Monte Carlo Tree Descent for Black-Box Optimization

Yaoguang Zhai
UCSD

Sicun Gao
UCSD

Abstract

The key to black-box optimization is to search efficiently through regions of inputs with potentially 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 further integrate sample-based descent for faster optimization. We design novel ways of expanding Monte Carlo search trees, with new descent methods at vertices that incorporate stochastic search and Gaussian Processes. We then propose new rules for balancing progress and uncertainty, branch selection, tree expansion, and backpropagation. The proposed search process puts more emphasis on sampling for faster descent, and uses localized Gaussian Processes as auxiliary metrics for both exploitation and exploration. We show empirically that the proposed algorithms can outperform state-of-the-art methods on many challenging benchmark problems.

1 Introduction

Black-box optimization (BBO) considers objective functions that are not known analytically and can only be evaluated, potentially at high cost. The generality of the formulation makes BBO makes applicable to a wide range of challenging problems in machine learning Zoph & Le (2016); Buşoniu et al. (2013); Kim et al. (2020) as well as many scientific and engineering problems Vanderplaats (2002); Yang et al. (2016); Kimura et al. (2005). BBO problems over compact domains are naturally NP-hard: in the worst cases, we may need to exhaustively search through the combinatorially-large number of local regions to find high-quality solutions. Thus, the goal of BBO algorithm design is to accelerate optimization progress with respect to 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 Bayesian Optimization 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. However, when dealing with high-dimensional spaces, constructing a global surrogate model requires a large number of samples, which is why many Bayesian based approaches utilize multiple local surrogate models with different initialization Eriksson et al. (2019). In spite of the fact that this improves the accuracy of surrogate models, much computational time is spent on the non-optimal local modeling. Model-free approaches include simulated annealing Henderson et al. (2003), cross-entropy methods De Boer et al. (2005), search gradient Salomon (1998), as well as traditional direct search methods such as Nelder-Mead Gao & Han (2012); Kolda et al. (2003). They aim to create a good probability distribution over samples in the search space without modeling. In spite of this, model-free approaches do not usually achieve rapid local convergence when any manner of modeling is absent Lewis et al. (2000).

Additionally, the lack of mechanisms of dealing with the combinatorially-large number of local regions has been the bottleneck of both approaches. 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

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

et al. (2020) for computing good partitioning of the search space. These works partition the search space into subspaces and generate samples only on the most appropriate leaf of the tree. However, space partition becomes increasingly pricey and challenging with the increase in dimensionality.

Our work proposes a new design for the MCTS framework that emphasizes sample-efficiency. As to the first fact, we conduct local optimization by combining both model-based and model-free approaches. Samples evaluated randomly in model-free approaches are also used to construct surrogate models; the surrogate models, derived from model-based approaches, can be used as oracles in model-free approaches. As a result of the double-use of a sample's information in both approaches, the efficiency of the analysis is increased. To the second fact, we design the MCTS framework in accordance with the selection of local optimizations. Unlike partition-based MCTS, in which state spaces are organized into a tree, our method instantiates multiple local optimization processes and puts them into the tree. At each step, we select the most appropriate process and perform local optimization on it. As a result, we sought to eliminate the need for space partitioning without compromising the balance between local exploitation and space exploration.

We evaluate the proposed methods with experiments on nonlinear deterministic optimization benchmarks Lavezzi et al. (2022), MuJoCo locomotion tasks Todorov et al. (2012), and neural architecture search benchmarks Dong & Yang (2020). We compare our algorithm with state-of-the-art methods such as TuRBO Eriksson et al. (2019) and La-MCTS Wang et al. (2020) as well other classical search methods like 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

Model-based methods. Bayesian optimization Jones et al. (1998); Shahriari et al. (2016) algorithms are an typical class which employs 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 Bayesian optimization algorithm commonly builds an acquisition function (e.g., Gaussian Process-Upper Confidence Bound, Expected Improvements and Most Probable Improvement Srinivas et al. (2012)) and selects the sample with highest acquisition function value Frazier (2018). The problem with Bayesian optimization 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 Bayesian optimization algorithm in high dimensional space Gardner et al. (2017); Rolland et al. (2018); Mutnỳ & Krause (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. The core to TuRBO is it intensively samples inside the local trust region, and adjust the size of the trust region after every sampling. By attempting with a good trust region size, TuRBO can improve the fitting of the local model and be able to allocate the extreme.

