Monte Carlo Tree Descent for Black-Box Optimization

Anonymous Author(s)
Affiliation
Address
email

Abstract

The key to black-box optimization is the efficient search among regions with widely-varying numerical properties to achieve low-regret descent and fast progress toward the optimum. Monte Carlo Tree Search (MCTS) methods have recently been introduced to improve Bayesian optimization (BO) by computing better partitioning of the search space and balancing exploration and exploitation. Extending this promising framework, we study how to integrate sample-based descent with BO for faster optimization. At the vertices of the search trees, we first introduce new descent methods that incorporate stochastic and direct search. We then design novel ways of balancing progress and uncertainty, and propose new branch selection, tree expansion, and backpropagation policies. Overall, the proposed MCTS puts more emphasis on sampling for faster descent, and uses localized Gaussian Processes as auxiliary metrics in both exploitation and exploration. We show experimentally that the proposed algorithms can outperform state-of-the-art methods on many challenging benchmark problems.

1 Introduction

Black-box optimization (BBO) problems assume that the objective function is not known analytically and can be evaluated at arbitrary inputs, potentially with high evaluation cost and stochastic outcomes. The generality of this formulation makes it applicable to a wide range of challenging problems in machine learning Zoph & Le (2016); Buşoniu et al. (2013); Kim et al. (2020) and many scientific and engineering disciplines Vanderplaats (2002); Yang et al. (2016); Kimura et al. (2005). These problems are naturally NP-hard: without analytic information of the objectives, we may need to exhaustively search through the combinatorially-large number of local regions to find high-quality solutions. In practice, the focus of BBO algorithm design is on accelerating optimization progress while minimizing the number of function evaluations.

Existing work on BBO can be categorized into model-based and model-free approaches. Most model-based approaches, typically in the framework of BO Jones et al. (1998); Rasmussen (2003), involve learning a surrogate function from samples of the unknown function and optimizing the surrogate rather than the original function. In spite of this, the results of BO are sensitive to the parameters of the surrogate model. Normally, they are fixed at a particular value, but this underestimates uncertainty. By contrast, one could compute the BO with astounding expense over entire distributions of hyperparameters. In addition, the number of required samples increases in proportion to the size of the search space, so problems in high-dimensional spaces are less sample-efficient. Model-free approaches include various search methods, such as simulated annealing, cross-entropy methods, search gradient, as well as traditional direct search methods such as Nelder-Mead Henderson et al. (2003); De Boer et al. (2005); Salomon (1998); Gao & Han (2012); Kolda et al. (2003). The key to finding better samples in direct search algorithms is to create a good probability distribution for the search space. In both approaches, the lack of mechanisms of dealing with the inherent combinatorially-large number of local regions has been the bottleneck of scaling these

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

approaches to high-dimensional problems with complex landscapes. Recent advances in stochastic tree search provides new opportunities for balancing local search and modeling with better global exploitation/exploration trade-off. In particular, MCTS has recently introduced in Munos (2011); Kim et al. (2020); Wang et al. (2020) for computing good partitioning of the search space.

We propose a new design of the MCTS framework with more emphasis on sample-driven descent methods for local exploitation first, and then use BO methods at vertices of the search tree for local modeling. We change the role of BO from global modeling to local metrics for balancing exploitation and exploration. We introduce new descent methods that incorporate ideas from stochastic and direct search, and then design novel ways of balancing progress and uncertainty. To integrate these approaches in the MCTS framework, we propose new branch selection, tree expansion, and backpropagation policies.

We evaluate the proposed methods with experiments on nonlinear optimization benchmarks Lavezzi et al. (2022), MuJoCo locomotion tasks Todorov et al. (2012), and neural architecture search benchmarks Dong & Yang (2020). We compare with state-of-the-art methods such as TuRBO Eriksson et al. (2019) and La-MCTS Wang et al. (2020) as well other search methods such as CMA-ES Hansen et al. (2019) and Nelder-Mead Gao & Han (2012). We observe clear benefits in the proposed designs in achieving the most efficient and robust descent progress across all benchmark sets.

2 Related Work

Many attempts have been made on the black-box function optimization problem in continuous space. Since 1960, direct search methods, a family of deterministic methods, have had a golden age in tackling the derivative-free optimization problems Lewis et al. (2000). In principle, they compare function values at candidate positions and adopt the best one iteratively. A classic scheme of this type is Nelder-Mead (NM) Gao & Han (2012). This method creates a simplex in R^N - the primary convex object in n dimensional space with $n + 1$ vertices - and gauges the function on each vertex. With every iteration of the algorithm, it mirrors the worst vertex over a hyperplane formed by the remaining n points, or shrinks the n worst vertices to the best one. While the NM method performs well in practice Rios & Sahinidis (2013), its ability to identify global or local optima is not guaranteed. For example, in McKinnon (1998) the authors demonstrated it cannot reach the stationary point.

