High-dimensional Bayesian Optimization via Covariance Matrix Adaptation Strategy

Anonymous authors
Paper under double-blind review

Abstract

Bayesian Optimization (BO) is an effective method for finding the global optimum of expensive black-box functions. However, it is well known that applying BO to high-dimensional optimization problems is challenging. To address this issue, a promising solution is to use a local search strategy that partitions the search domain into local regions with high likelihood of containing the global optimum, and then use BO to optimize the objective function within these regions. In this paper, we propose a novel technique for defining the local regions using the Covariance Matrix Adaptation (CMA) strategy. Specifically, we use CMA to learn a search distribution that can estimate the probabilities of data points being the global optimum of the objective function. Based on this search distribution, we then define the local regions consisting of data points with high probabilities of being the global optimum. Our approach serves as a *meta-algorithm* as it can incorporate existing black-box BO optimizers, such as `BO`, `TuRBO` (Eriksson et al., 2019), and `BAxUS` (Papenmeier et al., 2022), to find the global optimum of the objective function within our derived local regions. We evaluate our proposed method on various benchmark synthetic and real-world problems. The results demonstrate that our method outperforms existing state-of-the-art techniques.

1 Introduction

Optimizing expensive black-box functions is an important task that has various applications in machine learning, data science, and operational research. Bayesian Optimization (BO) (Jones et al., 1998; Brochu et al., 2010; Shahriari et al., 2016; Binois & Wycoff, 2022; Garnett, 2023) is a powerful approach to tackle this challenging problem in an efficient manner. It has been successfully applied in a wide range of applications, including but not limited to hyperparameter tuning of machine learning models (Snoek et al., 2012; Turner et al., 2020), neural architecture search (Jenatton et al., 2017; Kandasamy et al., 2018b), material design (Hernández-Lobato et al., 2017), robotics (Calandra et al., 2016), and reinforcement learning (Brochu et al., 2010; Parker-Holder et al., 2022).

BO operates in an iterative fashion by repeatedly training a *surrogate model* based on the observed data and using an *acquisition function* to suggest promising data points for evaluation (Garnett, 2023). This method is inspired by Bayes' theorem, which aims to improve the prior belief about the objective function by incorporating observed data to obtain a posterior with better information. In this way, BO selects the next data points by considering previous information and maximizes the knowledge gained about the objective function with new observations, making it sample-efficient in finding the objective function's global optimum.

Despite being a powerful optimization method, BO still suffers from various problems, including the curse of dimensionality issue, i.e., it often performs poorly when applied to high-dimensional problems (Rana et al., 2017; Eriksson et al., 2019; Binois & Wycoff, 2022; Papenmeier et al., 2022). One reason for this is that, as the search domain grows in dimensionality, more local optima may appear, making it difficult for the algorithm to find the global optimum. Additionally, a larger search space also implies more regions with high uncertainty, which can potentially cause the acquisition function to overemphasize exploration and fail to exploit potential regions within a fixed budget. Moreover, in high-dimensional spaces, the objective function is typically heterogeneous, making it difficult to fit a global surrogate model across the entire domain.

There have been various works attempting to make BO work well for high-dimensional optimization problems. One of the most promising approaches, which have demonstrated significant success, is the use of a local search strategy that partitions the search domain into promising local regions where the optimization process is performed within (Munos, 2011; Wang et al., 2014; Eriksson et al., 2019; Wang et al., 2020; Wan et al., 2021). These works, however, have certain limitations. For instance, the works in Munos (2011); Wang et al. (2014); Eriksson et al. (2019) employ a search space partition technique with fixed parameters that maybe difficult to optimally specify in advance, and thus, may not provide adequate flexibility for various problems (Wang et al., 2020). The work in Wang et al. (2020) learns promising local regions by partitioning the search space into non-linear boundary regions using an unsupervised classification algorithm, but there is no guarantee that these local regions can be learned accurately with a limited amount of training data.

In this paper, we follow the aforementioned local search approach to tackle the high-dimensional optimization problem. We propose to use the *Covariance Matrix Adaptation* (CMA) strategy to systematically define the local regions to be used within this local search approach. CMA is a technique developed in the Evolutionary Algorithm literature (Hansen & Ostermeier, 2001), aiming to estimate the probability distribution of data points in the search domain being the global optimum of the objective function (i.e., *search distribution*). Typically, CMA is combined with Evolutionary Strategy (ES) techniques like `CMA-ES` (Hansen & Ostermeier, 2001). These techniques leverage the search distribution derived from CMA to guide the search for the global optimum toward promising regions in the search domain, i.e., regions that highly likely contain the global optimum. It has been shown that CMA-based ES techniques, such as `CMA-ES`, perform very well in finding the global optima of high-dimensional optimization problems, demonstrating CMA's effectiveness in identifying promising regions that likely contain the global optima of high-dimensional optimization problems. A drawback of these techniques is that they generally require a large number of function evaluations (Loshchilov & Hutter, 2016; Nomura et al., 2021).

Inspired by the effectiveness of CMA in working with high-dimensional optimization problems, we propose to incorporate CMA into BO methods by using it to define the local regions in the BO local search approach. In particular, we define the local regions as the regions with the highest probabilities of containing the global optimum based on CMA's search distribution. Subsequently, we can use an existing BO optimizer, e.g., `BO`, `TuRBO` (Eriksson et al., 2019), `BAxUS` (Papenmeier et al., 2022), within these local regions to find the global optimum of the objective function. By leveraging information from CMA's search distribution, BO methods can focus the search within the local regions that have high likelihood of containing the global optimum of the optimization problem. CMA-based BO methods are therefore expected to work well with high-dimensional optimization problems, due to CMA's effectiveness, whilst preserving data-efficiency, a property lacking in CMA-based ES techniques. We derive the `CMA-BO`, `CMA-TuRBO` and `CMA-BAxUS` algorithms corresponding to the cases when we incorporate CMA with the optimizers `BO`, `TuRBO`, `BAxUS`, respectively. Our experimental results on various synthetic and real-world benchmark problems confirm that our proposed approach helps existing BO methods to work better for high-dimensional optimization problems whilst being data-efficient.

In summary, our contributions are as follows:

- Proposing a novel *meta-algorithm* using the local search approach and the CMA strategy to enhance the performance of existing BO methods for high-dimensional optimization problems;
- Deriving the CMA-based BO algorithms corresponding to the cases when we incorporate the proposed meta-algorithm with the state-of-the-art BO methods (e.g., `BO`, `TuRBO`, `BAxUS`);
- Conducting a comprehensive evaluation on various high-dimensional synthetic and real-world benchmark problems and demonstrating that our proposed CMA-based meta-algorithm outperforms existing state-of-the-art methods.

2 Background

In this section, we present the fundamental background of BO. Then we revisit two state-of-the-art BO methods (`TuRBO`, `BAxUS`) that we will incorporate into our CMA-based meta-algorithm.

2.1 Bayesian Optimization

Bayesian optimization (BO) is a powerful optimization method to find the global optimum of an expensive black-box objective function by sequential queries (Jones et al., 1998; Brochu et al., 2010; Shahriari et al., 2016; Binois & Wycoff, 2022; Garnett, 2023). Let us consider the minimization problem: given an unknown objective function $f : \mathcal{X} \to \mathbb{R}$ where $\mathcal{X} \subset \mathbb{R}^d$ is a compact space, the goal of BO is to find a global optimum $\boldsymbol{x}^*$ of the objective function f:

$$\boldsymbol{x}^* \in \arg\min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}). \tag{1}$$

BO solves an optimization problem in an iterative manner. First, the objective function f is approximated by a *surrogate model* trained with the current observed dataset $D_0 = \{\boldsymbol{x}_i, y_i\}_{i=1}^{t_0}$ with $y_i = f(\boldsymbol{x}_i) + \varepsilon_i$ and $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ being the corrupted noise. Then an *acquisition function* $\alpha : \mathcal{X} \to \mathbb{R}$ is constructed from the surrogate model to assign scores to all data points in the domain $\mathcal{X}$ based on their potential to improve the optimization process. The next data point to be evaluated, denoted as $\boldsymbol{x}_{\text{next}}$, is selected as the maximizer of the acquisition function. Subsequently, the objective function f is evaluated at $\boldsymbol{x}_{\text{next}}$ and the new observed data $(\boldsymbol{x}_{\text{next}}, y_{\text{next}})$, with $y_{\text{next}} = f(\boldsymbol{x}_{\text{next}}) + \varepsilon$ and $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, is added to the current observed dataset. The process is conducted iteratively until a pre-defined budget is exhausted, and then BO returns the best value found from the observed dataset as an estimate of the global optimum $\boldsymbol{x}^*$.

There are different choices for the surrogate models to be used in BO, including Gaussian Process (GP) (Rasmussen & Williams, 2006), Tree-structured Parzen Estimator (TPE) (Bergstra et al., 2011), and neural networks (Springenberg et al., 2016). In our work, we focus on the GP surrogate model, which is one of the most popular surrogate models in BO. GP is a probabilistic model that can provide both scalar predictions for the objective function values and their associated uncertainty. A GP is completely specified by its mean function $\mu(\boldsymbol{x})$ and covariance function (kernel) $k(\boldsymbol{x}, \boldsymbol{x}')$ (Rasmussen & Williams, 2006). While the mean function indicates the most probable values for the objective function values, the covariance function captures the properties of the objective function, e.g. its smoothness.

There is also a variety of common acquisition functions. Examples include Expected Improvement (EI) (Mockus et al., 1978), Probability of Improvement (PI) (Kushner, 1964), Upper Confidence Bound (UCB) (Srinivas et al., 2010), Thompson Sampling (TS) (Thompson, 1933), and Knowledge Gradient (KG) (Frazier et al., 2009). While each of these acquisition functions has its own strengths and weaknesses, they all aim to balance between exploration and exploitation. Exploration encourages the algorithm to look for promising values in highly uncertain locations, whilst exploitation favors refining the knowledge around the currently optimal locations. In this work, we use Thompson Sampling (TS) (Thompson, 1933) as the acquisition function following Eriksson et al. (2019); Papenmeier et al. (2022). In the next section, we will describe in detail the TS acquisition function which will be later used in our method.

2.2 The Thompson Sampling Acquisition Function

The Thompson Sampling (TS) acquisition function (Thompson, 1933), commonly used in BO research (Kandasamy et al., 2018a; Eriksson et al., 2019; Papenmeier et al., 2022), follows a stochastic policy. The main idea is to sample a random realization (i.e., *sample path*) of the objective function from its posterior distribution, then optimize this sample path to find the next data point to be evaluated.

Specifically, at iteration t, given the observed dataset D_t, the TS acquisition function $\alpha^{\text{TS}}(\boldsymbol{x}; D_t)$ can be defined as,

$$\alpha^{\text{TS}}(\boldsymbol{x}; D_t) = f^{(t)}(\boldsymbol{x}) \;\; \text{where } f^{(t)}(\boldsymbol{x}) \sim p(f|D_t), \tag{2}$$

and $p(f|D_t)$ denotes the posterior distribution of the objective function f given the observed data D_t. If GP is used as the surrogate model for the objective function, the TS acquisition function $\alpha^{\text{TS}}(\boldsymbol{x}; D_t)$ becomes $\alpha^{\text{TS}}(\boldsymbol{x}; D_t) = f^{(t)}(\boldsymbol{x})$ where $f^{(t)}(\boldsymbol{x}) \sim \text{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')|D_t)$.

For a minimization problem as defined in Eq. (1), the TS process is then to minimize the acquisition function $\alpha^{\text{TS}}(\boldsymbol{x}; D_t)$ to find the next data point $\boldsymbol{x}_{t+1}$ to be evaluated,

$$\boldsymbol{x}_{t+1} = \arg\min_{\boldsymbol{x} \in \mathcal{X}} \alpha^{\text{TS}}(\boldsymbol{x}; D_t). \tag{3}$$

The TS acquisition function selects the data point to be evaluated by drawing a random realization of the objective function f from its posterior distribution, therefore, it encourages exploitation of regions with higher probabilities of being optimal, while allowing for exploration of other regions, owing to its random nature. It thus satisfies the desirable property of balancing between exploitation and exploration, which is a requirement for any acquisition function in BO.

2.3 TurBO

`TuRBO` (Eriksson et al., 2019) is a state-of-the-art BO method, which proposes to use the local search strategy to solve the high-dimensional optimization problem. In `TuRBO`, the global search space is partitioned into smaller domains, called trust regions (TR) (Yuan, 2000), within which the surrogate model is believed to accurately model the objective function. `TuRBO` uses GP as the surrogate model and trains the GP using all the previous observed data. After that, `TuRBO` selects the next observed data point by optimizing the acquisition function locally within the TR.

This local strategy makes `TuRBO` more efficient than standard `BO` methods in high-dimensional problems, as it only requires modelling of local surrogate models, abandoning the global surrogate model used in standard `BO` methods. The local surrogate models of `TuRBO` do not suffer from the heterogeneity of the objective function, as they only need to accurately capture the objective function within the TR. Moreover, as the TR reduces the regions with large uncertainty, `TuRBO` mitigates the over-exploration issue commonly encountered by standard `BO` methods when solving high-dimensional optimization problems.

