Continuous Domain Reinforcement Learning using Rapidly-exploring Random Trees

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

The problem of efficient exploration for reinforcement learning is difficult, more so in domains that have continuous states and continuous actions. We present a novel algorithm RRT-VI that uses approximate value iteration on samples generated using Rapidly-exploring Random Trees to solve such problems. The optimal path planning algorithm RRT* can be thought of as a specific case of our algorithm. We also provide some preliminary experimental results to show the same.

1 Introduction

Reinforcement Learning(RL) deals with learning optimal control strategies through interactions with the world. One of the major hindrances in extending standard RL techniques to real world scenarios, is their inability to deal with continuous valued variables. Straightforward discretization of the space leads to several problems. The discretization might be too small or too large leading to large amounts of learning time needed or incorrect generalization correspondingly. Methods that use function approximators are also popular but they need to be fine tuned according to the domain. Least Squares Policy Iteration [1] is a popular algorithm that uses linear function approximation and offline data from several trajectories in the space to learn the solution. However this offline data needs to be sufficiently representative to arrive at the correct solution.
This leads us to another fundamental problem in reinforcement learning which is the problem of balancing exploration and exploitation. This becomes an even more daunting task in continuous domains as the amount of space to be explored can be huge. An algorithm that attempts to optimally balance explore-exploit is the UCT algorithm[2]. This however does not work in continuous spaces. In an attempt to make an online version of LSPI it has been combined with R-MAX, a technique that does efficient exploration in discrete domains[3] . However this has several implementation issues such as tuning exploration parameters and generalizing state visitation counters. Often in such domains a model or simulator is available and several model based algorithms exist. One such approach is PEGASUS [4] which makes a large problem more feasible by using a model of the world. We tackle the problem of exploration in continuous space by making use of Rapidly-exploring Random Trees(RRTs), a widely popular sample-based planning techniques in the robotics[5]. RRTs have several attractive properties that we exploit - they are parameter free and have provable exploration properties. We present an model based algorithm that solves continuous domain RL problems through an adaptive discretization of the space based on RRTs. This work is similar in spirit to the parti-game algorithm for adaptive discretization[6]. However the splitting criterion for our algorithm is exploration centric. Our algorithm is motivated by the RRT*[7] algorithm, which is an *optimal* version of the RRTs. The algorithm which we present is an generalized version of the RRT* algorithm as applied to RL problems and tries to achieve optimal corresponding optimal solutions. Section 2 will introduce some preliminaries as well as explain the RRT and RRT* algorithms. Our algorithm RRT-VI will be described in Section 3, followed by discussions and experimental results.

2 Preliminaries

A Markovian Decision Process(MDP) is described as $\langle S, A, P, R \rangle$, where $S \in \mathbb{R}^n$ is the domain of states and $A \in \mathbb{R}^m$ is the domain of actions. P is the state transition probability. R is a real valued reward function. The aim of the *RL agent* is to maximize the discounted cumulative reward obtained. A deterministic policy π is a mapping from the state space S to the action space A. $V^\pi(s)$ is the state value function and is equal to the expected long term rewards, by following π at s. Similarly, $Q^\pi(s,a)$ is the state-action value function. The value function might be expressed as,

$$\forall s \in S, \;\; V^\pi(s) = R(s) + \gamma \sum_{s'} P^{ss'}_{\pi(s)} V^\pi(s') \tag{1}$$

where s' is the state transitioned to and $\gamma \in (0,1)$ is the discount factor. A policy π^* is optimal if $\forall s, \pi \;\; V^{\pi^*}(s) \geq V^\pi(s)$ The optimal value function can be calculated by replacing the expectation in Equation 1 with $\max$. A similar set of equations exist for the Q function.

2.1 Rapidly-exploring Random Trees

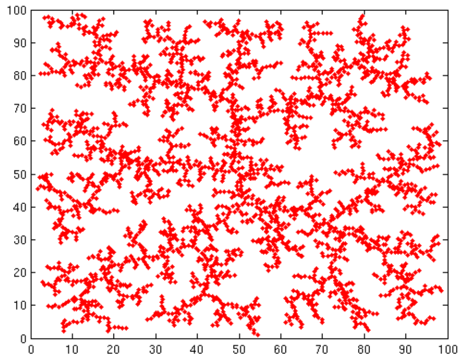

Figure 1: RRT with 300 nodes grown in $(0,100)^2$ starting from (50,50)

