2016 IEEE 55th Conference on Decision and Control (CDC)
ARIA Resort & Casino
December 12-14, 2016, Las Vegas, USA

Incremental Sampling-based Motion Planners Using Policy Iteration Methods

Oktay Arslan[1] Panagiotis Tsiotras[2]

Abstract— **Recent progress in randomized motion planning has led to the development of a new class of sampling-based algorithms that provide asymptotic optimality guarantees, almost surely, notably the RRT* algorithm among others. The recently proposed RRT$^\#$ algorithm utilizes dynamic programming ideas (namely, asynchronous value iteration) and implements them incrementally on randomly generated graphs to improve on RRT* and thus obtain high quality solutions with enhanced rates of convergence. In this work we explore a different class of dynamic programming algorithms for solving shortest-path problems on random graphs generated by iterative sampling, which utilize policy iteration instead of value iteration and that are better suited for massive parallelization.**

I. Introduction

Robot motion planning is one of the fundamental problems in robotics. It poses several challenges due to the high-dimensionality of the (continuous) search space, the complex geometry of unknown infeasible regions, the possibility of a continuous action space, and the presence of differential constraints [1]. Recent approaches use randomization to construct sparse graphs (hence avoiding gridding) of the search space [2], [3], [4]. These planners are simple to implement, require less memory, work efficiently for high-dimensional problems, but come with a relaxed notion of completeness, namely, *probabilistic completeness*. That is, the probability that the planner fails to return a solution, if one exists, decays to zero as the number of samples approaches infinity. On the other hand, the resulting paths could be arbitrarily suboptimal [5]

Recently, almost surely asymptotically optimal variants of these algorithms, such as PRM^* and RRT^* have been proposed [5], in order to remedy the undesirable behavior of the original RRT-like algorithms in terms of optimality guarantees. The seminal work of [5] has sparked a renewed interest in asymptotically optimal probabilistic, sampling-based motion planners. Several variants have been proposed that utilize the original ideas of [5]; a partial list includes [6], [7], [8], [9].

Careful analysis reveals that the optimality guarantees of RRT^* are the result of (at first glance) hidden ideas based on dynamic programming principles. Based on this key insight, the recently proposed $\mathrm{RRT}^\#$ algorithm [6], [10] utilizes a Gauss-Seidel version of asynchronous value iteration [6], [7] to speed up the convergence of RRT^*. Extensions based on similar ideas as the $\mathrm{RRT}^\#$ algorithm include the FMT* algorithm [8], the RRT$^\mathrm{x}$ algorithm [9], and the BIT* algorithm [11].

In this work, we depart from the previous VI-based algorithms and we propose, instead, a novel class of algorithms based on policy-iteration (PI). Policy iteration is an alternative to value iteration for solving dynamic programming problems and fits naturally into our framework, in the sense that a policy in a graph search problem amounts to an assignment of a (unique) parent to each vertex. Use of policy iteration has the following benefits: first, no queue is needed to keep track of the cost of each vertex. A subset of vertices is selected for Bellman updates, and policy improvement on these vertices can be done in parallel at each iteration. Second, for a given graph, determination of the *optimal* policy is obtained after a *finite* number of iterations since the policy space is finite [12]. The determination of the optimal value for each vertex, on the other hand, requires an infinite number of iterations. More crucially, and in order to find the optimal policy, only the correct ordering of the vertices is needed, not their exact value. This can be utilized to develop approximation algorithms that speed up convergence. Third, although policy iteration methods are often slower than value iteration methods, they tend to be better amenable for parallelization and are faster if the structure of the problem is taken into consideration during implementation.

II. Problem Formulation

Let $\mathcal{X}$ denote the configuration (search) space, which is assumed to be an open subset of $\mathbb{R}^d$, where $d \in \mathbb{N}$ with $d \geq 2$. The *obstacle region* and the *goal region* are denoted by $\mathcal{X}_{\mathrm{obs}}$ and $\mathcal{X}_{\mathrm{goal}}$, respectively, both assumed to be closed sets. The obstacle-free space is defined by $\mathcal{X}_{\mathrm{free}} = \mathcal{X} \setminus \mathcal{X}_{\mathrm{obs}}$. Elements of $\mathcal{X}$ are the states (or configurations) of the system. Let the *initial configuration* of the robot be denoted by $x_{\mathrm{init}} \in \mathcal{X}_{\mathrm{free}}$. The (open) neighborhood of a state $x \in \mathcal{X}$ is the open ball of radius $r > 0$ centered at x, that is, $B_r(x) = \{x' \in \mathcal{X} : \|x - x'\| < r\}$. Given a subset $S \subseteq V$ the notation $|S|$ denotes its cardinality, that is, the number of elements in S.

