Policy Iteration on Continuous Domains using Rapidly-exploring Random Trees

Author Name1 ABC@SAMPLE.COM and **Author Name2** XYZ@SAMPLE.COM
Address

Editor: Steven C.H. Hoi and Wray Buntine

Abstract

Reinforcement Learning on continuous states and continuous actions is a challenging task. Sampling based planners such as Rapidly exploring Random Trees have been widely successful in solving path planning problems. In this paper, we combine some of these planning based techniques to develop a novel policy iteration based algorithm to solve optimal control problems in continuous domains. We show that this algorithm is fundamentally sound and present experimental results on some standard domains.

Keywords: Reinforcement learning, Continuous state action, RRTs, model based

1. Introduction

Reinforcement Learning(RL) deals with learning optimal control strategies through interactions with the world. One of the major hindrances in applying existing RL techniques to real world scenarios, is their inability to deal with continuous state and action variables. This necessitates the use of state abstractions when applying these techniques. A simple and straightforward example of state abstraction is an uniform grid based discretization of the space. Such a discretization is seldom useful as a very fine grid runs into the curse of dimensionality whereas a course grid lacks sufficient representative power. To overcome this problem, variable resolution discretization techniques have been proposed (Remi Munos, 1999; Moore and Atkeson, 1995). These adopt a top-down approach where the algorithm begins with a single grid discretization and subsequently splits the space based on certain local and global measures. Although the resolution is adaptive, the inherent rectangular structure might lead to structural peculiarities in some domains. Another more popular technique involves using function approximation to represent the value functions instead of lookup tables. These involve the use of neural networks, linear approximators and other such parametric techniques which require careful selection and fine-tuning.
Sampling based path planning algorithms have been successfully applied in the domain of robotics to plan trajectories through multidimensional continuous spaces. One of the advantages of sampling based planning techniques such as Rapidly-exploring Random Trees(RRTs) and Probabilistic Road Maps(PRMs) is their ability scale well to higher dimensions (Lavalle, 1998). The aim of path planning is to find feasible trajectories to reach certain goal regions. In this paper we present an algorithm for solving discounted deterministic control problems, where the aim is to come up with control policies that maximize the long term accruement of a scalar reward. We present a novel method that incorporates several key features of sampling based path planning algorithms, namely RRTS, to solve continuous domain opti-

mal control problems. In particular, the space filling property of RRTs and the asymptotic completeness guarantees make them an ideal method of generating samples. This can also be thought of as an adaptive discretization technique where the set of samples represents the discretized states.

2. Related work

RRTs are simple and easy to implement planning tools that possess good scalability on higher dimensional domains. They are known to be asymptotically complete, i.e. guaranteed to find a solution if one exists as long as the algorithm is run long enough. A recent extension of the standard RRT algorithm is RRT*, an algorithm that is capable of producing asymptotically *optimal* paths while maintaining the asymptotic completeness property(Karaman and Frazzoli, 2011). The optimality of a given path is measured based on its length in the configuration space. This algorithm is said to work on a binary-cost map where, there are just obstacles and free space. It does not optimize on a arbitrarily defined reward. The idea of biasing RRTs based on a given cost map or a heuristic has been studied earlier in the continuous planning domain. In their work on heuristically biased RRTs (Urmson and Simmons, 2003), the authors present an algorithm that combines heuristic search techniques like the estimated cost-to-go with sampling based planning. Although they do not optimize on an arbitrary given cost and only look at reaching goal regions, ideas for biasing the RRT expansion are discussed. Transition based RRTs (Jaillet et al., 2010; Berenson et al., 2011) look to optimize paths based on a given cost by using transitions tests, similar to stochastic optimization techniques that favor more optimal paths. Different concepts of optimality itself are discussed in this work. The one that is closest to a conventional reinforcement learning is the Integral Cost. The authors on the other hand favor a minimum mechanical work criteria that seems to avoid high cost regions locally while sacrificing overall path length. In general, the aim of a conventional RL agent is to maximize the long term rewards. The reward signal is the only available feedback, and trying to optimize other local criteria might lead to distracting the agent from its global aim (Randløv and Alstrøm, 1998). Our algorithm draws a lot of inspiration from these works. In a rough sense, the algorithm presented here estimates the cost (or the overall reward) of states by sampling and then grows an RRT biased according to the so evaluated *value function*. This process is then carried out iteratively, similar to the policy iteration algorithm. The following section clearly defines the notion of long term reward, value functions and policy iteration.