In `TuRBO`, in each iteration, the TR is constructed as a hyper-rectangle centered at the optimum data point found so far. Each side length of the TR is initialized with a base side length L, and then scaled with the GP lengthscales in each dimension, while maintaining the overall hyper-volume. The size of the TR is critical, as it needs to be large enough to contain potential solutions, while being small enough to ensure the accuracy of the GP surrogate model. Therefore, `TuRBO` adopts an adaptation mechanism to expand or shrink the TR depending on whether the algorithm succeeds or fails to find a better solution. When `TuRBO` succeeds in finding better solutions for τ_{succ} consecutive times, the TR increases its current size, whereas it decreases its size after τ_{fail} consecutive failures. Furthermore, `TuRBO` also defines minimum and maximum thresholds, denoted as $L_{\min}$ and $L_{\max}$ respectively, for the TR base side length. The upper bound $L_{\max}$ is to prevent the TR from becoming too large, while the lower bound $L_{\min}$ serves as a restart criterion, such that when the side length $L < L_{\min}$, `TuRBO` discards the current TR and restarts a new TR from scratch. These hyperparameters (e.g., τ_{succ}, τ_{fail}, $L_{\min}$, $L_{\max}$) are set as some fixed values in `TuRBO`.

2.4 BAxUS

`BAxUS` (Papenmeier et al., 2022) is a state-of-the-art BO method that tackles the high-dimensional optimization problem using a subspace embedding approach (Wang et al., 2016; Nayebi et al., 2019; Letham et al., 2020). The main idea is to assume the existence of a low-dimensional subspace (*active subspace*) $\mathcal{Z} \subset \mathbb{R}^{d_e}$ ($d_e \leq d$), a function $g : \mathcal{Z} \to \mathbb{R}$ and a projection matrix $\boldsymbol{T} \in \mathbb{R}^{d_e \times d}$ such that $\forall \boldsymbol{x} \in \mathcal{X}, g(\boldsymbol{T}\boldsymbol{x}) = f(\boldsymbol{x})$. This property enables the optimization process to be conducted in the active subspace, which has a lower dimension compared to the original high-dimensional space.

In practice, the effective dimensionality d_e is generally unknown, therefore, existing approaches (Wang et al., 2016; Nayebi et al., 2019; Letham et al., 2020) randomly choose a subspace $\mathcal{V} \subset \mathbb{R}^{d_\mathcal{V}}$ (*target space*) to project the original space into, and perform optimization within this chosen subspace. The *target dimension* $d_\mathcal{V}$ must be chosen such that the probability of the target space containing the global optimum is high. In practice, choosing an appropriate value of the target dimension $d_\mathcal{V}$ is challenging as a small value of $d_\mathcal{V}$ does not guarantee that the target space contains the global optimum whilst a large value of $d_\mathcal{V}$ could be subject to the curse of dimensionality issue. `BAxUS` proposes an adaptive strategy to gradually increase the target dimension during the optimization process, guaranteeing a higher probability, compared to existing approaches, that its embedding contains the global optimum of the objective function.

`BAxUS` starts with a low value of the target dimension $d_\mathcal{V}$, and then conducts a BO process to search for the optimum in this target space within a specific evaluation budget before increasing the target dimension.

Additionally, BAxUS also employs TuRBO as its BO optimizer to perform optimization for high-dimensional problems. Therefore, in each iteration, BAxUS also constructs a TR and applies the TR adaptation mechanism, similar to TuRBO, when optimizing within the target space. BAxUS keeps most of the settings to be the same with TuRBO (e.g., hyper-rectangle TR, shrinkage factor, success tolerance), but redefines the failure tolerance τ_{fail} to make the search quicker in low-dimensional target spaces.

3 Related Work

Various research works have been conducted to address the challenge of applying BO to high-dimensional optimization problems. One approach is to exploit the additive structure of the objective functions, then construct and combine a large number of Gaussian Processes (GPs) to approximate the objective function (Kandasamy et al., 2015; Gardner et al., 2017). Some works suggest replacing GPs with surrogate models that might scale better with high-dimensional data such as random forests (Hutter et al., 2011), deep neural networks (Snoek et al., 2015), or Bayesian neural networks (Springenberg et al., 2016). Oh et al. (2018) propose BOCK whose main idea is to employ cylindrical transformation to transform the geometry of the search space, thus mitigating the over-exploration issue of BO in high-dimensional optimization problems. Other methods, such as the work by Garnett et al. (2014), REMBO (Wang et al., 2016), HeSBO (Nayebi et al., 2019), ALEBO (Letham et al., 2020), and BAxUS (Papenmeier et al., 2022) propose mapping the original high-dimensional space into a low-dimensional space and then conducting optimization within this low-dimensional space. More recently, Song et al. (2022) propose MCTS-VS, a meta-algorithm that employs Monte Carlo tree search (MCTS) to construct a low-dimensional subspace and then use a BO optimizer (e.g., BO, TuRBO) to optimize the objective function within this subspace.

Another approach that has recently attracted a lot of attention is the use of a local search strategy that partitions the search domain into promising local regions where the optimization can be performed within (Munos, 2011; Wang et al., 2014; Eriksson et al., 2019; Wang et al., 2020; Turner et al., 2020; Fröhlich et al., 2021; Wan et al., 2021; 2022). Notable methods in this direction include TuRBO (Eriksson et al., 2019), whose main idea is to construct local regions as hyper-rectangles centered around the best found values, and dynamically expands or shrinks these regions based on the function values. Wan et al. (2021) extend this idea for the categorical and mixed search space settings. Another method proposed in Wang et al. (2020), namely LA-MCTS, is a meta-level algorithm that uses an unsupervised K-mean algorithm to classify the search space into good and bad local regions, within which a BO optimizer such as BO or TuRBO can be employed.

Other related works, including the research by Müller et al. (2021); Nguyen et al. (2022), also employ a local search strategy. However, different compared to the local search approach of partitioning the search space into local regions, this approach aims to incorporate gradient information into the BO process to enhance its effectiveness. In particular, the work in Müller et al. (2021) develops a probabilistic model that can incorporate gradient information, and selects data points for evaluation as those that maximize the gradient. Building on this work, Nguyen et al. (2022) propose a new acquisition function designed to maximize the probability of gradient descent, thereby enabling BO to rapidly converge toward the local optimum region. The local strategies in these methods are different from the space partitioning mechanism used in TuRBO and LA-MCTS. These methods aim to perform local optimization using the current solution and seek to descend the objective function values via the gradient information computed from objective function values in nearby regions. These methods complement our proposed CMA-based meta algorithm, as they can be used as optimizers within our approach.

Evolutionary Algorithms (EAs) represent a widely-used family of algorithms for optimizing high-dimensional black-box functions. Among these, CMA-ES (Hansen & Ostermeier, 2001) is well-known for its impressive performance in finding global optimum of the objective function. In this paper, we propose a novel meta-algorithm that leverages the CMA technique of CMA-ES to define the local regions. There are several EA methods that also combine the CMA technique with the GP surrogate model to enhance the optimization performance. GPOP (Buche et al., 2005) is an EA method that uses CMA-ES to optimize a merit function defined by the predicted mean and standard deviation function of a trained GP. DTS-CMAES (Bajer et al., 2019), another EA method which has been shown to outperform GPOP, constructs a doubly-trained GP inside CMA-ES to select the data points for forming the mean vector and covariance matrix of CMA. Closely related

to our approach, BADS (Acerbi & Ma, 2017) is a global optimization method that adopts the mesh adaptive direct search framework (Audet & Dennis, 2006) which uses CMA-ES as the search oracle and a GP-based method to find the global optimum of the objective function.

In our experiments, we compare the CMA-based BO methods (created by combining our proposed CMA-based meta algorithm with the BO optimizers BO, TuRBO, BAxUS) with a comprehensive list of related methods: the standard BO method, TuRBO, BAxUS, LA-MCTS, MCTS-VS, CMA-ES, DTS-CMAES and BADS.

4 High-dimensional Bayesian Optimization via Covariance Matrix Adaptation

In this section, we first discuss the key ideas of the CMA strategy (Section 4.1), then we propose the CMA-based meta-algorithm (Section 4.2). Finally, we derive the CMA-based BO algorithms (CMA-BO, CMA-TuRBO, CMA-BAxUS) corresponding to the cases when we integrate the CMA-based meta-algorithm with the BO optimizer BO, TuRBO, and BAxUS (Sections 4.2.1, 4.2.2, and 4.2.3).

4.1 The Covariance Matrix Adaptation Strategy

The CMA strategy was initially developed in the Evolutionary Algorithm literature, particularly in the CMA-ES method (Hansen & Ostermeier, 2001). Its main idea is based on stochastic search, which maintains a search distribution, $p(\boldsymbol{x})$, to estimate the probabilities of data points in the search domain being the global optimum of the objective function. First, a search distribution $p^{(0)}(\boldsymbol{x})$ is initialized. Then a population of data is sampled from $p^{(0)}(\boldsymbol{x})$, and the search distribution is updated based on the objective function values of these data points. This process is conducted iteratively until the algorithm converges (Hansen & Ostermeier, 2001; Abdolmaleki et al., 2017). CMA provides well-established formulas (details below) for the updates of the search distribution, enabling the stochastic search algorithm to potentially converge to the true search distribution. It has been shown that CMA-based ES techniques, such as CMA-ES, perform very well in finding the global optima of high-dimensional optimization problems (Loshchilov & Hutter, 2016; Eriksson et al., 2019; Letham et al., 2020; Nomura et al., 2021).

In practice, the most popular choice for the search distribution used in CMA is the multivariate normal distribution (Hansen & Ostermeier, 2001; Hansen, 2016). Consequently, the focus of CMA is on updating the two principal moments of this distribution: the mean vector $\boldsymbol{m}$ and the covariance matrix $\boldsymbol{\Sigma}$. In CMA, the covariance matrix is normally decomposed as $\boldsymbol{\Sigma} = \sigma \boldsymbol{C}$ where $\sigma > 0$ is the overall standard deviation (step size) of the search distribution, thus the goal of CMA is then to update $\boldsymbol{m}, \sigma$, and $\boldsymbol{C}$. At iteration t, given λ observed data points $\{\boldsymbol{x}_i^{(t)}, y_i^{(t)}\}_{i=1}^{\lambda}$ sampled from the search distribution $\mathcal{N}\left(\boldsymbol{m}^{(t-1)}, \sigma^{(t-1)}\boldsymbol{C}^{(t-1)}\right)$ of the previous iteration $t-1$, the mean vector $\boldsymbol{m}^{(t)}$, covariance matrix $\boldsymbol{C}^{(t)}$, and step size $\sigma^{(t)}$ of the search distribution at iteration t are updated as follows (Hansen & Ostermeier, 2001; Hansen, 2016),

$$
\begin{aligned}
\boldsymbol{m}^{(t)} &= \boldsymbol{m}^{(t-1)} + c_m \sum_{i=1}^{\mu} w_i \left(\boldsymbol{x}_{i:\lambda}^{(t)} - \boldsymbol{m}^{(t-1)}\right), \\
\boldsymbol{C}^{(t)} &= (1 - c_1 - c_\mu)\, \boldsymbol{C}^{(t-1)} + \frac{c_\mu}{\sigma^{(t-1)}} \sum_{i=1}^{\lambda} w_i \left(\boldsymbol{x}_{i:\lambda}^{(t)} - \boldsymbol{m}^{(t-1)}\right)\left(\boldsymbol{x}_{i:\lambda}^{(t)} - \boldsymbol{m}^{(t-1)}\right)^{\intercal} + \frac{c_1}{\sigma^{(t-1)}} \boldsymbol{p}^{(t)} \boldsymbol{p}^{(t)\intercal}, \\
\sigma^{(t)} &= \beta^{(t)} \sigma^{(t-1)},
\end{aligned}
\tag{4}
$$

where

- $\boldsymbol{p}^{(t)} = \sum_{i=0}^{t} (\boldsymbol{m}^{(i)} - \boldsymbol{m}^{(i-1)})/\sigma^{(i)}$ denotes the evolution path which quantifies the overall movement of the search distribution, i.e., the movement of the mean vector across iterations,
- $\boldsymbol{x}_{i:\lambda}$ denotes the i-th best candidate out of λ data points $\{\boldsymbol{x}_i^{(t)}\}_{i=1}^{\lambda}$ based on their noisy function values, i.e., their corresponding noisy function values satisfy: $y_{1:\lambda}^{(t)} \leq y_{2:\lambda}^{(t)} \leq ... \leq y_{\lambda:\lambda}^{(t)}$,
- $\mu \leq \lambda$ is a hypeparameter denoting the number of data points selected to update the search distribution; by default, μ is usually set as $\lfloor \lambda/2 \rfloor$,

- $\{w_i\}_{i=1}^{\lambda}$ denotes the weight coefficients associated with the data points $\{\boldsymbol{x}_{i:\lambda}^{(t)}\}_{i=1}^{\lambda}$ such that $w_1 \geq w_2 \cdots \geq w_\mu > 0 > w_{\mu+1} \geq \cdots \geq w_\lambda$, $\sum_{i=0}^{\mu} w_i = 1$, and $\sum_{i=0}^{\lambda} w_i \approx 0$,
- c_m, c_1 and c_μ denote the learning rates at which the search distribution changes, where larger rates result in faster change and smaller rates reduce the adaptation rate of search distribution,
- $\beta^{(t)}$ denotes a modification rule for the step size, depending on the evolution path $\boldsymbol{p}^{(t)}$.

