Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search

- 作者：Linnan Wang，Rodrigo Fonseca，Yuandong Tian
- 机构：Brown University, Facebook AI research

- 会议：NIPS 2020
- 地址：http://arxiv.org/abs/2007.00708
- 代码：https://github.com/facebookresearch/LaMCTS

论文主要内容

摘要

高维黑盒优化受维度诅咒是个挑战问题，greedy search容易导致局部最优。本文的La-MCTS通过递归地把搜索空间用非线性决策边界划分为low/high function values区域，每个区域都用一个local model去拟合。La-MCTS作为一种*meta-algorithm*能够使用已有的许多黑盒优化器（如BO、TuRBO）作为local models，在通用黑盒优化、强化学习benchmark上表现突出，尤其是对于高维问题。

贡献

1. 提出了一种可学习、自适应的区域划分方法，避免在高维问题中的过分探索

缺陷

1. 处理noisy functions?

介绍

在黑盒优化问题中，我们有一个无显式表达式的函数 f 。目标是**用最小的代价**找使得函数达到最小的点 x^* ：

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in X} f(\mathbf{x})$$

直接去优化这样的黑盒函数最坏情况时间复杂度是指数级（grid search）。一种常用处理办法是通过*learning:*

1. 使用**少量**观测样本 $(x_i, f(x_i))$ 拟合一个surrogate regressor（代理模型） $\hat{f} \in \mathcal{H}$
    a. $\hat{f}$ 能比较好的近似 f
    b. Model class $\mathcal{H}$ 比较小（为了能用较少的样本拟合得到 $\hat{f}$ ）
2. 优化 $\hat{f}$ 代替优化 f

具体算法有Bayesian optimizeation等

Algorithm 1 Bayesian optimization

for $n = 1, 2, \ldots$ **do**
  select new $\mathbf{x}_{n+1}$ by optimizing acquisition function α

$$\mathbf{x}_{n+1} = \arg\max_{\mathbf{x}} \alpha(\mathbf{x}; \mathcal{D}_n)$$

  query objective function to obtain y_{n+1}
  augment data $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (\mathbf{x}_{n+1}, y_{n+1})\}$
  update statistical model
end for

研究内容

Motivation

- 当 f 高度非线性或高维的时候，需要非常大的 $\mathcal{H}$ (如GP，DNN)。也就需要非常多的样本去拟合
- 黑盒优化算法在高维问题中过度exploration（维度诅咒）
- 一些相关工作使用空间划分（利用**固定的准则**，独立于具体的优化问题）来处理高维问题

本文提出的算法通过*learning*得到空间划分

方法

定义

D_t 代表在第 t 次迭代下拥有的观测数据集 $(\mathbf{x}_i, f(\mathbf{x}_i))$

Ω 代表整个搜索空间

Tree node：

- 一个tree node表示了一个区域。如node A 表示区域 Ω_A
- 每个node记录被访问的次数 n_i 。如 n_A ，即 $\#(D_t \cap \Omega_A)$
  - 即落在该区域内的观测样本数
- 每个node拥有一个node value v_i 。 $v_i = \frac{1}{n_i} \sum f(\mathbf{x}_i), \forall \mathbf{x}_i \in D_t \cap \Omega_i$
  - 即在该区域内的观测样本performance均值

Latent actions:

- 将一个node表示的区域划分成两个区域（high performance、low performance）。
  - 即把一个节点A划分成左右子节点B、C。 $\Omega_A = \Omega_B \cup \Omega_C$
  - 默认左节点是high performance，右节点是low performance

用latent action可以将这个搜索空间 Ω 递归的划分成小区域 Ω_{leaves} ，并且可以很容易rank这些区域（从最左边的叶子节点到最右边的叶子节点）