3. Preliminaries

A Markov Decision Process(MDP) is described as $\langle S, A, T, R \rangle$. $S \in \mathbb{R}^n$ is the domain of states. Each state has a corresponding set of allowed actions A_s. This set maybe discrete or continuous. The set A is defined as $\bigcup_s A_s$. This essentially saying, the set of actions A is not characterized by labels. Performing action '1' in some state and performing the action '1' in another state does not mean the same action is performed in both states. T is the state transition function $T : S \times A \to S$. R is a real valued reward function $R : S \times A \to \mathbb{R}$. The aim of the *RL agent* is to maximize the discounted cumulative reward defined as the

return, given by $\sum_t \gamma^t r_t$, where $\gamma \in (0,1)$ is the discount factor. A policy π is a mapping from the state space S to the action space A. $\pi : S \times A_s \rightarrow (0,1)$ that is, it represents the probability of choosing some action at a given state s. $J^\pi(s)$ is the state value function and is equal to the expected long term rewards, by following policy π starting from state s. Similarly, $Q^\pi(s,a)$ is the state-action value function. The value function might be expressed as,

$$\forall s \in S, \quad J^\pi(s) = \sum_a \pi(s,a)(R(s,s') + \gamma J^\pi(s')) \tag{1}$$

where s' is the state transitioned to according to the transition dynamics T. A policy π^* is optimal if $\forall s, \pi \quad J^{\pi^*}(s) \geq J^\pi(s)$. The optimal value function J^* can be calculated by replacing the expectation with the max operator. Since we deal with discrete dynamics, we will use denote the max as taken over all possible next states s' from a given state s. With a slight abuse of notation we may write

$$\forall s \in S, \quad J^\pi(s) = \max_{s'}(R(s,s') + \gamma J^\pi(s')) \tag{2}$$

3.1. Approximate Policy Iteration

Policy Iteration is an iterative algorithm for determining the optimal policy given an MDP (Sutton and Barto, 1998). The first step is policy evaluation, which is evaluating the value function as given by Equation (1). Following which is the policy improvement step, where the policy is improved by making it greedy with respect to the value function.

$$\pi_{m+1}(s) = \arg_{s'} \max J^{\pi_m}(s')$$

Where π_m corresponds to the policy evaluated in the m^{th} iteration of the algorithm. This algorithm eventually converges to a policy that is optimal. Value iteration is an algorithm, that similarly evaluates the optimal value function J^* by turning Equation (2) into an iterative update.

When dealing with continuous states and actions, the value function as well as the policy cannot be represented completely using tables. Thus some approximate representation is used which leads to some error. The approximation can be introduced in representing the value function and representing the policy. This framework is referred to as Approximate Policy Iteration(API). API is a fundamentally sound algorithm, meaning if the errors in the mentioned approximations are bounded, then the algorithm converges to a policy close to the optimal policy. Also as the errors decrease, the algorithm converges to the original optimal policy. The following theorem has been presented in (Bertsekas and Tsitsiklis, 1996).

Theorem 1 *Let $\hat{\pi}_m$ be the policy and $J^{\hat{\pi_m}}$ be the corresponding value function approximated in the m^{th} iteration. Let ϵ and δ be the error in approximation in the value function and the policy correspondingly. If*

$$\forall m \quad \| \hat{J}^{\hat{\pi}_m} - J^{\hat{\pi}_m} \|_\infty \leq \epsilon$$

and

$$\forall m \quad \| \hat{\pi}_m - \pi_m \|_\infty \leq \delta$$

then the final policy converges to a solution whose performance is bounded as

$$\limsup_{m\to\infty} \| \hat{J}^{\hat{\pi}_m} - V^* \|_\infty \leq \frac{\delta + 2\gamma\epsilon}{(1-\gamma)^2}$$

We will show in the later sections that our algorithm also falls under the API framework and that it is fundamentally sound.

3.2. Rapidly-exploring Random Trees(RRTs)

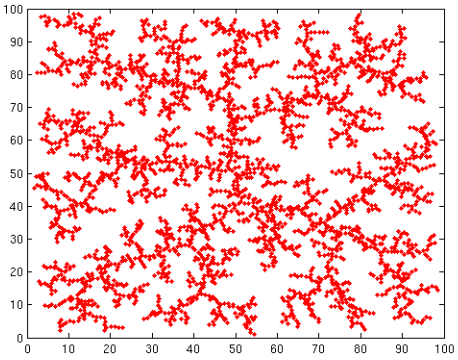

Figure 1: RRT with 300 nodes grown in $(0, 100)^2$ starting from (50,50)

The basic RRT construction is outlined in Algorithm 1. Figure 1 shows a RRT grown in a 2 dimensional space. The algorithm builds a Tree G of samples from some metric space S whose nodes and edges are represented by $V(G)$ and $E(G)$. The tree is rooted from a given state s_{start} and is grown as described in Algorithm 1. The `Extend` routine takes two points x and y and returns a point z such that

$$\mathcal{D}(x, y) > \mathcal{D}(z, y)$$

where $\mathcal{D}$ is a distance measure defined on S. The feasibility of a path from x to z is also checked. The amount by which the extension can take place is limited by some parameter η. i.e. $\mathcal{D}(x, y) \leq \eta$.

