Monte Carlo Tree Search in Continuous Spaces Using Voronoi Optimistic Optimization with Regret Bounds

Beomjoon Kim[1], **Kyungjae Lee**[2], **Sungbin Lim**[3], **Leslie Pack Kaelbling**[1], **and Tomás Lozano-Pérez**[1]
[1]MIT Computer Science and Artificial Intelligence Laboratory
{beomjoon,lpk,tlp}@mit.edu,
[2]Seoul National University
kyungjae.lee@cpslab.snu.ac.kr,
[3]KakaoBrain
leo.brain@kakaobrain.com

Abstract

Many important applications, including robotics, data-center management, and process control, require planning action sequences in domains with continuous state and action spaces and discontinuous objective functions. Monte Carlo tree search (MCTS) is an effective strategy for planning in discrete action spaces. We provide a novel MCTS algorithm (VOOT) for deterministic environments with continuous action spaces, which, in turn, is based on a novel black-box function-optimization algorithm (VOO) to efficiently sample actions. The VOO algorithm uses Voronoi partitioning to guide sampling, and is particularly efficient in high-dimensional spaces. The VOOT algorithm has an instance of VOO at each node in the tree. We provide regret bounds for both algorithms and demonstrate their empirical effectiveness in several high-dimensional problems including two difficult robotics planning problems.

Introduction

We are interested in finite-horizon deterministic planning problems with high-dimensional continuous action spaces, with possibly a discontinuous objective function. For example, consider the sequential robot mobile-manipulation planning problem shown in Figure 1 (left). In this domain, the objective function is defined to be the number of objects that the robot packs into the storage room while satisfying feasibility conditions, such as collision-free motions, and minimizing the total length of its trajectory. Another example is shown in Figure 1 (right), where the task is to clear obstacles from a region, and the objective is a function of the number of obstacles cleared and trajectory length. In both cases, the robot's action space is high dimensional, consisting of multiple pick or placement configurations of the robot.

More generally, such discontinuous objective functions are the sum of a finite set of step functions in a high-dimensional state-action space, where each step corresponds to the occurrence of an important event, such as placing an object. For classes of functions of this kind, standard gradient-based optimization techniques are not directly applicable, and even if we smooth the objective function, the solution is prone to local optima.

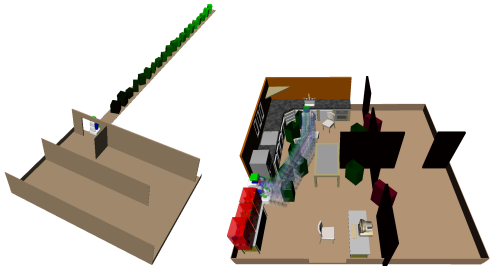

Figure 1: Packing domain: the task is to pack as many objects coming from a conveyor belt into the room (left). Object-clearing domain: obstacles must be cleared from the swept-volume of a path to the sink (right). In both domains, the robot needs to minimize the overall trajectory length.

Recently, several gradient-free approaches to continuous-space planning problems have been proposed (Buşoniu et al. 2011; Munos 2014; Weinstein and Littman 2012; Mansley, Weinstein, and Littman 2011), some of which have been proven to asymptotically find a globally optimal solution. These approaches either frame the problem as simultaneously optimizing a whole action sequence (Buşoniu et al. 2011; Weinstein and Littman 2012) or treat the action space in each node of a tree search (Mansley, Weinstein, and Littman 2011) as the search space for a budgeted-black-box function optimization (BBFO) algorithm, and use hierarchical-partitioning-based optimization algorithms (Munos 2011; Bubeck et al. 2011) to approximately find the globally optimal solution.

While these hierarchical-partitioning algorithms handle a richer class of objective functions than traditional methods (Pintér 1996), their main drawback is poor scalability to high-dimensional search spaces: to optimize efficiently, these algorithms sequentially construct partitions of the search space where, at each iteration, they create a finer-resolution partition inside the most promising cell of the current partition. The problem is that constructing a partition requires deciding the optimal dimension to cut, which is a difficult combinatorial problem especially in a high-dimensional space. Figure 2 (left) illustrates this issue with one of the algorithms, DOO (Munos 2011).

We propose a new BBFO algorithm called Voronoi Optimistic Optimization (VOO) which, unlike the previous approaches, only implicitly constructs partitions, and so scales to high-dimensional search spaces more effectively. Specifically, partitions in VOO are Voronoi partitions whose cells are implicitly defined as the set of all the points that are closer to the generator than to any other evaluated point. Figure 2 (right) shows an example.

Given as inputs a semi-metric, a bounded search space, and an exploration probability ω, VOO operates similarly to the previous partition-based methods: at each iteration, it selects (implicitly) a Voronoi cell based on a simple exploration-exploitation scheme, samples a point from the cell, and (implicitly) makes finer-resolution cells inside the selected cell based on the sampled point. The selection of a Voronoi cell is based on the given exploration probability: with probability ω, it explores by selecting a cell with probability proportional to the volume of the cell; with probability $1 - \omega$, it exploits by selecting the cell that contains the current best point. Unlike the previous methods, however, VOO never explicitly constructs the partitions: by using the definition of Voronoi partition and the given semi-metric, sampling from the best cell is implemented simply using rejection sampling. Sampling a point based on the volumes of the cells, which is also known as the *Voronoi bias* (Kuffner and LaValle 2000), is also simply implemented by sampling uniformly at random from the search space. Figure 1 (right) demonstrates this point. We prove the regret bound of VOO which shows that under some mild assumptions, the regret goes to zero.

Using VOO, we propose a novel continuous state-action-space Monte Carlo tree search (MCTS) algorithm, *Voronoi optimistic optimization applied to trees* (VOOT) that uses VOO at each node of the search tree to select the optimal action, in a similar fashion to HOOT (Mansley, Weinstein, and Littman 2011). HOOT, however, does not come with performance guarantees; we are able to prove a performance guarantee for VOOT, which is derived from a bound on the regret of VOO. The key challenge in showing this result is that, when VOO is used to optimize the state-action value function of a node in the tree, the value function is non-stationary, so that even when the environment is deterministic, its value changes as the policy at the sub-tree below the action changes. We address this problem by using the regret of VOO at the leaf nodes, whose value function is stationary, and computing how many re-evaluations at each depth is required to maintain the same regret at the root node as at the leaf node. We show this regret can be made arbitrarily small.

