Sample-Based Planning for Continuous Action Markov Decision Processes

Chris Mansley and **Ari Weinstein** and **Michael L. Littman**
Rutgers University
110 Frelinghuysen Road
Piscataway, NJ 08854 USA *

Abstract

In this paper, we present a new algorithm that integrates recent advances in solving continuous bandit problems with sample-based rollout methods for planning in Markov Decision Processes (MDPs). Our algorithm, Hierarchical Optimistic Optimization applied to Trees (HOOT) addresses planning in continuous-action MDPs. Empirical results are given that show that the performance of our algorithm meets or exceeds that of a similar discrete action planner by eliminating the problem of manual discretization of the action space.

Introduction

At present, most MDP planning algorithms are restricted to settings where the action space is finite. This limitation is driven by two issues: representing the continuous value function over the action space and performing the maximization over this function. A natural approach is to coarsely discretize the space and then plan in that modified space. The risk of this approach is that the chosen action discretization may not be appropriate for the domain and therefore impact the quality of the solution found. In this paper, we propose a planning algorithm that uses existing results from the bandit literature to perform a stochastic optimization over the action space via sampling combined with sample-based planning ideas. The resulting algorithm adaptively partitions the action space thus enabling it to avoid the pitfalls encountered in algorithms that use a fixed action discretization. The algorithm focuses quickly on regions of the action space with high value, while identifying regions of the action space that can be safety ignored.

Background

We describe some of the formal background on which our algorithm depends.

Markov Decision Processes

An MDP M is described by a five-tuple $\langle S, A, T, R, \gamma \rangle$, where $S \subseteq \mathbb{R}^N$ is the N-dimensional state space, $A \subseteq \mathbb{R}^D$ is the D-dimensional action space, T is the transition function, with $T(s, a, s')$ denoting the probability of reaching s' from s by taking action a, R is the reward function with $R(s, a)$ denoting the expected reward from taking action a in state s and $\gamma \in [0, 1)$ is the discount factor. A deterministic policy π is a mapping $\pi : S \to A$ from states to actions. The value of a policy, $V^\pi(s)$, is defined as the expected sum of discounted rewards starting at state s and following policy π. The discounted return from time $t = 0$ to $t = T$ for a horizon T is $\sum_{t=0}^{T} \gamma^t r_t$, where r_t is the reward obtained at step t. The optimal policy, π^*, is the policy that maximizes $V^\pi(s), \forall s \in S$.

Sample-Based Planners

Traditional methods for planning in a given MDP scale polynomially in the number of states and actions, which can be impractical in large or infinite state spaces. Sparse sampling (Kearns, Mansour, and Ng 1999) is a sample-based planner that produces provably near-optimal policies independent of the size of the state space but has costs exponential in the planning depth. In practice, the cost of building the full planning tree limits the depth to which Sparse Sampling can plan.

Rollout planners, another class of sample-based planner, do not suffer from computational costs as a function of the planning depth as acutely as sparse sampling. This property is important when planning must be carried out to a greater planning depth with restricted computational resources. In these methods, planning is done by conducting an entire rollout from the query state with length equal to the planning depth. The information gained from this trajectory can be used to refine the policy and then the process is repeated.

UCT

The Upper Confidence Bounds applied to Trees (UCT) algorithm was proposed as an instance of these rollout planners, also called Monte-Carlo planning (Kocsis and Szepesvári 2006). UCT uses a modified version of the UCB1 (Auer, Fischer, and Cesa-Bianchi 2002) bandit algorithm for action-selection in the search tree. The policy for the bandit component in UCT is defined as taking the action

$$\operatorname*{argmax}_{a \in A} \hat{R}(a) + 2C_p \sqrt{\ln(n)/n_a},$$

where $\hat{R}(a)$ is the sample mean of the reward observed from taking a, C_p is some constant greater than 0, and n_a is the

*The authors thank Ali Nouri for his fundamental insight into the relationship between HOO and UCT. This research was supported in part by National Science Foundation DGE-0549115.

number of times a has been taken.

One of these bandit agents is placed at each state and depth of the rollout. When a trajectory encounters a particular state and depth, each action is treated as a bandit arm, and the discounted return resulting from that action is treated as the reward by the bandit algorithm. Since the bandit component computes the mean discounted return and number of tries for each action, UCT is restricted to the discrete action setting.

Planning with Continuous Actions

Building on UCT, which takes actions during rollouts according to a discrete action bandit algorithm, a natural extension to this approach is the application of a continuous action bandit algorithm to rollout planning. We discuss an existing bandit algorithm and present our approach to integrating it into a rollout planner.

HOO

The Hierarchical Optimistic Optimization (Bubeck et al. 2009) or HOO strategy is a bandit algorithm that exploits a set of actions that forms a general topological space. This assumption allows HOO to be applied directly to many domains that traditional bandit algorithms such as UCB cannot, including those with a continuum of actions.