BO Jones et al. (1998); Shahriari et al. (2016) algorithms are the typical class by modeling with Gaussian Process (GP) Rasmussen (2003) as a local surrogate. GP uses a kernel function to evaluate the correlation between samples, and yields the sample prediction of the objective function. Such a BO algorithm commonly builds an acquisition function and selects the sample with highest acquisition function value Frazier (2018). The most commonly used acquisition functions are Gaussian Process-Upper Confidence Bound (GP-UCB), Expected Improvements (EI) and Most Probable Improvement (MPI) Srinivas et al. (2012). Moreover, a multi-objective acquisition may also be implemented, as in the Cowen-Rivers et al. (2020) who built one with evolutionary approach. The problem with BO approaches is that at every iteration, finding the global optimal of the acquisition function in high dimensional space is also challenging Oh et al. (2018). There have been many attempts to overcome the difficulty of BO algorithm in high dimensional space Gardner et al. (2017); Rolland et al. (2018); Mutnỳ & Krause (2019); Eriksson et al. (2019), and TuRBO Eriksson et al. (2019) is a widely accepted method. To address large-scale, highly dimensional problems, TuRBO combines Thompson sampling with Expected HyperVolume Improvement (EHVI) with a local probabilistic approach. As the trust region varies between iterations, TuRBO can improve the fitting of the local model and be able to allocate the sample of at the global extreme.

Another alternative is to optimize the black box function via sampling methods with domain partitions, such as Deterministic Optimistic Optimization (DOO) and Simultaneous Optimistic Optimization (SOO) in Munos (2011), and Hierarchical Optimistic Optimization (HOO) in Bubeck et al. (2011). Specifically, DOO divides up the search domain into partitions, each of which is represented by a point within it, and analyses are made on these points. Because DOO assumes a semi-metric input and a Lipschitz constant for the objective function, it can calculate an upper bound at other points within a partition cell via the distance between evaluated and unknown points. SOO does not require a specific character in the objective function, and HOO is a stochastic counterpart of DOO. Nevertheless, these approaches are applicable only to low-dimensional problems because of the high cost involved in creating perfect partition cells at each dimension. On the basis of these

prior works, Voronoi Optimistic Optimization (VOO) Kim et al. (2020) becomes more efficient in high-dimensional domains by including Voronoi partitioning in the tree search. LA-MCTS Wang et al. (2020) extends the work by learning a latent action within an existing partition. Such latent actions define a boundary between the high and low-performing regions of a cell, thereby making sampling preferential.

3 Preliminaries

Black-box optimization is a form of global optimization that makes the least requirement on the objective function. Without loss of generality, we assume that the objective $f(x)$ takes values in a box domain $\Omega = [lb_i, ub_i]^d, i = 1, ..., d$, lb_i and ub_i are lower and upper bounds of the domain at dimension i. We can evaluate $f(x)$ for any $x \in \Omega$, but do not have information about the analytic form of the function and can not directly evaluate its derivatives.

We use direct search sampling as part of the local descent optimization process. Direct search, in general, examines a set of points around the current point, seeking those with a function value lower than the current point. We also incorporate Stochastic Three Points (STP) Bergou et al. (2020) in order to increase sample efficiency. The STP algorithm effectively optimizes the objective function of a point x by sampling two additional points either side of x and selecting the one with the lowest value. Formally, descent approaches in direct search contains two steps in one iteration k: 1) sample $\{x_{k,i}\}, i = 1, ..., n$ with a probability distribution, and 2) select $x_{k+1} = argmin_{x \in D_k} f(x)$. For STP, the sampling set at each iteration is simply $D_k = \{x_k, x_k + s_k \cdot \alpha_k, x_k - s_k \cdot \alpha_k\}$ where s_k is a direction and α_k is its step size. As long as the amplitude of α_k is small enough, the relationship between $f(x_k + s_k \cdot \alpha_k)$, $f(x_k)$ and $f(x_k - s_k \cdot \alpha_k)$ is monotonically non-increasing or non-decreasing if the gradient of the continuous function f is not zero in the direction of s_k. With this method, it is possible to optimize the objective function iteratively with only two evaluations per step.

BO uses Gaussian Process regression (GP) to construct a surrogate model of the objective function. Using this model, we can now determine the predicted value and its uncertainty at a new point. In this way, promising samples can be identified from a large number of new candidates based on acquisition functions, e.g., by maximizing the expected improvement over all new candidates. For a finite collection of points $x_1, ..., x_k \in R^d$, GP constructs the mean vector μ_0 from the function f at each x_i, and the covariance matrix Σ_0 from a kernel function at each pair of x_i, x_j. The kernel reflects the correlation between the two points with the belief that two close points have similar values than points that are far apart. For any new point x, we can use Bayes' rule to compute the conditional distribution of $f(x)$:

$$f(x)|f(x_{1\sim k}) \sim N(\mu_0(x_{1\sim k}, x), \Sigma_0(x_{1\sim k}, x; x_{1\sim k}, x)) \tag{1}$$

This model allows us to define various acquisition functions Rasmussen (2003).