Algorithm 1: ConstructRRT(s_{start}, $rrtsize$)

$V(G) = s_{start}$, $E(G) = \emptyset$
repeat
  Uniformly sample a state s_{sample}
  $s_{near} \leftarrow$ `Nearest`$(V(G), s_{sample})$
  $s_{ext} \leftarrow$ `Extend`(s_{near}, s_{sample}) $V(G) = V(G) \cup s_{ext}$
  $E(G) = E(G) \cup (s_{near}, s_{ext})$
until $|V(G)| = rrtsize$

The construction of RRTs is very elegant and simple, yet they posses some interesting properties(Lavalle, 1998) as outlined below.

Theorem 2 *RRTs are probabilistically complete.*

$$\forall s \in S, \lim_{rrtsize \to \infty} Pr(s \in V(G)) = 1$$

That is, given enough time the RRT fills the space. In fact RRTs can be thought of as a online or *Monte-Carlo* space filling trees. The tree is biased towards exploring new areas in the state space. It can be seen that the nodes in the graph partition the space S into the corresponding Voronoi regions. When a point is sampled uniformly from the space, the probability that it falls into one of these regions and hence the probability that the corresponding node in the graph s_{near} is chosen for extension is proportional to the volume of its Voronoi space.

Theorem 3 *Probability of choosing a node s to be extended is* $\frac{1}{\eta} Vor(s)$

We will extend this basic RRT construction algorithm to solve optimal control problems by growing the tree in the state space of the MDP and using the samples to evaluate the value function.

4. The Algorithms

Given a control problem, represented as a MDP $\mathcal{M} = \langle S, A, T, R \rangle$, we present an algorithm RRT based Policy Iteration(RRTPI) that combines sampling methods such as RRTs in an iterative manner to find optimal trajectories in $\mathcal{M}$. We make the assumption that state space S is metric and has some distance measure $\mathcal{D}$ associated with it. We will first discuss a naive but intuitive algorithm, following which we will present the actual RRTPI algorithm. We begin by describing methods of generating samples and evaluating the value function based on these samples.

RRST

Algorithm 2: ConstructRRST$(p, rrtsize)$

$V(G) = s_{start}$, $E(G) = \emptyset$
while $|V(G)| \leq rrtsize$ **do**
  $s_{sample} \sim p(S)$
  $s_{near} \leftarrow$ `Nearest` $(s_{sample}, V(G))$
  $s_{ext} \leftarrow$ `Extend` (s_{near}, s_{sample})
  if $s_{ext} \neq \emptyset$ **then**
    $V(G) = V(G) \cup s_{ext}$
    $r \sim$ `SampleReward` (s_{near}, s_{ext})
    $E(G) = E(G) \cup (s_{near}, s_{ext}, r)$
  end
end

Rapidly-exploring Random Sample Trees(RRSTs), outlined in Algorithm 2, is a method to grow RRT like trees in the state space of the MDP $\mathcal{M}$. The algorithm takes as input

a probability distribution function p according to which random samples are generated in state space. It adds edges to the graph along with the reward r corresponding to the transition. This reward information is stored in the edges of the tree.

In order to implement this algorithm, we make the following two assumptions:

1. The algorithm needs access to a generative model of the world. This model allows us to sample the reward for arbitrary transitions.
2. A local approximate controller. This corresponds to the `Extend` subroutine. Given a starting state and a direction(denoted by a target state), the controller tries to generate feasible actions that leads to a state as defined in the Section 3.2. This assumption is similar to the local greedy controller assumption used in the parti-game algorithm (Moore and Atkeson, 1995).

Evaluating the Value function

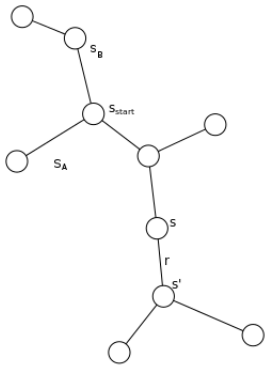

Figure 2: A small RRST. This corresponds to a discretization of the original MDP.

