A Implementation Details of Stochastic Three Point Approach

In Alg.1 , we can see the typical workflow of the stochastic three-point algorithm (STP).

Algorithm 1 Stochastic Three Point

while within Descent budget **do**
  dx ← random sampling
  X ← argmin(f(X), f(X + dx), f(X - dx))
end while

For our implementation, we have redesigned two versions of STP which are compatible with our search approach. In first place, the descent direction dx is not entirely random, but is sampled by Latin Hypercube Sampling within the domain of the step size. The length in each dimension is rescaled by the correlation length L from the local surrogate model, while maintaining the total length of dx: $dx = dx \cdot L \cdot ||dx||/||dx \cdot L||$. Further, the choice of dx is made by maximizing the expected improvement on the local surrogate model G: train $G(x), x \in$ collected samples at the node, and $dx = \underset{dx}{\operatorname{argmin}} G(X + dx)$, where X is the best point at the node.

The first implementation is fundamentally similar to the typical STP as in Alg.2, with the exception that it continues to test along the same dx whenever it find a better value for $G(X+dx)$ or $G(X-dx)$:

Algorithm 2 STP for local descent optimizer

$X \leftarrow$ best point at the node
Train surrogate model $G(x), x \in$ collected samples
while within Descent budget **do**
  $\mathcal{DX} \leftarrow$ Latin Sampling $\cdot$ Correlation Length $L \cdot$ Step Size α
  dx $\leftarrow \underset{dx \in \mathcal{DX}}{\operatorname{argmin}} G(X + dx)$
  $k \leftarrow 0$
  while $G(X + (k+1) \cdot dx) < G(X + k \cdot dx)$ **do**
    $k \leftarrow k + 1$
  end while
  X ← argmin($f(X + dx), f(X)$)
end while

Algorithm 3 Fine-grained STP

$X \leftarrow$ best point at the node
Train surrogate model $G(x), x \in$ collected samples
while within Descent budget **do**
  $\mathcal{DX} \leftarrow$ Latin Sampling $\cdot$ Correlation Length $L \cdot$ Step Size α
  dx $\leftarrow \underset{dx \in \mathcal{DX}}{\operatorname{argmin}} G(X + dx)$
  k_0, k_-, $k_+ \leftarrow$ 0.0, -1.0, +1.0
  while within computational budget **do**
    $k_0 \leftarrow \underset{k \in (k_0, k_-, k_+)}{\operatorname{argmin}} G(X + k \cdot dx)$
    update k_-, k_+
  end while
  X ← argmin($f(X + k_0 \cdot dx), f(X)$)
end while

The second one differs from the first implementation by always trying to test two more points, which have never been tested, on either side of the current point. As an example, if X, $X + dx$, and $X - dx$ are currently being compared, and $X + dx$ is the best point of the three, the two points that will be tested in the next step are $X + 2 \cdot dx$ and $X + 0.5 \cdot dx$ (since $X + dx$ has already been tested and we need to select two more points in either side of $X + dx$ for evaluation). Should X be the best point

among X, $X + dx$, and $X - dx$, the next round will be to test X, $X + 0.5 \cdot dx$, and $X - 0.5 \cdot dx$. The second version of the STP model, illustrated in Alg.3, may yield better results in fine-grain, however it is more computationally expensive. As a result, we switch to this fine-grained model when the function value drops below a threshold.

B Hyperparameters

In this chapter, we demonstrate the hyperparameters for various test functions in LaMCTS and MCDesent

B.1 LaMCTS

In LaMCTS, C_p is responsible for controlling the amount of exploration. Having a small C_p will make the search focus exclusively on the current best found value, but may result in being stuck at a local optimum. By contrast, a large C_p encourages LaMCTS to explore poor regions more frequently, but this can result in overexploration. The type of kernel and gamma determine the shape of the boundary drawn by the classifier in LaMCTS. Additionally, the leaf size determines the splitting threshold and the rate of tree growth. In all tests LaMCTS uses TuRBO-1 sampling method as default. All hyperparameters for LaMCTS are as listed below:

Table 1: Hyperparameters used in LaMCTS for each of the test functions

Functions	C_p	Leaf size	Num. of initial	Kernel type	Gamma type
Ackley-50d	1.	10	40	rbf	auto
Ackley-100d	1.	10	40	rbf	auto
Michalewicz-100d	10.	8.	40	rbf	auto
Hopper-33d	10	100	150	poly	auto
Walker-102d	20	10	40	poly	scale
Walker-204d	20	10	40	poly	scale
HalfCheetah-102d	20	10	40	poly	scale
CIFAR-10	10	8	10	poly	auto
CIFAR-100	10	8	10	poly	auto

B.2 MCDesent

As part of our MCTD approach, we have several hyperparameters that can be tuned during tests. As a first step, we may allow a specific number of computational calls from both the local descent optimizer and the local BO optimizer to the objective function. Typically, the total number of calls allowed in a single iteration is either 30 or 40 in order to ensure that enough steps are performed by the optimizers in one iteration. One may, however, want to combine the two algorithms to maximize the benefits. Such a situation could be addressed by splitting the budget among a local descent optimizer and a local BO optimizer according to different ratios. Furthermore, in local descent, we can specify the step size α for the optimizer, and when to change over to fine-grained descent optimization. The step size in our algorithm is relative to the dimensional length of the test function, and we set to use the fine-grained descent optimizer when the best found value on the node is below a threshold. As a third point, the UCT of each node is determined by the equation

$$uct_i = -y_i^* + C_d \cdot \sum_{j=1}^{J} (dy_{i,-j}) + C_p \cdot \sqrt{\log n_{parent}/n_i} \tag{1}$$

, and one can adjust the weight of recent improvement C_d and the weight for exploration C_p. As a final step, we must decide when to expand the branch and the leaf node when selecting a path from the tree. To do this, we compute an additional UCT value that has the following setting: $y^{*'} = \sum (y_i^*)/N$, $C_d^{'} = 0$, and $C_p^{'} \neq C_p$ at every branch node, where N is the number of children at the branch 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. And we apply the following

criteria after selecting a leaf node in order to determine whether it is worth exploring or exploiting:

$$-y^* + C_d^{''} \cdot \sum_{j=1}^{j^{''}} dy_{-j} > C_p^{''} \cdot \sqrt{\log n_{leaf}} \tag{2}$$

To summarize, we have the budget ratio, step size at local descent, function value at which using fine-grained descent, C_d and C_p at computing node UCT, $C_p^{'}$ for branch exploration, and $C_d^{''}$ and $C_p^{''}$ for leaf exploration. The hyperparamters used for each test is as in Tab.2:

Table 2: Hyperparameters used in MCTD for each of the test functions

Functions	Bud. Rat.	α	Switch at $f(x)$	C_d	C_p	$C_p^{'}$	$C_d^{''}$	$C_p^{''}$
Ackley-50d	1:1	0.2	10	10	0.5	0.1	50	0.1
Ackley-100d	1:1	0.2	4	20	0.5	0.1	5	0.1
Michalewicz-100d	1:2	0.02	-30	50	1.	0.2	1	10
Hopper-33d	1:2	0.1	-1000	100	1	10	100	200
Walker-102d	1:2	0.01	-100	100	0.1	50	50	50
Walker-204d	1:2	0.01	-100	100	0.1	50	50	10
HalfCheetah-102d	1:2	0.01	35000	50	1	1000	100	10
CIFAR-10	1:4	0.5	5	50	1	1	100	10
CIFAR-100	1:4	0.5	5	50	1	1	100	10

C Best Found Value in Test Functions

On each of the functions tested, we present the best results using different methods and the fewest steps to achieve that result. Note for function Ackley-50d, Ackley-100d, and Michalewicz-100d we want to minimize the function value. In contrast, for MuJoCo tasks and NAS tests we want to find the highest reward or the highest accuracy.

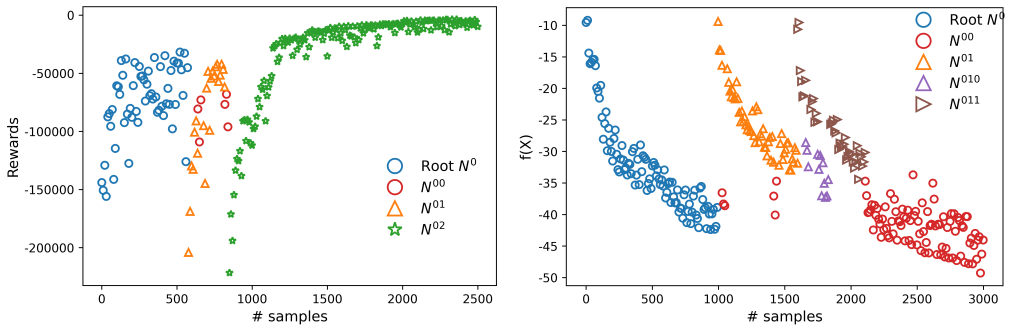
Table 3: Best Found Value / earliest step toward reaching that value from all tested functions. Bolded result is the best one among all tested methods.

