# A Trust Region Based Local Bayesian Optimization without Exhausted Optimization of Acquisition Function

**Qingxia Li**
Dongguan City College
**Anbing Fu**
Dongguan University of Technology
**Wenhong Wei** (weiwh@dgut.edu.cn)
Dongguan University of Technology
**Yuhui Zhang**
Dongguan University of Technology

**Research Article**

**Keywords:** Trust Region, Bayesian Optimization, Gaussian Process, UCB, Local Optimization

**Posted Date:** March 9th, 2022

**DOI:** https://doi.org/10.21203/rs.3.rs-1429011/v1

# A Trust Region Based Local Bayesian Optimization without Exhausted Optimization of Acquisition Function

**Qingxia Li**[1], **Anbing Fu**[2], **Wenhong Wei**[2] **and Yuhui Zhang**[2]

[1]School of Computer and Information, Dongguan City College, Dongguan, 523419, China
[2]School of Computer, Dongguan University of Technology, Dongguan, 523808, China

Corresponding author: Wenhong Wei (e-mail: weiwh@dgut.edu.cn).

**ABSTRACT** Bayesian optimization (BO) is an effective optimization technique for solving expensive black-box problems. Even though BO has remarkable success, its drawbacks are also obvious. First, the time complexity of the Gaussian process inference is higher than O($n^3$), where $n$ is the number of samples. Consequently, the running time of BO increases rapidly with the problem size. Second, due to the non-convexity and multimodality of the acquisition function, it costs a lot to achieve good results. To address the above problems, we develop a local Bayesian optimization algorithm based on the trust region idea (TRLBO). In TRLBO, two trust regions with dynamically changing sizes are used to enhance the algorithm's exploitation ability, while at the same time retaining the exploration ability. Specifically, one trust region is used to reduce the number of samples in the Gaussian process. The other is used to restrict the solution space of the candidates. Furthermore, some theoretical results were provided to enlighten the efficiency of the proposed algorithm. Experimental results on both benchmark functions and real-world problems show that TRLBO compares favorably with the state-of-the-art algorithms.

**INDEX TERMS** Trust Region, Bayesian Optimization, Gaussian Process, UCB, Local Optimization

## 1. Introduction

Many problems encountered in scientific research and engineering are expensive and non-convex, whose mathematical formation and the gradient information are usually not available. For example, one common problem in machine learning is to find the optimal parameters of neural networks so that they can perform well in certain tasks. The problem is posed as an optimization problem, and its mathematical formation is difficult to construct. In addition, evaluating one set of parameter values is very time-consuming, so traditional optimization algorithms are not suitable for this type of problem. In comparison, Bayesian optimization (BO) is specifically designed for black-box function optimization and very effective. In recent years, BO has been successfully applied to interactive user-interfaces, robotics, environmental monitoring, information retrieval, combinatorial optimization, automatic machine learning, sensor networks, adaptive Monte Carlo, experimental design, reinforcement learning, and so on [1].

The success of Bayesian optimization is attributed to two components: the probabilistic surrogate for modeling the objective function, and the acquisition function for balancing the exploration and exploitation abilities. The Gaussian process is currently the default choice of surrogates, and the time complexity of Gaussian process inference is $O(n^3)$, where $n$ is the number of samples. Therefore, Bayesian optimization is mainly used to solve the low dimension problems. However, recent years have witnessed the rapid growth in the computational power of GPUs. A more efficient algorithm that reduces the time complexity to O($n^2$) is developed by resorting the computational power of GPUs [2]. The development of the new algorithm makes it possible for BO to tackle high-dimensional problems.

But there are still challenges to overcome before BO can be successfully extended to the high-dimensional problems. First, due to the curse of dimensionality, the search space grows exponentially as the dimensionality of the objective function increases. Moreover, Brochu et al. [3] found that the tendency toward exploration is more salient when BO is handling higher dimensional problems, which leads to its poor performance. This is the major barrier that limits the applicability of BO. In the literature, many attempts have been made to extend BO to high-dimensional problems. Assuming that the objective function is in D-dimension and there are only $d$ active variables, where $d \ll D$, Chen et al. [4] used a hierarchical diagonal sampling method to perform both variable selection and objective optimization. Rolland [5] decomposed a high-dimensional objective function into multiple low-dimensional functions and used a graph to represent the dependency between the low-dimensional functions. An efficient information transfer algorithm is developed to optimize the acquisition function. Furthermore, Gibbs sampling method was employed to learn the structure of the dependency graph. Following the work of Rolland, Mutny et al.[6] proposed a quadrature Fourier features method to approximate the exponential square covariant function. It is proven that the approximation error will decrease exponentially as the number of features increase.

Besides the works mentioned above, there exist other approaches to extend BO to high-dimensional space [7-15]. Most of these methods assume the objective function has some additive structures. Then, different additive structures are adopted to train different Gaussian processes, so it becomes very time-consuming to draw a larger number of samples. Moreover, one obstacle lying in the way of applying BO to highdimensional problems is that the acquisition function is non-convex and difficult to optimize. Some researchers tried to circumvent the obstacle by making use of stochastic feature approximation [16]. In contrast, McIntire *et al.* [17] use a sparse Gaussian process as the probabilistic surrogate. More recently, Wang [18] proposed an ensemble Bayesian optimization algorithm (EBO) that resolves the problems of drawing a large number of samples, increasing dimensionality of objective functions, and balancing the diversity and accuracy of samples. Wang [18] also revealed the relationship between EBO and evolutionary algorithms (EAs), which brings innovation to the algorithm analysis. Eriksson et al. [19] developed a trust region local Bayesian optimization algorithm (TuRBO). The numerical results reported in [19] show that TuRBO can reach state-of-the-art performance when tackling high-dimensional problems, in which case a large number of samples are required. This paper follows this promising research avenue and makes further progresses. The differences between our approach and the existing work are as follows.

(1) During the iteration of TuRBO, all the sample-observation data is used to train the Gaussian process. In our approach, we extract part of the data to train the Gaussian process.

(2) Thompson sampling is adopted in TuRBO as the acquisition function. In our approach, the Gaussian process upper confidence bound (GP-UCB) is used instead.

(3) In TuRBO, one trust region is constructed to control the samples' solution space size. In our approach, the solution space size and the number of samples in the Gaussian process are controlled by two trust regions.

(4) We use a simple yet efficient way to extend GP-UCB so to draw samples in batches.

The rest of the paper is organized as follows. Section 2 gives a brief introduction to the Bayesian optimization algorithm, as well as the trust region method. The proposed TRLBO algorithm is described in detail under Section 3. Section 4 is devoted to the algorithmic analysis of TRLBO. The numerical experiments are conducted in Section 5. Finally, Section 6 discusses the remaining issues and concludes the paper.

## 2. Background

### 2.1. *Bayesian optimization*

Given an optimization problem: $\min f(x),\ x \in \Omega \subset \mathbb{R}^{\mathrm{D}}$, where $\Omega$ is a compact subset of $\mathbb{R}^{\mathrm{D}}$, $f\colon \Omega \to \mathbb{R}$ is a deterministic black box function whose mathematical formulation, gradient information, and the property (convexity) are unknown. Moreover, evaluating the function is very expensive. Without making any assumptions to the objective function, the only way to find the optimum is brute force search. It might happen that only small changes are made to the decision variables, but the objective function value changes dramatically, e.g., the Dirichlet function. In practice, we often introduce several assumptions to make the optimization problem more tractable. One common assumption is Lipschitz continuity: If for any $x \in \mathcal{P}, y \in \mathcal{P}$, there exists a constant $\ell$ such that $\|f(x) - f(y)\| \leq \ell\|x - y\|$, then $f$ is termed Lipschitz continuous in region $\mathcal{P}$. DIRECT [20] is an algorithm built upon the assumption. DIRECT constantly divides the solution space and abandons subspaces that do not contain the optimum. In this way, the search space is gradually reduced, and the algorithm can eventually find the optimum. However, in the process of reducing the search space size, numerous function evaluations are required. Therefore, the algorithm is not suitable for the above scenario. Different from the DIRECT algorithm, BO is a stochastic optimization technique. Although it cannot guarantee to find the global optimum in each run, it is able to find an ideal solution with a relatively small number of function evaluations. A more detailed description of BO can be found in [1], [21]. BO is composed of a probabilistic surrogate model and an acquisition function. In the following, we briefly introduce these two components.

The probabilistic surrogate model is used for modeling the objective function. One of the most commonly seen surrogate models is the Gaussian process. It is also used in our study. Gaussian process $\mathrm{GP}\big(\mu(x), k(x,x')\big)$ is a non-parametric model determined by its mean function $\mu(x)$ and covariant function $k(x,x')$. We assume that the objective function is sampled from a Gaussian process with mean 0 ($\mathrm{GP}\big(0, k(x,x')\big)$), namely, $f(x) \sim \mathrm{GP}\big(0, k(x,x')\big)$. The observed data is denoted by $D_n = \{x_t, y_t\}_{t=1}^{n}$, where $y_t = f(x_t) + \epsilon_t$ and $\epsilon_t$ are independent identically distributed variables drawn from Gaussian distribution, namely, $\epsilon_t \sim N(0, \sigma^2)$. According to the property of the Gaussian distribution, it is easy to deduce that the posterior distribution is also a Gaussian distribution whose mean and covariance are formulated as:

$$\mu_n(x) = \kappa_n(x)^{\mathrm{T}}(K_n + \sigma^2 I)^{-1} y_n \qquad (2.1)$$

$$\sigma_n^2(x) = \kappa(x, x) - \kappa_n(x)^{\mathrm{T}}(K_n + \sigma^2 I)^{-1} \kappa_n(x) \qquad (2.2)$$

where $K_n = \big[\kappa(x_i, x_j)\big]_{x_i, x_j \in D_n}$, $\kappa_n(x) = [\kappa(x_i, x)]_{x_i \in D_n}$. Please refer to [22] for more information about the Gaussian process. Modeling an objective function with a Gaussian process involves the matrix inversion operation, which has a time complexity of $O(n^3)$. This precludes the application of BO to scenarios where large numbers of samples are required. In the Gaussian process, the covariance function determines the smooth property of the sampling function, as well as the type it can fit. To increase the applicability of the Gaussian process, a common practice is to add some hyperparameters to the covariance function. During the training process, the hyperparameters can be tuned to make the surrogate model closer to the real objective function. One of the most important hyperparameters is the scale parameter. It is a vector that shares the same dimensionality as the input of the objective function. It is used to determine the degree of importance of the input variables. In our study, the scale parameter is also used to determine the corresponding trust region of the solution space of the acquisition function. Some commonly seen covariance functions are exponential covariance function, exponential square covariance function, and Martern covariance function.

The acquisition function is adopted to guide the selection of promising points for evaluation. There are different types of acquisition functions available for use, e.g, probability of improvement (PI) [23], expected improvement (EI) [24], Thompson search [25], entropy search [26-27], predictive entropy search, (PES) [28], knowledge gradient (KG) [29], Gaussian process upper confidence bound (GP-UCB) [30], and so on. GP-UCB is developed based on the idea of upper confidence bound (UCB) algorithm [31]. Although its formation is very simple, it can effectively strike a balance between exploration and exploitation. GP-UCB is formulated as follows:

$$a_{\mathrm{ucb}}(x; D_n) = \mu_n(x) + \beta_n \sigma_n(x) \qquad (2.3)$$

where $\mu_n(x)$ and $\sigma_n(x)$ denote the mean and variance of the posterior distribution respectively. The hyperparameter $\beta_n$ is used for balancing exploration and exploitation. Intuitively, the algorithm tends to explore when $\beta_n$ is large. Conversely, the algorithm tends to exploit when $\beta_n$ is small.