The RRST algorithm generates generates a tree in the state space of MDP $\mathcal{M}$ as shown in Figure 2. The state space can be discretized into partitions corresponding to the Voronoi regions of the nodes of the tree. The set of edges represent the possible transitions. Thus the process of growing the RRST corresponds to discretization of the state and action space of the given MDP $\mathcal{M}$. We can now define a discretized MDP whose states are represented by $V(G)$ and actions are represented by $E(G)$. We then describe a policy $\pi^{rrt}(s, s') \propto Vor(s')$ where $\pi(s, s')$ is the probability of choosing the edge between s and s' $Vor(s')$ is the volume of the Voronoi region of state s. Thus for eg. in Figure 2,

$$\frac{\pi^{rrt}(s_{start}, s_A)}{\pi^{rrt}(s_{start}, s_B)} = \frac{Vor(A)}{Vor(B)}$$

We can now evaluate the corresponding value function F^{rrt} by solving the bellman equations described in Equation (1). This can be done using iterative policy evaluation.

$$\forall s \in V(G), \quad F^{rrt}(s) = \sum_{s'} \pi(s, s')(R(s, s') + \gamma F^{rrt}(s')) \tag{3}$$

We then generalize the value function F^{rrt} to the original MDP $\mathcal{M}$ by defining it as

$$\forall s \in S, \quad \hat{F}(s) = F^{rrt}(s_{near})$$

where s_{near} is $\texttt{Nearest}(s, V(G))$. i.e. a piecewise constant approximation with the basis as the set of states defined in $V(G)$.

Naive RRTPI

We can now describe a simple algorithm that is similar to the policy iteration procedure.

1. Construct a RRST initially using uniform random distribution
2. Evaluate the value function as described in Equation (3)
3. Bias the probability distribution $p(s) \propto \hat{F}(s)$ and construct the RRST
4. Repeat steps 2 and 3 iteratively till convergence

Intuitively, step 2 of the algorithm corresponds to policy improvement, while step 3 corresponds to policy evaluation. This algorithm however does not perform as expected in practice. A closer look reveals some of the shortcomings of this naive algorithm. We discuss them and motivate the steps of the final RRTPI algorithm.

- Constructing a pdf from the Value function is non-trivial and requires some sort of bin or particle set approximation. This pulls the tree towards the globally optimal nodes, even through potentially low value nodes. Thus this step cannot be shown to correspond exactly with the policy improvement step. The value function incorporates the long term goodness of moving to a particular state and a policy that is locally greedy w.r.t to its own value function is indeed optimal (Sutton and Barto, 1998). So, the node to which the transition is made to - s_{ext}, must be chosen maximally based on its value $J(s_{ext})$.
- In order to evaluate $\pi(s, s')$ the volume of the Voronoi region of nodes must be computed. The original RRT algorithm, does this implicitly and biases the exploration proportionally to the volume of the Voronoi region. We modify the algorithm based on ideas from the Heuristic RRTs (Urmson and Simmons, 2003) and use rejection sampling in the `Extend` subroutine to only select new states with sufficiently good value. We can directly compute the optimal value function $F^*_{rrt}(s)$ corresponding to Equation (2) instead. Thus we have

  $$\forall s \in V(G), \quad F^*_{rrt}(s) = \max_{s'}(R(s, s') + \gamma F^*_{rrt}(s')) \tag{4}$$

  This can be done using the value iteration algorithm.

- Some MDPs are of finite horizon (such as the mountain car problem). Growing the particular branch of the tree must be stopped after the corresponding termination condition in the original condition has been reached. The leaves of the tree that have not reached the termination condition might lead to an incorrect value function, as during the evaluation step, these are considered as terminal states(since they have no further next states). Thus the tree must be pruned to contain only branches that have terminated. This may impose the condition that the tree has to be grown until at least one branch reaches termination which may require a very long time. The scarcity of rewards is a problem RL in general. RRTs are particularly suited in this case because, they posses excellent space filling properties and occupy space at an exponential rate.

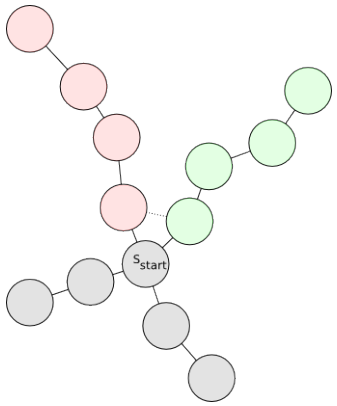

Figure 3: States in the green branch of the tree have high value, whereas the ones on the red branch have low value. The value between the states marked by dotted line is not generalized well even when it is possible to connect them using the controller.