4 Monte Carlo Tree Descent

4.1 Overview

We propose our partition-free algorithm for optimizing black-box functions by MCTS with local descent in this section. In a sample-based manner, our algorithm is like VOO and LaMCTS as far as constructing a Monte Carlo tree is concerned. Despite this, our approach does not rely on any partitioning concepts because of limitations in a high-dimensional space. Instead, our sampling method combines learning a local surrogate model and optimizing with direct search. In our approach, a node is commonly represented by samples and observed function values $\{(x, f(x)\}$. In each iteration, a path is chosen from the root to the leaf. During the selection of children for the branch node on the path, we apply the Upper Confidence Bounds applied to Trees (UCT) for the nodes in the child list of the branch node. Aside from the UCTs from these existing children, we also append an additional UCT value which indicates whether the tree is interested in exploring a new child at this branch node. As soon as this additional UCT is selected, a leaf node is created and added to the current node's child list, and this newly created leaf node is selected for optimization. When an existing leaf node is selected at the end of a path, a decision is also made concerning whether to explore a new sibling or optimize in the chosen leaf. The Alg. 1 is an overview of our approach.

Algorithm 1 MCDescent

```
while within total budget do
    node ← Root
    explore branch ← False
    while node.children exist do
        uct_i = −y_i^* + C_d · Σ_{j=1}^{k} (dy_{i,−j}) + C_p · √(log n_node / n_i)
        uct_explore = −Σ (y_i^*) / N_children + C_p' · √(log n_node)
        best uct ← max(uct_i, uct_explore)
        If best uct == uct_explore : explore branch ← True ; break
        node ← child with best uct
    end while
    explore leaf ← False
    if node is leaf then
        If −y^* + C_d'' · Σ_{j=1}^{k} (dy_{−j}) < C_p'' · √(log n_leaf) : explore leaf ← True
    end if
    if explore branch or explore leaf then
        x ← node.X ; xnew ← node.X + Distance · decay^{CurrNodeLevel}
        nodeNew (xnew) ; node.children.append(nodeNew)
        If explore leaf: nodeInherit(node.X) ; node.children.append(nodeInherit)
        node ← nodeNew
    end if
    while within Descent budget do
        GP.train(node.samples)
        dx ← argmin(GP(sampling)) * GP.correlation lengh * step size
        node.X ← argmin(f(node.X), f(node.X + dx), f(node.X - dx))
    end while
    while within BO budget do
        node.X ← TuRBO(node.X)
    end while
end while
```

4.2 Node Optimization

Local Descent We examined several direct search algorithms, including NM Gao & Han (2012) and STP Bergou et al. (2020), and selected STP with careful redesign. A general principle suggests that s_k will be selected at random from a uniform sphere in the absence of prior knowledge about the objective function. In our approach, however, we use the surrogate GPR model to assist in identifying the point with the highest expected improvement. By utilizing STP, we evaluate not only the point with the greatest expected improvement, but also the point that is positioned in the opposite direction, which will lead to an increased likelihood of finding a better solution. The step size α_k was, by suggestion from Bergou et al. (2020), inversely proportional to the square root of iteration steps. In our case, we set it to be proportional to the square root of the product of node visits and its level. In addition, the step size is also rescaled according to the correlation length in the surrogate GP model - a similar approach to when TuRBO defines the length of each dimension for a trust region. The descent optimizer, if it succeeds at finding a better solution, will continue to evaluate new points in the same direction until failure; if the step is not successful at the beginning of the step on the two points in opposite directions, the step size is halved.

Each node is trained with a local GPR model as a surrogate to approximate the objective function. We use the vanilla GP with RBF kernel in our implementation. In the case when the local descent optimizer selects candidates for evaluation, the surrogate may provide answers to the expected function value. It also serves to suggest the correlation length for rescaling the step size at descent.

Local Bayesian Optimization We modify TuRBO-1 Eriksson et al. (2019) for local BO. The TuRBO-1 model creates a hyper-rectangle Trust Region (TR) with volume L^N centered at the best sample found so far. Afterwards, it samples new candidates within the TR and queries the objective function for ground truth data. The length of L_i will either increase after successive "successes" or

decrease after consecutive "failures". We changes TuRBO-1 in three ways to fit it into our algorithm: 1) TuRBO-1 begins with collected samples of the node. Consequently, TuRBO-1 is compelled to optimize from the vicinity of the collected sample. 2) The trust region length has been preserved on the same node, so the local BO can continue from the previous epoch. 3) We do not perform restarts for TuRBO-1 in order to avoid TuRBO-1 restarting from random samples.