Model-free and direct search methods. Model-free methods iteratively create a set of candidate positions, compare function values at some or all candidate positions with the current, and adopt the best one. A classic scheme of this type is Nelder-Mead (NM) Gao & Han (2012). This method creates a simplex 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 towards the best one. Other direct search algorithms include simulated annealing (SA) Henderson et al. (2003), cross entropy (CE) De Boer et al. (2005), and Stochastic Three Point (STP) Bergou et al. (2020). The SA algorithm compares iteratively the outputs from the objective function with the current and a neighboring point in the domain. The neighboring point is adapted as the new base point if it is better, or if it is worse but wins a tossing game whose probability is decided by the difference of the function value at the two points. In this way SA is not prone to trapping in the local minima or maxima. In CE, a distribution is establishes on the input domain, and the cross entropy is minimized between this distribution and the objective function distribution. STP is a simple but effective way of direct search that compares function values at the base point, in one random direction, and in the opposite direction. Based on

the first two terms of the Taylor series of the real function, one can see that a real-valued function is increasing in one of two opposite directions as long as the gradient is non-zero.

Tree search methods. Tree search based BBO via sampling methods are typically integrated 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. DOO assumes a semi-metric input and a Lipschitz constant for the objective function, so it can calculate an upper bound at another point 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 and uses MCTS algorithm before selecting which partition the optimization should run on. The latent action defines a boundary between the high and low-performing regions in the subspace associated with a node, and create children according to this boundary. When continuing with such splitting, it sets a sampling preferential on every node in the tree. In every iteration, the search tree is traversed from root node to a leaf by selecting highest approximated value based on Upper Confidence Bounds for Trees (UCT) algorithm. The optimization is then performed from the subspace partition on the selected node.

3 Preliminaries

Black-box optimization is a form of global optimization: to find the global maximum or minimum of the objective $f(x) \in R$ in which $x \in \Omega \subseteq R^n$. We can evaluate the deterministic $f(x)$ for any $x \in \Omega$, but do not have information about the analytic form of the function and can not directly evaluate its derivatives.

In model-free approaches, the algorithm typically creates a set at each iteration $D_k = \{x_k^i\}, i = 1, 2, ...$ and selects to evaluate at certain points. In this work, we employ the STP Bergou et al. (2020) as the model-free local optimization. For STP, $D_k = \{x_k, x_k + s_k \cdot \alpha_k, x_k - s_k \cdot \alpha_k\}$ 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 function f is not zero in the direction of s_k.

The Gaussian Process regression (GPR) model is commonly used in Bayesian optimization, and is used in our work to construct a surrogate model for the local model-based optimization. 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 by a kernel at each pair of $(x_i, x_j), i, j = 1, 2, ...k$. With μ_0 and Σ_0 the prior distribution on f is:

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

For any new point x, we can use Bayes' rule to compute the conditional distribution of $f(x)$:

$$f(x)|f(x_{1\sim k}) \sim N(\mu_0(x_{1\sim k}, x), \Sigma_0(x_{1\sim k}, x; x_{1\sim k}, x)) \tag{2}$$

In MCTS, the simulation constitutes four ordered steps: namely, Selection, Expansion, Simulation and Backpropagation. During Selection, a search tree is traversed from root node to a leaf node. This traversal is made by selecting the node with the highest value based on UCT algorithm. For a node i, the uct_i is computed by:

$$uct_i = R_i/N_i + C \cdot \sqrt{2 \cdot \ln N_p/N_i} \tag{3}$$

in which R_i is the rewards on the node i; N_i and N_p denote the number of visits on node i and its parent node p, respectively; C is a constant to balance between exploitation and exploration. At each branch node p, the child to select is the one with highest uct value among all of its immediate children. At Expansion, a new child node is then added to expand the tree. During Simulation, a random simulation is run from the new child node until the terminal node is reached, and the simulation reward is approximated. Finally, the simulation reward is backpropagated through the selected nodes

to update the tree. The selection and expansion steps are managed by the tree policy, whereas default policy is responsible for the simulation step.