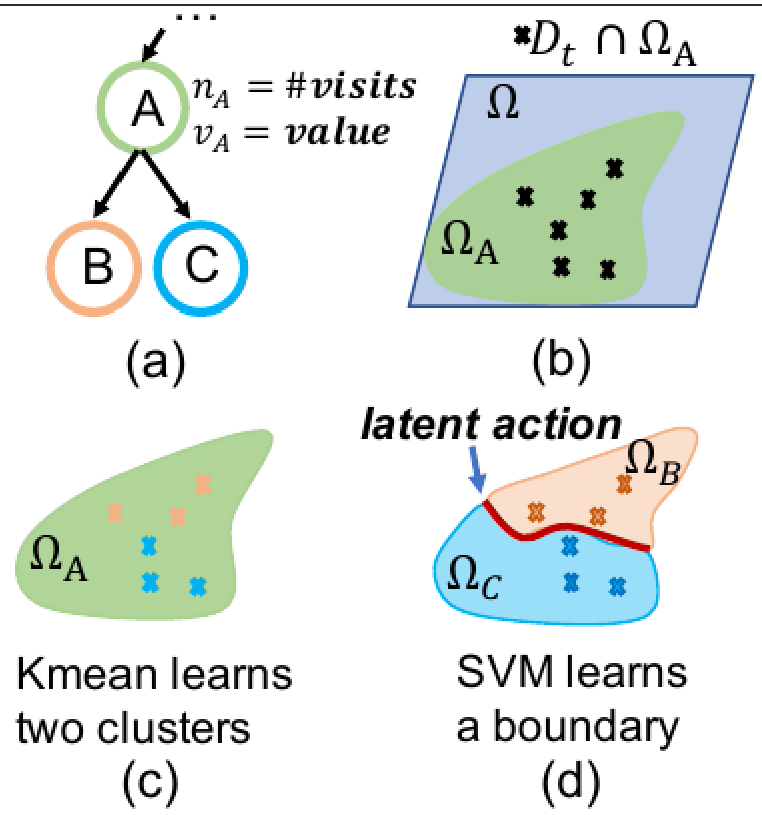


Figure 1: **the model of latent actions**: each tree nodes represents a region in the search space, and *latent action* splits the region into a high-performing and a low-performing region using $\mathbf{x}$ and $f(\mathbf{x})$.

- Latent action细节：
    - 准备训练数据 $\forall\left[\mathbf{x}_i, f\left(\mathbf{x}_i\right)\right], i \in D_t \cap \Omega_j$
    - kmeans在训练数据上训练（两个cluster），获得每个样本的簇标签 $\left[l_i, \mathbf{x}_i\right]$
    - 用 $\left[l_i, \mathbf{x}_i\right]$ 数据训练SVM得到分类器。即 $\forall \mathbf{x}_i \in \Omega$ 都能预测属于high-perfomace regin（左子节点）还是low-performance regin（右子节点）

算法流程

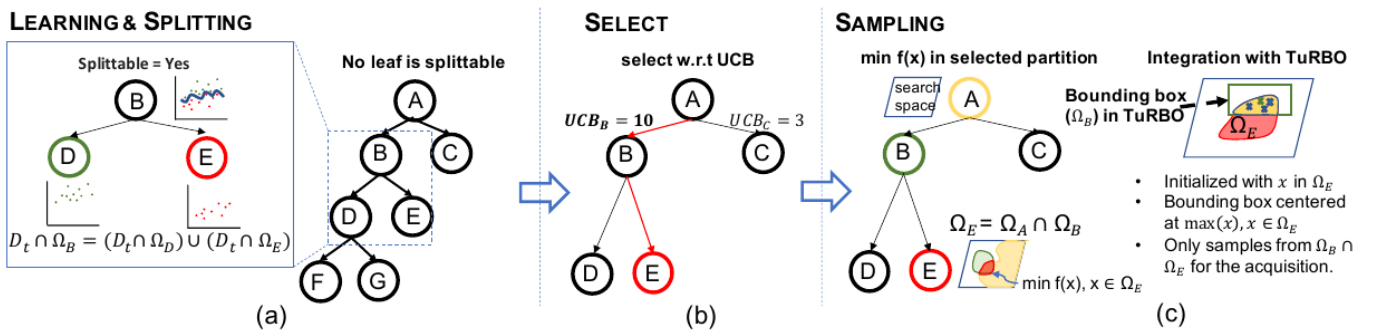


Figure 2: **the workflow of LA-MCTS**: In an iteration, LA-MCTS starts with building the tree via splitting, then it selects a region based on UCB. Finally, on the selected region, it samples by BO.