We compare VOO to several algorithms on a set of standard functions for evaluating black-box function optimization algorithms in which the number of dimensions of the search space is as high as 20, and show that VOO significantly outperforms the benchmarks, especially in high dimensions. To evaluate VOOT, we compare it to other continuous-space MCTS algorithms in the two sequential robot mobile-manipulation problems shown in Figure 1, and show that VOO computes significantly better quality plans than the benchmarks, within a much smaller number of iterations.

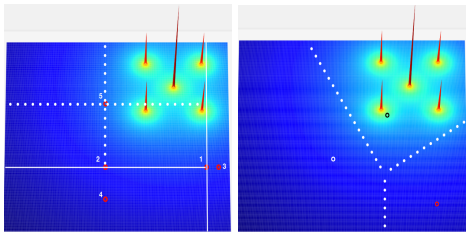

Figure 2: Left: Illustrations of a partition made by DOO when five points are evaluated to optimize a 2D Shekel function. Each solid line shows the partitions made by the point that is on it. Numbers indicate the order of evaluations. The dotted lines indicate the two possible partitions that can be made by the fifth point, and depending on this choice, the performance differs. Right: Illustration of the Voronoi partition implicitly constructed by VOO. We can sample from the best Voronoi cell (defined by the black point) by random-sampling points, and rejecting them until we obtain one that is closer to the black point than the other points. We can sample a point with Voronoi bias by uniformly sampling from the entire search space; the cell defined by the white point is most likely to be selected.

Related work

There are several planning methods that use black-box function optimization algorithms in continuous-space problems. We first give an overview of the BBFO algorithms, and then describe planning algorithms that use them. We then give an overview of progressive-widening approaches, which are continuous-space MCTS algorithms that do not use black-box function optimization methods.

Global optimization of black-box functions with budget

Several partition-based algorithms have been proposed (Munos 2011; Bubeck et al. 2011; Munos 2014). In (Munos 2011), two algorithms are proposed. The first algorithm is DOO, which requires as inputs a semi-metric and the Lipschitz constant for the objective function. It sequentially constructs partitions of the search space, where a cell in the partition has a representative point, on which the objective function is evaluated. Using the local-smoothness assumption, it builds an upper-bound on the un-evaluated points in each cell using the distance from the representative point. It chooses the cell with the highest-upper bound, and creates a finer-resolution cell inside of it, and repeats. The second algorithm proposed in (Munos 2011) is SOO, which does not require a Lipschitz constant, and evaluates all cells that might contain the global optimum. In (Bubeck et al. 2011), Hierarchical Optimistic Optimization (HOO) is proposed. Unlike SOO and DOO, HOO can be applied to optimize a noisy function, and can be seen as the stochastic counterpart of DOO. So far, these algorithms have been applied to problems with low-dimensional search spaces, because solving for the optimal sequence of dimensions to cut at each iteration is difficult. VOO gets around this problem by not explicitly building the partitions.

Alternatively, we may use Bayesian optimization (BO) algorithms, such as GP-UCB (Srinivas et al. 2010). A typical

BO algorithm takes as inputs a kernel function, and an exploration parameter, and assumes that the objective function is a sample from a Gaussian Process (GP). It builds an acquisition function, such as upper-confidence-bound function in GP-UCB (Srinivas et al. 2010), and it chooses to evaluate, at every iteration, the point that has the highest acquisition function value, updates the parameters of the GP, and repeats. The trouble with these approaches is that at every iteration, they require finding the global optimum of the acquisition function, which is expensive in high dimensions. In contrast, VOO does not require an auxiliary optimization step.

There have been several attempts to extend BO to high-dimensional search spaces (Wang et al. 2013; Kandasamy, Schneider, and Poczos 2015). However, they make a rather strong assumption on the objective function, such as that it lies on a low-dimensional manifold, or that it can be represented by a linear combination of functions of sub-dimensions, which are unlikely to hold in domains such as robotics, where all of the action dimensions contribute to its value. Also, these methods require extra hyperparameters that define the lower-dimensional search space that are tricky to tune. VOO requires neither the assumption or the hyperparameters for defining the low-dimensional search space.

There are also methods that try to combine BO and hierarchical partitioning methods, such as (Wang et al. 2014; Kawaguchi, Kaelbling, and Lozano-Pérez 2015). The idea is to use hierarchical partitioning methods to optimize the acquisition function of BO; unfortunately, for the same reason as hierarchical partitioning methods, they tend to perform poorly in higher dimensional spaces.

Optimal planning in continuous spaces using BBFO There are two approaches to continuous-space planning problems that use black-box function-optimization (BBFO) algorithms. In the first group of approaches, the entire sequence of actions is treated as a single search space for optimization. In (Weinstein and Littman 2012), the authors propose *hierarchical open-loop optimistic planning* (HOLOP), which uses HOO for finding finite-horizon plans in stochastic environments with continuous action space. In (Buşoniu et al. 2011), the authors propose an algorithm called *simultaneous optimistic optimization for planning* (SOOP), that uses SOO to find a plan when the environment is deterministic. These methods become very expensive as the length of the action sequence increases.

The second group of approaches, where our method belongs, performs a sample-based tree search with a form of continuous-space optimizer at each node. Our work most closely resembles *hierarchical optimistic optimization applied to trees* (HOOT) (Mansley, Weinstein, and Littman 2011), which applies *hierarchical optimistic optimization* (HOO) at every node in MCTS for the action-optimization problem, but does not provide any performance guarantees. These algorithms have been limited to problems with low-dimensional action space, such as the inverted pendulum. Our experiments demonstrate VOOT can solve problems with higher-dimensional action spaces much more efficiently than these algorithms.