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. Without loss of generality we use minimization problems as illustration. In our approach, a node N commonly saves pairs of samples and their corresponding function values $\{(x, y)\}_N$ that are obtained during executing local optimizations on it and its subtree. We denote the minimum function value at the current step on node N as y^*_N, and its corresponding domain value as x^*_N, and use $(x^*, y^*)_N$ to characterize this node. In the following context, if we say "a node A is dx away from the node B", it means the distance of x^*_A on node A and x^*_B on node B is dx.

The general algorithms are illustrated in Alg.1, and Fig. 4 is an illustration of the process. Here we use the superscript to denote the place where the node is located on the tree: the count of digits indicates the level of the node in the tree; the last digit is the index of the node in its parent's child list; the first to the last but second digit reflects its parent. As an example, node N^{021} is at level 2 in the tree (if the root node is at level 0). Its parent is node N^{02}, and it is the node with index 1 in N^{02}'s child list. We initialize a random sample in the state space Ω and affiliate it to the root node. In every iteration, a path is chosen from the root until a leaf by the UCT algorithm. The expansion of the tree is also included in this step. Thereafter, the local descent optimization and local Bayesian optimization are launched on the selected leaf. In the end, collected samples are backpropagated up to the root node. Note the optimization is only executed on the leaf node, while the branch node is used to store information on collected samples from its subtree. In the following three sections, we explain every step in detail.

4.2 Path selection

In every step we pick a leaf for optimization. In order to balance exploration and exploitation, our algorithm uses UCT to determine the path between the root and the leaf, as shown in the function **SELECT** in Alg.1. We modified the UCT formula for fitting our MCTD algorithm. For each child node N^{mi} with the parent node N^m , its UCT is given by:

$$uct_{mi} = -y^*_{mi} + C_d \cdot \sum_{j=1}^{J} dy_{mi,-j} + C_p \cdot \sqrt{\log n_m / n_{mi}} \tag{4}$$

Here, C_d is a weight factor controlling the importance of recent improvements during optimization, C_p is a hyper-parameter for the extent of exploration, n_m and n_{mi} are the number of visits to the parent node N^m and to the node N^{mi}, respectively. y^*_{mi} is the best value found at node N^{mi}, and $dy_{mi,-j}$ is the most recent j's improvement at N^{mi} after calling the objective function. For every call to the objective function during the optimization, we record the improvement on the function value from this call. If the value from this call is worse than the optimal value before the call, we set the improvement to zero; otherwise, we set the improvement as the absolute difference between the optimal value before and after the call. When computing the UCT value, only the last J improvements are taken into account.

We integrate the tree expansion in a similar way as the UCT algorithm. At a branch node N^m, in addition to examining the UCT of all its child nodes we add an artificial exploration node that has the UCT value as following: $uct_{exp,m} = -\sum_i (y^*_i)/nc_m + C^{'}_p \cdot \sqrt{\log n_m}$, where nc_m is the number of children at node N^m. This node represents if the branch decides to explore in a new domain, because the existing children are not performing well enough. When the exploration node is selected, a new child node under the branch node is created and selected. If a leaf that is not newly created is selected, we apply the following criteria to determine whether it is worth optimizing on it.

$$-y^* + C^{''}_d \cdot \sum_{j=1}^{J^{''}} dy_{-j} > C^{''}_p \cdot \sqrt{\log n_{leaf}} \tag{5}$$

Algorithm 1 Monte Carlo Tree Descent (MCTD)

function MCTD(objective: f, domain: Ω)
  $x \leftarrow$ Random Sample in Ω
  $y \leftarrow f(x)$
  root $\leftarrow (x, y)$
  for step = $1, ..., n$ **do**
    $N \leftarrow$ Select(root)
    Optimize(N)
    Backup(N)
  end for
  return y^*_{root}
end function

function OPTIMIZE(node: N)
  $N' \leftarrow N$
  while number of (x, y) on N' is too few **do**
    $N' \leftarrow$ parent of N'
  end while
  if number of (x, y) on N' is sufficient **then**
    oracle $\leftarrow$ Train GP by (x, y) on N'
    Set $stepsize$ in descent optimization $\propto$ correlation length in GP
  else
    oracle $\leftarrow$ None
  end if
  Descent Optimize with oracle and evaluation function f from $(x^*, y^*)_N$
  Bayesian Optimize on N starting from $\{(x, y)\}_N$
  N saves new $\{(x, y)\}'$ during optimization
  return
end function