In Bayesian optimization, we optimize the function to generate new evaluation points. In real-world applications, we hope that multiple promising points can be evaluated simultaneously in each iteration to fully use the computational power. Many studies have been conducted in this research avenue [32-38]. In this paper, to endow GP-UCB with the capability of batch sampling, we discard the exhausted optimization of the acquisition function. A random search algorithm is adopted instead. For the hyperparameter $\beta_n$ of GP-UCB, Srinivas derived its exact expression and proved its effectiveness from a theoretical perspective. However, the result is based on the premise that GP-UCB is optimized through some exact approach. This differs from the situation we face. To set $\beta_n$ to a value that effectively balances exploration and exploitation, we normalize $\mu_n(x)$ and $\sigma_n(x)$. The details are presented in section 3.

## 2.2. *Trust region algorithm*

Generally speaking, it is very difficult to globally fit a non-convex function. However, if we concentrate on a local region of the objective function at a time, it is easy to fit the region with a linear or quadratic function. The basic idea of the trust region algorithm is to use an approximation model to fit the objective function locally. The accuracy of the approximation model in the local region is trustable, and therefore the local region is termed "trust region". Trust region is usually a sphere or polytope centered at the current best solution, and its size is dynamically adjusted in each iteration. In short, the trust region algorithm uses a merit function to evaluate the fitting performance of the approximation model. If the approximation model fits the objective function very well, then the trust region grows. Conversely, the trust region shrinks. Instead of directly optimizing the objective function, the principle of the trust region algorithm is to optimize a simple local approximation model of the objective function. Quadratic approximation is a widely used approximation model. Using quadratic approximation, the original problem is reduced to the following constraint optimization problem.

$$\min m_k(d) = f_k + g_k^T d + \frac{1}{2} d^T B_k d \tag{2.4}$$

$$s.t. \|d\|_2 \leq \Delta_k \tag{2.5}$$

where $f_k$, $g_k^T$, $B_k$ denote the objective function value, the gradient, and the Hessian approximation matrix at point $x_k$ respectively. The notation $\Delta_k$ represents the radius of the trust region. It can be seen that the objective function of the transformed problem is a convex quadratic function. This function is easy to optimize. Assuming that $s_k$ is the solution to the transformed problem, we use the following evaluation function $r_k$ to determine the point $x_{k+1}$ and the trust region radius $\Delta_{k+1}$ of the next iteration.

$$r_k = \left(f(x_k) - f(x_k + s_k)\right) / \left(m_k(d) - m_k(s_k)\right) \tag{2.6}$$

The trust region algorithm repeats the above process until it meets the predefined termination criterion. The algorithm has a good convergence property. For a detailed description of the trust region algorithm, please refer to [39]. As in TuRBO [19], we use the Gaussian process to fit the objective function locally. However, instead of using all the data to train the Gaussian process, we use part of the data lies within the trust region.

## 3. Trust Region based Local Bayesian Optimization (TRLBO)

In this section, the proposed TRLBO is described in detail. Same as BO, our algorithm is composed of a surrogate model and an acquisition function. In TRLBO, the local Gaussian process serves as the surrogate model while the local GP-UCB serves as the acquisition function. Each component is associated with a trust region. The sizes of the two trust regions are controlled by a scale parameter. The two regions are different in shapes. The one associated with the local Gaussian process has a ball shape, while the other associated with the local GP-UCB is a hyper-rectangle. The details are presented in the following subsections.

### 3.1. *Local Gaussian Process*

The surrogate model plays an important role in Bayesian optimization. A suitable surrogate model can effectively fit the objective function and contribute to the good performance of the algorithm. Conversely, the algorithm will perform poorly if the surrogate model differs significantly to the objective function. Gaussian process is used for modeling functions, and it has a nice property: all joint distribution, marginal distribution, and posterior distribution of finite variables are Gaussian distribution. Therefore, the Gaussian process is the most commonly seen surrogate model in BO and has achieved good results in real-world applications. The performance of BO is largely attributed to the global Gaussian process that accurately fits the objective function. When handling low dimensional problems, a small number of samples are sufficient to train the global Gaussian process. However, when handling high-dimensional problems, a larger number of samples are required. This lowers the efficiency of the algorithm. Inspired by the trust region algorithm, we abandon the attempt to fit the entire objective function and try to fit a local region, this approach is similar to TuRBO [20]. The major difference is that TuRBO uses the entire data set to train the Gaussian process. Only when optimizing the acquisition function, it restricts the solution space to a trust region. In contrast, we extract part of the data to train the local Gaussian process. Specifically, denote the observed data as $D_n = \{(x_t, y_t)\}_{t=1}^n$, the scale parameter of the trust region as $\ell$, and the current best solution as $x_{opt}$. The superscript $n$ represents the $n$th iteration of the algorithm. We use the following formula to extract point set $D_n'$ from $D_n$ to train the local Gaussian process.

$$D_n' = \left\{(x, y) \mid \left\|x_{opt} - x\right\|_2 \leq \eta\ell, (x, y) \in D_n\right\} \tag{3.1}$$

In the formula, $\eta = \mathrm{Max}\,\lambda_\mathrm{i}, i \in \{1,2\ldots d\}$, $\lambda_\mathrm{i}$ is the scale hyperparameter of the Gaussian process, and $d$ is the dimension of the objective function input. The parameter $\eta$ is used to guarantee that the trust region associated with the local Gaussian process covers the trust region associated with the acquisition function. The scale parameter $\ell$ of the trust region gradually decreases as the number of iterations grows. Therefore, during the training process, the number of samples used the local Gaussian process will be much smaller than that in the global Gaussian process. This way, the running time is greatly reduced. In addition, according to our experiment, the local Gaussian process has similar or even better fitting quality than the global Gaussian process.

```
Algorithm 1: Local Gaussian process algorithm
Input: $D_n = \{(x_t, y_t)\}_{t=1}^{n}$: The observed data: $\ell$: The
       scale parameter of the trust region; $\eta$: The scaling
       factor; $x_{opt}$: The current best point
1: $n$ is the number of points in $D_n$, $D_n'$ is the valid points
   set.
2: for $i$ = 1 to $n$ do
3:     if $\|x_{opt} - x_i\| \leq \eta\ell$ then
4:         $D' = D' \cup \{x_i, y_i\}$
5:     end if
6: end for
7: use the update data $D'$ to train the Gaussian process
Output: The Posterior of the Multivariate Gaussian
        distribution
```

### 3.2. *Local GP-UCB*

In the previous subsection, by employing the trust region method, we have successfully applied the local Gaussian process to the local fitting of the objective function. To force the algorithm to pay more attention to local exploitation, we need to restrict the solution space of the newly generated samples. In other words, we hope that the next batch of samples can be generated near the current best solution. This is the effect of the trust region associated with the acquisition function. The approach used in [19] is adopted here to determine the trust region associated with the acquisition function. Let $\ell$ denote the scale parameter of the trust region, $\lambda$ denote the length-scale hyperparameter of the local Gaussian process, and $x_{opt}$ denote the current best solution. Then, the trust region is a hyper-rectangle centered at $x_{opt}$, with volume $\ell^d$. For each dimension, the side length of the hyper-rectangle is computed as $L^{(i)} = \lambda^{(i)} \ell / \left(\prod_{j=1}^{n} \lambda^{(j)}\right)^{1/d}$, where $d$ is the dimension of the input variables of the objective function. We have restricted the solution space of the new samples to the hyper-rectangle centered at the current best solution. Let $\Omega$ denote the solution space formed by the hyper-rectangle. We now illustrate the method used to generate the samples for evaluation.

The GP-UCB formulated in (2.3) is used in our algorithm as the acquisition function. The original Bayesian optimization algorithm equipped with GP-UCB can only generate one candidate in each iteration. We use a simple method to extend the algorithm, so that multiple candidate points can be generated at a time. Specifically, we generate a point set $\{x_i\}_{i=1}^{n}$ by uniformly sampling points from the solution space $\Omega$ of the acquisition function $a_{\mathrm{ucb}}(x; D_n)$. Then, we substitute it into the mean $\mu_n(x)$ and variance $\sigma_n(x)$ of the posterior distribution of the acquisition function and obtain the corresponding function value $\{\mu_i\}_{i=1}^{n}$ and $\{\sigma_i\}_{i=1}^{n}$. Subsequently, we use max-min normalization to normalize the posterior mean and variance. The normalized mean and variance are denoted by $\{\mu_i'\}_{i=1}^{n}$ and $\{\sigma_i'\}_{i=1}^{n}$ respectively. Finally, the acquisition function value is obtained by $\{a_i \mid a_i = \mu_i' + \beta_n \sigma_i'\}_{i=1}^{n}$. In order to generate $m (m > 1)$ candidate points at a time, we sort $\{a_i\}_{i=1}^{n}$ in ascending order and select the first $m$ acquisition function values. The points (represented by the input variables $x_i$) corresponding to the $m$ function values are collected to form the point set $\{x_i\}_{i=1}^{m}$ used for next evaluation. By uniformly sampling points in the restricted solution space of the acquisition function, we not only avoid optimizing the complex acquisition function, but also generate a set of samples without much computational cost. The theoretical analysis of the method is given in Section 4.

```
Algorithm 2: Optimizing GP-USB algorithm
Input: $\mu_n(x)$: The Posterior mean function; $\sigma_n(x)$: The
       Posterior variance function; $\beta_n$: To balance
       exploration; $m$: The batch size; $n$: The uniform
       sampling size
1: Generate $n$ samples $\{x_i\}_{i=1}^{n}$ by uniformly sampling
   from the solution space $\Omega$, $A=\emptyset$, $B=\emptyset$.
2: for $i$ = 1 to $n$ do
3:     $A = A \cup \mu_n(x_i)$, $B = B \cup \sigma_n(x_i)$
4: end for
5: Let $L_A$, $U_A$, $L_B$, $U_B$ denote the minimal, maximum values
```

```
    of set $A$, $B$ respectively, $C=\emptyset$.
6: for $i = 1$ to $n$ do
7:      $A_i = \frac{A_i - L_A}{U_A - L_A}$, $B_i = \frac{B_i - L_A}{U_A - L_A}$ // $A_i$, $B_i$ is the $i$-th element
        of set $A$, $B$ respectively.
8:      $C = C \cup (A_i + \beta_n B_i)$
9: end for
10: we sort $C$ in ascending order and select the first m
    acquisition function values.
Output: $m$ points $\{x_i\}_{i=1}^{n}$ corresponding to the $m$
        acquisition function values.
```

### 3.3. *Outline of TRLBO*

In the previous two subsections, we introduce trust regions into the probabilistic surrogate model and the acquisition function of BO. However, there is still one important step to emphasize before completing our algorithm, that is, the adjustment strategy of trust regions. It is worth noting that the trust regions associated with the local Gaussian process and the acquisition function are controlled by the same scale parameter $\ell$ and $\ell \in [\ell_{\min}, \ell_{\max}]$. The primitive trust region algorithm adjusts the trust region according to the merit function formulated in (2.6), which represents the ratio between the expected decrement and the actual decrement. In contrast, we use the stochastic model to locally approximate the objective function. Therefore, the adjustment strategy in [19] is adopted in TRLBO. In each iteration, among all samples, if one sample is better than the current best solution, then we termed it a successful trial. Otherwise, a failed trial is recorded. We define two threshold values $\tau_{\text{fail}}$ and $\tau_{\text{succ}}$ to trigger the process of adjusting the size of trust regions. If the number of successive failed trials reaches $\tau_{\text{fail}}$, we shrink the trust region by half: $\ell \leftarrow \ell/2$. Similarly, if the number of successive successful trials reaches $\tau_{\text{succ}}$, we enlarge the trust region by a factor of two: $\ell \leftarrow \min\{2\ell, \ell_{\max}\}$. The min operator is to guarantee that $r$ does not exceed the upper limit. Finally, all the procedures of our algorithm are shown as follows.