Widening techniques for MCTS in continuous action spaces There are progressive-widening (PW) algorithms that extend MCTS to continuous action spaces (Couëtoux et al. 2011; Auger, Couëtoux, and Teytaud 2013), but unlike the approaches above, their main concern is deciding when to sample a new action, instead of which action to sample. The action-sampler in these PW algorithms is assumed to be an external function that has a non-zero probability of sampling a near-optimal action, such as a uniform-random sampler.

Typically, a PW technique (Couëtoux et al. 2011) ensures that the ratio between the number of sampled actions in a node to the number of visits to the node is above a given threshold. In (Auger, Couëtoux, and Teytaud 2013), the authors show that a form of PW can guarantee that each state's estimated value approaches the optimal value asymptotically. However, this analysis does not take into consideration the regret of the action sampler, and assumes that the probability of sampling a near-optimal action is the same in every visit to the node. So, if an efficient action-sampler, whose regret reduces quickly at each visit, is used, their error bound would be very loose. Our analysis shows how the regret of VOO affects the planning performance.

Monte Carlo planning in continuous state-action spaces

We have a continuous state space S, a continuous action space U, a deterministic transition model of the environment, $T : S \times U \to S$, a deterministic reward function $R : S \times U \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1)$. Our objective is to find a sequence of actions with planning horizon H that maximizes the sum of the discounted rewards $\max_{u_0,\cdots,u_{H-1}} \sum_{t=0}^{H-1} \gamma^t r(s_t, u_t)$ where $s_{t+1} = T(s_t, u_t)$. Our approach to this problem is to use MCTS with an action-optimization agent, which is an instance of a black-box function-optimization algorithm, at each node in the tree.

We now describe the general MCTS algorithm for continuous state-action spaces, which is given in Algorithm 1. The algorithm takes as inputs an initial state s_0, an action-optimization algorithm $\mathcal{A}$, the total number of iterations N_{iter}, the re-evaluation parameter $N_r \in [0, N_{iter}]$, and its decaying factor $\kappa_r \in [0, 1]$. It begins by initializing the necessary data in the root node. U denotes the set of actions that have been tried at the initial node, $\hat{Q}$ denotes the estimated state-action value of the sampled actions, and n_r denotes the number of times we re-evaluated the last-sampled action. It then performs N_{iter} Monte Carlo simulations, after which it returns the apparently best action, the one with the highest estimated state-action value. This action is executed, and we re-plan in the resulting state.

Algorithm 1 MCTS($s_0, \mathcal{A}, N_{iter}, N_r, \kappa_r, H, \gamma$)

1: global variables: $T, R, H, \gamma, \mathcal{A}, N_{iter}, \kappa_r, H, \gamma$
2: $\mathcal{T}(s_0) = \{U = \emptyset, \hat{Q}(s_0, \cdot) = -\infty, n_r = 0\}$
3: **for** $i = 1 \to N_{iter}$
4: SIMULATE($s_0, 0, N_r$)
5: **return** $\text{argmax}_{u \in \mathcal{T}(s_0).U} \mathcal{T}(s_0).\hat{Q}(s_0, u)$

Algorithm 2 SIMULATE(s, h, N_r)

global variables: $T, R, H, \gamma, \mathcal{A}, N_{iter}, \kappa_r, H, \gamma$
if $s ==$ infeasible or $h == H$
  return 0
if $(|\mathcal{T}(s).U| > 0) \wedge (\mathcal{T}(s).n_r < N_r) \wedge (h \neq H-1)$
  // re-evaluate the last added action
  $u = \mathcal{T}.U$.get_last_added_element()
  $\mathcal{T}(s).n_r = \mathcal{T}(s).n_r + 1$
else
  // Perform action optimization
  $u \sim \mathcal{A}(\mathcal{T}(s).\hat{Q})$
  $\mathcal{T}(s).U = \mathcal{T}(s).U \cup \{u\}$
  $\mathcal{T}(s).n_r = 1$
$s' = T(s, u)$
$r = R(s, u)$
$\hat{Q}_{new} = r + \gamma \cdot$ SIMULATE$(s', h+1, N_r \cdot \kappa_r)$
if $\hat{Q}_{new} > \mathcal{T}(s).\hat{Q}(s, u)$
  $\mathcal{T}(s).\hat{Q}(s, u) = \hat{Q}_{new}$
return $\mathcal{T}(s).\hat{Q}(s, u)$

Procedure SIMULATE is shown in Algorithm 2. It is a recursive function whose termination condition is either encountering an infeasible state or reaching a depth limit. At the current node $\mathcal{T}(s)$, it either selects the action that was most recently sampled, if it has not yet been evaluated N_r times and we are not in the last layer of the tree, or it samples a new action. To sample a new action, it calls $\mathcal{A}$ with estimated Q-values of the previously sampled actions, $\mathcal{T}(s).\hat{Q}$. A transition is simulated based on the selected action, and the process repeats until a leaf is reached; Q-value updates are performed on a backward pass up the tree if a new solution with higher value has been found (note that, because the transition model is deterministic, the update only requires maximization.)

The purpose of the re-evaluations is to mitigate the problem of non-stationarity: an optimization algorithm $\mathcal{A}$ assumes it is given evaluations of a stationary underlying function, but it is actually given $\hat{Q}(s, a_t)$, whose value changes as more actions are explored in the child sub-tree. This problem is also noted in (Mansley, Weinstein, and Littman 2011). So, we make sure that $\hat{Q}(s, a_t) \approx Q^*(s, a_t)$ before adding an action a_{t+1} in state s by sampling more actions at the sub-tree associated with a_t. Since at the leaf node $Q^*(s, a_t) = R(s, a_t)$, we do not need to re-evaluate actions in leaf nodes. In section 5, we analyze the impact of the estimation error in $\hat{Q}$ on the performance at the root node.