The basic RRT construction is outlined in algorithm 1. The algorithm is straightforward and elegant. Figure 2.1 shows a RRT grown in a 2 dimensional space. The algorithm builds a Tree T of samples from some space S whose nodes and edges are represented by $V(T)$ and $E(T)$. The tree is rooted from a given state s_{start} and is grown as described. Extend the tree towards the closest point implies adding an new node in the direction of the sample point and connecting it with the nearest existing point on the tree. RRTs have been shown to be *asymptotically complete*[8].

$$\lim_{|V(T)| \to \infty} Pr(s \in V(T)) \to 1 \;, \forall s \in S$$

RRTs can be thought of as monte-carlo versions space-filling trees. Given a tree T the space S can be split into $|V(T)|$ regions each corresponding the Voronoi region of the nodes. When a point is sampled uniformly in the space, it can be seen that the node to be expanded in the third step of the RRT algorithm is chosen with a probability proportional to the volume of its Voronoi region. This implicitly directs exploration towards unexplored parts of the state space.

Algorithm 1 ConstructRRT

```
$V(T) = s_{start}$, $E(T) = \emptyset$
repeat
    Uniformly sample a state $s$ from space $S$
    Calculate closest point in tree to $s$
    Extend the tree from the closest point towards $s$
until termination
```

2.2 RRT*

RRT* is a variation of the standard RRT planner. It produces *asymptotically optimal* solutions as well as giving *asymptotically complete* guarantees of RRT[7]. The algorithm is similar to the original RRT algorithm with an added step - it 'rewires' the tree within some region as shown in steps 5-7 in Algorithm 2. Thus reducing the path length from the root of the tree to the new point added to the tree. The neighborhood in which the rewiring is done is a hypersphere of radius = $O((log(|V(T)|)/|V(T)|)^{1/d})$ where d is the dimensionality of the space S. Thus as the tree grows, the size of sphere falls such that the number of possible connections at every step is in $O(log(|V(T)|))$. Hence the additional computational overhead is not significantly more than the original RRT algorithm. The actual radius r in step 6 of the Algorithm 2 is

Algorithm 2 RRT*

```
1: $V(T) = s_{start}$, $E(T) = \emptyset$
2: repeat
3:     $s_{new} \leftarrow$ Uniform random sample
4:     $s_{nearest} \leftarrow$ Choose nearest point in tree from $s_{new}$
5:     $s_{center}$ Choose point to extend $s_{nearest}$ towards $s_{new}$
6:     $S_{near} \leftarrow$ Select all points in the tree within some radius $r$ from $s_{center}$
7:     Connect the closest point in the set $S_{near}$ to $s_{center}$
8: until termination
```

$\gamma(log(|V(T)|)/|V(T)|)^{1/d}$ where $\gamma > 2(1 + 1/d)^{1/d}(\mu(S)/\zeta_d)^{1/d}$. $\mu(S)$ is the Lebesgue measure of the space S and ζ_d is the volume of the d-dimensional unit ball. The RRT* algorithm produces asymptotically optimal paths to the goal region from some start point. In a similar manner, the our algorithm samples from the MDP and attempts to optimize trajectories w.r.t expected long term rewards i.e. the value function.

3 RRT-VI

Given an MDP with state space S we describe our algorithm for determining an optimal policy. There is some associated distance function $\mathcal{D}$ defined on the state space. In most domains, this corresponds to the Euclidean distance. We first describe an algorithm RRT-PE that evaluates a given policy by using RRT based sampling. We make the following assumptions.

1. A generative model $\mathcal{M}$ that allows us to sample the reward function given a state transition.
2. An *approximate local controller* that can move from a given state to any state in ϵ-neighborhood.

The local controller provides a *Steer(x,y)* function, where $x, y \in S$. The function returns z such that z is closer to y than x, $\mathcal{D}(x - y) > \mathcal{D}(y - z)$. Also for some $\delta, \epsilon > 0$, $\delta \leq \mathcal{D}(x - z) \leq \epsilon$

In our work we choose working with V value function over the Q function. In RL problems with continuous states and actions, the Q value for different actions at the same state will not differ as much as the Q value at different states given that the state-action space is smooth. As pointed out by Baird , as the time step between actions decreases, the policy implied by the Q values becomes more sensitive to bias in the function approximation[9].

RRT-PE is describe in Algorithm 4. The value function is denoted using $J(s)$ to avoid confusion with the vertex set $V(T)$. Each node in the tree represents a point in the state. Every edge corresponds to actions or transitions. Associated with every edge is a reward. Once a new node is added, the reward is sampled and the values of all the nodes along the path till the root of the tree are updated. The value function of a node can be calculated using the simple back described in Equation 1. Figure 2 shows one step of this algorithm. The value of a new node is taken to be zero. When a node has more than one children, the value is taken as the average of the two. This corresponds to an uniform policy. The policy being evaluated in this manner corresponds to the space filling exploratory policy of the RRT.