一次迭代过程：

1. Dynamic tree construction via splitting
    a. 一旦 $\#\left(D_t \cap \Omega_A\right)$ 超过阈值 θ (超参)，就split这个叶节点，直到不满足split条件
    b. 树的结构随着迭代动态变化。
2. Select via UCB (Upper Confidence Bound)
    a. 直接简单的在最左边的叶子节点搜索会导致over-exploiting
    b. 计算每个node的UCB得分，从root开始每次选择最大的UCB得分的子节点，形成一个path
    c. $ucb_j=\frac{v_j}{n_j}+2 C_p * \sqrt{2 \log \left(n_p\right) / n_j}$ ，其中的 C_p 是可调超参来控制exploration程度

```
    def get_uct(self, Cp = 10 ):
        if self.parent == None:
            return float('inf')
        if self.n == 0:
            return float('inf')
        return self.x_bar + 2*Cp*math.sqrt( 2* np.power(self.parent.n, 0.5) / self.n )
```

3. Sampling via Bayesian Optimizations
    a. 上一步中path的叶节点区域为 $\Omega_{selected}$ ，在这个限定区域内通过贝叶斯优化得到下一个 x_{new}
4. 评估黑盒函数在 x_{new} 上的值 $f(x_{new})$ 。 $D_{t+1}=D_t \cup\left(x_{new}, f\left(x_{new}\right)\right)$ ，进入下一轮迭代

实验结果

Mujoco locomotion tasks

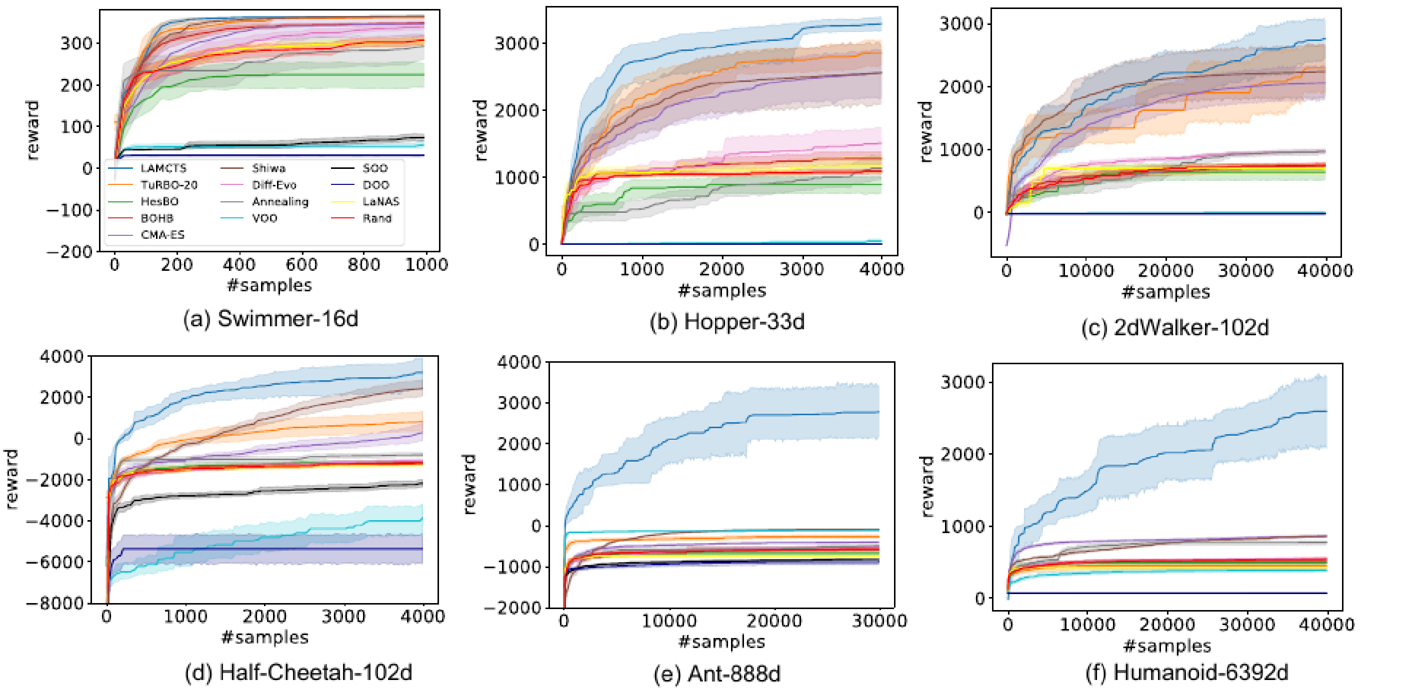


Figure 3: **Benchmark on MuJoCo locomotion tasks**: LA-MCTS consistently outperforms baselines on 6 tasks. With more dimensions, LA-MCTS shows stronger benefits (e.g. Ant and Humanoid). This is also observed in Fig. 4. Due to exploration, LA-MCTS experiences relatively high variance but achieves better solution after 30k samples, while other methods quickly move into local optima due to insufficient exploration.