One may wonder if it is worth it to evaluate the sampled actions same number of times, instead of more sophisticated methods such as Upper Confidence Bound (UCB), for the purpose of using an action-optimization algorithm $\mathcal{A}$. Typical continuous-action tree search methods perform progressive widening (PW) (Couëtoux et al. 2011; Auger, Couëtoux, and Teytaud 2013), in which they sample new actions from the action space uniformly at random, but use UCB-like strategies for selecting which of the previously-sampled actions to explore further. In this case, the objective for allocating trials is to find the highest-value action among a discrete set, not to obtain accurate estimates of the values of all the actions.

VOOT operates in continuous action spaces but performs much more sophisticated value-driven sampling of the continuous actions than PW methods. To do this, it needs accurate estimates of the values of the actions it has already sampled, and so we have to allocate trials even to actions that may currently "seem" suboptimal. Our empirical results show that this trade-off is worth making, especially in high-dimensional action spaces.

Voronoi optimistic optimization

Given a bounded search space $\mathcal{X}$, a deterministic objective function $f : \mathcal{X} \to \mathbb{R}$ and a numerical function evaluation budget n, our goal is to devise an exploration strategy over $\mathcal{X}$ that, after n evaluations, minimizes the *simple regret* defined as $f(x_\star) - \max_{t\in[n]} f(x_t)$, where $f(x_\star) = \max_{x\in\mathcal{X}} f(x)$, x_t is a point evaluated at iteration t, and $[n]$ is shorthand for $\{1, \cdots n\}$. Since our algorithm is probabilistic, we will analyze its expected behavior. We define the simple regret of a probabilistic optimization algorithm $\mathcal{A}$ as

$$\mathcal{R}_n = f(x_\star) - \mathbb{E}_{x_{1:t}\sim\mathcal{A}}\left[\max_{t\in[n]} f(x_t)\right]$$

Our algorithm, VOO (Algorithm 3), operates by implicitly constructing a Voronoi partition of the search space $\mathcal{X}$ at each iteration: with probability ω, it samples from the entire search space, to sample from a Voronoi cell with probability proportional to its volume; with probability $1-\omega$, it samples from the *best Voronoi cell*, which is the one induced by the current best point, $x_t^* = \arg\max_{i\in[t]} f(x_i)$.

Algorithm 3 VOO$(\mathcal{X}, \omega, d(\cdot,\cdot), n)$

for $t = 0 \to n-1$
  Sample $\nu \sim Unif[0, 1]$
  if $\nu \leq \omega$ **or** $t == 0$
    $x_{t+1} =$UNIFSAMPLE$(\mathcal{X})$
  else
    $x_{t+1} =$SAMPLEBESTVCELL$(d(\cdot,\cdot))$
  Evaluate $f_{t+1} = f(x_{t+1})$
return $\arg\max_{t\in\{0,\ldots,n-1\}} f_t$

It takes as inputs the bounded search space $\mathcal{X}$, the exploration probability ω, a semi-metric $d(\cdot,\cdot)$, and the budget n. The algorithm has two sub-procedures. The first one is UNIFSAMPLE, which samples a point from $\mathcal{X}$ uniformly at random, and SAMPLEBESTVCELL, which samples from the best Voronoi cell uniformly at random. The former implements exploration using the Voronoi bias, and the latter implements exploitation of the current knowledge of the function. Procedure SAMPLEBESTVCELL can be implemented using a form of rejection sampling, where we sample a point x at random from $\mathcal{X}$ and reject samples until $d(x, x_t^*)$ is the minimum among all the distances to the evaluated points. Efficiency can be increased by sampling from a Gaussian centered at x_t^*, which we found to be effective in our experiments.

To use VOO as an action optimizer in Algorithm 2, we simply let U be the search space, and use the semi-metric

$d(\cdot,\cdot)$. $f(\cdot)$ is now the value function $Q^*(s,\cdot)$ at each node of the tree, whose estimation is $\hat{Q}(s,\cdot)$. The consequence of having access only to $\hat{Q}$ instead of the true optimal state-action value function Q^* will be analyzed in the next section.

Analysis of VOO and VOOT

We begin with definitions. We denote the set of all global optima as $\mathcal{X}^\star$, the Voronoi cell generated by a point x as $\mathcal{C}(x)$. We define the *diameter* of $\mathcal{C}(x)$ as $\sup_{y\in\mathcal{C}(x)} d(x,y)$ where $d(\cdot,\cdot)$ is the semi-metric on $\mathcal{X}$.

Suppose that we have a Voronoi cell generated by x, $\mathcal{C}_0(x)$. When we randomly sample a point z from $\mathcal{C}_0(x)$, this will create two new cells, one generated by x, which we denote with $\mathcal{C}_1(x)$, and the other generated by z, denoted $\mathcal{C}_1(z)$. The diameters of these new cells would be random variables, because z was sampled randomly. Now suppose that we have sampled a sequence of n_0 points from the sequence of Voronoi cells generated by x, $\{\mathcal{C}_0(x),\mathcal{C}_1(x),\mathcal{C}_2(x),\cdots,\mathcal{C}_{n_0}(x)\}$. Then, we define the *expected diameter* of a Voronoi cell generated by x as the expected value of the diameter of the last cell, $\mathbb{E}[\sup_{y\in\mathcal{C}_{n_0}(x)} d(x,y)]$.

We write δ_{max} for the largest distance between two points in $\mathcal{X}$, $B_r(x)$ to denote a ball with radius r centered at point x, and $\bar{\mu}_B(r) = \frac{\mu(B_r(\cdot))}{\mu(\mathcal{X})}$ where $\mu(\cdot)$ is a Borel measure defined on $\mathcal{X}$. We make the following assumptions:

A 1. *(Translation-invariant semi-metric)* $d:\mathcal{X}\times\mathcal{X}\to\mathbb{R}^+$ *is such that* $\forall x,y,z\in\mathcal{X}$, $d(x,y)=d(y,x)$, $d(x,y)=0$ *if and only if* $x=y$, *and* $d(x+z,y+z)=d(x,y)$.