HOO operates by developing a piecewise decomposition of the action space, which is represented as a tree. When queried for an action to take, the algorithm starts at the root and continues to the leaves by taking a path according to the maximal score between the two children, called the B-value. At a leaf node, an action is sampled from any part of the range that the node represents. The node is then subdivided by adding two children. The process is repeated each time HOO is queried for an action selection.

The values computed for each node are a count, reward estimation, and reward bias, which are all combined to form the B-value. Let the pair (h, i) refer to the ith node at depth h, $C(h, i)$ be the set consisting of the node and its subtree, and let

$$N_{h,i}(n) = \sum_{t=1}^{n} \mathbb{I}_{\{(H_t, I_t) \in C(h,i)\}}$$

be the number of times node (h, i) has been visited up to time t, where (H_t, I_t) is the node which was split at t.

Let $\hat{R}_{h,i}(n)$ be the reward estimate of node (h, i) defined as

$$\hat{R}_{h,i}(n) = \frac{1}{N_{h,i}(n)} \sum_{t=1}^{n} \mathbb{I}_{\{(H_t, I_t) \in C(h,i)\}} r_t.$$

The upper bound on the estimate of the reward is

$$U_{h,i}(n) = \hat{R}_{h,i}(n) + \sqrt{\frac{2 \ln n}{N_{h,i}(n)}} + v_1 \rho^h$$

for $v_1 > 0$ and $0 < \rho < 1$. For nodes that have not yet been sampled, $\hat{R}_{h,i(n)} = U_{h,i}(n) = \infty$.

The B-value of a node is defined as

$$B_{h,i}(n) = \min \{U_{h,i}(n), \max \{B_{h+1,2i-1}(n), B_{h+1,2i}(n)\}\}$$

where for node (h, i), nodes $(h + 1, 2i - 1)$ and $(h + 1, 2i)$ are its children.

HOOT

We introduce Hierarchical Optimistic Optimization applied to Trees (HOOT), which integrates the HOO algorithm into the rollout planning structure. The use of HOO for action selection allows the algorithm to overcome the discrete action limitation of UCT.

Our algorithm can be described in relation to UCT as follows. The action selection mechanism of UCT operates by placing a UCB agent at each state and depth encountered during the rollout process. Instead of a discrete action bandit algorithm, HOOT places a continuous action bandit algorithm, HOO, at each state and depth in the rollout tree. Aside from this modification, all other aspects of the algorithm are the same.

The computation cost of HOO is greater than UCB, which is reflected in the running times of HOOT and UCT. Given the n samples at each node in the MCTS tree, UCB requires linear time to find the maximum, while HOOT must rebuild the HOO tree each time a sample is added to the tree, resulting in a quadratic complexity with respect to the number of samples. There exists a variant of HOO that makes the update $O(n \lg n)$ (Bubeck et al. 2010).

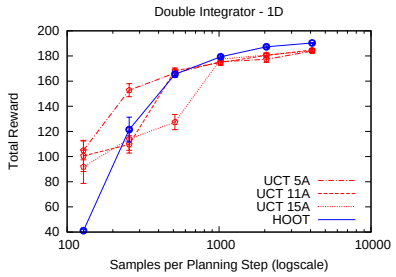
Experiments

We compared UCT with discretized actions to HOOT in a few continuous action MDPs. Since both HOOT and UCT require discrete states, all domains have 20 divisions per state dimension. The planner makes a decision based on the queries generated on that planning step only. Algorithms labeled UCT N, where N is 5, 11, 15, 20, represent the number of discretizations per action dimension used in UCT. After planning, the action taken by UCT was that with the highest mean return at the root. In HOOT, the action was taken by greedily following branches according to mean reward as opposed to B-value.

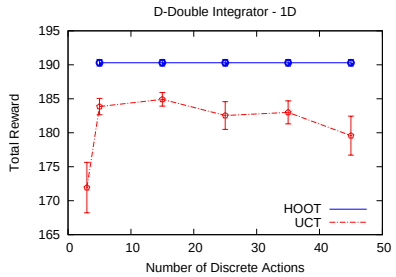
Double Integrator

The double integrator (Santamaría, Sutton, and Ram 1998) domain models the motion of an object along a surface. The dynamics of the system can be represented as a discrete time linear system with quadratic rewards. This system can be generalized to D dimensional systems to create D position dimensions and D velocity dimensions. We call this extension to the domain the D-double integrator.