与其他黑盒优化方法对比

- 在6个任务上都超过baseline
- 对于更高维度的任务，LA-MCTS的表现优势更明显
- 由于exploration，LA-MCTS相对有更高的variance，但是能够获得更好的solution。而其他方法很快陷入局部最优

Table 2: Compare with gradient-based approaches. Despite being a black-box optimizer, LA-MCTS still achieves good sample efficiency in low-dimensional tasks (*Swimmer*, *Hopper* and *HalfCheetah*), but lag behind in high-dimensional tasks due to excessive burden in exploration, which gradient approaches lack.

		The average episodes (#samples) to reach the threshold				
Task	**Reward Threshold**	**LA-MCTS**	**ARS V2-t [54]**	**NG-lin [55]**	**NG-rbf [55]**	**TRPO-nn [54]**
Swimmer-v2	325	**132**	427	1450	1550	N/A
Hopper-v2	3120	2897	1973	13920	8640	10000
HalfCheetah-v2	3430	3877	1707	11250	6000	4250
Walker2d-v2	4390	N/A($r_{best} = 3314$)	24000	36840	25680	14250
Ant-v2	3580	N/A($r_{best} = 2791$)	20800	39240	30000	73500
Humanoid-v2	6000	N/A($r_{best} = 3384$)	142600	130000	130000	unknown

N/A stands for not reaching reward threshold.
r_{best} stands for the best reward achieved by LA-MCTS under the budget in Fig. 3