function BACKUP(node$N$)
  while N has parent **do**
    Add N's new $\{(x, y)\}'$ to parent
    $N \leftarrow$ parent
  end while
end function

function SELECT(node: N)
  $N \leftarrow$ root
  while N has children **do**
    for child node i **do**
      $uct_i = -y^*_i + C_d \cdot \sum_{j=1}^{J}(dy_{i,-j}) + C_p \cdot \sqrt{\log n_N / n_i}$
    end for
    $uct_{exp} = -\sum(y^*_i)/nc_N + C'_p \cdot \sqrt{\log n_N}$
    if $\max(uct_i) < uct_{exp}$ **then**
      return Expand(N)
    end if
    $N \leftarrow \text{argmax}_{N\text{'s children}}\, uct_i$
  end while
  if $-y^*_N + C''_d \cdot \sum_{j=1}^{J''}(dy_{N,-j}) < C''_p \cdot \sqrt{\log n_N}$ **then**
    $N \leftarrow$ Expand(N)
  end if
  return N
end function

function EXPAND(node N)
  $x \leftarrow x^*_N$
  if N is leaf **then**
    Copy N without the parent and the child list
    Append the copied node to the child list of N with child index 0
  end if
  $lv \leftarrow$ level of N
  distance $d \propto \exp(-lv)$
  $x' \leftarrow$ Random Sample in Ω with distance d to x^*_N
  $y' \leftarrow f(x')$
  ExpandNode $N' \leftarrow (x', y')$;
  Append N' to the child list of N
  return N'
end function

In the event that the inequality holds, it is decided to optimize the selected leaf node; otherwise, we will make a leaf expansion.

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 at a point that is distant from the selected node. The minimum and maximum distances are set to 10% and 50% of the domain's dimensional length, by an exponential decay according to the node level. After the newly created child node is placed, it will be immediately 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 child 0 of node *L*. This 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 exploit around the best point found on *L* by the inherit node *L'*. These steps are in the function

EXPAND in Alg. 19, and 3 subplots in Fig.4 show an example. As in Fig.4 (c), the expansion takes place on the root node N^0. The node N^{01} is a new node for exploration, placed distant from N^0. Node N^{00} starts from x_0^*. Similarly, in 4 (d), node N^{010} starts from x_{01}^*, and node N^{011} is placed away from N^{01}, but the distance between node N^{01} and N^{010} is much smaller than the distance between node N^0 and N^{01} at node creation. Fig. 4 (e) shows a new leaf node is created below the branch (and root) node N^0.

4.4 Node Optimization

On every iteration, we run the modified STP Bergou et al. (2020) algorithm for local descent optimization, followed by the TuRBO-1 Eriksson et al. (2019) algorithm for local Bayesian optimization (BO) with limited computation budgets at the node for the selected leaf. It should be noted, however, that the two processes are not separate, but are tightly integrated. In BO, samples obtained from local descent optimization are used to construct the surrogate model GPR, and the GPR not only serves as an oracle for the local descent optimization, but also provides the correlation length according to which the local descent optimization scales its step sizes.

Local Descent Optimization In our work, we employed the direct search algorithm STP Bergou et al. (2020) with careful redesign. In the absence of prior knowledge about the objective function, the direction in direct search s_k is usually selected from a sphere with uniform distribution. In our approach, however, we use the surrogate GPR model to identify the point with the highest expected improvement. The steps of local descent optimization are as follows:

- Compute the step size α_k. In our case, we set α_k to be inversely proportional to the square root of the product of node visits and its level, as Bergou et al. (2020) suggested. We also rescaled it according to the correlation length in the surrogate GP model when possible.
- If the oracle is not available, random sample once in the box of step size, and use its direction as $s_k \cdot \alpha_k$ for checking ground truth.
- If the oracle is available, generate multiple samples in the box of step size, and choose the direction along the best point as $s_k \cdot \alpha_k$.
- Start one step of STP with selected direction of $s_k \cdot \alpha_k$.
- Depending on the optimization progress, one may decide to further optimize the objective function along the same direction with tuned step sizes in a fine-grain descent approach. See Alg. 2 and 3 in Supplementary Material.