A 2. *(Local smoothness of* f*)* *There exists at least one global optimum* $x_\star\in\mathcal{X}$ *of* f *such that* $\forall x\in\mathcal{X}, f(x_\star)-f(x)\le L\cdot d(x,x_\star)$ *for some* $L>0$.

A 3. *(Shrinkage ratio of the Voronoi cells) Consider any point* y *inside the Voronoi cell* $\mathcal{C}$ *generated by the point* x_0, *and denote* $d_0=d(y,x_0)$. *If we randomly sample a point* x_1 *from* $\mathcal{C}$, *we have* $\mathbb{E}[\min(d_0,d(y,x_1))]\le\lambda d_0$ *for* $\lambda\in(0,1)$.

A 4. *(Well-shaped Voronoi cells) There exists* $\eta>0$ *such that for any Voronoi cell generated by* x *with expected diameter* d_0 *contains a ball of radius* ηd_0 *centered at* x.

A 5. *(Local symmetry near optimum)* $\mathcal{X}_\star$ *consists of finite number of disjoint and connected components* $\{\mathcal{X}_\star^{(\ell)}\}_{\ell=1}^k$, $k<\infty$. *For each component, there exists an open ball* $B_{\nu_\ell}(x_\star^{(\ell)})$ *for some* $x_\star^{(\ell)}\in\mathcal{X}_\star^{(\ell)}$ *such that* $d(x,x_\star^{(\ell)})\le d(y,x_\star^{(\ell)})$ *implies* $f(x)\ge f(y)$ *for any* $x,y\in B_{\nu_\ell}(x_\star^{(\ell)})$.

We now describe the relationship between these assumptions and those used in the previous literature. A1 and A2 are assumptions also made in (Munos 2011). These make the weaker version of the Lipschitz assumption applied only to the global optima, instead of every pair of points in $\mathcal{X}$. A3 and A4 are also very similar to the assumptions made in (Munos 2011). In (Munos 2011), the author assumes that cells decrease in diameter as more points are evaluated inside of them and that each shell is well-shaped, in that it always contains a ball. Our assumption is similar, except that in our case, A3 and A4 are stated in terms of expectation, because VOO is a probabilistic algorithm.

A5 is an additional assumption that previous literature has not made. It assumes the existence of a ball inside of which, as you get closer to an optimum, the function values increase. It is possible to drastically relax this assumption to the existence of a sequence of open sets, instead of a ball, whose values increase as you get closer to an optimum. In our proof, we prove the regret of VOO in this general case, and Theorem 1 holds as the special case when A5 is assumed. We present this particular version for the purpose of brevity and comprehensibility, at the expense of generality.

Define $\nu_{min}=\min_{\ell\in[k]}\nu_\ell$. We have the following regret bound for VOO. All the proofs are in the appendix.

Theorem 1. *Let* n *be the total number of evaluations. If* $\frac{1-\lambda^{1/k}}{\bar{\mu}_B(\nu_{min})+1-\bar{\mu}_B(\eta\cdot\lambda\delta_{max})}<\omega$, *we have*

$$\begin{aligned}\mathcal{R}_n \le& L\delta_{max}C_1\left[\lambda^{1/k}+\omega(1-\bar{\mu}_B(\eta\cdot\lambda^n\delta_{max}))\right]^n\\ &+L\delta_{max}C_2[(1-\omega k\bar{\mu}_B(\nu_{min}))\cdot(1+\lambda^{1/k})]^n\end{aligned}$$

where C_1 *and* C_2 *are constants as follows*

$$C_1 := \frac{1}{1-\rho(\lambda^{1/k}+1-[1-\omega+\omega\bar{\mu}_B(\eta\cdot\lambda\delta_{max})])^{-1}},$$
$$\rho := 1-\omega\bar{\mu}_B(\nu_{min}),$$
$$\text{and } C_2 := \frac{\lambda^{-1/k}+1}{(\lambda^{-1/k}+1)-(1-\omega\bar{\mu}_B(\nu_{min})^{-1}}$$

Some remarks are in order. Define an *optimal cell* as the the cell that contains a global optimum. Intuitively speaking, when our best cell is an optimal cell, the regret should reduce quickly because when we sample from the best cell with probability $1-\omega$, we always sample from the optimal cell, and we can reduce our expected distance to an optimum by λ. And because of A5, the best cell is an optimal cell if we have a sample inside one of $B_{\nu_\ell}(x_\star)$.

Our regret bound verifies this intuition: the first term decreases quickly if λ is close to 0, meaning that if we sample from an optimal cell, then we can get close to the optimum very quickly. The second term says that, if $\bar{\mu}_B(\nu_{min})$, the minimum probability that the best cell is an optimal cell, is large, then the regret reduces quickly. We now have the following corollary showing that VOO is no-regret under certain conditions on λ and $\bar{\mu}_B(\nu_{min})$.

Corollary 1. *If* $\frac{\lambda^{1/k}}{(1+\lambda^{1/k})k\bar{\mu}_B(\nu_{min})}<\omega<1-\lambda^{1/k}$ *and* $\frac{\lambda^{1/k}}{1-\lambda^{2/k}}<k\bar{\mu}_B(\nu_{min})$, *then* $\lim_{n\to\infty}\mathcal{R}_n=0$.

The regret bound of VOOT makes use of the regret bound of VOO. We have the following theorem.

Theorem 2. *Define* $C_{max}=max\{C_1,C_2\}$. *Given a decreasing sequence* $\eta(h)$ *with respect to* h, $\eta(h)>0$, $h\in\{0\cdots H-1\}$ *and the range of* ω *as in Theorem 1, if* $N_{iter}=\prod_{h=0}^{H-1}N_r(h)$ *is used, where*

$$N_r(h)\ge\log\left(\frac{\eta(h)-\gamma\eta(h+1)}{2L\delta_{max}C_{max}}\right)\cdot\min(G_{\lambda,\omega},K_{\nu,\omega,\lambda})$$

$G_{\lambda,\omega} = (\log(\lambda^{1/k}+\omega))^{-1}$, *and* $K_{\nu,\omega,\lambda} = (\log([(1-\omega\bar{\mu}_B(\nu_{min}))(1+\lambda^{1/k})]))^{-1}$ *, then for any state* s *traversed in the search tree we have*

$$V_\star^{(h)}(s) - \hat{V}_{N_r(h)}^{(h)}(s) \leq \eta(h) \quad \forall h \in \{0, \cdots, H-1\}$$

This theorem states that if we wish to guarantee a regret of $\eta(h)$ at each height of the search tree, then we should use N_{iter} number of iterations, with $N_r(h)$ number of iterations at each node of height h.