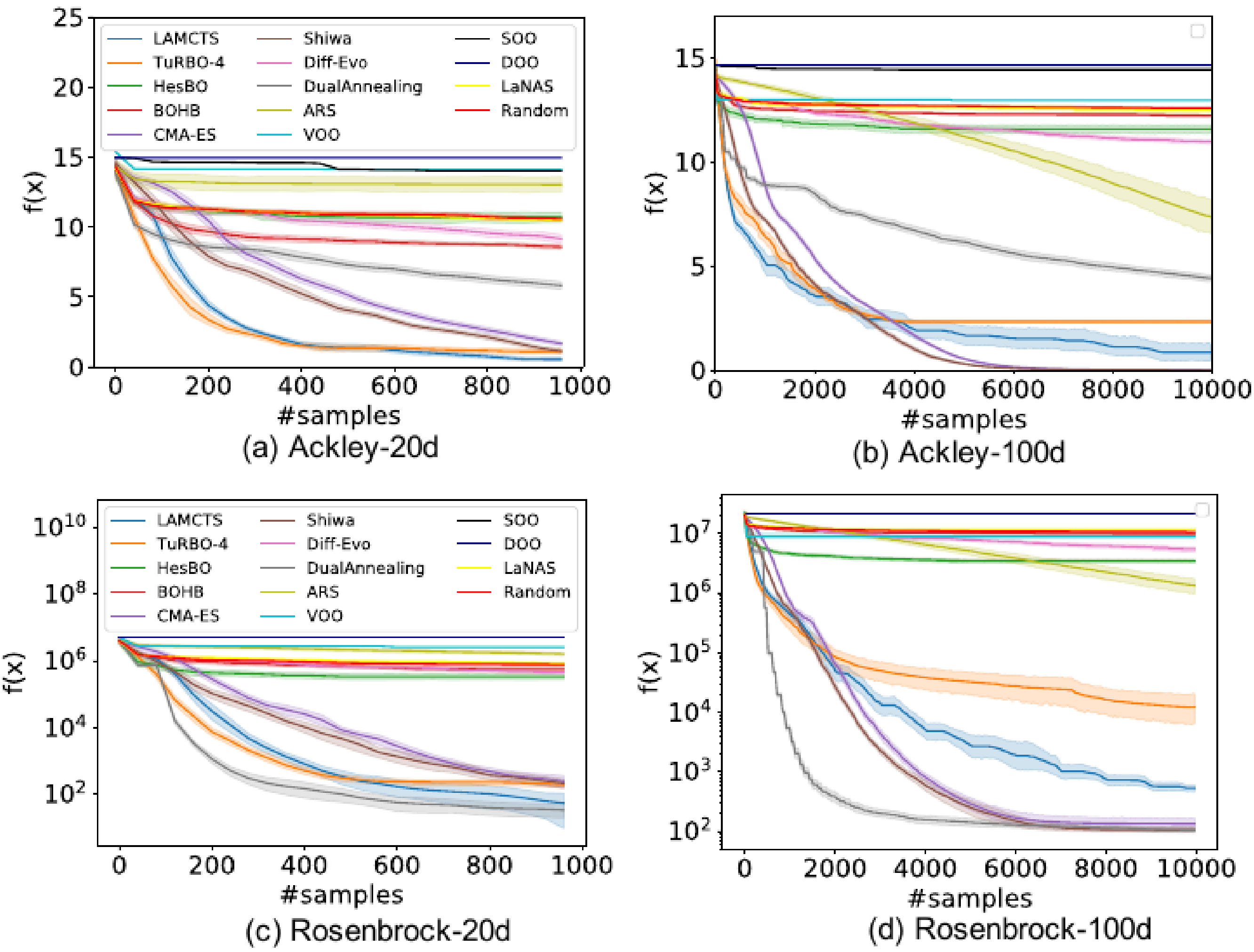
与gradient-based方法对比

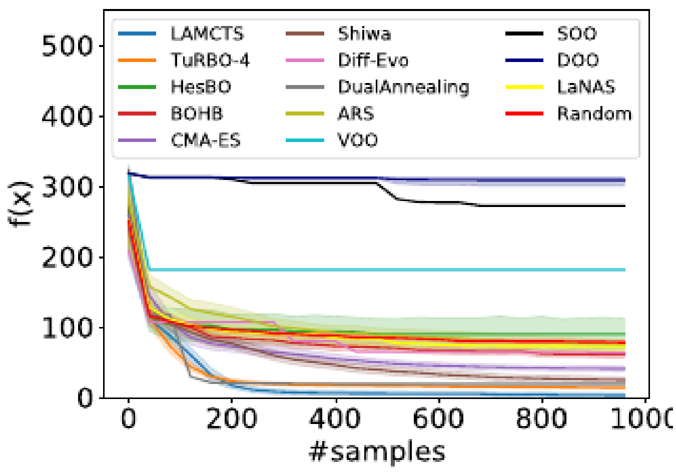
- 在低维任务上依然能有好的sample efficiency

- 但在高维任务落后于gradient-based方法（黑盒优化的exploration性质）

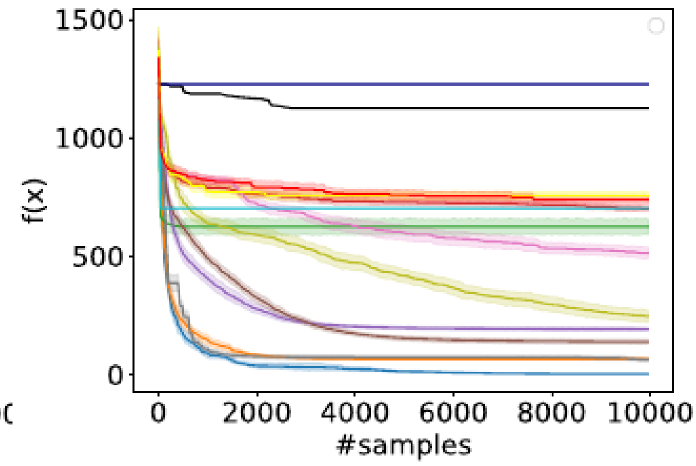
Small-scale Benchmarks

Synthetic functions

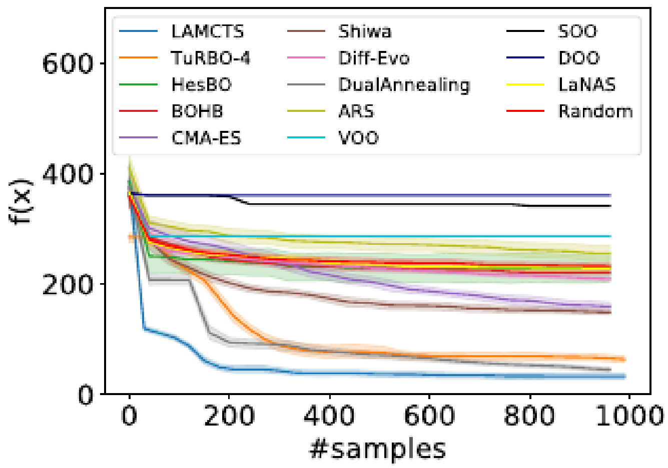




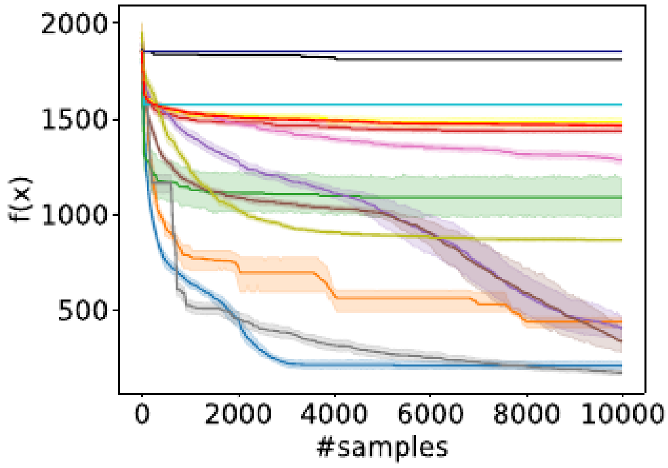
(e) Levy-20d