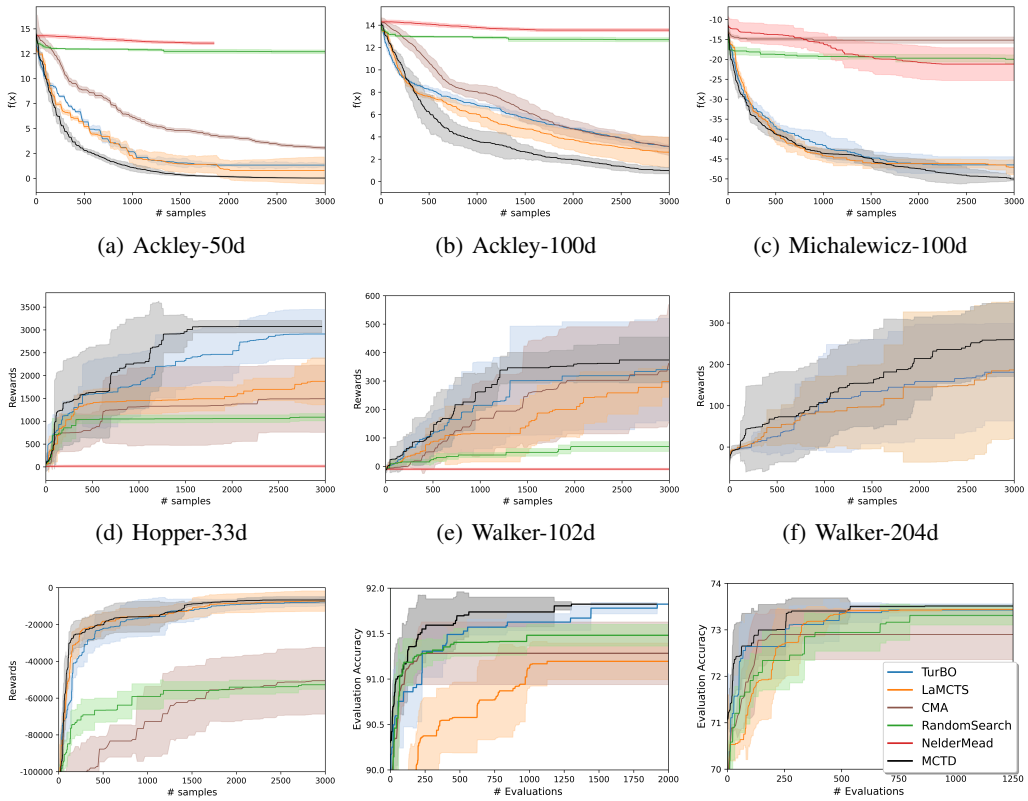
Note the last step is usually switched on when the optimization comes to fine-tuned phase with small variations on samples, so one can set a function threshold from which the search applies fine-grain descent approach.

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.

5 Experiments

5.1 Setup

Benchmarks The first benchmark set includes synthetic functions taken from standard nonlinear optimization benchmarks Lavezzi et al. (2022). There are numerous local minima, valleys, and ridges in each of these functions. We scale the functions up to 100-dimension.



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

(d) Hopper-33d (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 second benchmark set consists of MuJoCo locomotion environments Todorov et al. (2012). In our experiments, we chose 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-33d 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 matrices in the dimension of 102. In this case the Walker becomes an optimization problem in 204d. Since our approach considers deterministic results, we set the noise scale to zero in all MuJoCo environments to avoid randomness in rewards.

The third test set is from *NAS-Bench-201*. Many fields are using deep learning models, but implementing efficient neural networks requires a substantial amount of time and effort. Thus, Neural Architecture Search (NAS) problems have practical significance. We chose 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 this set, we created the following functions in the real domain: we replace each of the five types of layers with an integer, and the real-valued input is rounded up to the nearest integer. The evaluation accuracy of the architecture is set as the function value. 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 selected TuRBO Eriksson et al. (2019) as one baseline from the BO algorithms. La-MCTS Wang et al. (2020) is chosen as a major comparator since this algorithm also constructs trees 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.