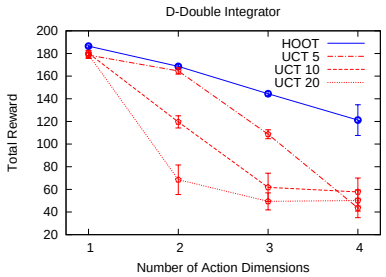
In Figure 1(a), the performance of HOOT and UCT is compared using different numbers of queries to the generative model during the planning step. Given enough samples, our algorithm performs better than simple discrete actions. This domain penalizes high magnitude actions, which forces HOOT to select lower magnitude actions. The curve also demonstrates our investment of samples in discovering the right discretization. At 256 samples, HOOT does poorly, similar to UCT11 or UCT15, while UCT5 does much better. By 2048 samples, HOOT can now begin to refine its estimate of the best actions to take, outperforming all discrete UCT.



(a) Variable number of samples per planning step.



(b) Variable number of discrete actions. Number of samples fixed to 2048.



(c) Increasing number of action dimensions. Number of samples fixed to 2048.

Figure 1: Performance of HOOT and UCT in the double integrator domain.

In Figure 1(b), the performance of HOOT and UCT is plotted for $D = 1$ as the number of discretized actions used by UCT varies. Since HOOT does not require actions to be discretized in advance, its performance is constant with respect to this variable. All uniform action discretizations used by UCT resulted in performance worse than HOOT. Note the characteristic "inverted-U" shape as the number of actions varies. The peak represents a location showing the trade-off between fine enough action discretization to perform the task and requiring increasingly more exploration due to the increase in the number of actions.

Figure 1(c) shows the performance of HOOT and UCT 5, 10 and 20 in the D-double integrator. As D increased, the performance of UCT degraded more rapidly than that of HOOT. This outcome can be attributed to the fact that the size of the discretized action set increases exponentially with D. For example, in 4-Double Integrator, UCT 20 must search over a space of 160000 discrete actions each step, which is greater than the number of samples allowed from the generative model. Since the HOO component prunes regions of the tree that appear to yield poor returns, HOOT can quickly focus exploration in regions of the action space where the optimal action is likely to lie and suffers less from the expansion of the dimension of the action space.

Inverted Pendulum

The inverted pendulum domain models the physics of a pendulum balancing on a pivot. The dynamics and reward function are defined by Pazis and Lagoudakis (2009). Noise in the actions in form of ± 20 Newtons uniformly distributed introduces stochasticity. The reward function penalizes policies that allow the pendulum to deviate from vertical, obtain high angular velocity and use large actions (large forces).

Figure 2(a) shows the performance of HOOT as compared to UCT as the number of discretized actions used by UCT varies. In this example, the performance of UCT is significantly worse than HOOT for all discretization levels, demonstrating the benefit of a variable resolution discretization scheme. Both algorithms are capable of preventing the pendulum from falling, but HOOT maintains more precise control using finer grained actions. The difference in value between a policy that barely keeps the pole balanced using large actions and a policy that keeps the pole perfectly balance with small, precise actions will be very small due to the rescaling of the original reward function.

Bicycle

The bicycle domain is a popular domain for testing higher dimensional states spaces and simulates the task of balancing a bicycle (Randløv and Alstrøm 1998; Li, Littman, and Mansley 2009). In this paper, we used a simplified version of the original domain that only focuses on the task of balancing the bicycle without steering toward a goal. There are five continuous state variables and two continuous action dimensions. The reward function provides a +1 each step for each step of balancing and the episode terminates if the bicycle falls. The run is limited to 2000 steps. The domain has noise along one of the action dimensions.

Figure 2(b) shows the performance of the bicycle domain as the number of actions varies. If too fine an action discretization is chosen, UCT has a difficult time performing the task. In the original paper, the number of actions was chosen to be 5, so these results are indicative of the small number of actions necessary to perform well in this domain.

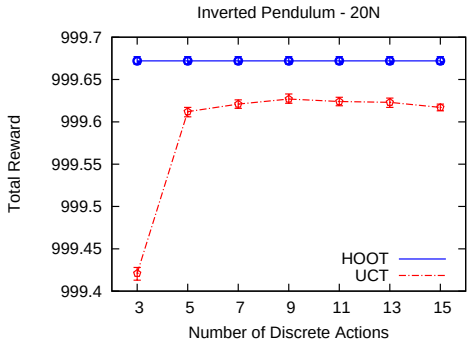
Future Work

It is possible to construct a version of HOO for use in the associative bandit setting (Kaelbling 1994), where the input (state) space is continuous. Since HOO utilizes only mean and count values, Weighted Hierarchical Optimistic Optimization (WHOO) introduces sample weights that allow for weighted means and counts. In addition to values recorded by HOO, the extension requires samples to record the state value s_t, on which a distance metric $K(s_1, s_2), s_1 s_2 \in S$ operates. This gives a similarity between different bandits that are close in input space.