To get an intuitive understanding of this, we can view the action optimization problem at each node as a BBFO problem that takes account of the regret of the next state. To see this more concretely, suppose that $H = 2$. First consider a leaf node, where the problem reduces to a BBFO problem because there is no next state, and the regret of the node is equivalent to the regret of VOO. We can verify that by substituting $N_r(H-1)$ to the bound in Theorem 1 the regret of $\eta(H-1)$ is guaranteed. Now suppose that we are at the root node at height $H-2$. There are two factors that contribute to the regret at this node: the regret at the next state in height $H-1$, and the regret that stems from sampling non-optimal actions in this node, which is the regret of VOO. Because all nodes at height $H-1$ have a regret of $\eta(H-1)$, to obtain the regret of $\eta(H-2)$, the regret of VOO at the node at height $H-2$ must be $\eta(H-2) - \gamma N_r(H-1)$. Again, by substituting $N_r(H-2)$ to the bound in Theorem 1, we can verify that that it would yield the regret of $\eta(H-2) - \gamma N_r(H-1)$ as desired.

Now, we have the following remark that relates the desired constant regret at each node and the total number of iterations.

Remark 1. *If we set* $\eta(h) = \eta, \quad \forall h \in \{0 \cdots H-1\}$*, and* $N_{iter} = (N_r)^H$ *where*

$$N_r = \log\left(\frac{\eta(1-\gamma)}{2L\delta_{max}C_{max}}\right) \cdot \min(G_{\lambda,\omega}, K_{\nu,\omega,\lambda})$$

then, for any state s *traversed in the search tree we have*

$$V_\star^{(h)}(s) - \hat{V}_{N_r(h)}^{(h)}(s) \leq \eta \quad \forall h \in \{0, \cdots, H-1\}$$

We draw a connection to the case of discrete action space with b number of actions. In this case, we can guarantee zero-regret at the root node if we explore all b^H number of possible paths from the root node to leaf nodes. In the continuous case, with assumptions A1-A5, it would require sampling infinite number of actions at a leaf node to guarantee zero-regret, rendering achieving zero-regret in problems with $H > 0$ impossible. So, this remark considers a positive expected regret of η. It show that to guarantee this, we need to explore at least $(N_r)^H$ paths from the root to leaf nodes, where N_r is determined by the regret-bound of our action-optimization algorithm VOO. Alternatively, if some other action-optimization algorithm such as DOO, SOO, or GP-UCB is used, then its regret bound can be readily used by computing the respective $N_r(h)$ values in Theorem 1, and its own N_r value in Remark 1. It is possible to prove a similar remark in an undiscounted case. Please see Remark 2 in our appendix.

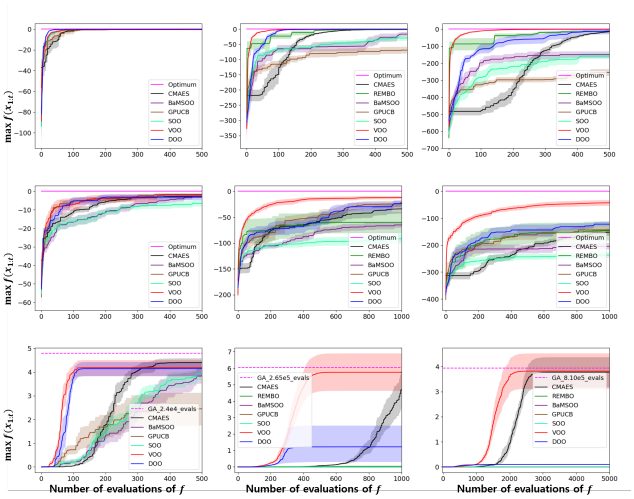

Figure 3: Griewank, Rastrigin, and Shekel functions (top to bottom) in 3, 10, and 20 dimensions (left to right)

Experiments

We designed a set of experiments with two goals: (1) test the performance of VOO on high-dimensional functions in comparison to other black-box function optimizers and (2) test the performance of VOOT on deterministic planning problems with high-dimensional action spaces in comparison to other continuous-space MCTS algorithms. All plots show mean and 95% confidence intervals (CIs) resulting from multiple executions with different random seeds.

Budgeted-black-box function optimization We evaluate VOO on three commonly studied objective functions from the DEAP (Fortin et al. 2012) library: Griewank, Rastrigin, and Shekel. They are highly non-linear, with many local optima, and can extend to high-dimensional spaces. The true optimum of the Shekel function is not known; to gauge the optimality of our solutions, we attempted to find the optimum for our instances by using a genetic algorithm (GA) (Qin and Suganthan 2005) with a very large budget of function evaluations.

We compare VOO to GP-UCB, DOO, SOO, CMA-ES, an evolutionary algorithm (Beyer and Schwefel 2002), REMBO, the BO algorithm for high-dimensional space that works by projecting the function into a lower-dimensional manifold (Wang et al. 2013), and BAMSOO, which combines BO and hierarchical partitioning (Wang et al. 2014). All algorithms evaluate the same initial point. We ran each of them with 20 different random seeds. We omit the comparison to HOO, which reduces to DOO on deterministic functions. We also omit testing REMBO in problems with 3-dimensional search spaces. Detailed descriptions of the implementations and extensive parameter choice studies are in the appendix.