Algorithm 3: TRLBO algorithm

```
1: Initialize the trust region scale parameter $\ell$, initialize q
   evaluation points $\{x_i\}_{i=1}^{q}$, compute their objective
   function values and store the observation data $D = \{(x_i, y_i)\}_{i=1}^{q}$.
2: Train the local Gaussian process (Algorithm 1).
3: Generate $m$ evaluation points (Algorithm 2), calculate
   their objective function value $\{x_i\}_{i=1}^{m}$, and add the new
   observation data to the data set $D$, $D = D \cup \{(x_i, y_i)\}_{i=1}^{m}$.
4: Adjust the trust region length $\ell$ according to the
   optimization results of Step 3, if $\ell \le \ell_{\min}$, terminate the
   algorithm; otherwise, go to Step 2.
```

## 4. Theoretical Analysis of TRLBO

### 4.1. *Exploration vs. Exploitation*

Trust region algorithm is a local optimization algorithm. Although it cannot guarantee to find the global optimum, it has a global convergence property. We introduce the main idea of the trust region algorithm to BO. From a theoretical point of view, our algorithm possesses the local optimization property. In each iteration of our algorithm, it is easy to infer from the trust region adjustment strategy that the trust region length gradually decreases. This property forces our algorithm to pay more attention to local exploitation. Overall, we have deduced several theoretical results. For the sake of illustration, we first introduce some relevant notations. We map the domain of the objective function to the space $\Omega = [0,1]^d$. The current best solution of the algorithm is denoted by $x_{opt}$ and the scale parameter of the trust region is denoted by $\ell(\ell \le 1)$. $\lambda$ is the scale hyperparameter of the Gaussian process, and $D$ is the observed data set. Moreover, we define the following quantities.

$$\Omega_{gp} = \left\{(x, y) \mid \left\|x_{opt} - x\right\|_2 < \ell, (x, y) \in D\right\} \quad (4.1.1)$$

$$\Omega_{acq} = \left\{(x, y) \mid x^{(i)} \in [l_i, u_i], (x, y) \in D, i \in \{1,2\ldots d\}\right\} \quad (4.1.2)$$

$$l_i = x_{opt}^{(i)} - \frac{1}{2}\lambda^{(i)}\ell \Big/ \left(\prod_{j=1}^{n} \lambda^{(i)}\right)^{1/d} \quad (4.1.3)$$

$$u_i = x_{opt}^{(i)} + \frac{1}{2}\lambda^{(i)}\ell \Big/ \left(\prod_{j=1}^{n} \lambda^{(i)}\right)^{1/d} \quad (4.1.4)$$

**Proposition 1** Let y denote the event $\Omega_{gp} \neq \emptyset$ and x denote the event $\left\{(x_i, y_i) \mid (x_i, y_i) \in \Omega_{gp}, (x_i, y_i) \notin \Omega_{acq}\right\} \neq \emptyset$, then we have:

$$\lim_{r \to 0} \mathrm{P}(\mathrm{x}|\mathrm{y}) = 0$$

**Proof:** Note that the trust region associated with the local Gaussian process is a sphere centered at the current best solution $x_{opt}$, with radius equals to $\eta\ell$. Its volume is computed as $V = \pi^{d/2}(\eta\ell)^d/\Gamma(1+d/2)$. The trust region associated with the acquisition function is a hyper-rectangle with its side length equals to $L^{(i)} = \lambda^{(i)}\,\ell/\left(\prod_{j=1}^{n}\lambda^{(j)}\right)^{1/d}$, where $d$ is the dimension of the objective function. Note also that the domain of the objective function is $[0,1]^d$. When $L^{(i)} \geq 1$, we take $L^{(i)} = 1$. Let $A = \{1,2\ldots d\}$, we discuss two different cases.

1）For every $i \in A, L^{(i)} < 1$. In this case, the volume of the hyper-rectangle is $\ell^d$, and we assume that the observed point set $\Omega_{gp}$ used for training the local Gaussian process is not empty (there is no need to train if it is empty). Furthermore, for simplicity, we assume that the sample points not in $\Omega_{gp}$ are uniformly distributed in the space $\Omega$ (the real case will be more complex). The probability that there exists observed data not in $\Omega_{gp}$ is computed as:

$$P(x|y) = \frac{V-\ell^d}{1} = \left(\frac{\pi^{d/2}\eta^d}{\Gamma(1+d/2)} - 1\right)\ell^d \quad (4.1.5)$$

It is obvious that when $\ell \to 0$, we have $\lim_{r\to 0}\mathbb{P}(x|y) = \lim_{r\to 0}\left(\frac{\pi^{d/2}\eta^d}{\Gamma(1+d/2)} - 1\right)\ell^d = 0$

2）There exists $i \in A$, $L^{(i)} \geq 1$. If for every $i \in A$, $L^{(i)} \geq 1$, since $L^{(i)} = \lambda^{(i)}\,\ell/\left(\prod_{j=1}^{n}\lambda^{(j)}\right)^{1/d}$, we have $\prod_{i=1}^{d}L^{(i)} = \prod_{i=1}^{d}\left(\lambda^{(i)}\,\ell/\left(\prod_{j=1}^{n}\lambda^{(j)}\right)^{1/d}\right) = \ell^d \geq 1$, this leads to a contradiction. Therefore we can assume that there exists a set $B \notin \emptyset$, for every $i \in B \subset \{1,2\ldots d\}$, $L^{(i)} = 1$. At this point, the volume of the hyper-rectangle can be calculated as $\ell^{|A|-|B|}\prod_{i\in A-B}\lambda^{(i)}/\left(\prod_{j=1}^{n}\lambda^{(j)}\right)^{1/d}$. Let $\zeta = \prod_{i\in A-B}\lambda^{(i)}/\left(\prod_{j=1}^{n}\lambda^{(j)}\right)^{1/d}$ and $\alpha = |A|-|B|$ ($\zeta$ and $\alpha$ are constants), then $P(x|y) = (V-\zeta\ell^\alpha)/1 = \pi^{d/2}(\eta\ell)^d/\Gamma(1+d/2) - \zeta\ell^\alpha$. It is easy to learn that $\lim_{r\to 0}P(x|y) = \lim_{r\to 0}\pi^{d/2}(\eta\ell)^d/\Gamma(1+d/2) - \zeta\ell^\alpha = 0$.

According to the corollary, as the scale parameter of the trust region decreases, our algorithm behaves similarly to the primitive Bayesian optimization algorithm whose solution space shrinks iteration by iteration. This property forces our algorithm to pay more attention to local exploitation. Conversely, when the scale parameter of the trust region is not so small, the solution space does not entirely overlap with the trust region used for determining the training data of the local Gaussian process. This guarantees the exploration ability of the algorithm.

## 4.2. *Effect of using random search to optimize the acquisition function*

From the theoretical point of view, it is very difficult to find the global minimum (maximum) of a non-convex function. Therefore, in practice, we change our goal to finding a suboptimal solution. Some common algorithms for optimizing the acquisition function are DIRECT, multi-start quasi-Newton hill-climbing approach, CMA-ES, and multi-start local search. In our algorithm, we use random search to optimize the acquisition function. In the following, we analyze the efficiency of the approach. We first provide some basic definitions and illustrations.

**Definition 1** Let $x_{min} = \arg\min f(x), x \in \Omega$, $\Omega$ is a compact subset of $R^n$, $x$ is termed an $\epsilon -$optimal solution if it satisfies the following conditions:
There exists a hyper-cube $A = \{x|\ \|x\|_\infty = \epsilon\} \subset \Omega$, such that $x \in A$ and $x_{min} \in A$, namely $x_{min}$ and $x$ are in the same grid.

Now we regard the scale parameter of the trust region as a random variable $\mathcal{L}$, its corresponding value is $\ell$. Let $d$ denote the dimension of the input variables of the objective function. We divide the solution space of the new samples into $k = \lceil \ell^d/\epsilon^d \rceil$ grids with size $\epsilon^d$. When $\epsilon$ is sufficiently small, the dividing error can be omitted. Moreover, when using random search to optimize the acquisition function, the number of samples is set to $100d$. We use $m$ to represent the number. Supposing $y$ is the event that an $\epsilon -$optimal solution is obtained by using the sampling method, we have

$$P(y|\mathcal{L}=\ell) = 1 - \left(\frac{k-1}{k}\right)^m \approx 1 - \left(1 - \left(\frac{\epsilon}{\ell}\right)^d\right)^m \quad (4.2.1)$$

The probability is very small. However, $P(y|\mathcal{L}=\ell)$ will increase as $\ell \to \epsilon$. Next, we analyze the expected number of iterations the algorithm required to find a local optimal solution or successively find solutions better than the current best solution. We first assume that when $\ell$ reduces to a threshold value, say $\epsilon$, then the algorithm has found a local optimal solution. Otherwise, if the current solution is not a local optimum, when $\epsilon$ is sufficiently small, the probability that the random search finds a better solution is $p \approx 1/2$. When there are $100d$ sample points, the probability of finding a better solution becomes $q = 1-(1-p)^{100d}$. It can be inferred from the expression of $q$ that, when $p = 0.01, d = 10$, $q$ exceeds 0.99. Therefore, we can take the fact that the current solution is the local optimum.

Note that as the scale parameter $\ell$ of the trust region changes, it only takes two values, $\ell$ and $2\ell$. Therefore, when $\ell$ is changing within the interval $[\epsilon, 1]$, its choices are very limited. We can regard it as a random variable whose value are taken from a finite state space $\mathrm{S} = \{0,1,2 \ldots n\}$. We denote the random variable as $\{\mathcal{L}_i\}$. State $i (i \in \mathrm{S})$ indicates the case $\ell = 2^i \epsilon$. When $\epsilon$ is sufficiently small, $n$ is approximated as $n = \lceil log_2(1/\epsilon) \rceil$. For the convenience of analysis, we now take $\{\mathcal{L}_i\}$ as a random walking process. Let $\mathrm{P}_{i,i+1} = p$ denote the probability of moving a step right. Starting from state $j$, the probability that $\{\mathcal{L}_i\}$ reaches state $n$ before reaching state 0 is $\alpha(j)$. According to property of random walking, we have

$\alpha(j) = p\alpha(j+1) + (1-p)\alpha(j-1)$, $\mathrm{j} = 1,2, \ldots n-1$,

It can be simplified to

$$\alpha(j+1) - \alpha(j) = \frac{1-p}{p}\big(\alpha(j) - \alpha(j-1)\big) \quad (4.2.2)$$

Given that $\alpha(0) = 0$, we have,

$$\alpha(2) - \alpha(1) = \frac{1-p}{p}\big(\alpha(1) - \alpha(0)\big) = \frac{1-p}{p}\alpha(1) \quad (4.2.3)$$

$$\alpha(3) - \alpha(2) = \frac{1-p}{p}\big(\alpha(2) - \alpha(1)\big) = \left(\frac{1-p}{p}\right)^2 \alpha(1) \quad (4.2.4)$$

.
.
.

$$\alpha(n) - \alpha(n-1) = \left(\frac{1-p}{p}\right)^{n-1} \alpha(1) \quad (4.2.5)$$

Adding the above $n$-1 equations and substituting $\alpha(\mathrm{n}) = 1$, we can deduce that when $p \neq 1/2$,

$$\alpha(j) = \frac{1-\left(\frac{1-p}{p}\right)^j}{1-\left(\frac{1-p}{p}\right)^n} \quad (4.2.6)$$

when $p = 1/2$,

$$\alpha(j) = j/n \quad (4.2.7)$$

Let $B$ denote the number of steps required for $\{L_i\}$ to start from state $j$, and finally reach state 0 or $n$. We then compute the expected value of $B$, namely, $E(B|L_0 = j)$. Let $X_i \in \{-1,1\}, i \geq 1$ denote the outcome of $L$ after the $j$th change, and for $B$ we have,

$$B = \min\left\{m \middle| \sum_{i=1}^{m} X_i = -j \; or \sum_{i=1}^{m} X_i = n-j\right\} \quad (4.2.8)$$

1) when $p \neq 1/2$.