- Consider the case where there are two branches of the tree that are very close to each other in distance as shown in Figure 3. One reaches a highly rewarding state, whereas the other does not. Thus the value backed up by the value iteration algorithm will be dramatically different along the states in the branch. Although these states are close to each-other distance wise, their value is poorly generalized. There may have been a critical decision in the history of the two branches that leads to this disparity, although we observe that this is not the case as there can be many valid transitions between the nodes of the two branches. This was observed to be true in the continuous grid world domain. This seems to hint that growing a directed acyclic graph(DAG) instead of a tree might lead to better generalization. This is similar to the asymptotically optimal path planning technique RRT* (Karaman and Frazzoli, 2011). In fact this is related to the fact that the plain RRT algorithm can be proved to be non-optimal for path planning (Karaman and Frazzoli, 2011). Constructing such a DAG is shown in Algorithm 3. We use the parameter η that defines the extent of the local controller. Connections are attempted between all nodes in the graph that are within an η-radius of s_{ext} to s_{ext}, where s_{ext} is the new node being added to the DAG. Successful connections are added to the edge set and the corresponding rewards are

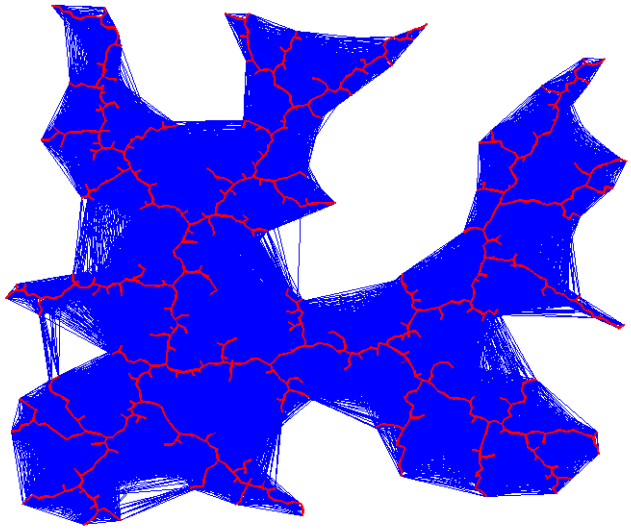

Figure 4: Forming a DAG instead of a Tree. The DAG is shown in blue, the corresponding the Tree is shown in red.

stored as well. An example of such a DAG constructed in a 2 dimensional space is shown in Figure 4. Constructing a DAG involves some additional work, but as shown, the extra edges improve the quality of solution. η maybe be chosen to decrease, as the number of nodes in the graph increase. Typically, choosing η to be proportional to $dfracn\log n$ results in the addition of $O(\log n)$ edges per node, and thus the number of edges increase to $O(n \log n)$ compared to $O(n)$ for the tree.

Algorithm 3: ConstructRRSG$(p, rrtsize)$

$V(G) = s_{start}$, $E(G) = \emptyset$
while $|V(G)| \leq rrtsize$ **do**
  $s_{sample} \leftarrow$ `Sample` (p)
  $s_{near} \leftarrow$ `Nearest` $(s_{sample}, V(G))$
  $s_{ext} \leftarrow$ `Extend` (s_{near}, s_{sample})
  $S_\eta \leftarrow \eta$-`Near` $(s_{ext}, V(G))$
  $V(G) = V(G) \cup s_{ext}$
  for *valid* $s' \in S_\eta \cup s_{near}$ **do**
    $r \sim$ `SampleReward` (s', s_{ext})
    $E(G) = E(G) \cup (s', s_{ext}, r)$
  end
end

RRTPI

The RRTPI algorithm is presented in Algorithm 4. This incorporates solutions to all issues discussed above. At any time in the construction of the DAG, the new node to be added s_{ext} is chosen according to its value. This is achieved by first scaling $\hat{F}$ to $(0, 1)$ and using rejection sampling. The value function $\hat{F}$ itself is learned by evaluating the optimal value function F^*_{rrt} in the discretized MDP and then generalizing it as a piecewise constant function. Thus by iteratively evaluating and improving the policy, the algorithm eventually converges to a optimal trajectory. Convergence can be determined by checking if the change in the average value of the nodes along the optimal path of the graph is within some bound. Figure 5 shows the algorithm finding a near optimal path in a simple 2 dimensional continuous gridworld. The controller corresponds to straight line paths within some distance. The reward function is -1 everywhere and on reaching the goal, the termination flag is raised along with a reward of +100. Hitting obstacles or the boundary gives a reward of -10 and the episode terminates. The value function that is learnt is shown in Figure 6. In the next section we will show some properties of this algorithm and then finally present results on standard benchmarks comparing the RRTPI algorithm with other algorithms.