Experiment Setup For CMA optimization, $fmin2$ from the CMA-ES package is used with its default parameters. We implement our own version of the Nelder-Mead algorithm, and set its expansion coefficient, contraction inside the simplex, contraction outside the simplex, and shrink coefficient as $2.0, 0.5, 0.5,$ and 0.5, respectively. TuRBO is initialized with 20 random samples selected using Latin Hypercube sampling, and its Automatic Relevance Determination (ARD) is set to *True*. For La-MCTS, we use different settings and include them in the supplementary material, as well as our MCTD approach. Benchmarks are made mainly on Google Colab with Tesla P100 graphic card.

5.2 Overall Performance

Evaluation Metrics For each benchmark function, we run baselines and our algorithm by at least five different random seeds. Due to the limit on the computational power available to us, we set the number of calls to the objective function to 3000. Our study evaluates the best found value at every step and computes the mean and standard deviation of all runs. As a result, we can compare the best found value at the end of the run as well as the speed at which each algorithm is capable of reaching the most optimal result. There is a possibility that some algorithms will find the optimal value before 3000 calls, which will result in an early stop.

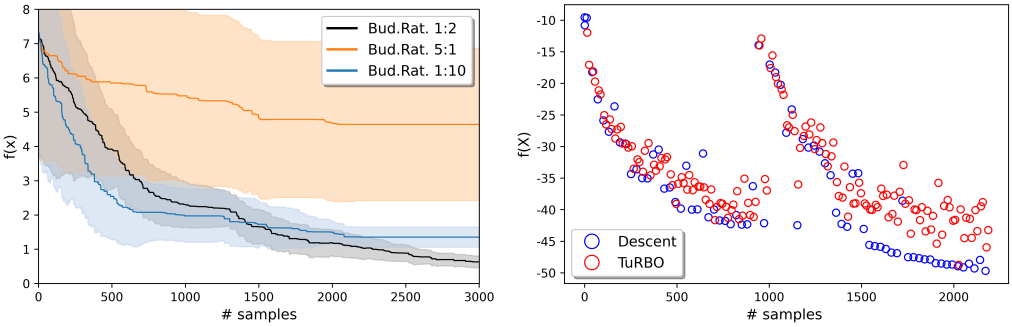
Efficiency Fig. 1 illustrates the comparison between our model and baselines on benchmark sets. It was found that in general, random search, CMA, and NM methods performed poorly in these cases due to the fact that they do not cooperate with any approach that may potentially improve the efficiency of the sample.

According to Fig. 1 (a), (b) and (c), MCTD significantly improves the speed of finding better results for the set of synthetic functions compared to TuRBO and La-MCTS. In particular, the Ackley synthetic function exhibits a noticeable improvement when we balance local optimization exploitation and state space exploration. Michalewicz is improved moderately through descent optimization, and MCTS helps improve the optimization consistently.

In the reinforcement learning set, problems are typically difficult for global optimization due to their complexity. In spite of this, our approach is competitive with TuRBO and La-MCTS on this set, and has moderate improvement over the average value on functions Hopper-33d, Walker-204d, and Cheetah-102d. In particular, the combination of local BO and local descent optimization speeds up the optimization during its early stages. It is, however, difficult to balance local exploitation and space exploration by picking the correct weigths to bring recent improvements, explore terms, and objective function values into the same order of magnitude. This is because we use the absolute value of the objective function that varies significantly at different optimization steps. In light of this, we see a large variation from different runs in this set, as in Fig.1 (d), (e), (f), and (g).

In CIFAR-10/CIFAR-100, MCTD reaches the optimal solution by a small amount of samples, which is critical for NAS searches. The combination of descent and modeling approaches facilitates the search for the optimal solution more quickly than if only one method was used.

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. 2(a) illustrates the optimization history of Ackley-100d when budget ratios are 1:2, 5:1, and 1:10. It is demonstrated that a model with a high budget for local descent suffers from a low optimization rate. In contrast, the model with a high budget in local Bayesian optimizer may have difficulty escaping the local optimal point.



(a) Optimization by different budget ratio

(b) Queried value from local optimizers

Figure 2: (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.