According to Wald equation, we can get

$$E\left(\sum_{i=1}^{B} X_i\right) = E(X_i) * E(B|L_0 = j) \quad (4.2.9)$$

Given $\mathrm{P}(\sum_{i=1}^{m} X_i = n-j) = \alpha(j)$, $\mathrm{P}(\sum_{i=1}^{m} X_i = -j) = 1-\alpha(j)$, $E(X_i) = 2p-1$, the expression becomes

$$\mathrm{E}\left(\sum_{i=1}^{B} X_i\right) = n\alpha(j) - j \quad (4.2.10)$$

Substitute the above formula into 4.2.9, the expected value of $B$ is given by

$$E(B|L_0 = j) = \frac{1}{2p-1}\left\{\frac{\left[1-\left(\frac{1-p}{p}\right)^j\right] * n}{1-\left(\frac{1-p}{p}\right)^n} - j\right\} \quad (4.2.11)$$

2) when $p = 1/2$.

Let $G(j) = E(B|L_0 = j)$, it is obvious that $G(0) = G(n) = 0$, and consider the case of one step of movement, we have

$$G(j) = 1 + \frac{1}{2}G(j+1) + \frac{1}{2}G(j-1) \quad (4.2.12)$$

(4.2.12) is an in-homogeneous linear difference equation. Solving the equation gives:

$$G(j) = E(B|L_0 = j) = j(n-j) \quad (4.2.13)$$

Note that the maximum value of $j(n-j)$ is $n^2/4$ and $n = \lceil log_2(1/\epsilon) \rceil$. Even $\epsilon$ takes a very small value, the value of $n$ is acceptable. Refer to [40] and [41] for more detailed descriptions of random walking.

From the above analysis it can be seen that even in the worst case, the number of iterations taken by our algorithm to find a local optimum is acceptable. In addition, according to the experimental results, given the same number of iterations, the running time spent by our algorithm is shorter than other algorithms which perform exact optimization of the acquisition function.

## 5. Numerical Experiment

In the experiment, three commonly used synthetic functions (i.e., Levy, Griewank, and Ackley) and two real-world problems (i.e., a 14D robot pushing problem and a 60D rover trajectory planning problem) are used to test the performance of algorithms. For the synthetic functions, three different numbers of dimensions (i.e., 10, 20 and 50) are tested. The main part of our algorithm is built upon the framework of TuRBO [19]. Besides the parameters of the algorithmic framework, we introduce two new parameters, the acquisition function parameter $\beta$ of GP-UCB and the radius $r$ associated with the local Gaussian process. Their values are set to $d\ell$ and $||\lambda||_{\infty}\ell$ respectively, where $d$ is dimension of the input variables of the objective function and $\lambda$ is the scale hyper-parameter of the local Gaussian process. In addition, we generate new sample points in a batch manner and the batch size is fixed at 10. The number of the initial sample points is 20. See [19] for more detailed descriptions of the other parameters. To evaluate the performance of our algorithm, we compare it with TuRBO1, TuRBOm, CMA-ES, and RS (Random Search). TuRBO1, proposed by Eriksson [19], is a Bayesian optimization algorithm with a single trust region. The trust region is used to restrict the solution space of the acquisition function. TuRBOm extends TuRBO1 by increasing the number of trust regions to $m$. In our experiment, $m$=5. CMA-ES is one of the most successfully evolutionary strategy algorithms. It has exhibited good performance in real-world applications, especially when handling median-scale complex optimization problems. CMA-ES is a random search algorithm. Its key idea is to use a multivariable Gaussian distribution in the search space to guide the search direction, and adjust the mean and covariance matrix of the multivariable Gaussian distribution in an adaptive manner. In the experiment, we use the CMA-ES implementation provided in the pycma library. Random search algorithms are algorithms that focus solely on exploration. It does not require any information about the objective function and has a wide range of applications. It performs very well in some types of problems. In the experiment, we uniformly sample batches of points in the solution space, and the batch size is set to 10. We run the algorithms 30 times for each test function. All source code is available at https://github.com/agier9/TRLBO.