Detailed information for the suggested settings of these hyperparameters can be found in Appendix A.1. From Eq. (4), it can be seen that in CMA, the mean vector $\boldsymbol{m}^{(t)}$ is updated based on the previous mean vector $\boldsymbol{m}^{(t-1)}$ and the highest-ranking observed data points $\{\boldsymbol{x}_{i:\lambda}^{(t)}\}_{i=1}^{\mu}$. The covariance matrix $\boldsymbol{C}^{(t)}$ is updated based on the previous covariance matrix $\boldsymbol{C}^{(t-1)}$, all the observed data points $\{\boldsymbol{x}_{i:\lambda}^{(t)}\}_{i=1}^{\lambda}$, and the evolution path $\boldsymbol{p}^{(t)}$ of the search distribution from previous iterations. The step size $\sigma^{(t)}$ is updated based on β, which depends on the overall movement of the search distribution (the evolution path $\boldsymbol{p}^{(t)}$). If the evolution path is short, e.g., when the vectors $\Delta\boldsymbol{m}^{(i)} = \boldsymbol{m}^{(i)} - \boldsymbol{m}^{(i-1)}$ in consecutive iterations cancel each other out, the step size σ is decreased, as the search distribution is likely to start converging toward a solution. On the contrary, when the evolution path is long, e.g., the vectors $\Delta\boldsymbol{m}^{(i)}$ are in the same direction, the step size is increased, as the search distribution is likely to be far away from the true one.

Theoretically, it has been shown that the CMA strategy can be interpreted as a natural gradient learning method that updates the parameters (mean, covariance matrix, step size) of the search distribution $p(\boldsymbol{x})$ to minimize the expected function value $\mathbb{E}[f(\boldsymbol{x})]$ under this distribution (Akimoto et al., 2010; Nomura et al., 2021). With the updates in Eq. (4), CMA tends to maximize the probability of generating successful data points $\{\boldsymbol{x}_{i:\lambda}^{(t)}\}_{i=1}^{\mu}$ (e.g., data points with lower function values for a minimization problem) in the subsequent iterations (Hansen & Auger, 2011). Empirically, as discussed at the beginning of this section, CMA-based ES techniques like `CMA-ES` perform very well in finding the global optimum of high-dimensional optimization problems (Loshchilov & Hutter, 2016; Nomura et al., 2021), demonstrating the effectiveness of CMA in deriving search distributions that can estimate the probabilities of data points being the global optimum of the objective function.

4.2 The CMA-based Meta-algorithm

In this section, we present our proposed CMA-based meta-algorithm. We first discuss the overall process of this meta-algorithm, then we derive three CMA-based BO methods where we integrate the proposed meta-algorithm with the three state-of-the-art BO optimizers (`BO`, `TuRBO`, `BAxUS`).

Overall Process. We illustrate the CMA-based meta-algorithm in Fig. 1 and the pseudocode in Algorithm 1. First, an initial search distribution $\mathcal{N}(\boldsymbol{m}^{(0)}, \sigma^{(0)}\boldsymbol{C}^{(0)})$ is set (line 6), and a local region $\mathcal{S}^{(0)}$ is computed based on this search distribution and the chosen BO optimizer `bo_opt` (line 9). Then the BO optimizer `bo_opt` is used within the local region $\mathcal{S}^{(0)}$ to suggest λ data points to be evaluated (lines 11-14). The search distribution is then updated based on these λ observed data points (line 15). The process is conducted iteratively until the evaluation budget is depleted, and the algorithm terminates. Note that a restart strategy is also included to restart the algorithm when the current optimization process is stuck at a local minimum (line 17).

Local Region Formulation. We first define the base local region $\mathcal{S}_b$ for the CMA-based meta-algorithm. Depending on the employed BO optimizer, we will further derive the local region $\mathcal{S}$ corresponding to that particular BO optimizer in the later sections (Sections 4.2.1, 4.2.2, and 4.2.3). As discussed in Section 4.1, the search distribution by CMA can assign higher probabilities to more promising data points; therefore, we can define the local regions as the regions containing data points with high probability values from this search distribution. As CMA's search distribution is a multivariate normal distribution, we propose defining the local region as the α-level confidence hyper-ellipsoid centered at the mean vector of this search distribution, containing α percent of the population of data points that follows this search distribution. Specifically, at iteration t, given the multivariate normal search distribution $\mathcal{N}(\boldsymbol{m}^{(t-1)}, \boldsymbol{\Sigma}^{(t-1)})$, where $\boldsymbol{\Sigma}^{(t-1)} = \sigma^{(t-1)}\boldsymbol{C}^{(t-1)}$,

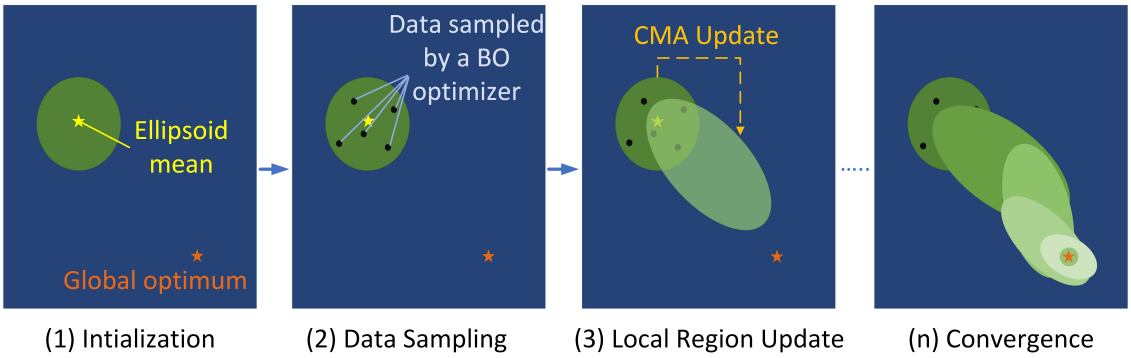

Figure 1: Illustration of the proposed CMA-based meta-algorithm. In step (1), a hyper-ellipsoid local region is initialized. In step (2), a BO optimizer (e.g., `BO`, `TuRBO`, `BAxUS`) is used in this local region to collect a population of candidates (data points to be evaluated). In step (3), the local region is updated using the CMA technique. The process is conducted iteratively until the evaluation budget is depleted.

obtained in the previous iteration, the base hyper-ellipsoid local region $\mathcal{S}_b^{(t)}$ can be computed as,

$$\mathcal{S}_b^{(t)} = \left\{ \boldsymbol{x} \mid \Delta^{(t-1)}(\boldsymbol{x}) \leq \chi^2_{1-\alpha,d} \right\}, \tag{5}$$

where $\Delta^{(t-1)}(\boldsymbol{x}) = \sqrt{\left(\boldsymbol{x} - \boldsymbol{m}^{(t-1)}\right)^\intercal \left(\boldsymbol{\Sigma}^{(t-1)}\right)^{-1} \left(\boldsymbol{x} - \boldsymbol{m}^{(t-1)}\right)}$ is the Mahalanobis distance (Mahalanobis, 1936) from $\boldsymbol{x}$ to the search distribution $\mathcal{N}(\boldsymbol{m}^{(t-1)}, \boldsymbol{\Sigma}^{(t-1)})$ and $\chi^2_{1-\alpha,d}$ is the Chi-squared $1-\alpha$ critical value with d degree of freedom. In our proposed CMA-based meta-algorithm, we set α to be 99.73%, corresponding to the 3-sigma rule that is commonly used in practice. With this setting, the selected observed data in each iteration will always fall within the three standard deviations of the mean vector of the search distribution.

Local Optimization. In each iteration t, given the multivariate normal search distribution $\mathcal{N}(\boldsymbol{m}^{(t-1)}, \boldsymbol{\Sigma}^{(t-1)})$ obtained in the previous iteration, we first sample a pool of data points that follow this search distribution and are within the local region $\mathcal{S}^{(t)}$. Then, we use the employed BO optimizer, `bo_opt`, to select λ data points from this pool of data points. The rationale behind this step is that in the CMA strategy, when updating the search distribution, the λ observed data points are required to be sampled from the previous search distribution (Eq. (4)). In our proposed CMA-based meta-algorithm, we also aim to ensure that the data points chosen by the employed BO optimizer `bo_opt` are drawn from the previous CMA's search distribution. Finally, after obtaining λ observed data points, we update the CMA's search distribution following Eq. (4). It is worth noting that, in our proposed approach, in each iteration, we sample and evaluate λ data points rather than just one as in standard BO methods, i.e., in our algorithm, one iteration is equal to λ iterations in standard BO methods.

Restart Strategy. A local search strategy is typically biased toward the starting point, and the optimization process can be trapped in local minima (Eriksson et al., 2019). To enable global optimization for the CMA-based meta-algorithm, we use the CMA's restart strategy: a new local search will be initialized when the current one is stuck at a local minimum (Auger & Hansen, 2005; Hansen, 2016). The conditions for a restart in CMA normally involve checking if the objective function values are flat for a number of iterations or if some numerical indicators (e.g., condition number) of the search distribution are violated. Furthermore, since some BO optimizers (e.g., `TuRBO`, `BAxUS`) have their own restart strategies, we also incorporate these restart strategies when applying our proposed CMA-based meta-algorithm to the corresponding BO optimizers (detailed information in the subsequent sections).

4.2.1 CMA-BO: CMA-based Meta-algorithm with Standard BO

In this section, we describe `CMA-BO`, the corresponding CMA-based BO method obtained when incorporating the proposed CMA-based meta-algorithm with the standard `BO` optimizer. Note that from this section, we remove the superscript denoting the iteration index for brevity.

Algorithm 1 The CMA-based meta-algorithm.

Input: Objective function $f(.)$, search domain $[l, u]^d$, maximum number of function evaluations N, number of initial points n_0, BO optimizer `bo_opt`
Output: The optimum $\boldsymbol{x}^*$
Set $t \leftarrow 0$, $T \leftarrow \lfloor (N - n_0)/\lambda \rfloor$, global dataset $D \leftarrow \emptyset$, local dataset $\Omega \leftarrow \emptyset$, population size λ
while $t \leq T$ **do**
  Sample n_0 initial data points D_0 ▷ Latin hypercube
  Set the initial search distribution $\mathcal{N}(\boldsymbol{m}^{(t)}, \sigma^{(t)}\boldsymbol{C}^{(t)})$ based on D_0 ▷ Sec. 5.1
  Update $D \leftarrow D \cup D_0$, $\Omega \leftarrow D_0$, `restart` $\leftarrow False$
  while $t \leq T$ **and** not `restart` **do**
    Compute the local region $\mathcal{S}^{(t)}$ from $\mathcal{N}(\boldsymbol{m}^{(t)}, \sigma^{(t)}\boldsymbol{C}^{(t)})$ depending on `bo_opt` ▷ Eqs. (5),(6),(8)
    Initialize a dataset to collect λ observed data points $D_\lambda \leftarrow \emptyset$
    for i=1:λ **do**
      Apply BO optimizer `bo_opt` to propose an observed data $\{\boldsymbol{x}_i, y_i\}$ from a pool of data points
      Update $\Omega \leftarrow \Omega \cup \{\boldsymbol{x}_i, y_i\}$, $D_\lambda \leftarrow D_\lambda \cup \{\boldsymbol{x}_i, y_i\}$
    end for
    Update $\{\boldsymbol{m}^{(t+1)}, \boldsymbol{C}^{(t+1)}, \sigma^{(t+1)}\} \leftarrow \mathrm{CMA}(\{\boldsymbol{m}^{(t)}, \boldsymbol{C}^{(t)}, \sigma^{(t)}\}, D_\lambda)$ ▷ Eq. (4)
    Update $D \leftarrow D \cup D_\lambda$, $t \leftarrow t + 1$
    Update `restart` $\leftarrow True$ **if** stopping criteria satisfied
  end while
end while
Return $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}_i \in D} \{y_i\}_{i=1}^N$ from $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$

Local Region Formulation. For `CMA-BO`, we define the local region $\mathcal{S}$ equal to the base local region $\mathcal{S}_b$ described in Eq. (5), i.e., the local region is the α-level confidence hyper-ellipsoid of the search distribution $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma})$. The value α is also set at 99.73%, corresponding to the 3-sigma rule.

Local Optimization. Following the base algorithm described in Section 4.2, in each iteration, we first sample a pool of data points that (1) follow the previous search distribution $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma})$, and, (2) are within the local region $\mathcal{S}$. Then we sequentially apply BO with the TS acquisition function to select the best λ data points from this pool. Note that when training the GP, as in Eriksson et al. (2019), we also use all the observed data in all the previous iterations. The pseudocode of the local optimization step in `CMA-BO` is in Appendix A.2, Algorithm 2.

Restart Strategy. `CMA-BO` has the same restart strategy with CMA as described in the base algorithm.

4.2.2 CMA-TuRBO: CMA-based Meta-algorithm with TuRBO

We derive `CMA-TuRBO`, the CMA-based BO method obtained when incorporating our CMA-based meta-algorithm with the `TuRBO` optimizer. Compared to `CMA-BO`, `CMA-TuRBO` is slightly different in the local region formulation and restart steps as `TuRBO` has another method to set its local regions and the restart mechanism.