4.3 Tree Expansion

In the event that exploration is decided on a node, a new child node will be created. The new child node is randomly created on a point that is distant from the best point of the selected node. Minimum and maximum distances are set to 10% and 50% of the domain's dimensional length, by an exponential decay on node level. After the new child node is created, it will be selected as the node for optimization at the current step.

In the scene the selected node to explore is a leaf node *L*, a new child node *M* is created in the same way as above, making *L* a branch node. At this time, a new node *L'*, starting from the current best point at *L*, is also created as the second child of node *L*. The new node *L'* inherits a batch of samples that are near its starting point, as well as the latest improvement history on *L*. The reduced number of samples forces the inheriting node *L'* to focus on optimizing in the neighborhood of the starting point, while the newly created node *M* is optimizing in a distant region. By adopting this approach, the tree grows at node *L* while maintaining the potential to optimize around the best point found on *L* by the inherit node *L'*.

4.4 Path selection

In order to balance exploration and exploitation, our algorithm uses UCT to determine the path between the root and a node. We modified the UCT formula for fitting our MCDescent algorithm:

$$-y^* + C_d \cdot \sum_{i=1}^{k} dy_{-i} + C_p \cdot \sqrt{\log n_{parent}/n_{node}} \tag{2}$$

Here, C_p is a hyper-parameter for the extent of exploration, n_{parent} is the number of visits to the parent node, y^* is the best value found at this node, and dy_{-k} is the most recent k's improvement at this node: $dy_i = y^*_{i-1} - y_i$ if $y_i < y^*_{i-1}$, else 0.0. As an example, if the best found value at a node improves over eight calls to the objective function, there will be eight improvements in total, and the last k improvements will be taken into account at computing UCT. At a parent node, it picks the nodes that has the highest UCT score. When $C_d = 0$ and $C_p = 0$, our approach degenerates into a greedy policy that only optimizes around the current best value.

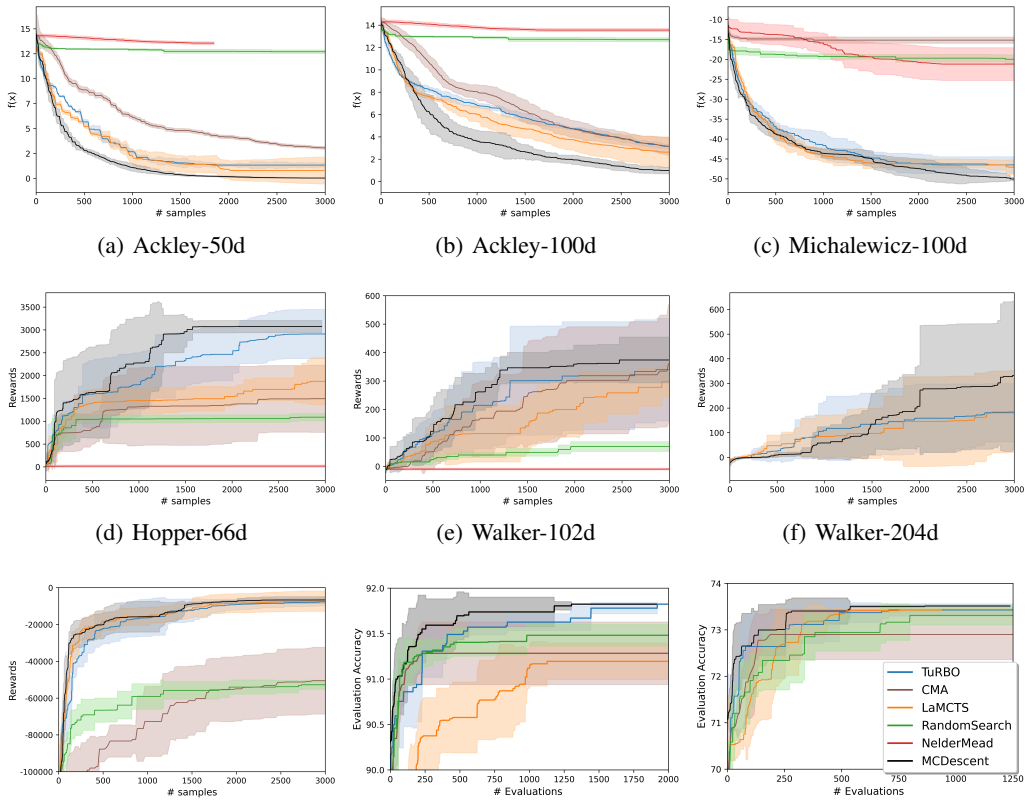
When a branch node is selected, it examines the UCT of all its child nodes as well as an additional UCT value that has the following setting: $y^{*'} = \sum (y^*_i)/N$, $C_d^{'} = 0$, and $C_p^{'} \neq C_p$, where N is the number of children at the current node. This additional UCT value represents if the branch decides to explore in a new child domain, because the existing children are not performing well enough. When this branch exploration UCT is the highest among all other child nodes, the search for the path is terminated, and the branch node is selected to create a child node. If no exploration is need on every branch node along the path, we apply the following criteria after selecting a leaf nodein order to determine whether it is worth exploring or exploiting:

$$-y^* + C_d^{''} \cdot \sum_{i=1}^{k^{''}} dy_{-i} > C_p^{''} \cdot \sqrt{\log n_{leaf}} \tag{3}$$

In the event that the inequality holds, it is decided to optimize the selected leaf node. Otherwise, two new nodes are created as children of this leaf node, as discussed above.

5 Experiments

Benchmarks Our first benchmark set includes synthetic functions from standard nonlinear optimization benchmarks Lavezzi et al. (2022). These functions all contain numerous local minima, valleys, and ridges. We scale the functions up to 100-dimensional.



(a) Ackley-50d (b) Ackley-100d (c) Michalewicz-100d

(d) Hopper-66d (e) Walker-102d (f) Walker-204d

(g) HalfCheetah-102d (h) CIFAR-10 (i) CIFAR-100

Figure 1: Overall performance of the baselines and our method. For Ackley and Michalewicz in (a), (b) and (c), the goal is to optimize for the lowest function values; in MuJoCo tasks (d) (e) (f) and (g), we aim to maximize the rewards; and for CIFAR-10 in (h) and CIFAR-100 in (i) we want to find the architecture with highest accuracy as quickly as possible

The MuJoCo locomotion environments Todorov et al. (2012) are among the most popular benchmarks for search and optimization. In our experiments, we choose Hopper, Walker, and HalfCheetah for tests. Hopper has 3 dimensions in action space a and 11 in observation s. We choose a linear policy $a = Ws$ in which W is the weighting matrix to search for maximizing the reward, therefore, the search space for Hopper is in the dimension of $3 \cdot 11 = 33$. Similarly, we set linear policies in both Walker-102d and HalfCheetah-102d. In addition to the above linear policy, we double the weighting matrix space dimension in Walker from 102 to 204, such that $a = W_1 s + W_2 s$ where W_1 and W_2 are matrix both in dimension 102. In this case the Walker becomes an optimization problem in 204d. Note since our approach does only consider deterministic result, we set the noise scale to zero in all MuJoCo environment to avoid the randomness in rewards.

The third test set is from *NAS-Bench-201*. The use of deep learning models is becoming popular in many fields, however the implementation of efficient neural networks requires lots of time and efforts. Therefore, Neural Architecture Search (NAS) problems have a practical significance. We choose the CIFAR-10 and CIFAR-100 datasets from *NAS-Bench-201* Dong & Yang (2020) for our benchmark. Each network in the datasets consists of three stacks of searching cells, and each cell has six positions where one can select one type of layer from five different types: (1) zeroize, (2) skip connection, (3) 1-by-1 convolution, (4) 3-by-3 convolution, and (5) 3-by-3 average pooling layer. Overall, there are $5^6 = 15625$ different types of architectures, and each architecture is trained and evaluated on both CIFAR-10 and CIFAR-100. The accuracy of training and evaluation is recorded. In order to benchmark on these datasets, we create following functions in real domain: we replace each of the five types of layers with an integer, and use the evaluation accuracy as the function value at corresponding point for the performance of a particular type of architecture in the set. When the input

is a real number, it is rounded up to the nearest integer and the function value at that point is returned. As an example, we set the input domain to $\{[0.5, 5.5]^6\}$, and $f([1.1]^6) = f([1]^6)$, where each $1-5$ corresponds to one type of the layers. It should be noted that in this method different inputs may refer to the same network architecture; therefore, the number of unique architectures examined is less than the number of functions called.