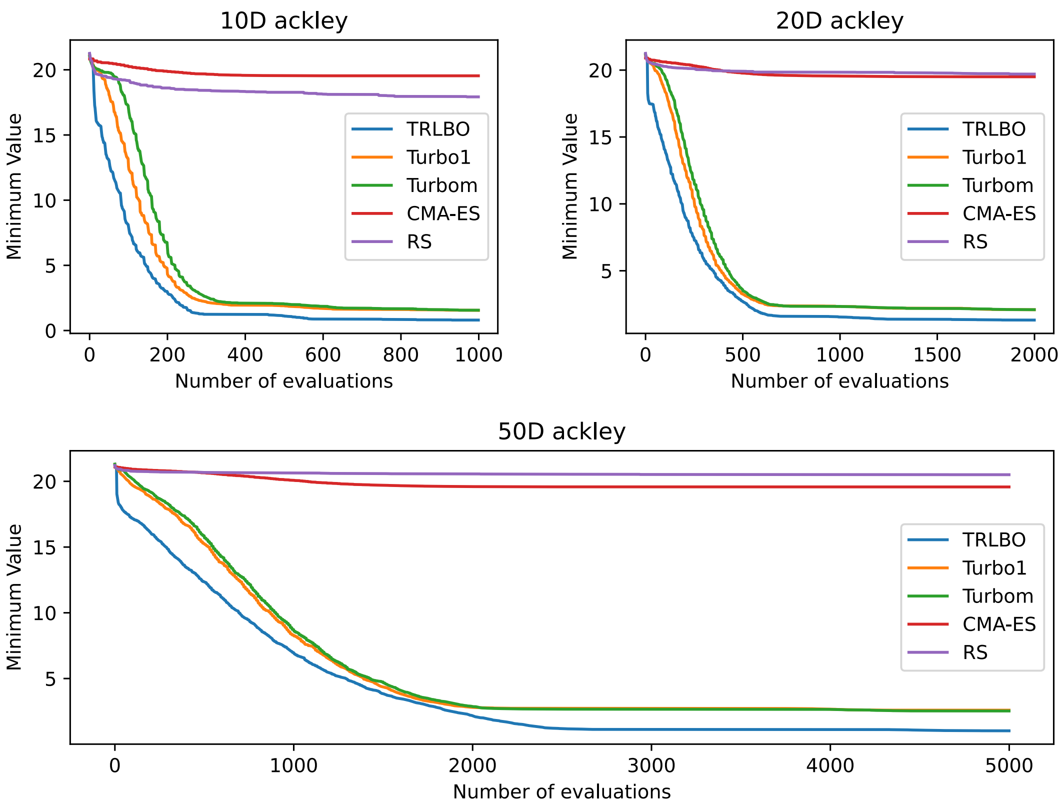

Fig. 1 10D, 20D, and 50D Ackley function

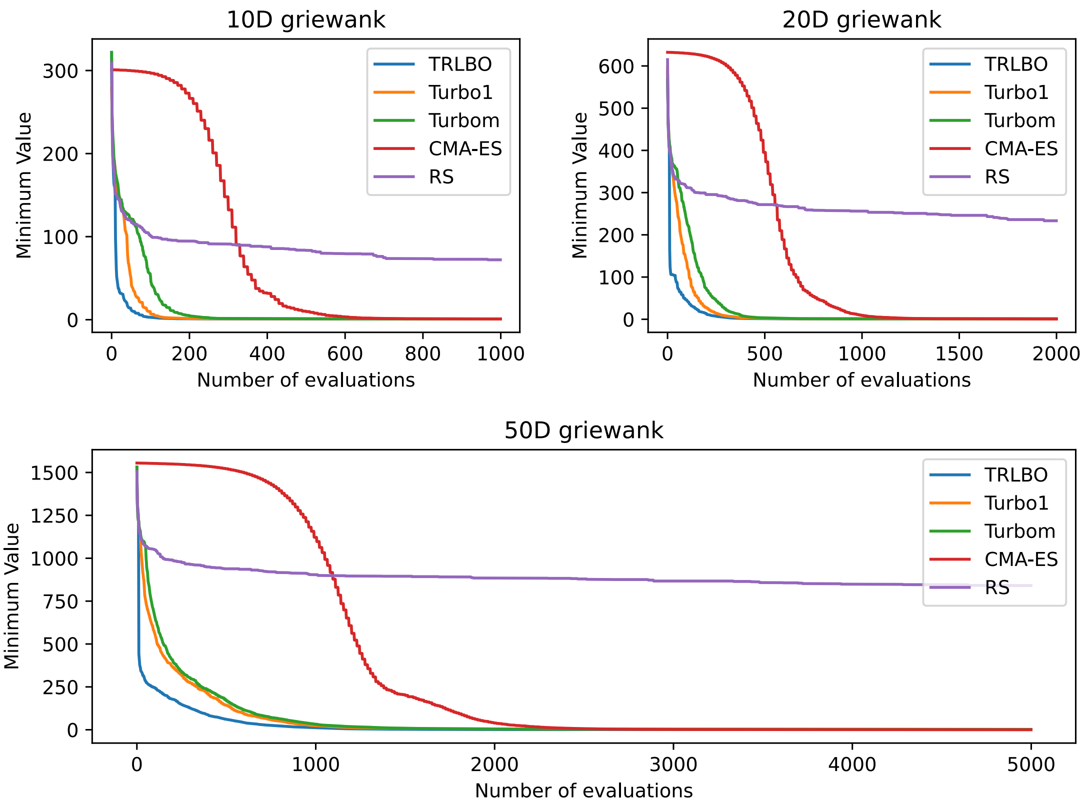

Fig. 2 10D, 20D, and 50D Griewank

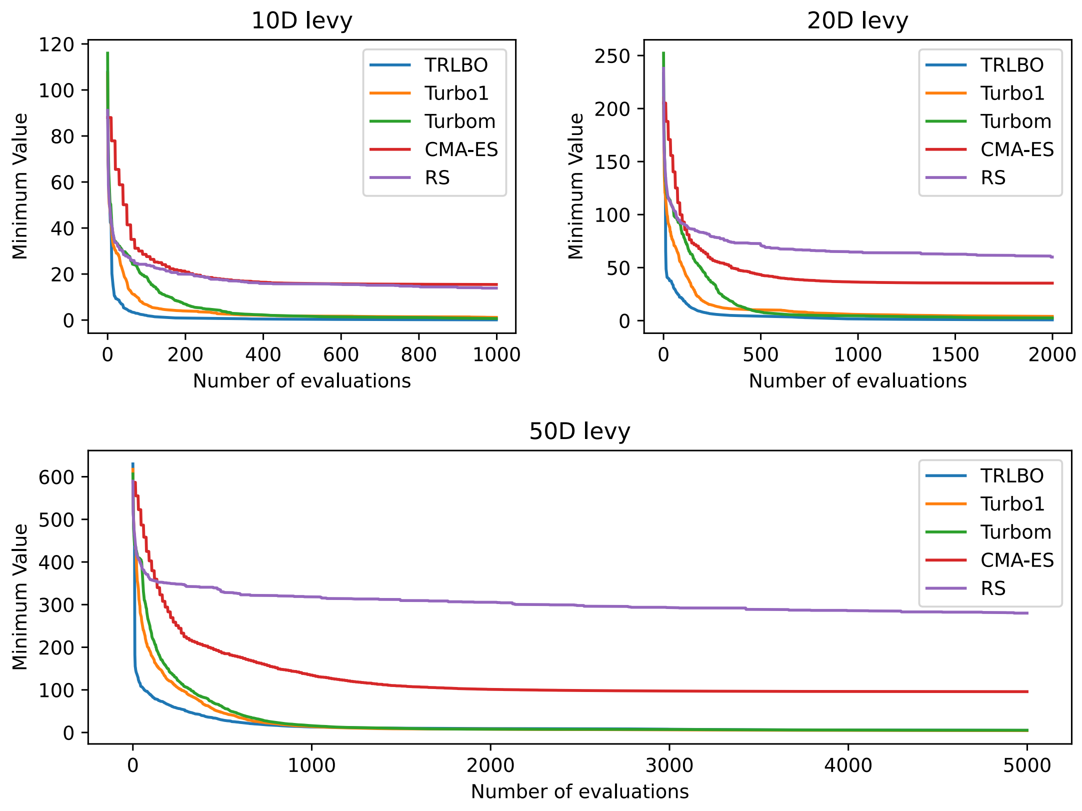

Fig. 3 10D, 20D, and 50D levy

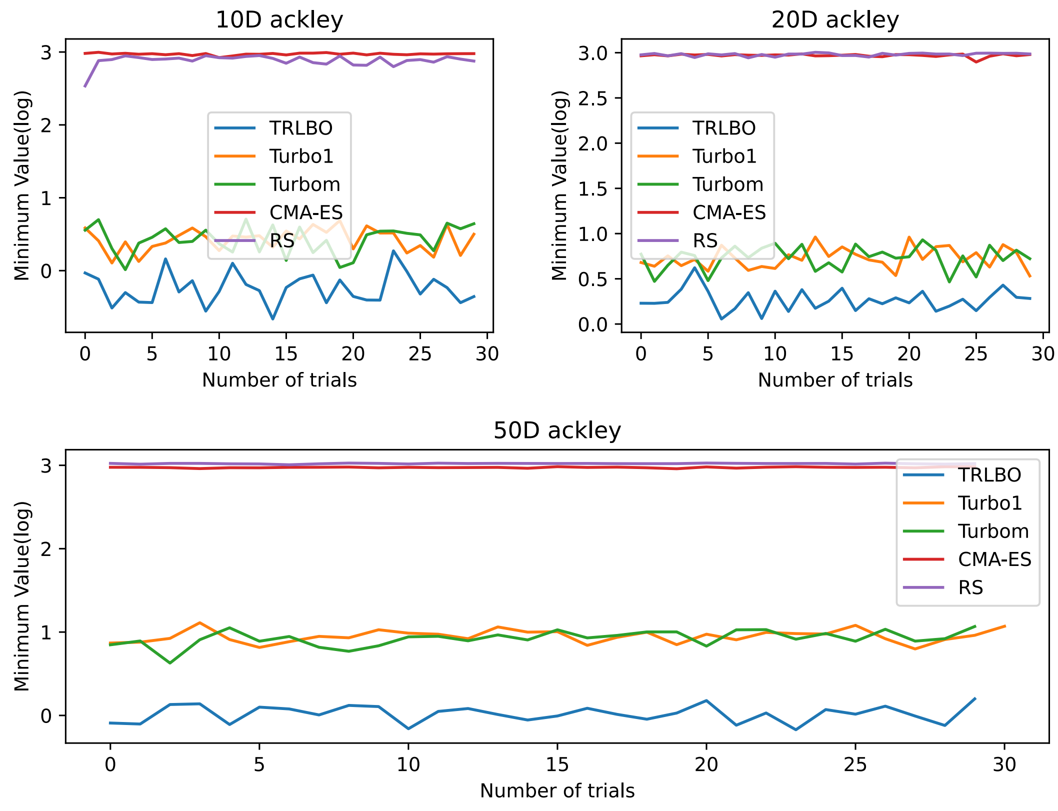

Fig. 4 Minimum value of Ackley function in 30 trials

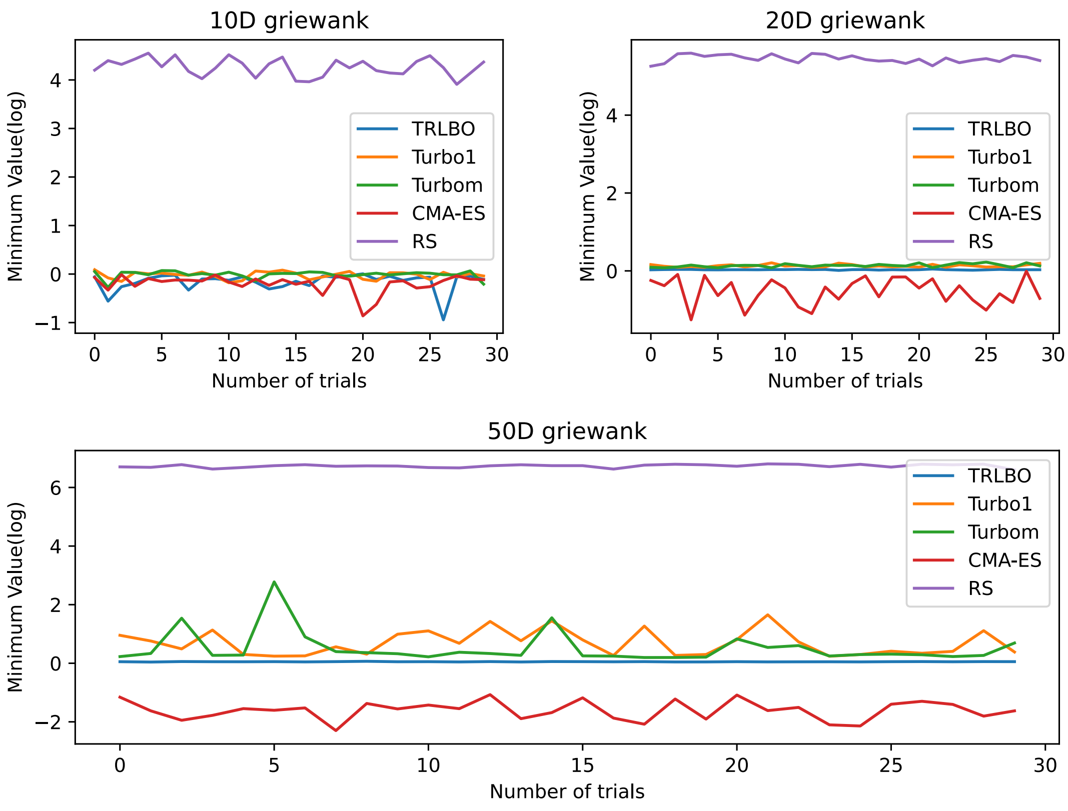

Fig. 5 Minimum value of Griewank function in 30 trials

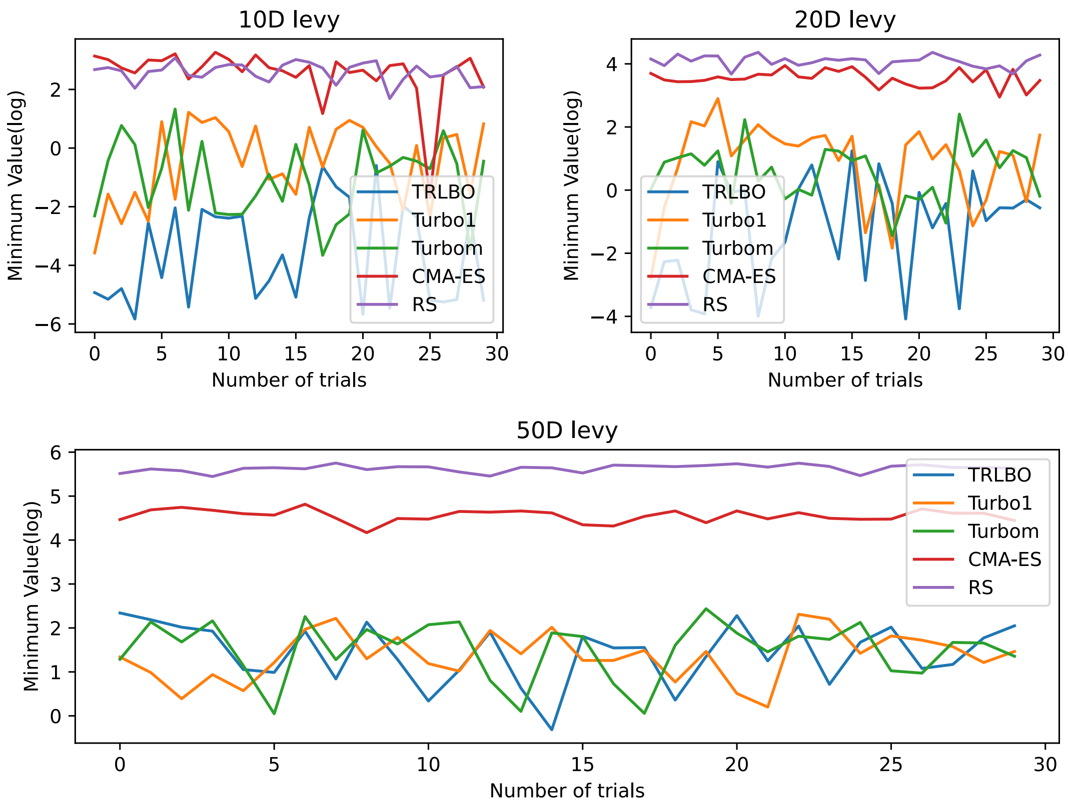

Fig. 6 Minimum value of levy function in 30 trials

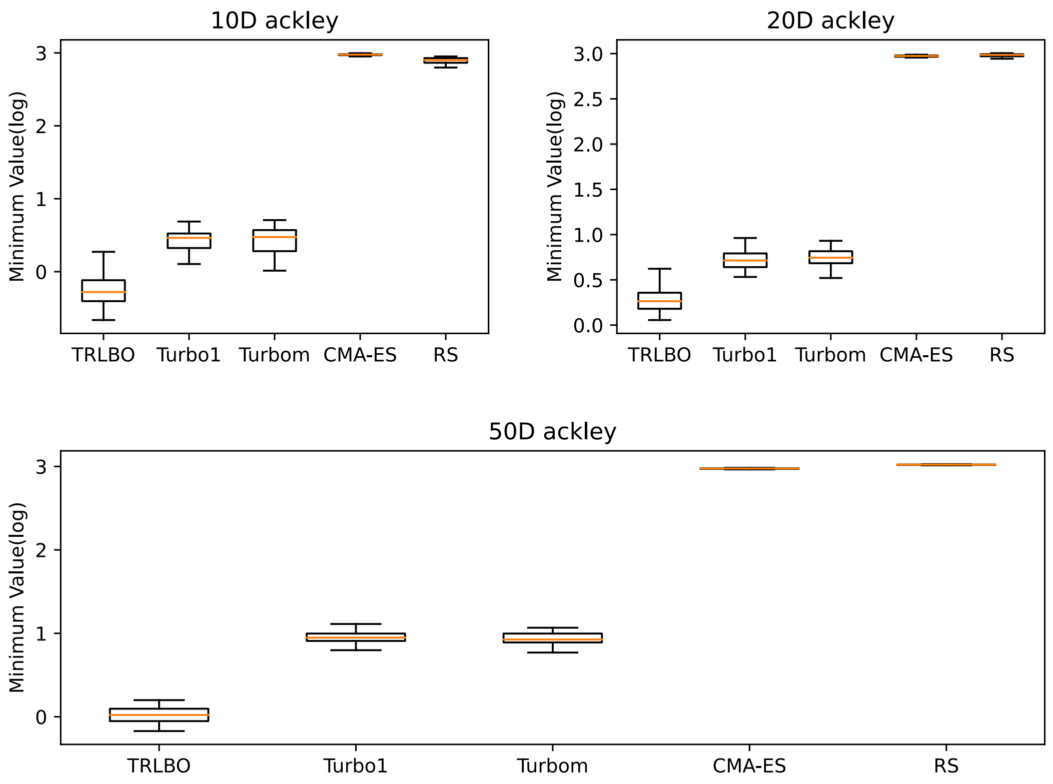

Fig. 7 Boxplot of the best value found by all algorithms with 30 trials for function Ackley