Local Region Formulation. For `CMA-TuRBO`, we incorporate `TuRBO`'s local region adaptation mechanism, which is based on the success and failure state of the optimization process, with the local region strategy defined by CMA. Specifically, with the search distribution $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma})$, we define the local region $\mathcal{S}_{\texttt{CMA-TuRBO}}$ as the hyper-ellipsoid with: (1) the center being at the mean vector $\boldsymbol{m}$, (2) the radii (lengths of the semi-axes of the hyper-ellipsoid) computed based on the covariance matrix $\boldsymbol{\Sigma}$ and scaled with a factor L that is based on the success and failure state of the optimization, similar to the local region adaptation mechanism in `TuRBO`. In particular, L is initially set as 0.8, and after τ_{succ} consecutive success, L is doubled, while it is halved when the optimization fails to progress after τ_{fail} consecutive times. Therefore, compared to the local regions defined by `CMA-BO`, the local regions of `CMA-TuRBO` are scaled based on the historical optimization success record as in `TuRBO`. With a scale factor L, the local regions of `CMA-TuRBO` are defined as follows,

$$\mathcal{S}_{\texttt{CMA-TuRBO}} = \left\{ \boldsymbol{x} \mid \Delta_{\texttt{CMA-TuRBO}}(\boldsymbol{x}) \leq \chi^2_{1-\alpha,d} \right\}, \tag{6}$$

where $\Delta_{\texttt{CMA-TuRBO}}(\boldsymbol{x}) = \sqrt{(\boldsymbol{x}-\boldsymbol{m})^\intercal \boldsymbol{\Sigma}_{\texttt{CMA-TuRBO}}^{-1}(\boldsymbol{x}-\boldsymbol{m})}$ is the Mahalanobis distance from $\boldsymbol{x}$ to the scaled search distribution $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma}_{\texttt{CMA-TuRBO}})$ and $\boldsymbol{\Sigma}_{\texttt{CMA-TuRBO}} = L^2\boldsymbol{\Sigma}$. The derivation of this scaled covariance matrix is as follows. By definition, the radii of the hyper-ellipsoid constructed by $\boldsymbol{\Sigma}$ is $\boldsymbol{r} = \text{diag}(\boldsymbol{\Lambda}^{1/2})$, where $\boldsymbol{\Lambda}$ is the diagonal matrix whose diagonal elements are the eigenvalues of $\boldsymbol{\Sigma}$. Note that, using the eigendecomposition, we have that, $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^{-1}$ with $\boldsymbol{U}$ being the eigenvector matrix. Thus, when scaling the radii of this hyper-ellipsoid by L, i.e., $\boldsymbol{r}_{\texttt{CMA-TuRBO}} = L\boldsymbol{r}$, the covariance matrix $\boldsymbol{\Sigma}_{\texttt{CMA-TuRBO}}$ becomes $\boldsymbol{U}(L^2\boldsymbol{\Lambda})\boldsymbol{U}^{-1} = L^2\boldsymbol{\Sigma}$. Finally, similar to TuRBO, we also set upper and lower bounds for L, i.e., L cannot exceed a threshold $L_{\max}$ and when L becomes smaller than a threshold $L_{\min}$, the algorithm restarts.

Local Optimization. After obtaining the scaled search distribution $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma}_{\texttt{CMA-TuRBO}})$ and the local region $\mathcal{S}_{\texttt{CMA-TuRBO}}$, the optimization process is conducted similarly as in the base algorithm described in Section 4.2. Specifically, we first sample a pool of data points from the search distribution $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma}_{\texttt{CMA-TuRBO}})$, and then apply BO with the TS acquisition function to select λ data points from this pool. Besides, when training the GP, as with Eriksson et al. (2019), we use all the observed data points so far. The pseudocode of the local optimization step in CMA-TuRBO is in Appendix A.2, Algorithm 3.

Restart Strategy. Apart from the restart strategy of CMA, we also employ the restart strategy of TuRBO, i.e., when L shrinks below a minimum threshold $L_{\min}$, we terminate the CMA local region $\mathcal{S}_{\texttt{CMA-TuRBO}}$ and restart it at a new location randomly.

4.2.3 CMA-BAxUS: CMA-based Meta-algorithm with BAxUS

We present CMA-BAxUS, the method resulted when incorporating our proposed CMA-based meta-algorithm with the BAxUS optimizer. Compared to CMA-BO and CMA-TuRBO, CMA-BAxUS is significantly different in the local optimization step as BAxUS optimizes the objective function within the target space (with lower dimension) instead of the original high-dimensional search space.

Local Region Formulation. As described in Section 2.4, the core idea of BAxUS is to perform optimization in the target space $\mathcal{V}$ of dimension $d_{\mathcal{V}}$ rather than in the original search space $\mathcal{X}$ of dimension d ($d \geq d_{\mathcal{V}}$). Therefore, to incorporate BAxUS into our proposed CMA-based meta-algorithm, we need to compute CMA's local regions in the target space $\mathcal{V}$. Note that since BAxUS obtains the objective function evaluations in the original search domain $\mathcal{X}$, thus, using the CMA's update formula in Eq. (4), we can only compute the search distribution in $\mathcal{X}$. To compute the CMA's local regions in the target space $\mathcal{V}$, our main goal is to project the search distribution $\mathcal{N}_{\mathcal{X}}(\boldsymbol{m}_{\mathcal{X}}, \boldsymbol{\Sigma}_{\mathcal{X}})$ from $\mathcal{X}$ to $\mathcal{V}$, and then use the projected search distribution $\mathcal{N}_{\mathcal{V}}(\boldsymbol{m}_{\mathcal{V}}, \boldsymbol{\Sigma}_{\mathcal{V}})$ to construct the local regions. When performing function evaluation in $\mathcal{X}$, BAxUS uses a sparse embedding matrix $\boldsymbol{Q} : \mathcal{V} \to \mathcal{X}$, so that for any vector $\boldsymbol{v} \in \mathcal{V}$, we can compute the corresponding vector $\boldsymbol{x} \in \mathcal{X}$ as $\boldsymbol{x} = \boldsymbol{Q}\boldsymbol{v}$, and thus evaluate the objective function value $f(\boldsymbol{x})$. Our problem is then to find a projection matrix $\boldsymbol{P}$ that maps $\mathcal{X}$ to $\mathcal{V}$. This is basically a linear regression problem, and the solution can be derived as $\boldsymbol{P} = (\boldsymbol{Q}^\intercal\boldsymbol{Q})^{-1}\boldsymbol{Q}^\intercal$ (Golub & Van Loan, 2013). Having defined the linear transformation $\boldsymbol{P} : \mathcal{X} \to \mathcal{V}$, given the search distribution $\mathcal{N}_{\mathcal{X}}(\boldsymbol{m}_{\mathcal{X}}, \boldsymbol{\Sigma}_{\mathcal{X}})$, we can compute the projected search distribution $\mathcal{N}_{\mathcal{V}}(\boldsymbol{m}_{\mathcal{V}}, \boldsymbol{\Sigma}_{\mathcal{V}})$ as follows (Tong, 1990),

$$\begin{aligned} \boldsymbol{m}_{\mathcal{V}} &= \boldsymbol{P}\boldsymbol{m}_{\mathcal{X}}, \\ \boldsymbol{\Sigma}_{\mathcal{V}} &= \boldsymbol{P}\boldsymbol{\Sigma}_{\mathcal{X}}\boldsymbol{P}^\intercal. \end{aligned} \tag{7}$$

Furthermore, note that BAxUS employs TuRBO as their optimizer, so when defining the local regions for BAxUS, we also make use of the local region adaptation mechanism in TuRBO, which is to include a scale factor L to scale the local region based on the success and failure state of the optimization process. With this, the local region $\mathcal{S}_{\mathcal{V},\texttt{CMA-BAxUS}}$ can be defined as,

$$\mathcal{S}_{\mathcal{V},\texttt{CMA-BAxUS}} = \left\{\boldsymbol{v} \mid \Delta_{\texttt{CMA-BAxUS}}(\boldsymbol{v}) \leq \chi^2_{1-\alpha,d}\right\}, \tag{8}$$

where $\Delta_{\texttt{CMA-BAxUS}}(\boldsymbol{v}) = \sqrt{(\boldsymbol{v}-\boldsymbol{m}_{\mathcal{V}})^\intercal \boldsymbol{\Sigma}_{\texttt{CMA-BAxUS}}^{-1}(\boldsymbol{v}-\boldsymbol{m}_{\mathcal{V}})}$ is the Mahalanobis distance from $\boldsymbol{v} \in \mathcal{V}$ to the scaled search distribution $\mathcal{N}_{\mathcal{V}}(\boldsymbol{m}_{\mathcal{V}}, \boldsymbol{\Sigma}_{\texttt{CMA-BAxUS}})$ and $\boldsymbol{\Sigma}_{\texttt{CMA-BAxUS}} = L^2\boldsymbol{\Sigma}_{\mathcal{V}}$.

Local Optimization. After defining the local region $\mathcal{S}_{\mathcal{V},\texttt{CMA-BAxUS}}$ in the target space $\mathcal{V}$, we perform the optimization process similarly to the base algorithm described in Section 4.2, which is to sample a pool of data points from the search distribution $\mathcal{N}_{\mathcal{V}}(\boldsymbol{m}_{\mathcal{V}}, \boldsymbol{\Sigma}_{\texttt{CMA-BAxUS}})$ and use BO with the TS acquisition function to pick λ data points in the target space $\mathcal{V}$. Note that, to make use of the observed data collected in previous target spaces, we employ the same splitting strategy as `BAxUS` to transform the data obtained in previous target spaces into the current target space, and add them to the observed dataset of the current target space. Finally, after collecting λ observed data points in the target space $\mathcal{V}$, we then find the corresponding data points in the original search space $\mathcal{X}$ by using the projection $\boldsymbol{x} = \boldsymbol{Q}\boldsymbol{v}$ and evaluate their objective function values $f(\boldsymbol{x})$. From here, we can update the search distribution of CMA, $\mathcal{N}_{\mathcal{X}}(\boldsymbol{m}_{\mathcal{X}}, \boldsymbol{\Sigma}_{\mathcal{X}})$, using Eq. (4), and then recompute the local region $\mathcal{S}_{\mathcal{V},\texttt{CMA-BAxUS}}$ via the projected search distribution $\mathcal{N}_{\mathcal{V}}(\boldsymbol{m}_{\mathcal{V}}, \boldsymbol{\Sigma}_{\mathcal{V}})$. The pseudocode of the local optimization step in `CMA-BAxUS` is in Appendix A.2, Algorithm 4.

Restart Strategy. Apart from the restart strategy of CMA, we also employ the restart strategy of `BAxUS`. Specifically, when the local region of the largest target dimension shrinks smaller than the minimum threshold, i.e., when $d_{\mathcal{V}} = d$ and $L < L_{\min}$, we restart the embedding with target dimension $d_{\mathcal{V}} = d$ and identity embedding matrix $\boldsymbol{Q} = \boldsymbol{I}_d$.

5 Experiments

5.1 Experimental Setup

We use Matérn 5/2 ARD kernels for the GPs in all methods. The input domains of all problems are scaled to have equal domain lengths in all dimensions as in Loshchilov & Hutter (2016). The output observations are normalized following a Normal distribution $y \sim \mathcal{N}(0, 1)$.

For the hyperparameters of the CMA strategy in all CMA-based BO and ES methods, we set them using the suggested values in Hansen (2016). Specifically, the population size λ is set to be $4 + \lfloor 3 + \ln d \rfloor$. The initial mean vector $\boldsymbol{m}^{(0)}$ is selected by minimizing 20 initial data points following a Latin hypercube sampling (Jones, 2001). The covariance matrix $\boldsymbol{C}^{(0)}$ is initialized with an identity matrix $\boldsymbol{I}_d$, and the initial step size $\sigma^{(0)}$ is set to $0.3(u-l)$ where u, l denote the upper and lower bounds of the search domain $\mathcal{X}$, i.e., $\mathcal{X} = [l, u]^d$.

To ensure fair comparison between the CMA-based BO methods and the corresponding BO optimizers, we set the hyperparameters of the CMA-based BO methods to be the same as those of the corresponding BO optimizers. Specifically, for `BO` and `CMA-BO`, the hyperparameter settings of the GP and the TS acquisition function of these methods are the same. For `TuRBO` and `CMA-TuRBO`, we follow the same setting suggested by `TuRBO` (Eriksson et al., 2019) to set the initial TR base side length L_0, the maximum and minimum TR side lengths $L_{\max}$ and $L_{\min}$, and the success and failure threshold τ_{succ} and τ_{fail}. For `BAxUS` and `CMA-BAxUS`, we also follow the same setting suggested by `BAxUS` (Papenmeier et al., 2022) to set the initial TR base side length L_0, the maximum and minimum TR side lengths $L_{\max}$ and $L_{\min}$, the success and failure thresholds τ_{succ} and τ_{fail}, and the bin size b. All the developed CMA-based BO methods (`CMA-BO`, `CMA-TuRBO`, `CMA-BAxUS`) are implemented using GPyTorch (Gardner et al., 2018) as with `TuRBO` and `BAxUS`. All the Python-based methods are run with the same Python package versions.