(f) Levy-20d

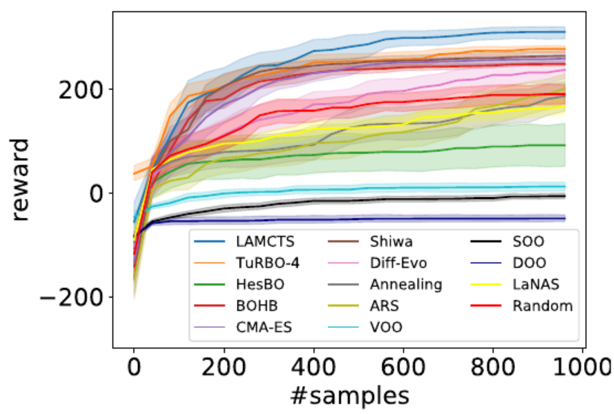


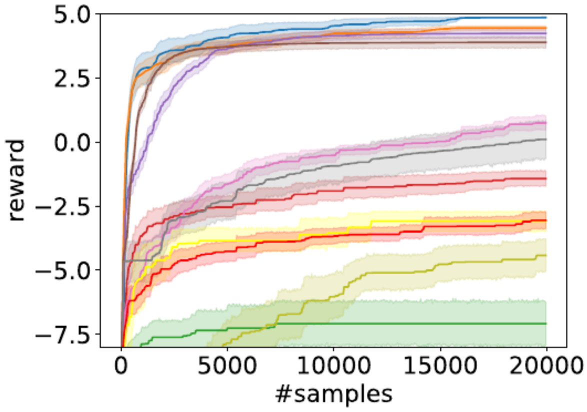
(g) Rastrigin-20d



(h) Rastrigin-100d

Lunar Landing & Rover-60d





(a) Lunar landing, #params = 12 (b) Rover trajectory planning, #params = 60

Figure 9: **evaluations on Lunar landing and Trajectory Optimization**: LA-MCTS consistently outperforms baselines.