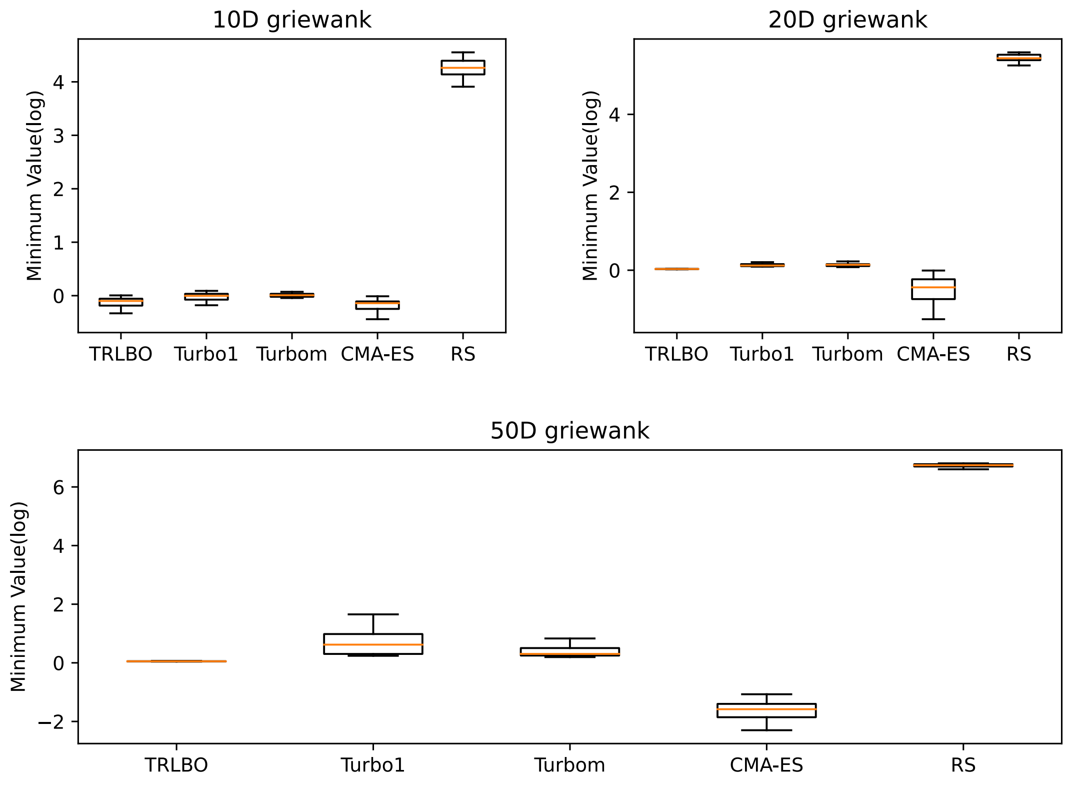

Fig. 8 boxplot of the best value found by all algorithms with 30 trials for function griewank

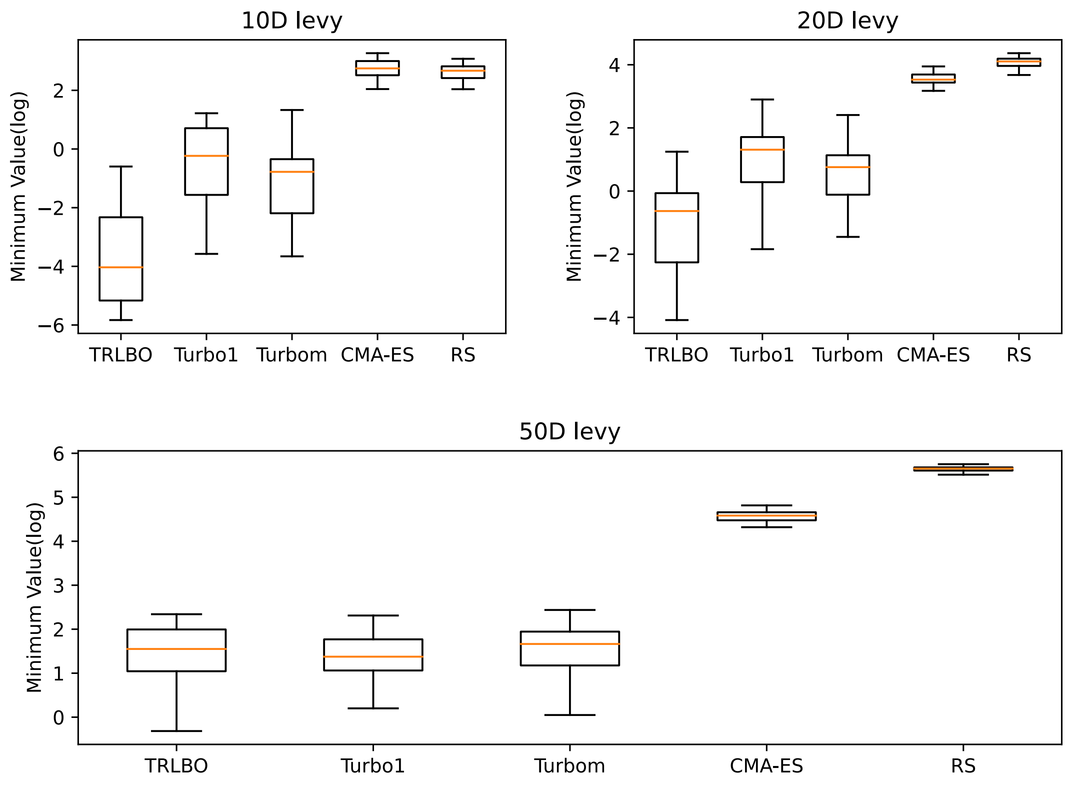

Fig. 9 Boxplot of the best value found by all algorithms with 30 trials for function levy

**Synthetic function** When handling synthetic function with 10, 20, and 50 dimensions, the number of function evaluations are set to 1000, 2000, and 5000 respectively. From the convergence graphs (Figs. 1-3）TRLBO, Turbo1, and Turbom perform much better than CMA-ES and RS. In addition, according to the convergence curve of RS, its performance is not satisfying, this also shows that the tendency towards exploration when BO is handling high-dimensional problems is the main reason for its poor performance. Figs. 4-6 shows the optima found by the algorithms over the 30 runs and Figs. 7-9 presents the corresponding box plots.

**Robot pushing.** The problem is about controlling a robot to push two objectives to their target positions. The moving trajectory of the robot is determined by a function with 14 parameters. The problem is first proposed by Wang [42] in 2017. Here, we set the number of objective function evaluations to 10K. It can be observed from the convergence graph that TRLBO, Turbo1, and Turbom are superior to CMA-ES and random search. It is surprising that the performance of CMA-ES is not as good as the random search. Figs. 10-12 give the corresponding results.

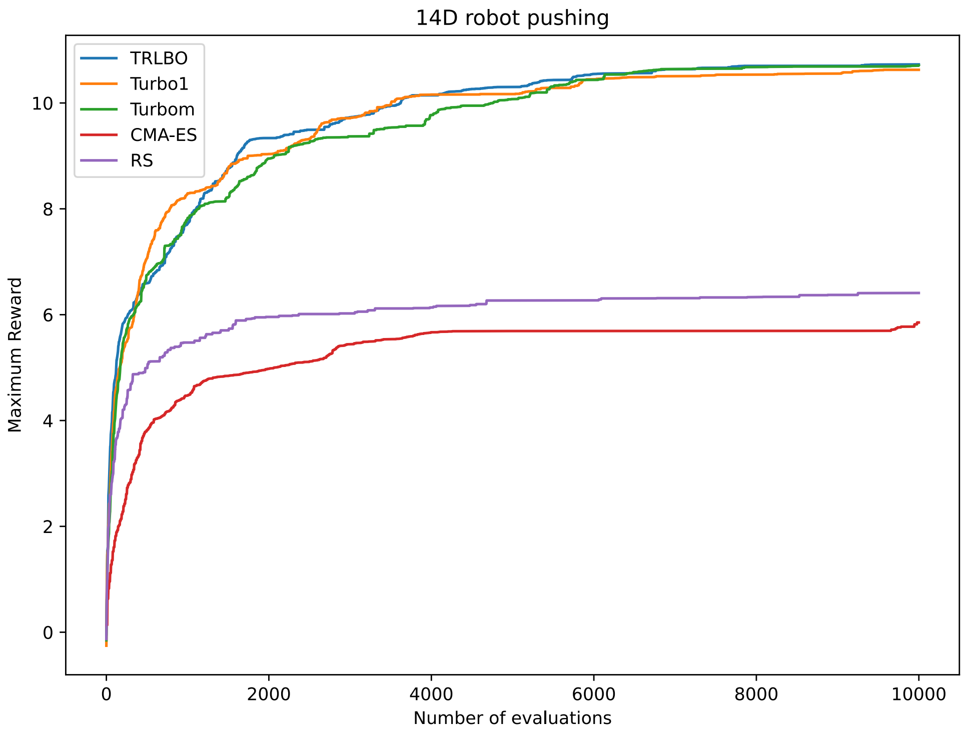

Fig. 10 robot pushing

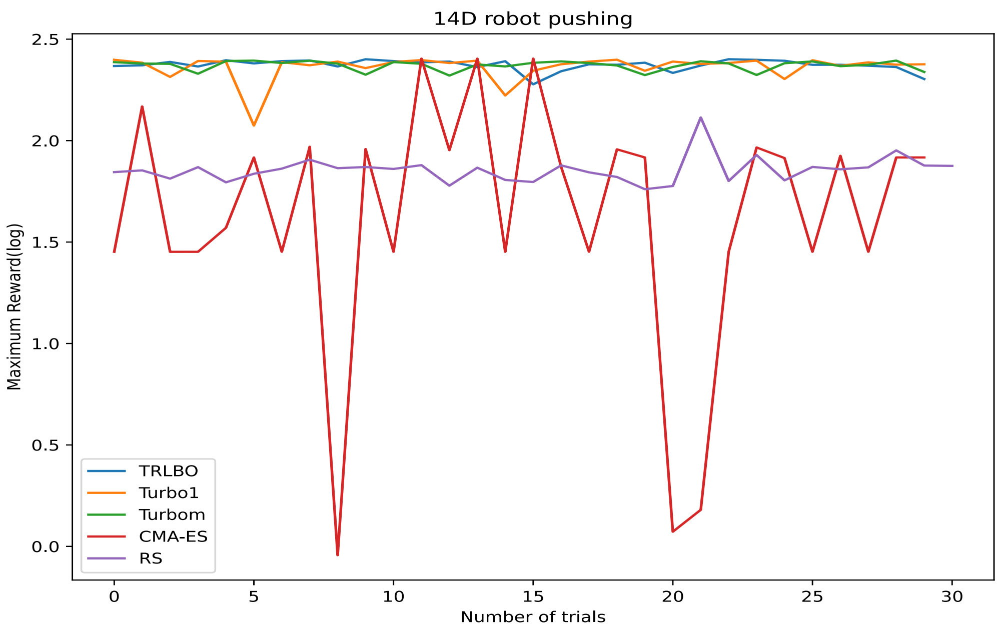

Fig. 11 Maximum reward of robot pushing problem in 30 trials.