As shown in the case of Fig.2(a), when we use the budget that emphasizes local descent (budget 5:1), the performance is less comparable with that of emphasizing local BO (budget 1:20) in the term of optimization speed. Based on the budget ratio for every function in supplementary material Tab.2, it is generally advantageous to use at least the same (or even more) amount of computational budget on local BO as on local descent. This may be one challenge for the local descent approach, since this indicates that local descent may require local BO as the oracle when the function landscape is difficult, such as for the complicated functions in MoJoCo locomotion and for non-continuous functions in NAS sets.

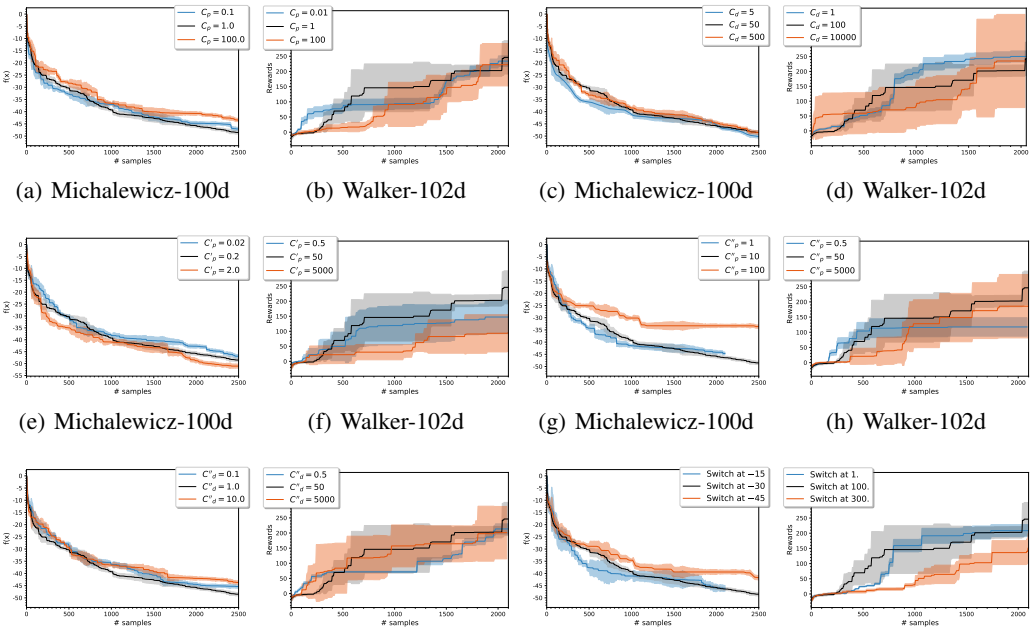
However, the local descent approach still proves beneficial despite these factors. From Fig.2(b) we can see the optimization improvement of the local BO becomes insignificant when the process is close to the local optimum. Local descent, on the other hand, can contribute steadily to the discovery of a superior solution. To conclude, using a balanced approach can yield better results than using each approach separately.

5.3 Ablation Study

The purpose of this section is to describe the ablation study of six hyperparameters in three categories. The first category includes C_p' (the weight in uct_{exp}) C_p'', and C_d'' (the weights on leaf exploration in Eq.5) that control the expansion of the tree. The second set of values, C_d and C_p in Eq.4, balance local exploitation and space exploration. Lastly, the threshold value determines when fine-grain descent is required. We use the synthetic function Michalewicz-100d and the locomotion Walker-102d for the ablation study, and each case runs with at least 3 different seeds. Please note that results are based on runs with different hyperparameters than those presented in Section 5.2. We found that a wise choice on C_p, C_p', and C_p'' is critical to improving performance, while C_d and C_d'' are less significant. The switching threshold value is highly dependent upon the objective function's properties.

Ablation Study: state space exploration The parameters C_d and C_p balance exploration and exploitation of the existing tree. As shown in Fig.3(a), the moderate choice on C_p improves the overall performance slightly; however, this is not clearly observed in Fig.3(b). From Fig.3(c) and 3(d), we can see a variation in C_d may not help significant changes in the overall performance. Even so, we can observe a contribution from C_d and C_p: from Fig. 3(b) and Fig.3(d), we can see that path selection with low values of C_d and C_p leads to little variation between runs, since the path selection tends to select the node where the current best-known value resides, thus limiting the path selection.