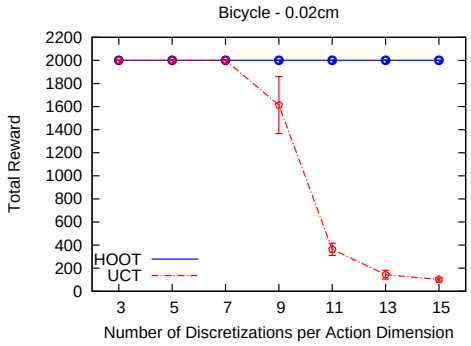
Using this distance metric, we can define equations for $N, \hat{R}, U$, and B for the associative bandit setting. Let

$$N_{h,i}(n,s) = \sum_{t=1}^{n} \mathbb{I}_{\{(H_t,I_t)\in C(h,i)\}} K(s,s_t)$$

$$\hat{R}_{h,i}(n,s) = \frac{1}{N_{h,i}(n,s)} \sum_{t=1}^{n} \mathbb{I}_{\{(H_t,I_t)\in C(h,i)\}} r_t K(s,s_t)$$



(a) Inverted Pendulum



(b) Bicycle

Figure 2: Performance of HOOT and UCT in two domains as the number of actions varies. The number of queries was fixed at 2048.

$$U_{h,i}(n,s) \quad = \quad \hat{R}_{h,i}(n,s) + \sqrt{\frac{2\ln N_{0,1}(n,s)}{N_{h,i}(n,s)}} + v_1\rho^h$$

with $B_{h,i}(n,s)$ being defined similar to HOO.

This associative bandit algorithm can be integrated into a sample-based planning structure that creates a planner that functions in both continuous action and state spaces, which results in the Weighted Hierarchical Optimistic Optimization applied to Trees (WHOOT) algorithm. It is constructed by placing one WHOO algorithm at each depth. This is one possible method for extending the methods presented in this paper into continuous state spaces.

The utilization of the HOO algorithm as a stochastic optimization algorithm is not limited to the sample-based planning framework outlined in this paper. This work should be extended into traditional temporal-differencing (TD) methods as a mechanism for computing the maximum over the value function. This algorithm could be worked into the Dyna framework.

Conclusion

This paper introduces HOOT, an algorithm for planning in continuous-action MDPs. The main benefit of using the proposed algorithm is its ability to adaptively partition the action space and focus exploration on regions in the action space where the highest returns are likely to lie. This removes one parameter (the discretization of the action space) that must be tuned in planning algorithms that assume a discrete state space. In comparison, the choice of action discretization used by UCT must balance the number of actions used. In many domains, too few actions makes a good policy inexpressible, while too many actions makes the space too large to search effectively, especially because there is no generalization over actions in UCT. HOOT, on the other hand, does generalize over actions.

Finally, it seems the greatest benefits of using HOOT instead of an approach that requires discretization stems from its behavior in high dimensional action spaces. Although the regret of UCB is $O(\log t)$, the cost is also linear in the number of discrete actions k. This cost is generally not considered significant because it is assumed t is much greater than k. Since UCT builds on UCB, this issue arises in the planning setting as well. In domains with high-dimensional action spaces, this dependency on the exponentially growing number of actions becomes a significant factor. HOO, on the other hand, does not discretize actions and so has no k. It also has regret independent of the number of action dimensions D under certain assumptions (Bubeck et al. 2009). This property mitigates the combinatorial explosion that arises in planning in domains with high dimensional action spaces.

References

Auer, P.; Fischer, P.; and Cesa-Bianchi, N. 2002. Finite-time analysis of the multi-armed bandit problem. *Machine Learning* 47.

Bubeck, S.; Munos, R.; Stoltz, G.; and Szepesvári, C. 2009. Online optimization in X-armed bandits. In *Advances in Neural Information Processing Systems 23*.

Bubeck, S.; Munos, R.; Stoltz, G.; and Szepesvári, C. 2010. X-armed bandits. *CoRR* abs/1001.4475.

Kaelbling, L. 1994. Associative reinforcement learning: Functions in k-dnf. *Machine Learning* 15(3).

Kearns, M.; Mansour, S.; and Ng, A. 1999. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *IJCAI*.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*.

Li, L.; Littman, M. L.; and Mansley, C. R. 2009. Online exploration in least-squares policy iteration. In *International Conference on Autonomous Agents and Multiagent Systems*.

Pazis, J., and Lagoudakis, M. 2009. Binary action search for learning continuous-action control policies. In *International Conference on Machine Learning*.

Randløv, J., and Alstrøm, P. 1998. Learning to drive a bicycle using reinforcement learning and shaping. In *International Conference on Machine Learning*.

Santamaría, J. C.; Sutton, R.; and Ram, A. 1998. Experiments with reinforcement learning in problems with continuous state and action spaces. In *Adaptive Behavior 6*.