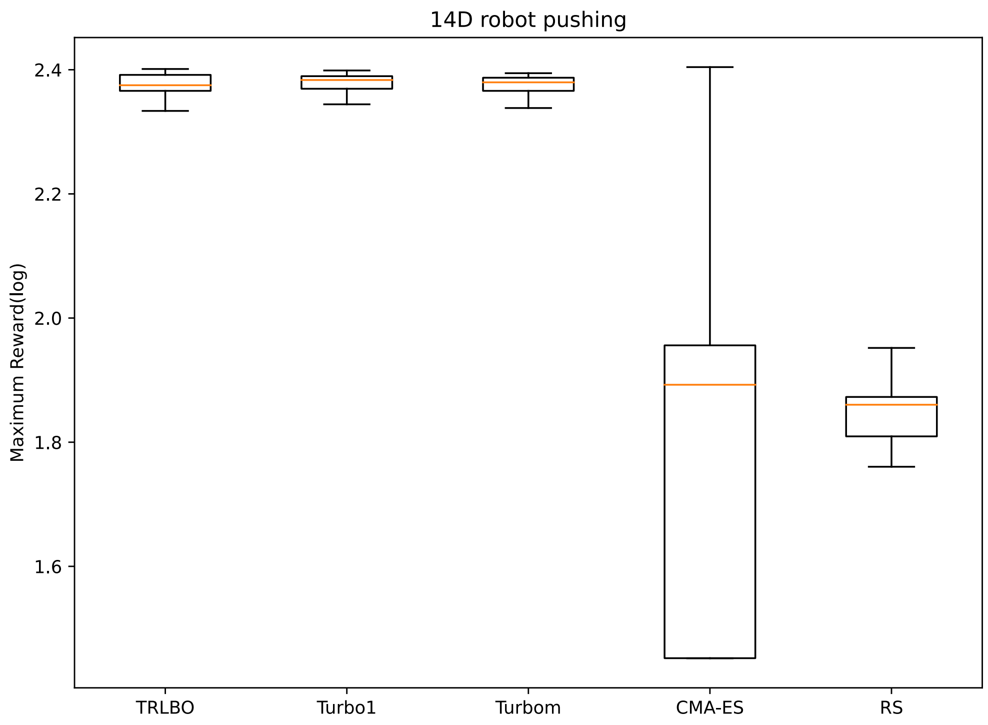

Fig. 12 Boxplot of the best value found by all algorithms with 30 trials for robot pushing problem.

**Rover trajectory planning.** This is a trajectory optimization problem in 2D, meant to emulate a rover navigation task. The objective function is a non-smooth, discontinuous function. More detailed descriptions of the problem can be found in [43]. In our experiment, the number of objective function evaluations is set to 20K. From the convergence graphs depicted in Fig. 5 (the top subgraph), it can be seen that TRLBO, Turbo1, Turbom, and CMA-ES have similar performance, followed by the random search. Fig. 13-15 show the optima found by the algorithms over 30 runs and the right subgraph in Figs. 13-15 present the corresponding box plot.

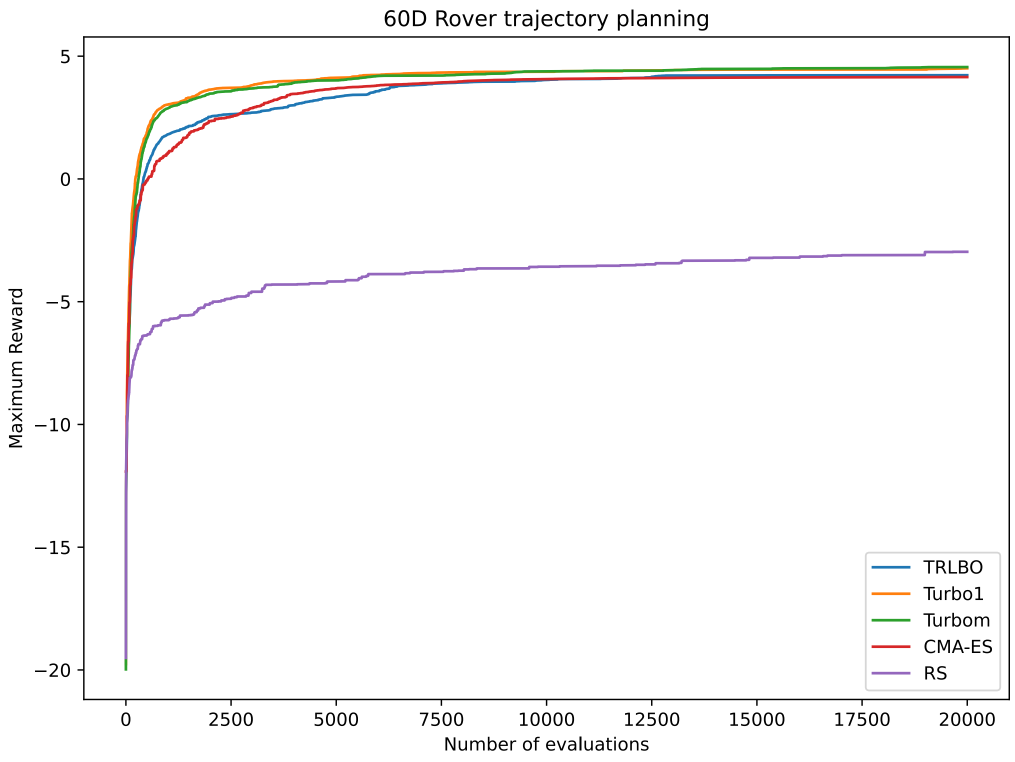

Fig. 13 Convergence result of the rover trajectory planning problem

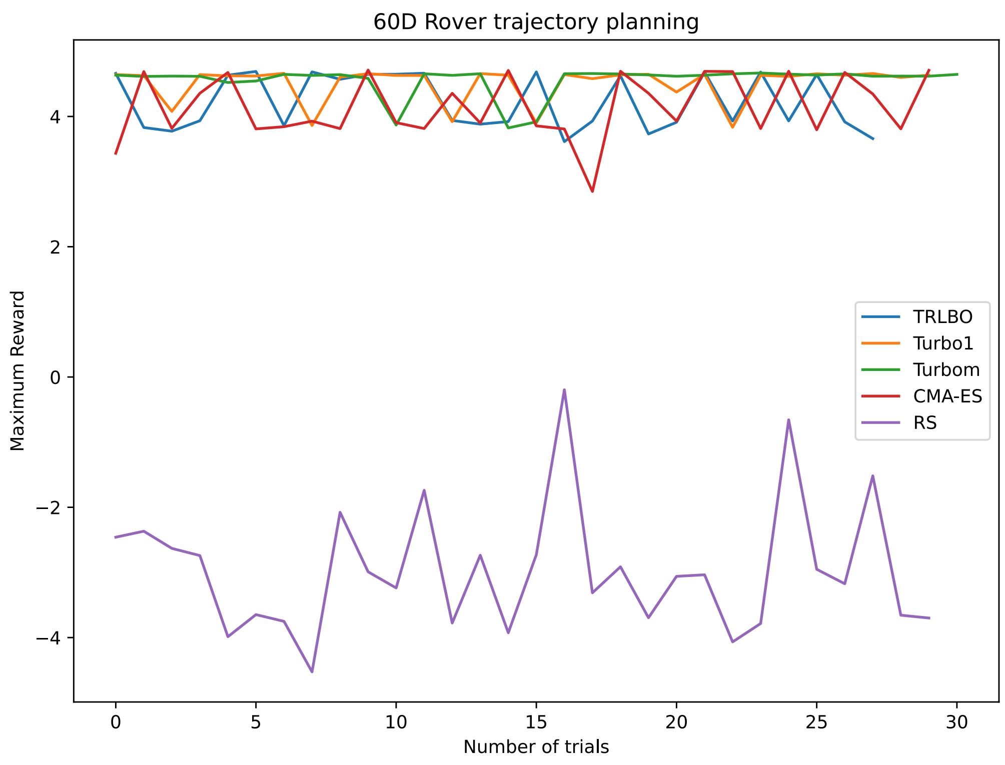

Fig. 14 Maximum reward of rover trajectory planning problem in 30 trials

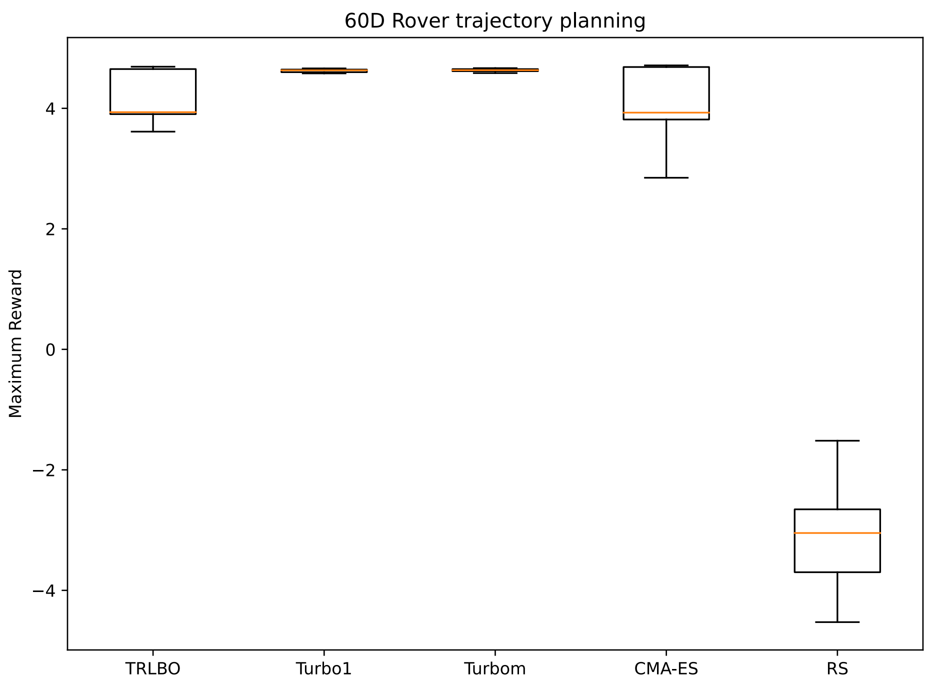

Fig. 15 Boxplot result of the rover trajectory planning problem in 30 trials

Moreover, we record the best, worst, and mean results of the algorithm on all test problems over 30 runs. From the numerical results, when solving the majority of the synthetic functions, TRLBO is the best among all the compared to the rest algorithms. CMA-ES excepts to perform extremely well on the Griewank function. Given a sufficient number of function evaluations, the solutions found by CMA-EA are the near global optimal solution. For the 10D, 20D, and 50D problems, the number of function evaluations is set to 1000, 2000, and 5000. The performance of CMA-EA on low-dimensional problems is not as good as in high-dimensional problems. In addition, when solving the robot pushing problem, TRLBO performs slightly better than Turbo1 and Turbom. For the Rover trajectory planning problem, Turbo1 and Turbom perform the best, followed by TRLBO and CMA-ES. Random Search is the worst among all the algorithms. The median performance of TRLBO is probably due to the non-smooth, non-continuous property of the objective function. The experimental results are listed in Table 1, where the best results are marked in bold.