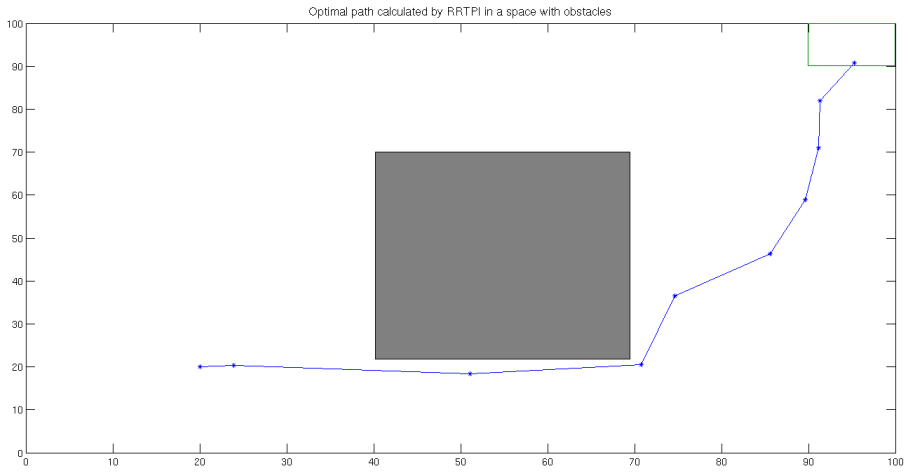

Figure 5: RRTPI solving a 2 dimensional gridworld. The gray region is an obstacle and the green box represents the goal regions.

5. Discussions

We present some results on the RRTPI algorithm.

By construction it can be seen that the DAG version has all the properties of the RRT algorithm discussed in Section 3.2. Namely asymptotic completeness and bias towards larger Voronoi regions.

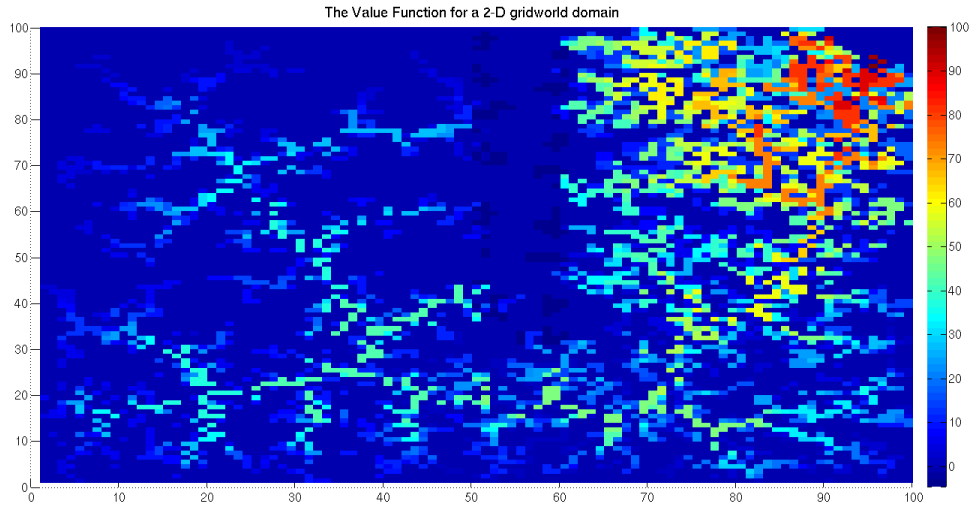

Figure 6: Heat map showing the value function learnt on the state space of the gridworld MDP

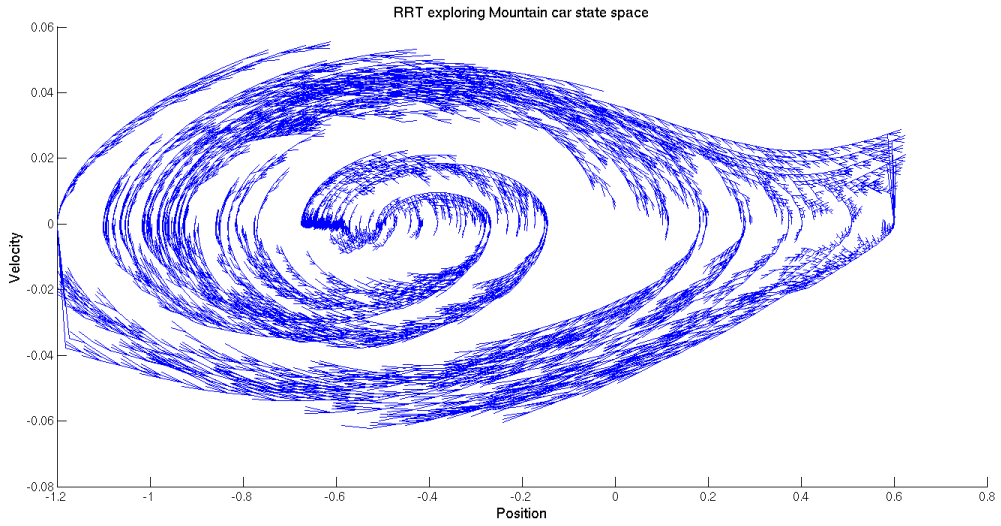

Figure 7: The first iteration of the rrtpi algorithm on the mountain car domain

Theorem 4 *RRTPI is a fundamentally sound algorithm.*

Proof We will prove the theorem by adapting the generic proof of Approximate Policy Iteration as shown in Section 3.1. It suffices to show that the error in approximating the policy as well as the value function is bounded.
We first show that the constructing the graph implicitly corresponds to policy improvement.