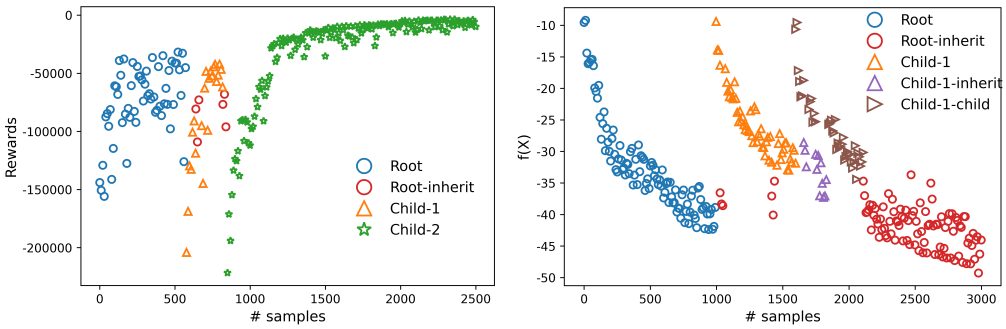
Baselines We evaluated our MCTS-Descent by comparing it to baselines from various algorithm categories. From among the BO algorithms, we selected TuRBO Eriksson et al. (2019), which is also part of our own algorithm. La-MCTS Wang et al. (2020) in MCTS class is selected as a major comparator, since this algorithm also constructs the tree in a similar manner. Moreover, CMA-ES Hansen et al. (2019) from the Evolutionary Algorithm category, Nelder-Mead Gao & Han (2012) from Direct Search algorithms, as well as the Random Search algorithm are selected for comparison as baselines.

Evaluation Metrics For each benchmark set, we run each baseline and our algorithm using up to five different random seeds. We assume that there is a limit on the number of calls to the objective function, which we set to 3,000. In our study, we evaluate the best found value until every step, and compute the mean and standard deviation for all runs. In this manner, we can compare the best found value at the end of the run as well as the speed at which each algorithm can reach its goal. It is possible that some algorithms will find the optimal value before 3000 calls, which will lead to an early stop in such a case.

Experiment Setup The function $fmin2$ from CMA-ES package is applied for CMA optimization with its default parameters. We implement our own version of Nelder-Mead algorithm, and set its expansion coefficient, contraction outside of simplex, contraction inside the simplex, and shrink coefficient as $2.0, 0.5, 0.5$ and 0.5, respectively. TuRBO is initialized with 20 random samples chosen by Latin Hypercube sampling, and its Automatic Relevance Determination (ARD) is set to *True*. La-MCTS uses different exploration setting for different test functions, so as our MC-Descent algorithm, and we put them in supplementary material. Our approach allows us to adjust not only the coefficients for the UCT and tree expansion, but also the allocation of resources between local descent and local BO. We set up running our approach on Google Colab with Tesla-V100 graphic card.

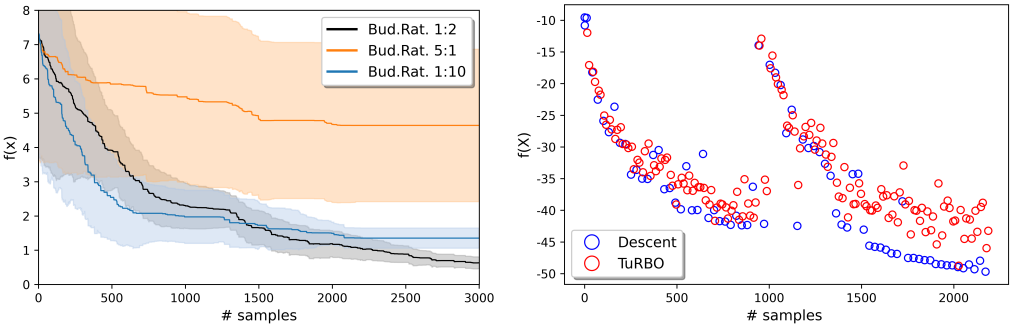
Overall Performance Fig. 1 shows the comparison between our model and baselines on different datasets. It is seen that MCDescent can substantially improve the speed of finding better results for Ackley-50d, Ackley-100d, Hopper-66d, CIFAR-10 and CIFAR-100. Particularly for Ackley and CIFAR-10/CIFAR-100, MCDescent comes very close to the optimal solution in a fast manner, which is crucial for NAS search. In the remaining test cases, we find that the approach is capable of achieving similar or even better mean scores earlier than TuRBO and LaMCTS. It may, however, display significant variability due to the challenging complexity of the landscape for the descent approach. As seen in the late stage of optimization on Walker-102d and Walker-204d, the reward curves exhibit large variation between the runs of MCDescent. Consequently, the performance of MCDescent may be limited in some high dimensional spaces that are inhospitable to descent approaches. Overall, these results show clear benefits of the designs in the proposed methods in achieving faster descent progress and more optimal final results.

Ablation Study: Tree expansion It is important to justify the expansion of the tree. Fig. 2 illustrates the nodes from which the query is made for the objective function. In Fig. 2(a), the root node is optimized for the first 200 queries; however, no significant improvement is evident for the next 300 queries. At this point, the tree decides to expand, so it creates a new child node, *child-1*, and starts optimizing from this child. Nonetheless, the optimization is also stuck after 200 more queries. Therefore, our tree abandons to optimize in *child-1*, and adds a new child node named *child-2*. On *child-2*, the optimization procedure is significant, and a new best value is found. The tree in Fig. 2(b) attempts to optimize in the *child-1* and its new exploration child *child-1-child*, however, the improvements on these nodes are insignificant. Consequently, the tree decides to optimize from the root inherit node. In light of the newly gathered samples upon exploring *child-1* and *child-1-child*, optimization is able to proceed at the root inherit node. They demonstrate that the tuned tree model is capable of optimizing by selecting a correct node.