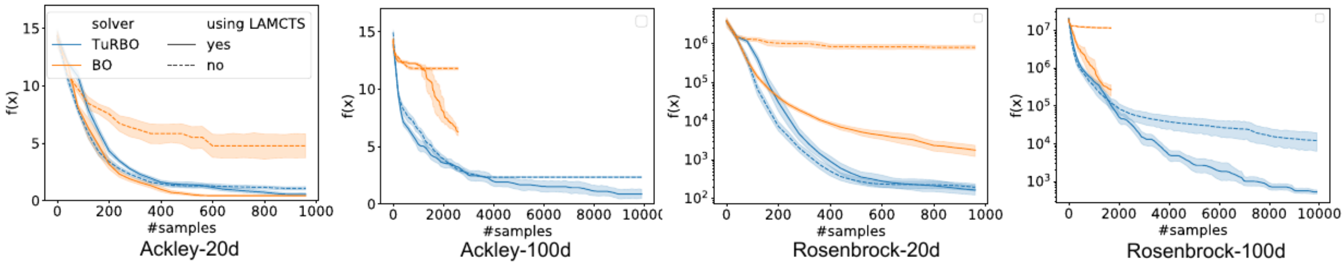
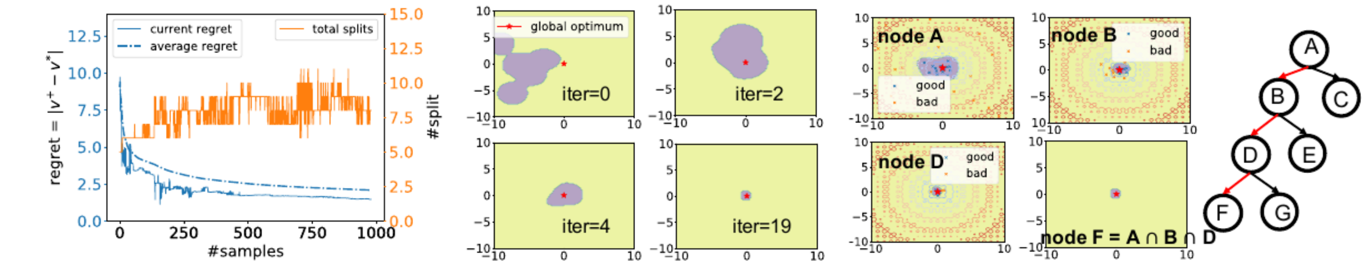


Figure 4: **LA-MCTS as an effective meta-algorithm**. LA-MCTS consistently improves the performance of TuRBO and BO, in particular in high-dimensional cases. We only plot part of the curve (collected from runs lasting for 3 day) for BO since it runs very slow in high-dimensional space.

- 最为一个meta-algorithm，使用BO作为子算法也是有效的



(a) the progress of the value of selected node v^+ to the global optimum $v^* = 0$, and the #splits. (b) the visualization of selected region at different search iterations (c) the selected region (iter=19 in (b)) is collectively bounded by SVMs in the path.

Figure 5: **Validation of LaMCTS**: (a) the value of selected node becomes closer to the global optimum as #splits increases. (b) the visualization of $\Omega_{selected}$ in the progress of search. (c) the visualization of $\Omega_{selected}$ that takes the intersection of nodes on the selected path.

- 随着样本增加，每次迭代选中的叶子节点的performance估计逐渐接近实际值，图（a）
- 选中区域也逐渐收敛到全局最优点，图（b）
- 可视化svm决策边界，图（c）