Functions	MCTD	TuRBO	LaMCTS	CMA	Nealder-Mead	RandomSearch
Ackley-50d	**0.07/2342**	1.33/1889	0.80/2018	0.13/3000	13.21/1225	12.32/1311
Ackley-100d	**0.29/2826**	2.81/2891	1.89/2971	1.77/3401	13.34/1616	12.37/1326
Michalewicz-100d	**-51.13/2975**	-49.08/1776	-49.87/2945	-40.35/9015	-27.69/2017	-21.22/1759
Hopper-33d	3204/1890	**3397/2574**	2802/2858	3043/4128	67/4603	1220/193
Walker-102d	490/2472	**665/1316**	379/2056	657/3264	-4/414	91/1957
Walker-204d	**993/2884**	862/1673	498/1830	551/3386	-	-
HalfCheetah-102d	**-3268/2446**	-4679/2064	-4034/2970	-22062/3145	-101228/2271	-50542/1145
CIFAR-10	**91.82/86**	91.82/1296	91.48/2724	91.70/198	-	91.56/458
CIFAR-100	**73.52/22**	73.52/80	73.49/2433	73.52/135	-	73.52/338

Tab.3 illustrates that our MCTD approach obtains the best value with relatively fewer steps in most of the test cases out of five attempts. In particular, our approach performs reasonably well for cases where descent optimization can significantly improve the optimization performance. However, it should be noted that in some instances our approach leads to large variations between the different attempts. The adjustment of sample methods may provide one method for improving the descent optimization on those functions.

D Selection of Nodes

It is important to justify the expansion of the tree. Fig. 1 illustrates the nodes from which the query is made for the objective function. In Fig. 1(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, N^{01}, and starts optimizing from this child. Nonetheless,



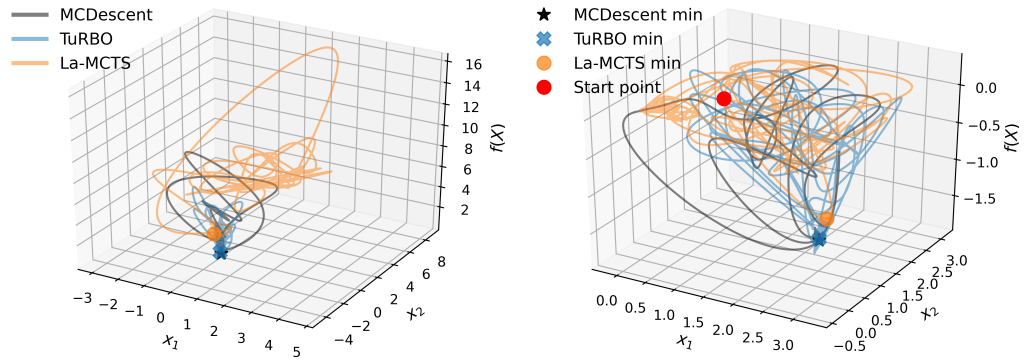
(a) Node queries on HalfCheetah-102d (b) Node queries on Michalewicz-100d

Figure 1: Illustration of nodes at queries to the objective function

the optimization is also stuck after 200 more queries. Therefore, our tree abandons to optimize in N^{01}, and adds a new child node named N^{02}. On N^{02}, the optimization procedure is significant, and a new best value is found. The tree in Fig. 1(b) attempts to optimize in the N^{01} and its new exploration child N^{011}, 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 N^{01} and N^{011}, 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.

E Optimization route

Fig. 2 illustrates how MCTD, 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. MCTD, on the other hand, samples much closer to the global optimal point and thus finds the solution more rapidly.



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

Figure 2: Search paths of different algorithms. The black star, the blue cross, and the orange circle indicate the best values found by MCTD, TuRBO, and LaMCTS, respectively; the red dot represents the starting point of all three methods.