5.2 Baselines

We compare our proposed CMA-based meta-algorithm against a comprehensive list of related baselines, including `BO`, `TuRBO` (Eriksson et al., 2019), `BAxUS` (Papenmeier et al., 2022), `LA-MCTS` (Wang et al., 2020), `MCTS-VS` (Song et al., 2022), `CMA-ES` (Hansen & Ostermeier, 2001), `DTS-CMAES` (Bajer et al., 2019), and `BADS` (Acerbi & Ma, 2017). Since `LA-MCTS` and `MCTS-VS` are also meta-algorithms like our proposed method, we therefore compare against the corresponding methods obtained when incorporating these meta-algorithms with the `BO` and `TuRBO` optimizers, resulting in `LAMCTS-TuRBO`, `MCTS-VS-BO`, `MCTS-VS-TuRBO`. Note that neither `LA-MCTS` nor `MCTS-VS` provide guidance on how to incorporate `BAxUS` as a BO optimizer, so we are unable to include the `BAxUS`-based methods with these two meta-algorithms. Besides, we are also unable to include `LAMCTS-BO` as its running time is prohibitively slow on the problems used in this paper (each repeat takes approximately 3 days to run).

To evaluate the baseline methods, we use the implementation and hyperparameter settings provided in the authors' public source code and their respective papers. Note that for `DTS-CMAES` and `BADS`, the authors' implementation source code is in Matlab, so to ensure consistency in the objective function evaluation process with other baselines, we call Python from Matlab to evaluate the objective function values. All the methods are initialized with 20 initial data points and are run for 10 repeats with different random seeds. All experimental results are averaged over these 10 independent runs. We then report the mean and the standard error of the simple regret or the best optimal value found. More details on the implementation of all the baseline methods can be found in the Appendix A.3.

5.3 Synthetic and Real-world Benchmark Problems

To evaluate all methods, we conduct experiments on five synthetic and three real-world benchmark problems.

Synthetic Problems. We use Levy-100D, Alpine-100D, Branin2-500D, and two modified versions, Shifted-Levy-100D and Shifted-Alpine-100D. For Branin2-500D, we use the implementation from Papenmeier et al. (2022); Wang et al. (2016) where the function is created by adding additional 498 dummy dimensions to the original Branin 2D function, resulting in a function with the dimension d to be 500. The Levy-100D and Alpine-100D are common test functions[1] used in BO research, and we set the dimension d to be 100 for each function. Additionally, for Levy-100D and Alpine-100D, we create two new versions, namely Shifted-Alpine-100D and Shifted-Levy-100D, where we shift the global optimum away from the original global optimum by uniformly random shifting, i.e., we set $f_{\text{shifted}}(\boldsymbol{x}) = f_{\text{original}}(\boldsymbol{x}+\boldsymbol{\delta})$ with $\boldsymbol{\delta} = [\delta_1, \ldots, \delta_d] \in [l, u]^{100}$ and $\delta_i \sim \mathcal{U}(l, u)$. The search domains of these two functions, $\mathcal{X} = [l, u]^d$, are kept the same as in the original functions. The motivation behind including these two shifted synthetic problems for evaluation is that, based on our observations, some sparse embedding methods (e.g., `BAxUS`) have considerable advantages when the global optimum is at the center of the search domain, thus, we also evaluate all methods on problems where the global optima are not at the search domain's center.

Real-world Problems. We use the following real-world problems: Half-cheetah-102D, LassoDNA-180D and Rover-100D. For Half-cheetah-102D, we use the same implementation as described in Song et al. (2022), which parameterizes the Half-cheetah-v4 Mujoco environment from the Gym package[2] into a 102D reinforcement learning (RL) problem. The goal of this problem is to optimize the parameters of a linear policy designed to solve the RL task. These Mujoco RL tasks have been used in many works, such as Wang et al. (2020); Nguyen et al. (2020); Song et al. (2022). For LassoDNA-180D, we use the implementation from the Python LassoBench library (Šehić et al., 2022) as in Papenmeier et al. (2022). The problem LassoDNA-180D solves the Least Absolute Shrinkage and Selection Operator (LASSO) problem using the DNA dataset from a microbiology problem. The LassoBench suite has been used in several previous works, such as Papenmeier et al. (2022); Ziomek & Ammar (2023). For Rover-100D, we use the implementation provided by Wang et al. (2018). This problem optimizes the locations of 50 points in a 2D-plane trajectory of a rover, resulting in a 100D benchmark function. The goal is to maximize the reward calculated based on the number of collisions along the rover trajectory. The Rover function has been used in various research works, e.g., Wang et al. (2018); Eriksson et al. (2019); Eriksson & Poloczek (2021); Nguyen et al. (2022).

5.4 Experimental Results

5.4.1 Effectiveness of the CMA-based Meta-algorithm

In this section, we aim to evaluate whether the proposed CMA-based meta-algorithm enhances the performance of the BO optimizers and whether it is better than existing meta-algorithms. Specifically, we compare the performance of the CMA-based BO methods (`CMA-BO`, `CMA-TuRBO`, `CMA-BAxUS`) with the corresponding BO optimizers (`BO`, `TuRBO`, `BAxUS`), as well as other methods created by applying existing meta-algorithms to the associated BO optimizers (`LAMCTS-TuRBO`, `MCTS-VS-BO`, `MCTS-VS-TuRBO`). The experimental results are shown in Fig. 2.

[1] `https://www.sfu.ca/~ssurjano/optimization.html`
[2] `https://www.gymlibrary.dev/index.html`

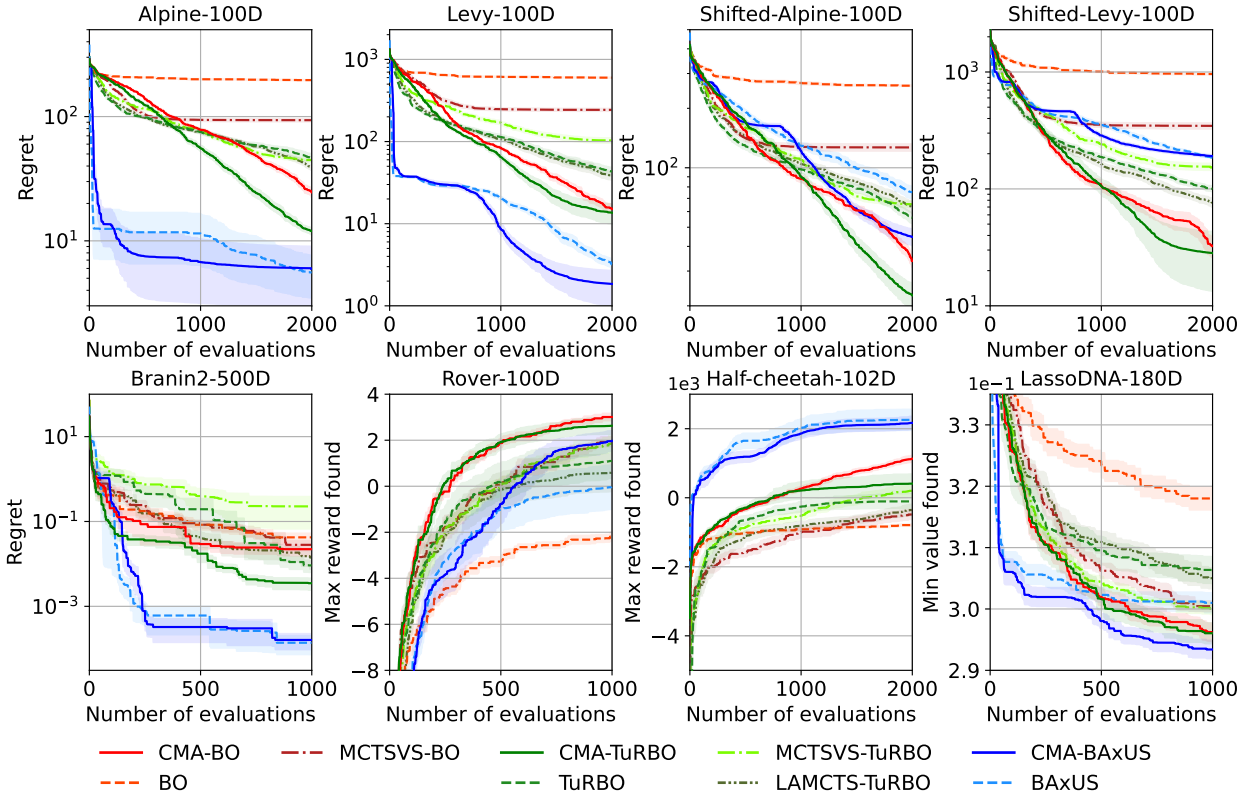

Figure 2: Comparison between the CMA-based BO methods (`CMA-BO`, `CMA-TuRBO`, `CMA-BAxUS`) against (1) the original BO optimizers (`BO`, `TuRBO`, `BAxUS`) and (2) other meta-algorithms (`LA-MCTS`, `MCTS-VS`). Plotting the mean and standard error over 10 repetitions. Red colors represent methods with the `BO` optimizer, green colors are for methods with the `TuRBO` optimizer and blue colors are for methods with the `BAxUS` optimizer. The solid lines (the CMA-based BO methods) are always among the best methods of each color group.

Firstly, it can be clearly seen that our proposed CMA-based meta-algorithm significantly enhances the performance of the corresponding optimizers. `CMA-BO` and `CMA-TuRBO` outperform `BO` and `TuRBO` by a very high margin across all 8 benchmark problems. `CMA-BAxUS` outperforms `BAxUS` significantly on 4 problems (Levy-100D, Shifted-Alpine-100D, Rover-100D, LassoDNA-180D) and performs similarly on 4 problems (Alpine-100D, Shifted-Levy-100D, Branin2-500D, Half-cheetah-102D). Besides, it is worth noting that, in the shifted functions, both the performance of `BAxUS` and `CMA-BAxUS` degrade drastically. This is because the global optimum is not at the center of the search domain and these methods no longer have the advantageous benefit of the sparse embedding technique. However, `CMA-BAxUS` still outperforms `BAxUS` in Shifted-Alpine-100D and has a similar performance in Shifted-Levy-100D.

Secondly, compared to existing meta-algorithms (`LA-MCTS` and `MCTS-VS`), our proposed CMA-based meta-algorithm also outperforms these state-of-the-art meta-algorithms significantly. It can be clearly seen that `CMA-TuRBO` outperforms both `LAMCTS-TuRBO` and `MCTS-VS-TuRBO` by a very high margin on all of the problems. Similarly, `CMA-BO` also outperforms `MCTS-VS-BO` on all of the problems by a very high margin. Note that, as mentioned in Section 5.2, we are unable to include `LAMCTS-BO` due to its prohibitively slow running time (approximately 3 days per one repeat). Furthermore, these meta-algorithms do not suggest on how to incorporate `BAxUS` as a BO optimizer, so we are also unable to compare them with `CMA-BAxUS`.

5.4.2 Comparison against all Baselines

Now, we compare the performance of our proposed CMA-based meta-algorithm with all baselines, including the BO optimizers, the state-of-the-art meta-algorithms, and other related methods such as the CMA-based ES methods (`CMA-ES`, `DTS-CMAES`) and `BADS`, a global optimization method combining `BO` and `CMA-ES`.

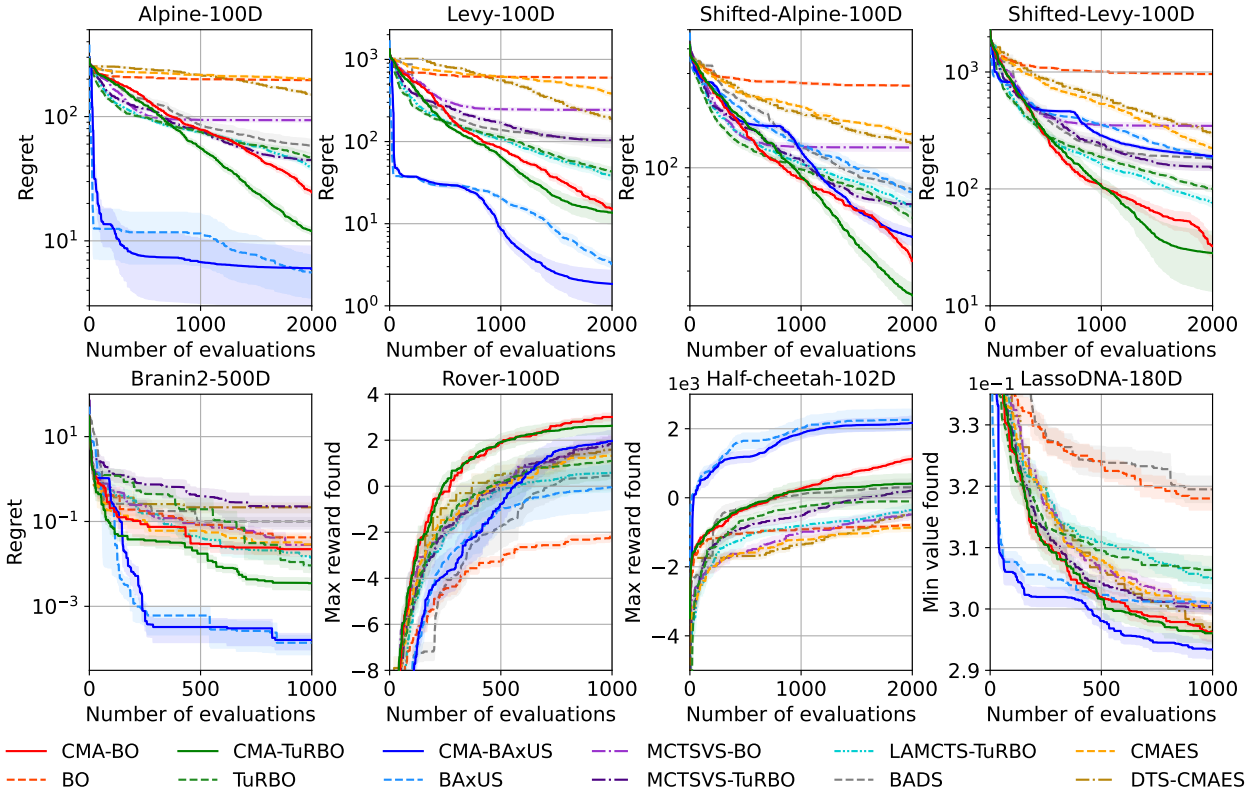

Figure 3: Comparisons among all baselines on all benchmark functions. Plotting the mean and standard error over 10 repetitions. The solid lines represent the CMA-based BO methods (`CMA-BO`, `CMA-TuRBO`, `CMA-BAxUS`). CMA-based BO methods significantly outperform other baselines including the BO optimizers, state-of-the-art meta-algorithms, the CMA-based ES methods (`CMA-ES`, `DTS-CMAES`) and `BADS`, a global optimization method which combines `BO` and `CMA-ES`.