Results are shown in Figure 3. In the 3-dimensional cases, most algorithms work fairly well with VOO and DOO performing similarly. But, as the number of dimensions increases, VOO is significantly better than all other methods.

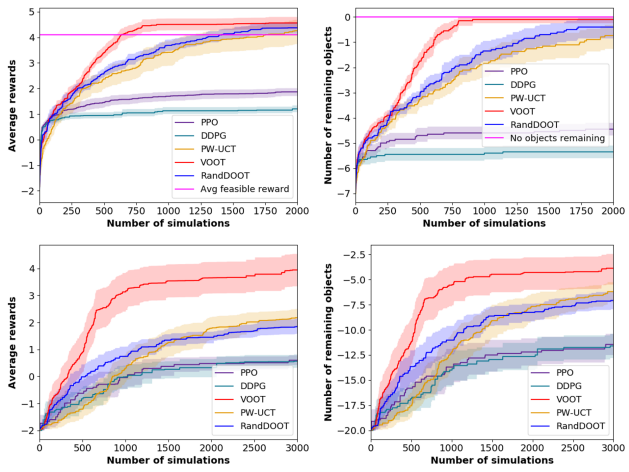

Figure 4: (Top-left) max sum of rewards vs. N_{iter} for the object clearing domain (Bottom-left) that for the packing domain. (Top-right) minus the number of remaining objects that need to be moved vs. N_{iter} in the object clearing domain (Bottom-right) that for the packing domain.

Purely hierarchical partitioning methods, DOO and SOO suffers because it is difficult to make the optimal partition, and SOOsuffers more than DOO because it does not take advantage of the semi-metric; the mixed approach of BO and hierarchical partitioning, BAMSOO, tends to do better than SOO, but still is inefficient in high dimensions for the same reason as SOO. GP-UCB suffers because in higher dimensions it becomes difficult to globally optimize the acquisition function. REMBO assumes that the objective function varies mostly in a lower-dimensional manifold, and there are negligible changes in the remaining dimensions, but these assumptions are not satisfied in our test functions, and VOO, which doesn't make this assumption, outperforms it. CMA-ES performs a large number of function evaluations to sustain its population, making it less suitable for *budgeted*-optimization problems where function evaluations are expensive.

This trend is more pronounced in the Shekel function, which is flat over most of its domain, but does increase near the optimum (see the 2D version in Figure 2). DOO, SOO, and BAMSOO perform poorly because they allocate samples to large flat regions. GP-UCB performs poorly because in addition to the difficulty of optimizing the acquisition function, the function is not well modeled by a GP with a typical kernel, and the same goes for REMBO. VOO has neither of these problems; as soon as VOO gets a sample that has a slightly better value, it can concentrate its sampling to that region, which drives it more quickly to the optimum. We do note that CMA-ES is the only method besides VOO to perform at all well in high-dimensions.

Sequential mobile manipulation planning problems We now study two realistic robotic planning problems. We compare VOOT to DOOT, which respectively use VOO and DOO and as its action-optimizer in Algorithm 2, and a single-progressive-widening algorithm that uses UCT (PW-UCT) (Couëtoux et al. 2011). But to make DOOT work in these problems, we consider a randomized variant called RAND-DOOT which samples an action uniformly in the cell to be evaluated next, instead of always selecting the mid-point, which could not solve any of these problems.

The objective of comparing to PW-UCT is to verify our claim that using an efficient action-optimizer, at the expense of uniform re-evaluations of the sampled actions, is better evaluating sampled actions with UCB at the expense of sampling new actions uniformly. The objective of comparing to RAND-DOOT is to verify our claim that VOOT can scale to higher dimensional problems for which RAND-DOOT does not.

In addition to the state-of-the-art continuous MCTS methods, we compare VOO to the representative policy search methods typically used for continuous-action space problems, PPO (Schulman et al. 2017) and DDPG (Lillicrap et al. 2016). We train the stochastic policy using the same amount of simulated experience that the tree-search algorithms use to find a solution, and report the performance of the best trajectory obtained.

The action-space dimensions are 6 and 9 in the object-clearing and packing domains, respectively. The detailed action-space and reward function definitions, and extensive hyper-parameter value studies are given in the appendix. The plots in this section are obtained with 20 and 50 random seeds for object-clearing and packing problems, respectively.

We first consider the object-clearing problem (s_0 is shown in Figure 1 (right)). Roughly, the reward function penalizes infeasible actions and actions that move an obstacle but do not clear it from the path; it rewards actions that clear an object, but with value inversely proportional to the length of the clearing motion. The challenging aspect of this problem is that, to the right of the kitchen area, there are two large rooms that are unreachable by the robot; object placements in those rooms will be infeasible. So, the robot must clear obstacles within the relatively tight space of the kitchen.

Figure 4 (Top-left) shows the results. In this case, PW-UCT samples from the whole space, concentrating far too many of them in the unreachable empty rooms. RAND-DOOT also spends time partitioning the big unreachable regions, due to its large exploration bonus; however it performs better than PW-UCT because once the cells it makes in the unreachable region get small enough, it starts concentrating in the kitchen region. However, it performs worse than VOOT for similar reasons as in the Shekel problems: as soon as VOOT finds the first placement inside the kitchen (i.e. first positive reward), it immediately focuses its sampling effort near this area with probability $1 - \omega$. This phenomenon is illustrated in Figure 5, which shows the values of placements. We can also observe from Figure 4 (Bottom-left) that VOOT clears obstacles much faster than the other methods; it clears almost all of them with 750 simulations, while others require more than 1700, which is about a factor of 2.3 speed-up.

The reinforcement learning algorithms, PPO and DDPG, perform poorly compared to the tree-search methods. We can see that within the first 250 simulations, their rewards

Figure 5: $\hat{Q}(s, a)$ of PW-UCT, RAND-DOOT, and VOOT (left to right) after 50 visits to the place node for the first object. Blue and purple bars indicate values of infeasible and feasible placements, respectively. Solid robot indicates the current state of the robot, and the transparent robots indicate the placements sampled. Notice VOOT has far fewer samples in infeasible regions.

grow just as quickly as for the search algorithms, but they seem to get stuck at local optima, clearing only one or two obstacles. This is because the problem has two challenging characteristics: large future delayed rewards and sparse rewards.