Ablation Study: tree expansion C_p', C_p'' and C_d'' are responsible for expanding the current tree. As a result of setting the parameters C_p', C_p'' to large values and C_d'' to a small value, it is likely that a new sibling leaf will be created at the selected node in order to explore the state space. Alternatively, the path will tend to select the node that has the current optimal value. Since a leaf always has zero children, even though C_d'' and C_p'' have the same functionality as C_p', the criteria for tree exploration



(a) Michalewicz-100d (b) Walker-102d (c) Michalewicz-100d (d) Walker-102d

(e) Michalewicz-100d (f) Walker-102d (g) Michalewicz-100d (h) Walker-102d

(i) Michalewicz-100d (j) Walker-102d (k) Michalewicz-100d (l) Walker-102d

Figure 3: Ablation studies on function Michalewicz-100d with hyper parameters (a) C_p, (c) C_d, (e) C'_p, (g) C''_p, (i) C''_d and (k) switching to fine-grain STP at function value; ablation studies on function Walker-102d with hyper parameters (b) C_p, (d) C_d, (f) C'_p, (h) C''_p, (j) C''_d and (l) switching to fine-grain STP at function value

are different for branch and leaf nodes. According to Fig.3(f) and Fig.3(h), it is evident that a good choice on C'_p and C''_p can improve the optimization performance by exploring new state space distant from the local optimal value. Conversely, when their values are set either too large (orange lines in Fig.3(f) and Fig.3(g)) or too small (blue line in Fig.3(h)), this would adversely affect the overall performance of the optimization process. The effect of C''_d is less noticeable. However, a small C''_d results in a small variation between different runs - a similar behavior as C_d.

Ablation Study: switching at function value The fine-grain STP can be beneficial in certain cases, as the orange lines shows in Fig.3(k) and Fig.3(l). In these two lines, switching takes place at a late stage of optimization, which results in an excessive use of normal STP. Generally, fine-grain STP can be used as soon as possible. However, in some experiments, the fine-grain STP highlighted a small neighborhood at an early stage of optimization, resulting in an insufficient amount of information for BO to model.

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 MCTD, 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.

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.

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.

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.

Appendix A Tree expansion illustration

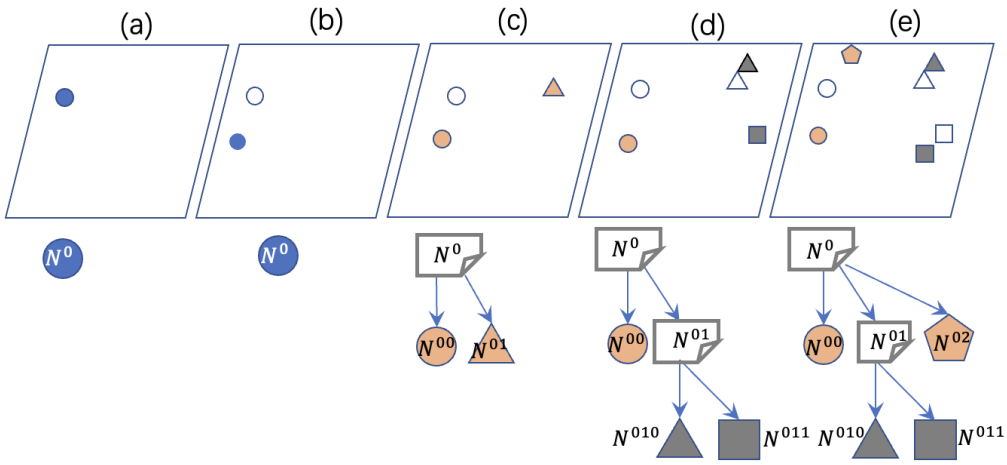


Figure 4: Top: Illustration of the nodes in the input domain; Bottom: Illustration of the tree expansion. (a) The *root* N^0 begins with a random sample. (b) Optimization is carried out on the *root* N^0. (c) Leaf exploration on the *root* N^0 creates two nodes N^{00} and N^{01}. N^{00} starts from x_0^*, and N^{01} starts from a point distant from x_0^*. (d) Leaf exploration on the node N^{01}, generating two new node N^{010} and N^{011}. N^{010} starts from x_{01}^* while N^{011} starts at a point away from x_{01}^*. (e) Branch exploration at *root* N^0 creates a new child node N^{02}.