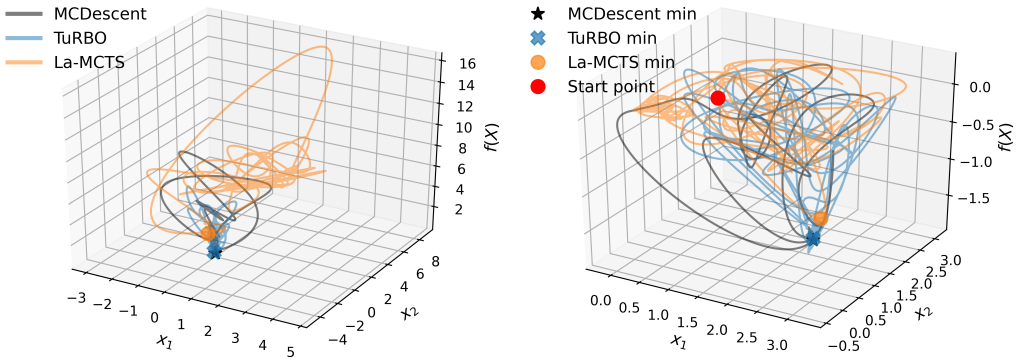
(a) Node queries on HalfCheetah-102d (b) Node queries on Michalewicz-100d

Figure 2: Illustration of nodes at queries to the objective function



(a) Optimization by different budget ratio (b) Queried value from local optimizers

Figure 3: (a) illustrates the optimization curves for Ackley-100d when the computational budget is divided between local descent and local BO in the ratio of 1:2, 5:1, and 1:10; (b) shows the values of Michalewicz-100d from local descent and local BO at each query.



(a) Search path on Ackley-2d (b) Search path on Michalewicz-2d

Figure 4: Search paths of different algorithms. The black star, the blue cross, and the orange circle indicate the best values found by MCDescent, TuRBO, and LaMCTS, respectively; the red dot represents the starting point of all three methods.

Ablation Study: Descent optimizer and Bayesian optimizer We examine the performance of our approach when the computational budget is divided between a local descent model and a local Bayesian optimizer TuRBO. Fig. 3(a) illustrates the optimization history of Ackley-100d when budget ratios are 1:2, 5:1, and 1:10. It is demonstrated that the model with a high budget in local

descent suffers from a low optimization rate, and the model with a high budget in local Bayesian optimizer may have difficulty escaping from the local optimal point. Having a balanced budget between local descent and BO speeds up the optimization process and overcomes the limits of the BO method at the local optimal. To investigate further how the local descent can enhance BO, we examine the function value of Michalewicz-100d at queries from both models in Fig. 3(b). From the figure we seen the local Bayesian optimizer has significantly better epoch improvement than local descent at an early stage of optimization, but when it reaches the local optimum, the optimization improvement provided by the local Bayesian optimizer becomes insignificant, while local descent can contribute steadily towards a superior solution. The two models can be combined to achieve better results than when they are used separately.

Ablation Study: Optimization route Fig. 4 illustrates how MCDescent, TuRBO, and LaMCTS optimize Ackley-2d and Michalewicz-2d in the first 30 samples after initialization. In both plots, LaMCTS explores a wide range of input domains, making it less likely to find a solution by a small number of calls. TuRBO locates efficiently the area where the optimal point is located in the beginning, however, its subsequent samples are diverse and fail to identify the global optimal solution. MCDescent, on the other hand, samples much closer to the global optimal point and thus finds the solution more rapidly.

6 Conclusion

In this paper, we present a sample-efficient approach to the optimization of deterministic black-box functions. Our proposed algorithm comprises descent methods as well as BO, and is not based on the input domain partition. The performance of our approach is evaluated against baselines from BO Eriksson et al. (2019), partition-based MCTS Wang et al. (2020), Evolutionary Algorithms Hansen et al. (2019), and Direct Search Gao & Han (2012). The benchmark is applied by using synthetic functions, MuJoCo locomotion tasks, and NAS-201 datasets. The result shows MCDescent, our approach, is comparable to baselines in terms of the best found values within fewer evaluation function calls, but provides the solution faster. Specifically, we examine how the descent model assists BO in increasing optimization speed as well as how the tree structure balances exploration and exploitation of a search domain.