For both synthetic and real-world problems, we can see that for all problems, a CMA-based BO method is always the best method (or one of the best methods). For instance, for Alpine-100D, Levy-100D, Branin2-500D, Half-cheetah-102D, and LassoDNA-180D, `CMA-BAxUS` is the best method (or one of the best methods). In the case of Shifted-Alpine-100D and Shifted-Levy-100D, `CMA-TuRBO` is the best method whilst for Rover-100D, `CMA-BO` is the best method. The variation in the best-performing CMA-based BO method arises because their performance depend on the performance of the corresponding BO optimizers; in some problems, `TuRBO` or `BO` is better whilst in some other problems, `BAxUS` is better. Nevertheless, it is evident that our proposed CMA-based meta-algorithm can significantly enhance the performance of these state-of-the-art BO optimizers, enabling them to perform as the best or among the best across all the problems.

It is worth mentioning that these results also demonstrate that the CMA-based BO methods outperform the related CMA-based ES methods in the EA literature such as `CMA-ES` and `DTS-CMAES`. This improvement could be attributed to the use of BO optimizers to select data points for the CMA strategy, instead of randomly sampling, as is the case with these evolutionary algorithms. This approach makes the CMA-based BO methods to be more data-efficient. Other global optimization methods like `BADS` seem to struggle with the high-dimensional optimization problems with limited data and perform poorly on most of our problems.

5.5 The Trajectory of the Local Regions by the CMA-based Meta-algorithm

We conduct a study to understand the trajectories of the local regions defined by the CMA-based meta-algorithm in various 2D problems. Note that we only plot the local regions for two derived CMA-based BO methods, `CMA-BO` and `CMA-TuRBO`, as the behaviour of `CMA-BAxUS` in 2D problems is similar to that of

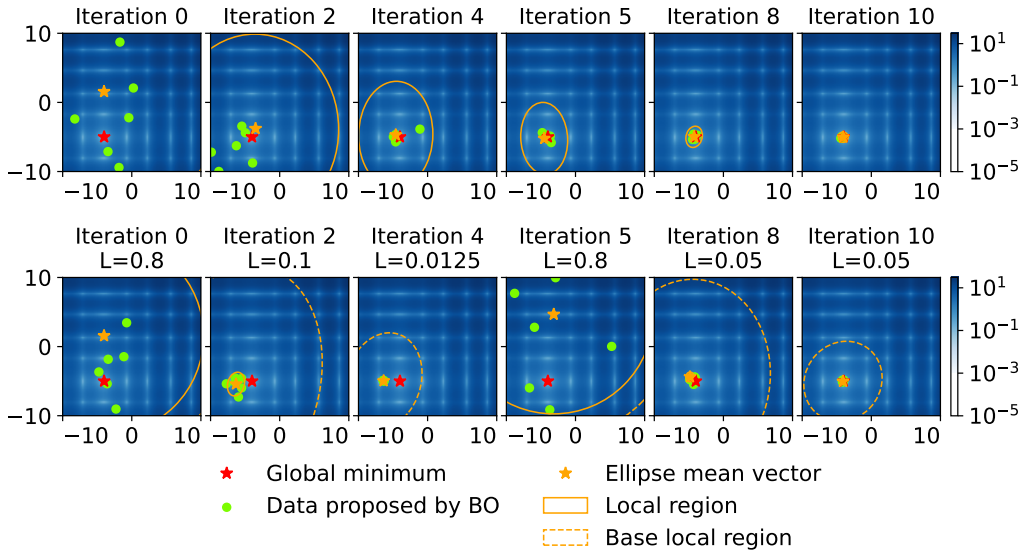

Figure 4: The trajectories of the local regions defined by the proposed CMA-based meta-algorithm when paired with `BO` (upper row) and `TuRBO` (lower row) for the Shifted-Alpine-2D function. In `CMA-BO`, the local regions are the same as the base local regions. In `CMA-TuRBO`, the local regions are the base local regions scaled with L (in terms of radii). In this case, the local regions of `CMA-BO` gradually move towards the global minimum of the objective function whilst the local regions of `CMA-TuRBO` quickly converge to a sub-optimal location, then restart and move toward to the global optimum.

`CMA-TuRBO`. In Fig. 4, we show the local trajectories for the problem Shifted-Alpine-2D. Additional results for all other synthetic problems can be found in Appendix A.4.

We can see that at the beginning (Iteration 0), when the prior information is insufficient for BO, the selected data points scatter randomly throughout the search domain. In the later iterations, owing to the use of `BO` or `TuRBO` combining with the local regions defined by the CMA strategy, the selected data points converge closer to the global optimum. Note that in these plots, both `CMA-BO` and `CMA-TuRBO` start with the same CMA's search distribution, however, the local regions in `CMA-TuRBO` are further scaled by a factor of L compared to the base local region due to its local region adaptation mechanism. In the example plotted here, the local regions of `CMA-BO` gradually shrink toward the global optimum whilst the local regions of `CMA-TuRBO` shrink much faster, converge to a local optimum at Iteration 4 (with $L = 0.0125$), then restart and ultimately converge to the global optimum.

6 Conclusion

In this paper, we propose a novel CMA-based meta-algorithm to address the high-dimensional BO problem by incorporating a local search strategy and the CMA strategy to enhance the performance of existing BO methods. We further derive the CMA-based BO algorithms for the cases in which our proposed meta-algorithm is applied to some common state-of-the-art BO optimizers such as `BO`, `TuRBO`, and `BAxUS`. Our extensive experimental results demonstrate the effectiveness and efficiency of the proposed CMA-based meta-algorithm, which can significantly improve the BO optimizers and outperform other state-of-the-art meta-algorithms and related methods.

References

Abbas Abdolmaleki, Bob Price, Nuno Lau, Luís Paulo Reis, and Gerhard Neumann. Deriving and improving CMA-ES with information geometric trust regions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 657–664. ACM, 2017. doi: 10.1145/3071178.3071252.

Luigi Acerbi and Wei Ji Ma. Practical bayesian optimization for model fitting with bayesian adaptive direct search. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Youhei Akimoto, Yuichi Nagata, Isao Ono, and Shigenobu Kobayashi. Bidirectional Relation between CMA Evolution Strategies and Natural Evolution Strategies. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I*, pp. 154–163, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3642158439.

Charles Audet and J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

Anne Auger and Nikolaus Hansen. A restart cma evolution strategy with increasing population size. In *2005 IEEE congress on evolutionary computation*, volume 2, pp. 1769–1776. IEEE, 2005.

Lukáš Bajer, Zbyněk Pitra, Jakub Repický, and Martin Holeňa. Gaussian process surrogate models for the cma-es. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 17–18. Association for Computing Machinery, 2019. ISBN 9781450367486. doi: 10.1145/3319619.3326764.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24*, pp. 2546–2554, 2011.

Mickael Binois and Nathan Wycoff. A survey on high-dimensional gaussian process modeling with application to bayesian optimization. *ACM Transactions on Evolutionary Learning and Optimization*, 2(2):1–26, 2022.

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, 2010.

Dirk Buche, Nicol N Schraudolph, and Petros Koumoutsakos. Accelerating evolutionary algorithms with gaussian process fitness function models. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(2):183–194, 2005.

Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian optimization for learning gaits under uncertainty: An experimental comparison on a dynamic bipedal walker. *Annals of Mathematics and Artificial Intelligence*, 76:5–23, 2016.

David Eriksson and Matthias Poloczek. Scalable Constrained Bayesian Optimization. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130, pp. 730–738. PMLR, 2021.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable Global Optimization via Local Bayesian Optimization. In *Advances in Neural Information Processing Systems*, pp. 5496–5507, 2019.

Peter I. Frazier, Warren B. Powell, and Savas Dayanik. The Knowledge-Gradient Policy for Correlated Normal Beliefs. *INFORMS J. Comput.*, 21(4):599–613, 2009. doi: 10.1287/ijoc.1080.0314.

Lukas P. Fröhlich, Melanie N. Zeilinger, and Edgar D. Klenske. Cautious bayesian optimization for efficient and scalable policy search. In *Proceedings of the 3rd Annual Conference on Learning for Dynamics and Control, L4DC 2021, 7-8 June 2021, Virtual Event, Switzerland*, volume 144 of *Proceedings of Machine Learning Research*, pp. 227–240, 2021.

Jacob R. Gardner, Chuan Guo, Kilian Q. Weinberger, Roman Garnett, and Roger B. Grosse. Discovering and Exploiting Additive Structure for Bayesian Optimization. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pp. 1311–1319. PMLR, 2017.

Jacob R. Gardner, Geoff Pleiss, Kilian Q. Weinberger, David Bindel, and Andrew Gordon Wilson. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. In *Advances in Neural Information Processing Systems 31*, pp. 7587–7597, 2018.

Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

Roman Garnett, Michael A. Osborne, and Philipp Hennig. Active learning of linear embeddings for gaussian processes. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, pp. 230–239. AUAI Press, 2014.

Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial, 2016.

Nikolaus Hansen and Anne Auger. CMA-ES: Evolution Strategies and Covariance Matrix Adaptation. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, pp. 991–1010. Association for Computing Machinery, 2011. ISBN 9781450306904. doi: 10.1145/2001858.2002123.

Nikolaus Hansen and Andreas Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398.

José Miguel Hernández-Lobato, James Requeima, Edward O. Pyzer-Knapp, and Alán Aspuru-Guzik. Parallel and Distributed Thompson Sampling for Large-scale Accelerated Exploration of Chemical Space. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 1470–1479. PMLR, 2017.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization - 5th International Conference*, volume 6683, pp. 507–523. Springer, 2011. doi: 10.1007/978-3-642-25566-3_40.

Rodolphe Jenatton, Cedric Archambeau, Javier González, and Matthias Seeger. Bayesian Optimization with Tree-structured Dependencies. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 1655–1664. PMLR, 2017.

Donald R. Jones. A Taxonomy of Global Optimization Methods Based on Response Surfaces. *J. Glob. Optim.*, 21(4):345–383, 2001. doi: 10.1023/A:1012771025575.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *J. Glob. Optim.*, 13(4):455–492, 1998. doi: 10.1023/A:1008306431147.

Kirthevasan Kandasamy, Jeff G. Schneider, and Barnabás Póczos. High Dimensional Bayesian Optimisation and Bandits via Additive Models. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pp. 295–304. JMLR, 2015.

Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabas Poczos. Parallelised bayesian optimisation via thompson sampling. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 84 of *Proceedings of Machine Learning Research*, pp. 133–142, 2018a.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric P. Xing. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. In *Advances in Neural Information Processing Systems*, pp. 2020–2029, 2018b.

H. J. Kushner. A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. *Journal of Basic Engineering*, 86(1):97–106, 1964. ISSN 0021-9223. doi: 10.1115/1.3653121.

Ben Letham, Roberto Calandra, Akshara Rai, and Eytan Bakshy. Re-examining linear embeddings for high-dimensional bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 1546–1558, 2020.

Ilya Loshchilov and Frank Hutter. CMA-ES for Hyperparameter Optimization of Deep Neural Networks, 2016.

Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The Application of Bayesian Methods for Seeking the Extremum. *Towards Global Optimization*, 2(117-129):2, 1978.

Sarah Müller, Alexander von Rohr, and Sebastian Trimpe. Local policy search with Bayesian optimization. In *Advances in Neural Information Processing Systems 34*, pp. 20708–20720, 2021.

