jun 1999.

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

[7] 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+, 1964.

[8] Cédric Malherbe and Nicolas Vayatis. Global optimization of lipschitz functions. In *ICML*, 2017.

[9] Regina Hunter Mladineo. An algorithm for finding the global maximum of a multimodal, multivariate function. *Mathematical Programming*, 34(2):188–200, Mar 1986.

[10] Jonas Mockus. On bayesian methods for seeking the extremum. In *Proceedings of the IFIP Technical Conference*, pages 400–404, London, UK, UK, 1974. Springer-Verlag.

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

[12] J. Pintér. Globally convergent methods for n-dimensional multiextremal optimization. *Optimization*, 17(2):187–202, 1986.

[13] S. A. Piyavskii. An algorithm for finding the absolute extremum of a function. *USSR Computational Mathematics and Mathematical Physics*, 12(4):57–67, 1972.

[14] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[15] B. Shubert. A sequential method seeking the global maximum of a function. *SIAM Journal on Numerical Analysis*, 9(3):379–388, 1972.

[16] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Gaussian process bandits without regret: An experimental design approach. *CoRR*, abs/0912.3995, 2009.

[17] G. R. Wood and B. P. Zhang. Estimation of the lipschitz constant of a function. *Journal of Global Optimization*, 8(1):91–103, Jan 1996.

[18] Jiawei Zhang. Derivative-free optimization algorithms: Population based optimization algorithms and random search algorithms, 2019.

[19] Jiawei Zhang. Gradient descent based optimization algorithms for deep learning models training, 2019.

[20] Jiawei Zhang and Fisher B. Gouza. GADAM: genetic-evolutionary ADAM for deep neural network optimization. *CoRR*, abs/1805.07500, 2018.