Table 1 10D synthetic function problems result

| Problem | Result | TRLBO | Turbo1 | Turbom | CMA-ES | Random serach |
|---|---|---|---|---|---|---|
| Ackley | Best | **0.515** | 1.11 | 1.013 | 18.604 | 12.608 |
| | Mean | **0.802** | 1.548 | 1.56 | 19.54 | 17.925 |
| | Worst | **1.312** | 1.988 | 2.028 | 20.008 | 19.134 |
| Griewank | Best | 0.389 | 0.835 | 0.759 | **0.422** | 49.863 |
| | Mean | 0.865 | 0.978 | 0.992 | **0.833** | 72.141 |
| | Worst | 1.005 | 1.093 | 1.074 | **0.988** | 94.849 |
| Levy | Best | **0.002** | 0.028 | 0.025 | 0.184 | 5.41 |
| | Mean | **0.089** | 1.158 | 0.675 | 15.464 | 13.89 |
| | Worst | **0.55** | 3.384 | 3.78 | 26.226 | 21.652 |

Table 2 20D synthetic function problems result

| Problem | Result | TRLBO | Turbo1 | Turbom | CMA-ES | Random serach |
|---|---|---|---|---|---|---|
| Ackley | Best | **1.057** | 1.702 | 1.592 | 18.117 | 18.979 |
| | Mean | **1.315** | 2.089 | 2.093 | 19.501 | 19.702 |
| | Worst | **1.863** | 2.617 | 2.537 | 19.843 | 20.175 |
| Griewank | Best | 1.012 | 1.095 | 1.081 | **0.284** | 191.637 |
| | Mean | 1.031 | 1.139 | 1.149 | **0.624** | 233.343 |
| | Worst | 1.049 | 1.229 | 1.253 | **0.991** | 268.245 |
| Levy | Best | **0.016** | 0.061 | 0.234 | 19.064 | 39.483 |
| | Mean | **0.731** | 4.02 | 2.537 | 35.351 | 59.969 |
| | Worst | **3.478** | 18.173 | 11.117 | 51.794 | 78.741 |

Table 3 50D synthetic function problems result

| Problem | Result | TRLBO | Turbo1 | Turbom | CMA-ES | Random serach |
|---|---|---|---|---|---|---|
| Ackley | Best | **0.842** | 2.22 | 1.874 | 19.273 | 20.216 |
| | Mean | **1.024** | 2.593 | 2.533 | 19.573 | 20.504 |
| | Worst | **1.22** | 3.04 | 2.906 | 19.76 | 20.64 |
| Griewank | Best | 1.042 | 1.27 | 1.216 | **0.1** | 736.473 |
| | Mean | 1.053 | 2.185 | 2.162 | **0.21** | 840.669 |
| | Worst | 1.068 | 5.225 | 16.073 | **0.342** | 902.208 |
| Levy | Best | **0.729** | 1.221 | 1.049 | 64.608 | 231.775 |
| | Mean | **5.019** | 4.489 | 5.291 | 95.843 | 280.086 |
| | Worst | 10.389 | **10.086** | 11.441 | 123.404 | 314.921 |

Table 4 14D Robot pushing problems result

| Problem | Result | TRLBO | Turbo1 | Turbom | CMA-ES | Random serach |
|---|---|---|---|---|---|---|
| 14D Robot pushing | Best | **11.036** | 11.01 | 10.962 | 11.07 | 8.281 |
| | Mean | **10.729** | 10.627 | 10.708 | 5.852 | 6.41 |
| | Worst | 9.752 | 7.959 | **10.182** | 0.958 | 5.815 |

## 6. Conclusion

Bayesian optimization is an effective black-box optimization algorithm. It shows good performance when optimizing low-dimensional objective functions. However, extending BO to tackle high-dimensional problems is not an easy task. In this paper, we proposed a local Bayesian optimization algorithm based on the idea of the trust region. The algorithm incorporates the local Gaussian process and eliminates the need for exact optimization of the acquisition function. The experimental result on a number of test problems shows that our algorithm has similar or even better performance than other Bayesian optimization algorithms. Due to the limited computational resources, we did not test our algorithm on higher dimensional problems. In the future, we intend to adopt multiple local Gaussian processes to fit different parts of the search space of the objective function. From the optimization point of view, the information in local data is not necessarily lower than global data.

## Compliance with Ethical Standards statements

**Ethical approval** This paper does not contain any studies with human participants or animals performed by any of the authors.

**Funding details** This work was supported by the Key Project of Science and Technology Innovation 2030 supported by the Ministry of Science and Technology of China (Grant No. 2018AAA0101301), the Key Projects of Artificial Intelligence of High School in Guangdong Province (No. 2019KZDZX1011), Innovation Project of High School in Guangdong Province (No. 2018KTSCX314), Dongguan Social Development Science and Technology Project (No. 20211800904722) and Dongguan Science and Technology Special Commissioner Project (No. 20201800500442).

**Conflict of interest** The authors declare that they have no conflict of interest.

**Informed Consent** The authors declare that they give informed consent of this paper.

**Code availability** Codes are available and could be shared at any time when readers ask. They can send a mail on the address displayed in the top of the article. Codes are programmed under Python language.

## Authorship contributions

All authors contributed to the study conception and design. Experiments, data collection and written were performed by Anbing Fu. The conceptualization and methodology were performed by Qingxia Li. The reviewing and editing were performed by Wenhong Wei and Yuhui Zhang. The all authors read and approved the final manuscript.

## Reference

[1] Shahriari, B., Swersky, K.,Wang, Z., Adams, R.P., & Freitas, N.D. (2016). Taking the Human Out of the Loop: A Review of Bayesian Optimization. Proceedings of the IEEE, 104, 148-175.

[2] Gardner, J.R., Pleiss, G., Bindel, D.S., Weinberger, K.Q., & Wilson, A.G. (2018). GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. NeurIPS.

[3] Brochu, E., Freitas, N.D., & Ghosh, A. (2007). Active Preference Learning with Discrete Choice Data. NIPS.

[4] Chen, B., Castro, R.M., & Krause, A. (2012). Joint Optimization and Variable Selection of High-dimensional Gaussian Processes. ICML.

[5] Rolland, P., Scarlett, J., Bogunovic, I., & Cevher, V. (2018). High-Dimensional Bayesian Optimization via Additive Models with Overlapping Groups. ArXiv, abs/1802.07028.

[6] Mutný, M., & Krause, A. (2018). Efficient High Dimensional Bayesian Optimization with Additivity and Quadrature Fourier Features. NeurIPS.

[7] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M.M., Prabhat,& Adams, R.P. (2015). Scalable Bayesian Optimization Using Deep Neural Networks. ICML.

[8] Lakshminarayanan, B., Roy, D.M., & Teh, Y.W. (2016). Mondrian Forests for Large-Scale Regression when Uncertainty Matters. ArXiv, abs/1506.03805.

[9] Binois, M., Ginsbourger, D., & Roustant, O. (2015). AWarped Kernel Improving Robustness in Bayesian Optimization Via Random Embeddings. LION.

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

[11] CGardner, J.R., Guo, C., Weinberger, K.Q., Garnett, R., & Grosse, R.B. (2017). Discovering and Exploiting Additive Structure for Bayesian Optimization. AISTATS.

[12] Kandasamy, K., Schneider, J.G., & Póczos, B. (2015). High Dimensional Bayesian Optimisation and Bandits via Additive Models. ICML.

[13] Nayebi, A., Munteanu, A., & Poloczek, M. (2019). A Framework for Bayesian Optimization in Embedded

Subspaces. ICML.
[14] Wang, Z., Zoghi, M., Hutter, F., Matheson, D., & Freitas, N.D. (2016). Bayesian Optimization in a Billion Dimensions via Random Embeddings. J. Artif. Intell. Res., 55, 361-387.
[15] Wang, Z., Gehring, C., Kohli, P., & Jegelka, S. (2018). Batched Large-scale Bayesian Optimization in High-dimensional Spaces. ArXiv, abs/1706.01445.
[16] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M.M., Prabhat, & Adams, R.P. (2015). Scalable Bayesian Optimization Using Deep Neural Networks. ICML.
[17] McIntire, M., Ratner, D., & Ermon, S. (2016). Sparse Gaussian Processes for Bayesian Optimization. UAI.
[18] Wang, Z., Gehring, C., Kohli, P., & Jegelka, S. (2018). Batched Large-scale Bayesian Optimization in High-dimensional Spaces. ArXiv, abs/1706.01445.
[19] Eriksson, D., Pearce, M., Gardner, J.R., Turner, R.D., & Poloczek, M. (2019). Scalable Global Optimization via Local Bayesian Optimization. NeurIPS.
[20] Jones, D.R., Perttunen, C.D., & Stuckman, B.E. (1993). Lipschitzian optimization without the Lipschitz constant. Journal of Optimization Theory and Applications, 79, 157-181.
[21] Brochu, E., Cora, V.M., & Freitas, N.D. (2010). A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. ArXiv, abs/1012.2599.
[22] Rasmussen, C.E., & Williams, C.K. (2009). Gaussian Processes for Machine Learning. Adaptive computation and machine learning.
[23] Kushner, H.J. (1964). A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. Journal of Basic Engineering, 86, 97-106.
[24] Mockus, J. (1977). On Bayesian Methods for Seeking the Extremum and their Application. IFIP Congress.
[25] Thompson, W.R. (1933). ON THE LIKELIHOOD THAT ONE UNKNOWN PROBABILITY EXCEEDS ANOTHER IN VIEW OF THE EVIDENCE OF TWO SAMPLES. Biometrika, 25, 285-294.
[26] Villemonteix, J., Vázquez, E., & Walter, E. (2009). An informational approach to the global optimization of expensive-to-evaluate functions. Journal of Global Optimization, 44, 509-534.
[27] Hennig, P., & Schuler, C.J. (2012). Entropy Search for Information-Efficient Global Optimization. ArXiv, abs/1112.1217.
[28] Hernández-Lobato, J., Hoffman, M.W., & Ghahramani, Z. (2014). Predictive Entropy Search for Efficient Global Optimization of Black-box Functions. NIPS.
[29] Scott,W.R., Frazier, P.,&Powell,W.B. (2011). The Correlated Knowledge Gradient for Simulation Optimization of Continuous Parameters using Gaussian Process Regression. SIAM J. Optim., 21, 996-1026.
[30] Srinivas, N., Krause, A., Kakade, S.M., & Seeger, M.W. (2010). Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. ICML.
[31] Kocsis, L., & Szepesvari, C. (2006). Bandit Based Monte-Carlo Planning. ECML.
[32] Contal, E., Buffoni, D., Robicquet, A., & Vayatis, N. (2013). Parallel Gaussian Process Optimization with Upper Confidence Bound and Pure Exploration. ECML/PKDD.
[33] Desautels, T., Krause, A., & Burdick, J.W. (2012). Parallelizing Exploration-Exploitation Tradeoffs with Gaussian Process Bandit Optimization. J. Mach. Learn. Res., 15, 3873-3923.
[34] Ginsbourger, D., Riche, R.L., & Carraro, L. (2010). Kriging is well-suited to parallelize optimization.
[35] Snoek, J., Larochelle, H., & Adams, R.P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. NIPS.
[36] Marmin, S., Chevalier, C., & Ginsbourger, D. (2015). Differentiating the Multipoint Expected Improvement for Optimal Batch Design. MOD.
[37] Shah, A., & Ghahramani, Z. (2015). Parallel Predictive Entropy Search for Batch Global Optimization of Expensive Objective Functions. NIPS.
[38] Wang, J., Clark, S.C., Liu, E., & Frazier, P. (2020). Parallel Bayesian Global Optimization of Expensive Functions. Oper. Res., 68, 1850-1865.
[39] Conn, A.R., Scheinberg, K., & Toint, P.L. (1997). Recent progress in unconstrained nonlinear optimization without derivatives. Mathematical Programming, 79, 397-414.
[40] Ross, S. M. (1985). Stochastic processes. 2nd ed. Journal of the American Statistical Association, 80(389).
[41] Lawler, G. F. . (2006). Introduction to Stochastic Processes, Second Edition. Houghton Mifflin Co.
[42] Zi Wang, Chengtao Li, Stefanie Jegelka, and Pushmeet Kohli. Batched high-dimensional Bayesian optimization

via structural kernel learning. In International Conference on Machine Learning (ICML), 2017.

[43] Z. Wang, C. Gehring, P. Kohli, and S. Jegelka. Batched largescale Bayesian optimization in highdimensional spaces. In International Conference on Artificial Intelligence and Statistics, pages 745–754, 2018.