Rémi Munos. Optimistic Optimization of a Deterministic Function without the Knowledge of its Smoothness. In *Advances in Neural Information Processing Systems 24*, pp. 783–791, 2011.

Amin Nayebi, Alexander Munteanu, and Matthias Poloczek. A Framework for Bayesian Optimization in Embedded Subspaces. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 4752–4761. PMLR, 2019.

Quan Nguyen, Kaiwen Wu, Jacob Gardner, and Roman Garnett. Local Bayesian optimization via maximizing probability of descent. In *Advances in Neural Information Processing Systems*, volume 35, pp. 13190–13202. Curran Associates, Inc., 2022.

Vu Nguyen, Sebastian Schulze, and Michael Osborne. Bayesian optimization for iterative learning. *Advances in Neural Information Processing Systems*, 33:9361–9371, 2020.

Masahiro Nomura, Shuhei Watanabe, Youhei Akimoto, Yoshihiko Ozaki, and Masaki Onishi. Warm starting cma-es for hyperparameter optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9188–9196, 2021.

ChangYong Oh, Efstratios Gavves, and Max Welling. BOCK: Bayesian Optimization with Cylindrical Kernels. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 3865–3874. PMLR, 2018.

Leonard Papenmeier, Luigi Nardi, and Matthias Poloczek. Increasing the scope as you learn: Adaptive bayesian optimization in nested subspaces. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. Automated Reinforcement Learning (AutoRL): A Survey and Open Problems. *J. Artif. Intell. Res.*, 74:517–568, 2022. doi: 10.1613/jair.1.13596.

Santu Rana, Cheng Li, Sunil Gupta, Vu Nguyen, and Svetha Venkatesh. High dimensional bayesian optimization with elastic gaussian process. In *International conference on machine learning*, pp. 2883–2891. PMLR, 2017.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006. ISBN 026218253X.

Kenan Šehić, Alexandre Gramfort, Joseph Salmon, and Luigi Nardi. Lassobench: A high-dimensional hyperparameter optimization benchmark suite for lasso. In *International Conference on Automated Machine Learning*, pp. 2–1. PMLR, 2022.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE*, 104(1):148–175, 2016. doi: 10.1109/JPROC.2015.2494218.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable Bayesian Optimization Using Deep Neural Networks. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pp. 2171–2180. JMLR.org, 2015.

Lei Song, Ke Xue, Xiaobin Huang, and Chao Qian. Monte carlo tree search based variable selection for high dimensional bayesian optimization. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=SUzPos_pUC`.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian Optimization with Robust Bayesian Neural Networks. In *Advances in Neural Information Processing Systems 29*, pp. 4134–4142, 2016.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias W. Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 1015–1022. Omnipress, 2010.

William R Thompson. On the likelihood that on unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933. ISSN 0006-3444. doi: 10.1093/biomet/25.3-4.285.

Yung Liang Tong. *Fundamental properties and sampling distributions of the multivariate normal distribution.* Springer, 1990.

Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track*, volume 133, pp. 3–26. PMLR, 2020.

Xingchen Wan, Vu Nguyen, Huong Ha, Binxin Ru, Cong Lu, and Michael A Osborne. Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. *International Conference on Machine Learning (ICML) 38*, 2021.

Xingchen Wan, Cong Lu, Jack Parker-Holder, Philip J Ball, Vu Nguyen, Binxin Ru, and Michael Osborne. Bayesian generational population-based training. In *International Conference on Automated Machine Learning*, pp. 14–1. PMLR, 2022.

Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search. In *Advances in Neural Information Processing Systems*, volume 33, pp. 19511–19522. Curran Associates, Inc., 2020.

Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. Batched large-scale Bayesian optimization in high-dimensional spaces. In *International Conference on Artificial Intelligence and Statistics*, pp. 745–754. PMLR, 2018.

Ziyu Wang, Babak Shakibi, Lin Jin, and Nando de Freitas. Bayesian Multi-Scale Optimistic Optimization. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33, pp. 1005–1014. JMLR.org, 2014.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando De Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

Ya-xiang Yuan. A review of trust region algorithms for optimization. In *Iciam*, volume 99, pp. 271–282, 2000.

Juliusz Krzysztof Ziomek and Haitham Bou Ammar. Are random decompositions all we need in high dimensional bayesian optimisation? In *International Conference on Machine Learning*, pp. 43347–43368. PMLR, 2023.

A Appendix

A.1 Additional Information for the CMA Update Formula

Here, we provide additional information about how to set the hyperparmeters for the CMA formula in Eq. (4) based on Hansen & Ostermeier (2001). Given the problem dimension as d and the population size $\lambda = 4 + \lfloor 3 + \ln d \rfloor$, let us define some additional terms as,

$$\begin{aligned} w'_i &= \ln \frac{\lambda+1}{2} - \ln i, \text{ for } i = 1, \dots, \lambda, \\ \mu_{\text{eff}} &= \frac{(\sum_{i=1}^{\mu} w'_i)^2}{\sum_{i=1}^{\mu} {w'_i}^2}, \\ \mu_{\text{eff}}^- &= \frac{(\sum_{i=\mu+1}^{\lambda} w'_i)^2}{\sum_{i=\mu+1}^{\lambda} {w'_i}^2}. \end{aligned} \tag{9}$$

The learning rates coefficient are as follows,

$$\begin{aligned} c_m &= 1, \\ c_1 &= \frac{2}{(d+1.3)^2 + \mu_{\text{eff}}}, \\ c_\mu &= \min\left(1 - c_1, 2\frac{\mu_{\text{eff}} - 2 + 1/\mu_{\text{eff}}}{(d+2)^2 + \mu_{\text{eff}}}\right). \end{aligned} \tag{10}$$

The weight coefficients w_i is set as,

$$w_i = \begin{cases} \frac{1}{\sum_{i=1}^{\mu} w'_i} w'_i & \text{if } w'_i > 0, \\ \min\left(1 + \frac{c_1}{c_\mu}, 1 + \frac{2\mu_{\text{eff}}^-}{2+\mu_{\text{eff}}}, \frac{1-c_1-c\mu}{nc_\mu}\right) \frac{1}{\sum_{i=\mu+1}^{\lambda} w'_i} w'_i & \text{if } w'_i < 0. \end{cases} \tag{11}$$

Regarding the the covariance matrix update (second line in Eq. (4)), in practice, the evolution path $\boldsymbol{p}^{(t)} = \sum_{i=0}^{t} (\boldsymbol{m}^{(i)} - \boldsymbol{m}^{(i-1)})/\sigma^{(i)}$ is computed via exponential smoothing. Initialized with $\boldsymbol{p}^{(0)} = 0$, the exact formula of the evolution path is as follows,

$$\boldsymbol{p}^{(t)} = (1 - c_c)\boldsymbol{p}^{(t-1)} + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}} \frac{\boldsymbol{m}^{(t)} - \boldsymbol{m}^{(t-1)}}{\sigma^{(t-1)}}, \tag{12}$$

where

$$c_c = \frac{4 + \mu_{\text{eff}}/d}{d + 4 + 2\mu_{\text{eff}}/d}. \tag{13}$$

Regarding the β coefficient in the step size update (third line in Eq. (4)), the exact formula is as follows,

$$\beta^{(t)} = \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}_\sigma^{(t)}\|}{\sqrt{d}} - 1\right)\right), \tag{14}$$

where

$$\begin{aligned} c_\sigma &= \frac{2 + \mu_{\text{eff}}}{d + 5 + 2\mu_{\text{eff}}}, \\ d_\sigma &= 1 + 2\max\left(0, \sqrt{\frac{\mu_{\text{eff}-1}}{d+1}} - 1\right) + c_\sigma, \\ \boldsymbol{p}_\sigma^{(t)} &= (1 - c_c)\boldsymbol{p}_\sigma^{(t-1)} + \sqrt{c_c(2 - c_c)\mu_{\text{eff}}} \boldsymbol{C}^{(t-1)^{-\frac{1}{2}}} \frac{\boldsymbol{m}^{(t)} - \boldsymbol{m}^{(t-1)}}{\sigma^{(t-1)}}, \text{ with } \boldsymbol{p}_\sigma^{(0)} = 0. \end{aligned} \tag{15}$$

A.2 Pseudocode of the CMA-based BO Algorithms

We present the pseudocode for the local optimization steps of `CMA-BO`, `CMA-TuRBO` and `CMA-BAxUS`. These are the detailed implementation of line 12 in Algorithm 1 depending on the BO optimizer `bo_opt`. Note that, as discussed in the base algorithm in Section 4.2, when performing the local optimization step, we first need to sample a pool of n_c data points following the previous search distribution, then perform BO to select the data points. In practice, we set $n_c = \min(100d_c, 5000)$ where $d_c = d$, the dimensionality of the problem, for `CMA-BO` and `CMA-TuRBO` or $d_c = d_{\mathcal{V}}$, the current target dimensionality, for `CMA-BAxUS`. This value of n_c is the same as in `TuRBO` when selecting sampling data points for the TS acquisition function.

Algorithm 2 Local Optimization for `CMA-BO`.

1: **Input:** Objective function $f(.)$, search distribution $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma})$, local region $\mathcal{S}$, dataset Ω, number of sampling points n_c
2: **Output:** A new observed data $\{\boldsymbol{x}_{\text{next}}, y_{\text{next}}\}$
3: Train a GP from Ω
4: Sample n_c data points $\mathcal{A} = \{\boldsymbol{x}_j\}_{j=1}^{n_c}$ from $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma})$ and constrained within $\mathcal{S}$
5: Propose a next observed data $\boldsymbol{x}_{\text{next}} = \arg\min_{\boldsymbol{x}\in\mathcal{A}} \alpha^{\text{TS}}(\boldsymbol{x})$
6: Evaluate the observed data $y_{\text{next}} = f(\boldsymbol{x}_{\text{next}}) + \varepsilon$
7: Return $\{\boldsymbol{x}_{\text{next}}, y_{\text{next}}\}$

Algorithm 3 Local Optimization for `CMA-TuRBO`.

1: **Input:** Objective function $f(.)$, search distribution $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma}_{\texttt{CMA-TuRBO}})$, local region $\mathcal{S}_{\texttt{CMA-TuRBO}}$, dataset Ω, number of sampling points n_c
2: **Output:** A new observed data $\{\boldsymbol{x}_{\text{next}}, y_{\text{next}}\}$
3: Train a GP from Ω
4: Sample n_c data points $\mathcal{A} = \{\boldsymbol{x}_j\}_{j=1}^{n_c}$ from $\mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma}_{\texttt{CMA-TuRBO}})$ and constrained within $\mathcal{S}_{\texttt{CMA-TuRBO}}$
5: Propose a next observed data $\boldsymbol{x}_{\text{next}} = \arg\min_{\boldsymbol{x}\in\mathcal{A}} \alpha^{\text{TS}}(\boldsymbol{x})$
6: Evaluate the observed data $y_{\text{next}} = f(\boldsymbol{x}_{\text{next}}) + \varepsilon$
7: Return $\{\boldsymbol{x}_{\text{next}}, y_{\text{next}}\}$

Algorithm 4 Local Optimization for `CMA-BAxUS`.

1: **Input:** Objective function $f(.)$, search distribution $\mathcal{N}_{\mathcal{V}}(\boldsymbol{m}_{\mathcal{V}}, \boldsymbol{\Sigma}_{\texttt{CMA-BAxUS}})$, local region $\mathcal{S}_{\mathcal{V},\texttt{CMA-BAxUS}}$, dataset Ω, number of sampling points n_c, embedding matrix $\boldsymbol{Q} : \mathcal{V} \to \mathcal{X}$
2: **Output:** A new observed data $\{\boldsymbol{x}_{\text{next}}, \boldsymbol{v}_{\text{next}}, y_{\text{next}}\}$ in both $\mathcal{X}$ and $\mathcal{V}$
3: Train a GP from $\{\boldsymbol{v}_i, y_i\}_{i=1}^{|\Omega|} \in \Omega$
4: Sample n_c data points $\mathcal{A} = \{\boldsymbol{v}_j\}_{j=1}^{n_c}$ from $\mathcal{N}(\boldsymbol{m}_{\mathcal{V}}, \boldsymbol{\Sigma}_{\texttt{CMA-BAxUS}})$ and constrained within $\mathcal{S}_{\mathcal{V},\texttt{CMA-BAxUS}}$
5: Propose a next observed data $\boldsymbol{v}_{\text{next}} = \arg\min_{\boldsymbol{v}\in\mathcal{A}} \alpha^{\text{TS}}(\boldsymbol{v})$ and $\boldsymbol{x}_{\text{next}} = \boldsymbol{Q}\boldsymbol{v}_{\text{next}}$
6: Evaluate the observed data $y_{\text{next}} = f(\boldsymbol{x}_{\text{next}}) + \varepsilon$
7: Return $\{\boldsymbol{x}_{\text{next}}, \boldsymbol{v}_{\text{next}}, y_{\text{next}}\}$

A.3 Detailed Implementation of the Baselines

Details of the implementation for each baselines in the paper are as follows.

BO. This is the standard `BO` method with the TS acquisition function. The GP is constructed with the Matérn 5/2 ARD kernel and is fitted using the Maximum Likelihood method. The domains of the input variables in all problems are scaled to have equal domain lengths in all dimensions (Loshchilov & Hutter, 2016). The output observations $\{y_i\}$ are normalized following a Normal distribution $\mathcal{N}(0, 1)$.