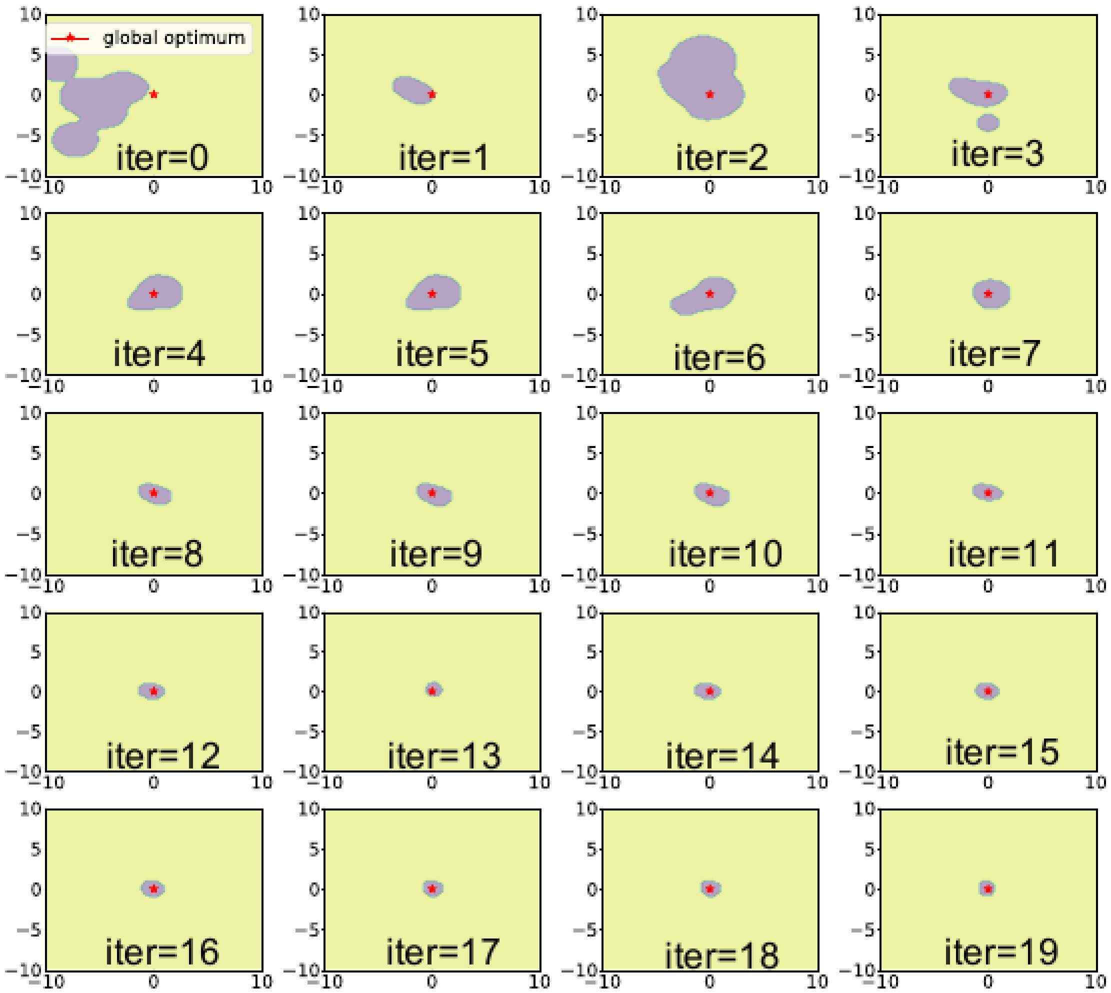
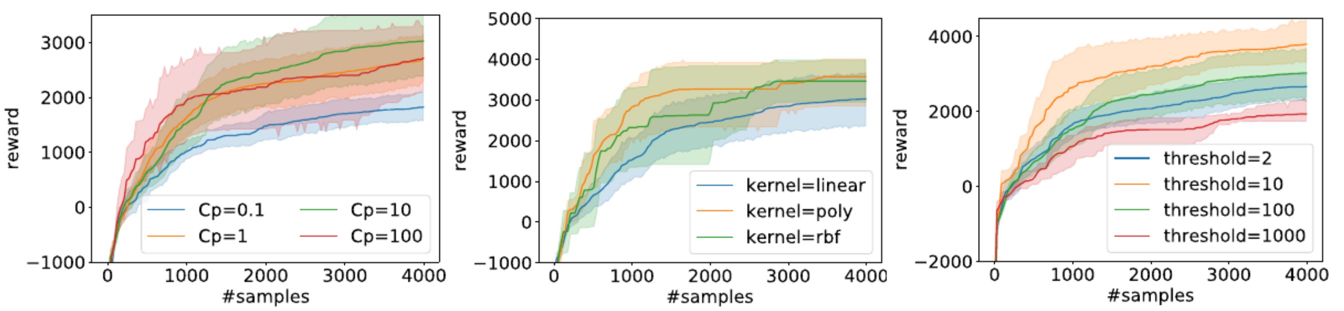


Figure 10: **the visualization of LA-MCTS in iterations 1→20**: the purple region is the selected region $\Omega_{selected}$, and the red star represents the global optimum.

消融实验

超参：UCB中的 C_p 、SVM中的kernel type、splitting阈值 θ



(a) Ablation on Cp (b) Ablation on SVM kernel (c) Ablation on splitting threshold

Figure 6: Ablation studies on hyper-parameters of LAMCTS.

- C_p 设f (x)最大值的 1%~10%
- kernel类型决定了划分边界的形状。poly和rbf有非线性性质，能够生成任意形状的区域边界
- θ 决定了树生成的速度，更小的阈值会产生更深的树

- 对于大的搜索空间，小的阈值能够快速将区域划分成小区域
- 但阈值过小容易造成performance和边界估计不可靠