Sample-based Planning for Continuous Action Markov Decision Processes

Ari Weinstein AWEINST@RUTGERS.EDU
Chris Mansley CMANSLEY@CS.RUTGERS.EDU
Michael Littman MLITTMAN@CS.RUTGERS.EDU
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, directing the exploration of the search tree using insights from recent bandit research. 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.

1. Introduction

At present, most MDP planning algorithms are restricted to settings where the action space is finite. When used in domains where actions are continuous, 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 adaptively partitions the action space and enables it to avoid the pitfalls encountered in algorithms that use a fixed action discretization. The result is an algorithm that focuses quickly on regions of the action space with high value.

Submitted to the ICML 2010 Workshop on Reinforcement Learning and Search in Very Large Spaces.

2. Background

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

2.1. 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 $V^\pi(s), \forall s \in S$.

2.2. Sample-Based Planners

Sparse sampling (Kearns et al., 1999) is a sample-based planner that produces provably near-optimal policies but has costs exponential in the planning depth. To plan, a tree is built with a branching factor at least $|A|$ and with height equal to the planning depth. In practice, due to this cost, the depth to which Sparse Sampling can plan is quite shallow..

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. The simplest example of this style of algorithm is Monte-Carlo planning (Kocsis & Szepesvári, 2006).

2.3. UCT

The Upper Confidence Bounds applied to Trees (UCT) algorithm was proposed as an instance of Monte-Carlo planning (Kocsis & Szepesvári, 2006). UCT uses a modified version of the UCB1 (Auer et al., 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 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 plans over a discrete number of actions, UCT is restricted to the discrete action setting.

3. Sample-Based Planning in Continuous Action MDPs

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.

3.1. 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 described 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.

3.2. 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.

4. 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. To keep the implementation as direct and simple as possible, at each step all previous planning was discarded and started anew. UCT 5, UCT 10, and UCT 20 refer to UCT where there are 5, 10, and 20 discretizations across each action dimension, respectively. 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.

4.1. Double Integrator

The double integrator (Santamaría et al., 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 as follows:

$$\mathbf{x_{t+1}} = A\mathbf{x_t} + B\mathbf{u}$$

where $A = \begin{bmatrix} 1 & 0 \\ \Delta t & 1 \end{bmatrix}$, $B = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix}$.

The reward signal is quadratic and is defined as

$$-\frac{1}{D}\left(\mathbf{x_t}^T Q \mathbf{x_t} + \mathbf{u}^T R \mathbf{u}\right)$$

where $Q = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$, $R = [1]$.

This system can be generalized to D dimensional systems by extending the A, B, Q and R matrices to create D position dimensions and D velocity dimensions. We call this extension to the domain the D-double integrator.

In both experiments in this domain, 2048 queries were allowed to the generative model per step. In Figure 1, 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.

Figure 2 shows the performance of HOOT and UCT 5, 10 and 20 in the D-double integrator. As the number of action dimensions 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 far greater than the number of samples allowed from the generative model. Since the HOO component prunes regions of that 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.

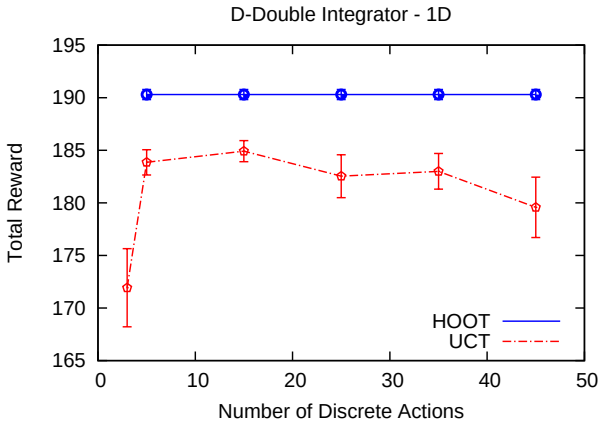

Figure 1. Performance of HOOT and UCT in the double integrator domain as the number of discretized actions used by UCT varies.

4.2. D-Link Swimmer

In the D-link swimmer domain (Tassa et al., 2007), a simulated swimmer is made up of a chain of D links where $D-1$ joint torques are applied between links in order to propel the swimmer to a goal point. The total size of the state space is $2D+4$-dimensional. As in the D-double integrator, we tested UCT 5, 10, and 20. In this domain, 8192 queries to the generative model were allowed per step. For all values of D tested, UCT 10 and 20 were not statistically different from that of a random policy, so of these only the performance of the random policy is plotted for reference.

Figure 3 shows the performance of HOOT, UCT 5, and the random policy. As in the D-double integrator, the performance of UCT 5 is competitve with HOOT while the number of action dimensions is small. However, once the number of action dimensions grows to four or larger, the performance of UCT 5 is not different from that of a random policy with statistical significance. HOOT, on the other hand continues to perform well with even the largest number of action dimensions tested.

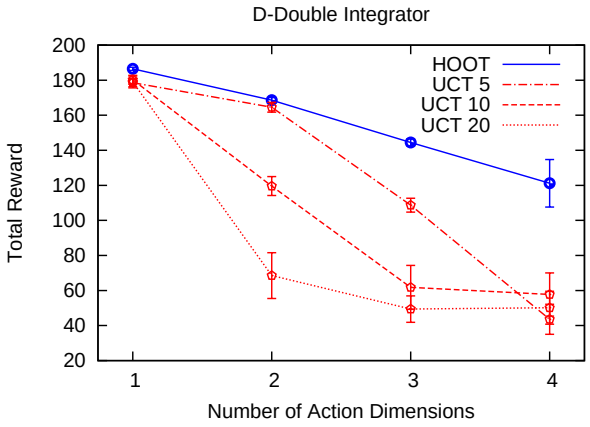

Figure 2. Performance of HOOT and UCT in the D-double integrator domain as the number of action dimensions increases.

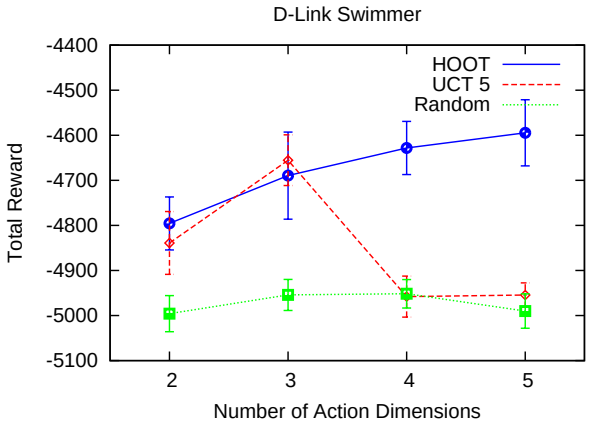

Figure 3. Performance of HOOT and UCT in the D-link swimmer domain as the number of action dimensions increases.

5. Conclusion and Discussion

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.

Acknowledgments

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.

References

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

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

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

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

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

Tassa, Yuval, Erez, Tom, and Smart, William D. Receding horizon differential dynamic programming. In *Advances in Neural Information Processing Systems 21*. 2007.