Algorithm 4: RRTPI($rrtsize$)

Initialize $\forall s \hat{F}(s) \leftarrow 0$
repeat
  $V(G) = s_{start}$, $E(G) = \emptyset$
  while $|V(G)| \leq rrtsize$ **do**
    $s_{sample} \leftarrow$ Sample uniformly from S
    $s_{near} \leftarrow$ `Nearest` $(s_{sample}, V(G))$
    $s_{ext} \leftarrow$ `Extend` (s_{near}, s_{sample})
    if $\hat{F}(s_{ext}) < rand(0,1)$ **then**
      continue
    end
    $S_\epsilon \leftarrow \epsilon$-Near $(s_{ext}, V(G))$
    $V(G) = V(G) \cup s_{ext}$
    for *valid* $s' \in S_\epsilon \cup s_{near}$ **do**
      $r \sim$ `SampleReward` (s', s_{ext})
      $E(G) = E(G) \cup (s', s_{ext}, r)$
    end
  end
  solve $\forall s \in V(G), \;\; F^*_{rrt}(s) = \max_{s'}(R(s,s') + \gamma F^*_{rrt}(s'))$
  $\forall s \in S, \;\; \hat{F}(s) = F^{rrt}(s_{nearest})$ where $s_{nearest} = \text{Nearest}(s, V(G))$
  `Scale` $\hat{F}$ to $[0,1]$
until *convergence*

The probability of forming an edge between s_{ext} and s_{near} is given by

$$Pr(\text{Choosing } s_{near}) \times Pr(\text{Choosing } s_{ext})$$

$$\propto Vor(s_{near}) \times \hat{F}(s_{ext})$$

and $\hat{F}$ corresponds to the optimal value function of the discretized MDP represented by the Graph.

The construction of the graph possesses properties similar to RRTs. Thus,

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

The sampling graph constructed in the RRTPI algorithm discretizes the given MDP $\mathcal{M} = \langle S, A, T, R \rangle$ into a discrete MDP $\mathcal{M}' = \langle V(G), E(G), T', R \rangle$ By construction of the graph and the asymptotic completeness property,

$$\lim_{rrtsize \to \infty} V(G) \to S$$

$$\lim_{rrtsize \to \infty} E(G) \to A$$

If F^* is the optimal value function corresponding to the original MDP and $\hat{F}$ is the value function described in RRTPI algorithm then from the above result it follows that

$$\forall s \in S, \quad \lim_{rrtsize \to \infty} |F^*(s) - \hat{F}(s)| \to 0$$

Thus the error in approximating the value function can be trivially bound as there exists some *rrtsize* for which the above error is within a given bound ϵ. The policy is implicitly represented using the value function itself. Thus we can conclude that the algorithm fits into the API framework and hence is fundamentally sound.

The main issue with forming a DAG as mentioned above, is that connecting two arbitrary states in the space might be a difficult task. This is especially true in the case of under-actuated systems, where it may sometimes even be impossible to do so. For some under actuated domains such as the mountain car, we do not construct a DAG but instead use the same RRRTPI described above with a tree, omitting the part where connections are made to other existing nodes in the graph. Such a tree is shown in Figure 7. This algorithm still converges to good solutions as can be seen in experimental results section. ■

6. Experimental results

The performance of the algorithm was first evaluated in a simple continuous grid world as shown in the previous section. The algorithm produces policies that are similiar to the optimal paths produced my the RRT* algorithm. The algorithm was also compared with Q learning on a uniform discritized grid and an adaptive resolution technique presented in (Remi Munos, 1999). The comparison is performed on a standard 2 dimensional domain, the mountain-car. The domain is described in (Singh et al., 1996). The results are summarized in Figure 8. The RRTPI algorithm is run with a limit on the number of nodes in the graph. The algorithm shows close to optimal performance on this domain with a relatively low number of states. The performance measure is the average return of the initial state, averaged across 50 runs.
Next we compare the performance on the acrobot domain. The domain used here is described in Chapter 11 of Sutton and Barto (1998). The results are summarized in Figure 9. As can be seen the RRTPI algorithm is again able to generate good solutions with relatively smaller number of samples.