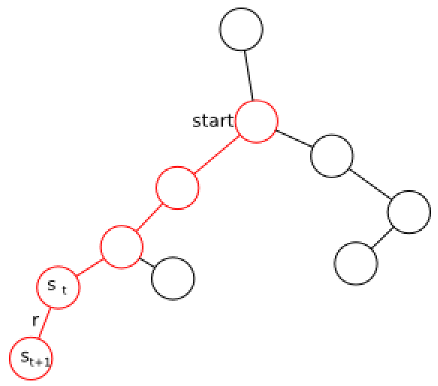

Figure 2: Representation of RRT based sample tree. Node s_{t+1} is newly added node to the tree. All values along the path to the root *start* have to be updated using the simple formula in Equation 1

Algorithm 3 RRT-PolicyEval

```
1:  $V(T) \leftarrow s_{start}$, $E(T) \leftarrow \emptyset$
2:  $J(s_{start}) \leftarrow 0$
3:  **repeat**
4:      $s_{new} \leftarrow$ Uniform random sample
5:      $s_{nearest} \leftarrow$ Choose nearest point in tree from $s_{new}$
6:      $s_{extn} \leftarrow Steer(s_{nearest}, s_{new})$
7:      $V(T) \leftarrow V(T) \cup s_{extn}$, $E(T) \leftarrow E(T) \cup (s_{nearest}, s_{extn})$
8:      $r \leftarrow \mathcal{G}(s_{nearest}, s_{extn})$
9:      $J(s_{start}) \leftarrow 0$
10:     $S_{trace} \leftarrow$ Nodes along the path from $s_{extn}$ to the root $s_{start}$
11:     Update values of all nodes in $S_{trace}$
12: **until** termination
```

We can now describe our main algorithm by extending RRT-PE in a way similar to the RRT* algorithm. An extra step corresponding to policy improvement is done on every iteration. The algorithm is described below. We will call this algorithm RRT-Value Iteration, as it resembles the standard Value Iteration procedure in RL. Just as in RRT* we consider all points around a newly added node within a radius q, and rewire the tree such that the value of the node is maximized.

Algorithm 4 RRT-VI

```
1:  $V(T) \leftarrow s_{start}$, $E(T) \leftarrow \emptyset$
2:  $J(s_{start}) \leftarrow 0$
3:  **repeat**
4:      $s_{new} \leftarrow$ Uniform random sample
5:      $s_{nearest} \leftarrow$ Choose nearest point in tree from $s_{new}$
6:      $s_{extn} \leftarrow Steer(s_{nearest}, s_{new})$
7:      $V(T) \leftarrow V(T) \cup s_{extn}$
8:      $S_{near} \leftarrow$ Select all points in the tree within some radius $q$ from $s_{extn}$
9:      $s_{best} \leftarrow$ Select the node such that $J(s_{extn})$ is maximized.
10:     $E(T) \leftarrow E(T) \cup (s_{extn}, s_{best})$
11:     $S_{trace} \leftarrow$ Nodes along the path from $s_{extn}$ to the root $s_{start}$
12:     Update values of all nodes in $S_{trace}$
13: **until** termination
```

Discussion

The addition step of rewiring the tree in RRT-VI is similar to the policy improvement step in approximate policy iteration. This is similar to Value Iteration as the improvement as well as evaluation

steps are done for a state at the same time. It can be shown that this is equivalent to value iteration on a discretized version of the MDP. The tree induces an adaptive discretization of the space, it splits it into the Voronoi regions corresponding to each node, and approximates the value of the entire to be constant and equal to J value calculated at that node. The larger the Voronoi region of the node, the more likely it is to be explored and split further.
We can conjecture that similar to RRT* producing an optimal path, RRT-VI will produce asymptotically optimal trajectories w.r.t the MDP as the number of nodes added to the tree increases.

4 Experimental Results

5 Conclusion and Future Work

References

[1] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[2] Levente Kocsis and Csaba Szepesvri. Bandit based monte-carlo planning. In *Proceedings of European Conference on Machine Learning 2006*, pages 282–293, 2006.

[3] Lihong Li, Michael L. Littman, and Christopher R. Mansley. Online exploration in least-squares policy iteration. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, pages 733–739, 2009.

[4] Andrew Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

[5] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Dept, Iowa State University, 1998.

[6] Andrew W. Moore and Christopher G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:199–233, 1995.

[7] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, June 2011.

[8] James J. Kuffner Jr. and Steven M. Lavalle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 995–1001, 2000.

[9] Leemon C. Baird. Advantage updating. Technical report, Wright Laboratory, 1993.