References

Bergou, E. H., Gorbunov, E., and Richtárik, P. Stochastic three points method for unconstrained smooth minimization. *SIAM Journal on Optimization*, 30(4):2726–2749, 2020.

Bubeck, S., Munos, R., Stoltz, G., and Szepesvári, C. X-armed bandits. *Journal of Machine Learning Research*, 12(5), 2011.

Buşoniu, L., Daniels, A., Munos, R., and Babuška, R. Optimistic planning for continuous-action deterministic systems. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 69–76. IEEE, 2013.

Cowen-Rivers, A. I., Lyu, W., Wang, Z., Tutunov, R., Jianye, H., Wang, J., and Ammar, H. B. Hebo: Heteroscedastic evolutionary bayesian optimisation. *arXiv e-prints*, pp. arXiv–2012, 2020.

De Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

Dong, X. and Yang, Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.

Eriksson, D., Pearce, M., Gardner, J., Turner, R. D., and Poloczek, M. Scalable global optimization via local bayesian optimization. *Advances in Neural Information Processing Systems*, 32:5496–5507, 2019.

Frazier, P. I. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Gao, F. and Han, L. Implementing the nelder-mead simplex algorithm with adaptive parameters. *Comput. Optim. Appl.*, 51(1):259–277, jan 2012.

Gardner, J., Guo, C., Weinberger, K., Garnett, R., and Grosse, R. Discovering and exploiting additive structure for bayesian optimization. In *Artificial Intelligence and Statistics*, pp. 1311–1319. PMLR, 2017.

Hansen, N., Akimoto, Y., and Baudis, P. CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634, February 2019. URL `https://doi.org/10.5281/zenodo.2559634`.

Henderson, D., Jacobson, S. H., and Johnson, A. W. *The Theory and Practice of Simulated Annealing*, pp. 287–319. Springer US, Boston, MA, 2003.

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

Kim, B., Lee, K., Lim, S., Kaelbling, L., and Lozano-Perez, T. Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):9916–9924, Apr. 2020. doi: 10.1609/aaai.v34i06.6546.

Kimura, S., Ide, K., Kashihara, A., Kano, M., Hatakeyama, M., Masui, R., Nakagawa, N., Yokoyama, S., Kuramitsu, S., and Konagaya, A. Inference of s-system models of genetic networks using a cooperative coevolutionary algorithm. *Bioinformatics*, 21(7):1154–1163, 2005.

Kolda, T. G., Lewis, R. M., and Torczon, V. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM review*, 45(3):385–482, 2003.

Lavezzi, G., Guye, K., and Ciarcià, M. Nonlinear programming solvers for unconstrained and constrained optimization problems: a benchmark analysis, 2022. URL `https://arxiv.org/abs/2204.05297`.

Lewis, R. M., Torczon, V., and Trosset, M. W. Direct search methods: Then and now. *Journal of computational and Applied Mathematics*, 124(1-2):191–207, 2000.

McKinnon, K. I. Convergence of the nelder–mead simplex method to a nonstationary point. *SIAM Journal on optimization*, 9(1):148–158, 1998.

Munos, R. Optimistic optimization of a deterministic function without the knowledge of its smoothness. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

Mutnỳ, M. and Krause, A. Efficient high dimensional bayesian optimization with additivity and quadrature fourier ffeatures. *Advances in Neural Information Processing Systems 31*, pp. 9005–9016, 2019.

Oh, C., Gavves, E., and Welling, M. Bock: Bayesian optimization with cylindrical kernels. In *International Conference on Machine Learning*, pp. 3868–3877. PMLR, 2018.

Rasmussen, C. E. Gaussian processes in machine learning. In *Summer school on machine learning*, pp. 63–71. Springer, 2003.

Rios, L. M. and Sahinidis, N. V. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.

Rolland, P., Scarlett, J., Bogunovic, I., and Cevher, V. High-dimensional bayesian optimization via additive models with overlapping groups. In *International conference on artificial intelligence and statistics*, pp. 298–307. PMLR, 2018.

Salomon, R. Evolutionary algorithms and gradient search: Similarities and differences. *IEEE Transactions on Evolutionary Computation*, 2(2):45–55, 1998.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

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

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Vanderplaats, G. Very large scale optimization. In *8th Symposium on Multidisciplinary Analysis and Optimization*, pp. 4809, 2002.

Wang, L., Fonseca, R., and Tian, Y. Learning search space partition for black-box optimization using monte carlo tree search. *arXiv preprint arXiv:2007.00708*, 2020.

Yang, Z., Sendhoff, B., Tang, K., and Yao, X. Target shape design optimization by evolving b-splines with cooperative coevolution. *Applied Soft Computing*, 48:672–682, 2016.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See section *Overall performance*
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See section 5.4
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All results in section 5.5 and 5.6 include uncertainty range
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In section 5.4

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]