7. Conclusion and Future Work

We present RRTPI, a novel algorithm that combines the properties of sampling based continuous domain planners to solve problems of optimal control. This algorithm is shown to have competitive performance in standard domains. The algorithm was proved to be asymptotically complete and fundamentally sound. Further interesting analysis, can be carried out on the rate of convergence and the sample complexity. By using some of the properties of Random Geometric Graphs as discussed for the RRT* algorithm (Karaman and Frazzoli, 2011), we may be able to prove that the graph constructed possesses fast mixing properties. Thus providing a starting point for arriving at results on sample complexity and rate of convergence.

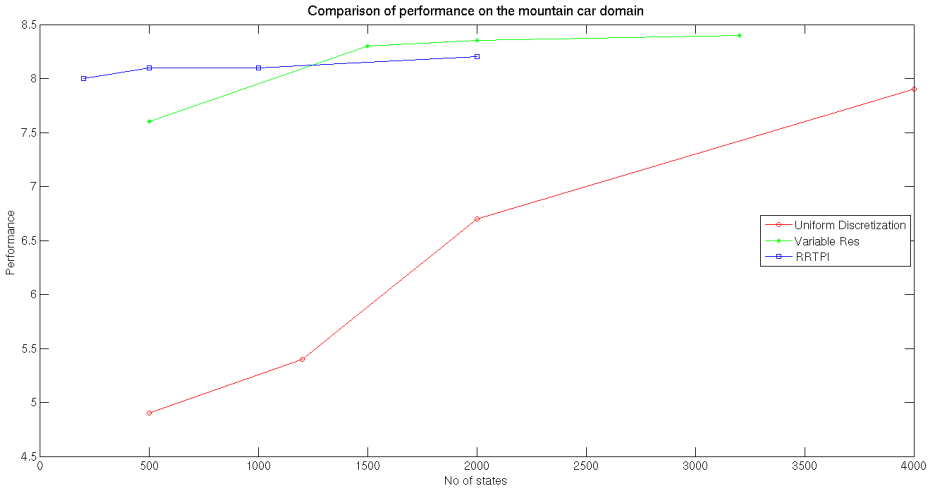

Figure 8: RRTPI performs competitively on the standard mountain car domain

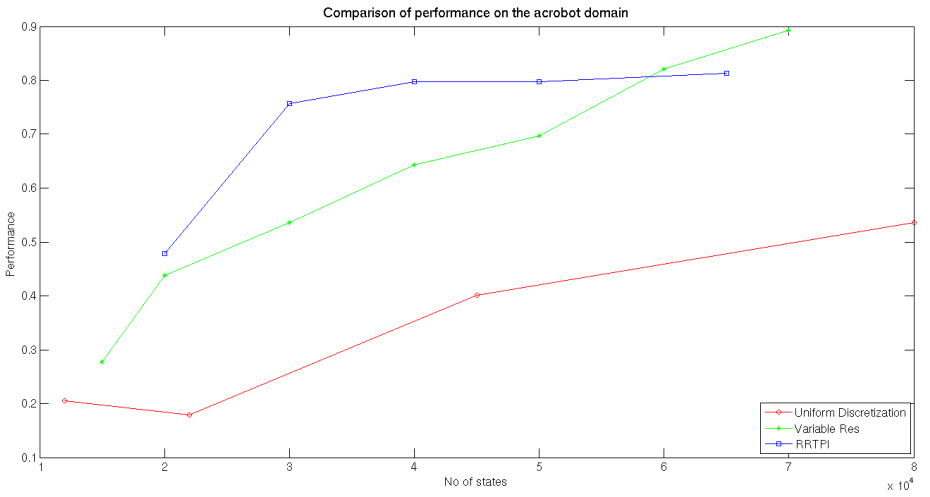

Figure 9: RRTPI performs competitively on the standard mountain car domain

References

Dmitry Berenson, Thierry Simeon, and Siddhartha Srinivasa. Addressing cost-space chasms in manipulation planning. In *IEEE International Conference on Robotics and Automation (ICRA '11)*, May 2011.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming (Optimization and Neural Computation Series, 3)*. Athena Scientific, May 1996.

Leonard Jaillet, Juan Cortes, and T. Simeon. Sampling-based path planning on configuration-space costmaps. *IEEE Transactions on Robotics*, 26(4):635–646, August

2010.

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

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

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.

Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 463–471, 1998.

Andrew Moore Remi Munos. Variable resolution discretizations for high-accuracy solutions of optimal control problems. In *Proceedings of the International Joint Conference on Artificial Intelligence, Stockholm*, pages 1348–1355, 1999.

Satinder Singh, Richard S. Sutton, and P. Kaelbling. Reinforcement learning with replacing eligibility traces. In *Machine Learning*, pages 123–158, 1996.

R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 28. MIT press, 1998.

Christopher Urmson and Reid Simmons. Approaches for heuristically biasing rrt growth. In *IEEE/RSJ IROS 2003*, October 2003.