The problem has sparse rewards because most of the actions are unreachable placements, or kinematically infeasible picks. It has large delayed rewards because the reward function is inversely proportional to the length of the clearing motion, but the first few objects need to be moved far away from their initial locations to make the subsequent objects accessible. Unfortunately, the RL methods come with an ineffective exploration strategy for long-horizon planning problems: Gaussian random actions[1]. This strategy could not discover the delayed future rewards, and the policies fell into a local optima in which they try to clear the first two objects with the least possible cost, but blocking the way to the subsequent objects.

We now consider the conveyor belt problem (s_0 shown in Figure 1 (left)). The challenge is the significant interdependence among the actions at different time steps: the first two boxes are too big to go through the door that leads to the bigger rooms, so the robot must place them in the small first room, so that there is still room to move the rest of the objects into the bigger rooms. Figure 4 (row 5, left) shows the results. VOOT achieves the reward of a little more than 3 with 1000 simulations, while other methods achieve below 1 ; even with 3000 simulations, their rewards are below 2, whereas that of VOOT goes up to approximately 4. Figure 4 (row 5, right) shows that VOOT finds a way to place as many as 15 objects within 1000 simulations, whereas the alternative methods have only found plans for placing 12 or 13 objects after 3000 simulations. We view each action-optimization problem (line 10 of Alg. 2) as a BBFO problem, since we only have access to the values of the actions that have been simulated, and the number of simulations is limited to N_{iter}. The RL approaches suffer in this problem as well, packing at most 8 boxes, while the worst search-based method packs 13 boxes. Again, the reason is the same as in the previous domain: sparse and delayed long-term rewards.

[1]In order to get the RL methods to perform at all well, we had to tailor the exploration strategy to compensate for the fact that many of the action choices are completely infeasible. Details are in the appendix.

Future work and conclusions

We proposed a continuous MCTS algorithm in deterministic environments that scales to higher-dimensional spaces, which is based on a novel and efficient BBFO VOO. We proved a bound on the regret for VOO, and used it to derive a performance guarantee on VOOT. The tree performance guarantee is the first of its kind for search methods with BBFO-type algorithms at the nodes. We demonstrated that both VOO and VOOT significantly outperform previous methods within a small number of iterations in challenging higher-dimensional synthetic BBFO and practical robotics problems.

We believe there is a strong potential for combining learning and VOOT to tackle more challenging tasks in continuous domains, much like combining learning and Polynomial UCT has done in the game of Go (Silver et al. 2016). We can learn from previous planning experience a policy π_θ, which assigns high probabilities to promising actions, using a reinforcement-learning algorithm. We can then use VOO with π_θ, instead of uniform sampling.

Acknowledgement

We gratefully acknowledge support from NSF grants 1523767 and 1723381; from AFOSR grant FA9550-17-1-0165; from ONR grant N00014-18-1-2847; from Honda Research; and from the MIT-Sensetime Alliance on AI. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

Auger, D.; Couëtoux, A.; and Teytaud, O. 2013. Continuous Upper Confidence Trees with polynomial exploration - consistency. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

Beyer, H.-G., and Schwefel, H.-P. 2002. Evolution strategies a comprehensive introduction. *Natural Computing*.

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

Buşoniu; Daniels, A.; Munos, R.; and Babuška, R. 2011. Optimistic planning for continuous-action deterministic systems. *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*.

Couëtoux, A.; Hoock, J.-B.; Sokolovska, N.; Teytaud, O.; and Bonnard, N. 2011. Continuous upper confidence trees. *International Conference on Learning and Intelligent Optimization*.

Fortin, F.-A.; De Rainville, F.-M.; Gardner, M.-A.; Parizeau, M.; and Gagné, C. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*.

Kandasamy, K.; Schneider, J.; and Poczos, B. 2015. High dimensional bayesian optimisation and bandits via additive models. *International Conference on Machine Learning*.

Kawaguchi, K.; Kaelbling, L.; and Lozano-Pérez, T. 2015. Bayesian optimization with exponential convergence. In *Advances in Neural Information Processing Systems*.

Kuffner, J., and LaValle, S. 2000. RRT-connect: An efficient approach to single-query path planning. In *International Conference on Robotics and Automation*.

Lillicrap, T. P.; J. J. Hunt, A. P.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*.

Mansley; Weinstein, A.; and Littman, M. 2011. Sample-based planning for continuous action Markov Decision Processes. *International Conference on Automated Planning and Scheduling*.

Munos, R. 2011. Optimistic optimization of a deterministic function without the knowledge of its smoothness. *Advances in Neural Information Processing Systems*.

Munos, R. 2014. From bandits to Monte-Carlo Tree Search: the optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning*.

Pintér, J. 1996. *Global Optimization in Action (Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications)*. Springer US.

Qin, A., and Suganthan, P. 2005. Self-adaptive differential evolution algorithm for numerical optimization. *IEEE Congress on Evolutionary Computation*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv*.

Silver, D.; Huang, A.; Maddison, C.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*.

Srinivas, N.; Krause, A.; Kakade, S.; and Seeger, M. 2010. Gaussian Process optimization in the bandit setting: no regret and experimental design. *International Conference on Machine Learning*.

Wang, Z.; Zoghi, M.; Hutter, F.; Matheson, D.; and Freitas, N. 2013. Bayesian optimization in high dimensions via random embeddings ziyu. *International Conference on Artificial Intelligence and Statistics*.

Wang, Z.; Shakibi, B.; Jin, L.; and Freitas, N. 2014. Bayesian multi-scale optimistic optimization. *International Conference on Artificial Intelligence and Statistics*.

Weinstein, A., and Littman, M. 2012. Bandit-based planning and learning in continuous-action markov decision proceses. *International Conference on Automated Planning and Scheduling*.