TuRBO (Eriksson et al., 2019). We set all the hyperparameters of `TuRBO` as suggested in their paper. This includes the upper and lower bound for TR side length $L_{\max} = 1.6$, $L_{\min} = 2^{-7}$, batch size $b = 1$ and

the TR adaptation threshold $\tau_{\text{succ}} = 3$, $\tau_{\text{fail}} = \lceil \max(4/b, d/b) \rceil$ where d is the dimension of the problem. We use their implementation that is made available at `https://github.com/uber-research/TuRBO`.

BAxUS (Papenmeier et al., 2022). We set all the hyperparameters of `BAxUS` as suggested in their paper. This includes the upper and lower bound for TR side length $L_{\max} = 1.6$, $L_{\min} = 2^{-7}$, the TR adaptation threshold $\tau_{\text{succ}} = 3$ and bin size $b = 3$, budget to input dim m_D is set to the maximum budget. We use their implementation that is made available at `https://github.com/LeoIV/BAxUS`.

LA-MCTS (Wang et al., 2020). We set all the hyperparameters of `LA-MCTS` as suggested in their paper. This includes the exploration factor in UCB $C_p = 1$, the kernel type of SVM is RBF and the splitting threshold $\theta = 20$. For Levy function, we use different settings, which is recommended in the author's implementation code[3], i.e., $C_p = 10$, polynomial kernel, $\theta = 8$. We use their implementation that is made available at `https://github.com/facebookresearch/LaMCTS`.

MCTSVS (Song et al., 2022). We set all the hyperparameters of `MCTS-VS` as suggested in their paper. This includes the exploration factor in UCB $C_p = 1$, the fill-in strategy of "best-k" with $k = 20$, the feature batch size $N_v = 2$, the sample batch size $N_s = 3$, the tree re-initialization threshold $N_{bad} = 5$, the node splitting threshold $N_{split} = 3$. We use their implementation that is made available at `https://github.com/lamda-bbo/MCTS-VS`.

CMAES (Hansen & Ostermeier, 2001). We use the default settings as suggested in the paper, which is similar to our settings for `CMA-BO`. This includes the population size $\lambda = 4 + \lfloor 3 + \ln d \rfloor$ where d is the problem dimension, the random initial mean vector $\boldsymbol{m}^{(0)}$ selected from the minimum of the 20 random initial points, the identity initial covariance matrix $\boldsymbol{C}^{(0)} = \boldsymbol{I}_d$ and the initial step-size $\sigma^{(0)} = 0.3(u - lb)$ where the domain is scaled to uniform bound of $[l, u]^d$. We also activate the restart mechanism of `CMA-ES` so that the algorithm can restart when it converges to a local minimum. We use their implementation that is made available at `https://github.com/CMA-ES/pycma`.

DTS-CMAES (Bajer et al., 2019). We set all the hyperparameters of `DTS-CMAES` as suggested in their paper. We use the doubly-trained GP configuration with the population size $\lambda = 8 + \lfloor 6 + \ln d \rfloor$ where d is the problem dimension, initial mean vector $\boldsymbol{m}^{(0)}$ selected from the minimum of the 20 random initial points, the identity initial covariance matrix $\boldsymbol{C}^{(0)} = \boldsymbol{I}_d$ and the initial step-size $\sigma^{(0)} = 0.3(u - lb)$ where the domain is scaled to uniform bound of $[l, u]^d$ and fixed learning rate $\beta = 0.05$. We use their implementation that is made available at `https://github.com/bajeluk/surrogate-cmaes`.

BADS (Acerbi & Ma, 2017) We set all the hyperparameters of `BADS` as suggested in their paper and the Matlab package. We use their implementation that is made available at `https://github.com/acerbilab/bads`.

A.4 Additional Trajectory Plots of the Local Regions by the CMA-based Meta-algorithm

We show the remaining trajectory plots of the local regions defined by `CMA-BO` and `CMA-TuRBO` in the remaining 4 synthetic functions: Alpine-2D, Levy-2D, Branin-2D and Shifted-Levy-2D.

[3] `https://github.com/facebookresearch/LaMCTS/blob/489bd60886f23b0b76b10aa8602ea6722f334ad6/LA-MCTS/functions/functions.py`

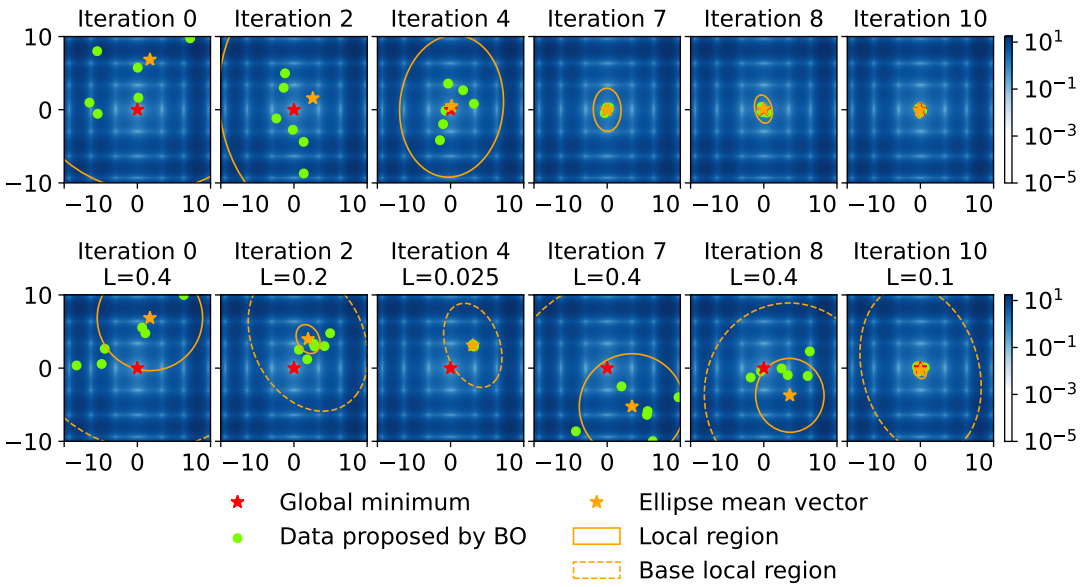

Figure 5: Trajectories of the local regions defined by CMA-based BO methods, `CMA-BO` (upper) and `CMA-TuRBO` (lower), for Alpine 2D function. The function global optimum is at $[0, 0]$.

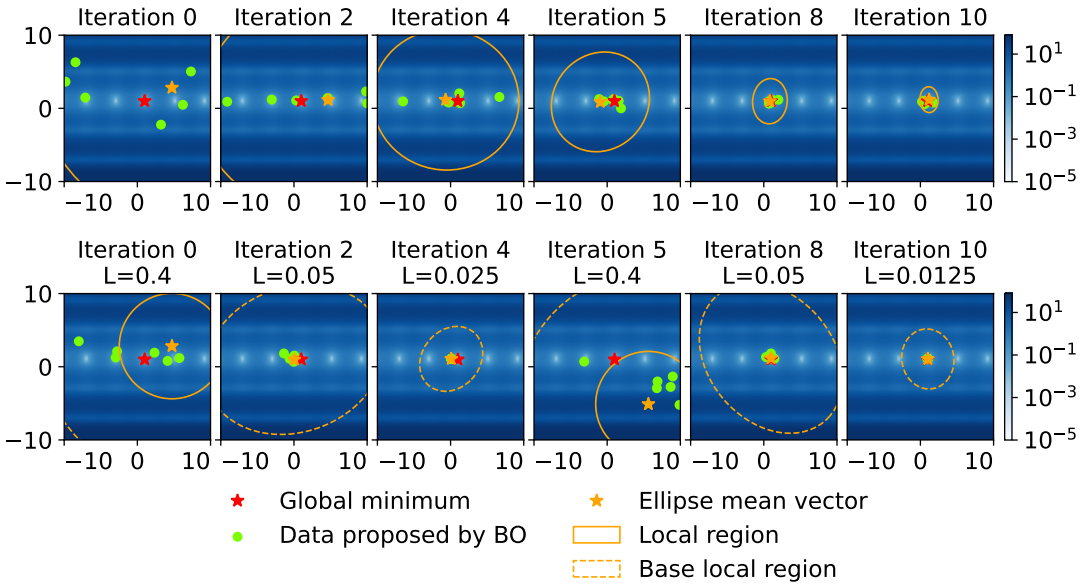

Figure 6: Trajectories of the local regions defined by CMA-based BO methods, `CMA-BO` (upper) and `CMA-TuRBO` (lower), for Levy 2D function. The function global optimum is at $[1, 1]$.

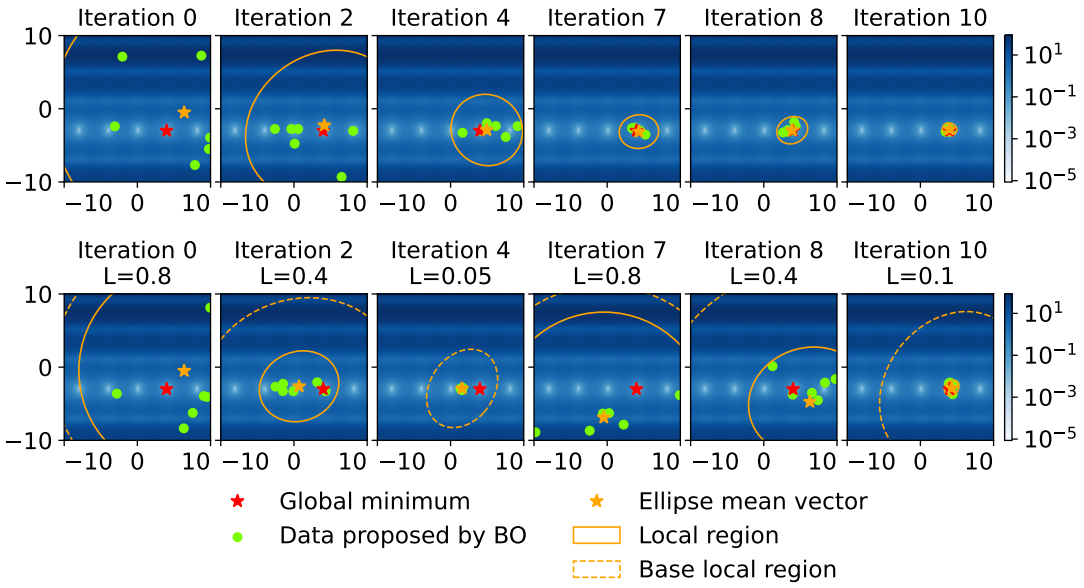

Figure 7: Trajectories of the local regions defined by CMA-based BO methods, `CMA-BO` (upper) and `CMA-TuRBO` (lower), for Shifted-Levy 2D function. The function global optimum is at $[3, -4]$.

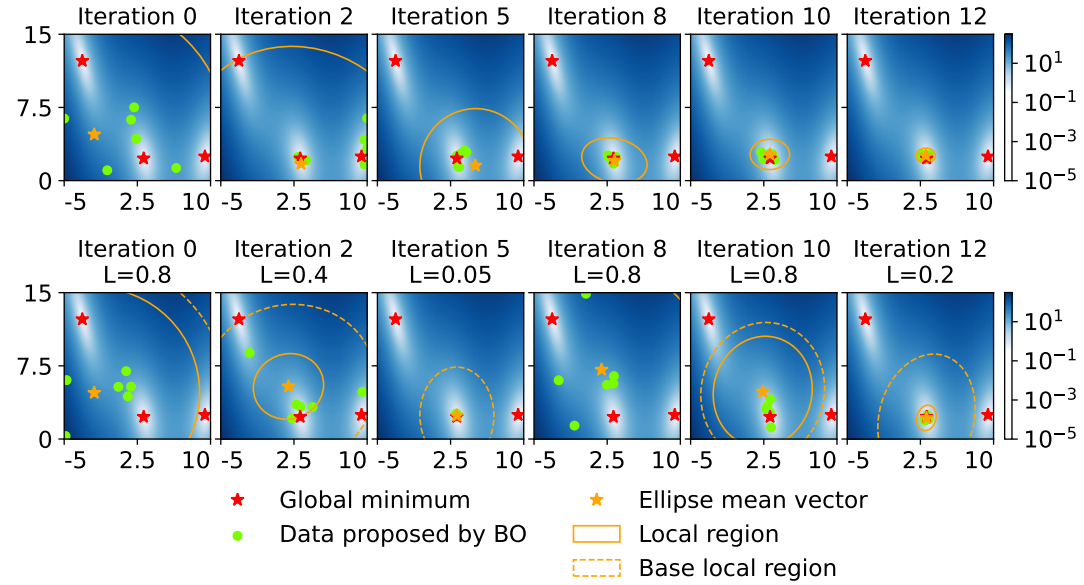

Figure 8: Trajectories of the local regions defined by CMA-based BO methods, `CMA-BO` (upper) and `CMA-TuRBO` (lower), for Branin 2D function. The function has 3 global optima at $[-\pi, 12.275]$, $[\pi, 2.275]$ and $[9.42478, 2.475]$.