We will approximate $\mathcal{X}_{\mathrm{free}}$ with an increasingly dense sequence of discrete subsets of $\mathcal{X}_{\mathrm{free}}$. Each such discrete approximation of $\mathcal{X}_{\mathrm{free}}$ will be encoded in a graph $\mathcal{G} = (V, E)$ with V being the set of vertices (the elements of the discrete approximation of $\mathcal{X}_{\mathrm{free}}$) and with edge set $E \subseteq V \times V$ encoding allowable transitions between elements of V. Hence, $\mathcal{G}$ is a directed graph. Transitions between two vertices x and x' in V are enabled by a control action $u \in U(x)$ such that x' is the successor vertex of x in $\mathcal{G}$

[1] Robotics PhD student with the Institute for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332-0150, USA. Oktay Arslan is currently a Robotics Technologist with the Jet Propulsion Laboratory at the California Institute of Technology, Pasadena, CA 91109-8099, USA, Email:oktay.arslan@jpl.nasa.gov

[2]Panagiotis Tsiotras is with the faculty of D. Guggenheim School of Aerospace Engineering and the Institute for Robotics and Intelligent Machines at the Georgia Institute of Technology, Atlanta, GA 30332-0150, USA, Email:tsiotras@gatech.edu

under the action u so that $(x, x') \in E$. Let $U = \cup_{x \in V} U(x)$. We use the mapping $f : V \times U \to V$ given by

$$x' = f(x, u), \qquad u \in U(x), \tag{1}$$

to formalize the transition from x to x' under the control action u. In this case, we say that x' is the successor of x and that x is the predecessor of x'. The set of predecessors of $x \in V$ will be denoted by $\texttt{pred}(\mathcal{G}, x)$, and the set of successors of x will be denoted by $\texttt{succ}(\mathcal{G}, x)$. Also, we let $\overline{\texttt{pred}}(\mathcal{G}, x) = \texttt{pred}(\mathcal{G}, x) \cup \{x\}$.

Once we have abstracted $\mathcal{X}_{\text{free}}$ using the graph $\mathcal{G}$, the motion planning problem becomes one of a shortest path problem on the graph $\mathcal{G}$. To this end, we define the path σ in $\mathcal{G}$ to be a sequence of vertices $\sigma = (x_0, x_1, \ldots, x_N)$ such that $x_{k+1} \in \texttt{succ}(\mathcal{G}, x_k)$ for all $k = 0, 1, \ldots, N-1$. The length of the path is N, denoted by $\texttt{len}(\sigma) = N$. When we want to specify explicitly the first node of the path we will use the first node as an argument, and we write $\sigma(x_0)$. The kth element of σ will be denoted by σ_k. That is, if $\sigma(x_0) = (x_0, x_1, \ldots, x_N)$ then $\sigma_k(x_0) = x_k$ for all $k = 0, 1, \ldots, N$. A path is rooted at x_{init} if $x_0 = x_{\text{init}}$. A path rooted at x_{init} terminates at a given goal region $\mathcal{X}_{\text{goal}} \subset \mathcal{X}_{\text{free}}$ if $x_N \in \mathcal{X}_{\text{goal}}$.

To each edge (x, x') encoding an allowable transition from $x \in \mathcal{X}_{\text{free}}$ to $x' \in \texttt{succ}(\mathcal{G}, x)$, we associate a finite cost $\texttt{c}(x, x')$. Given a path $\sigma(x_0)$, the cumulative cost along this path is then

$$\sum_{k=0}^{N-1} \texttt{c}(x_k, x_{k+1}). \tag{2}$$

Given a point $x \in \mathcal{X}$, a mapping $\mu : x \mapsto u \in U(x)$ that assigns a control action to be executed at each point x is called a *policy*. Let $\mathcal{M}$ denote the space of all such policies. Under some assumptions on the connectivity of the graph $\mathcal{G}$ and the cost of the directed edges, one can use DP algorithms and the corresponding Bellman equation in order to compute optimal policies. Note that a policy $\mu \in \mathcal{M}$ for this problem defines a graph whose edges are $(x, f(x, \mu(x))) \in E$ for all $x \in V$. The policy μ is proper if and only if this graph is acyclic, i.e., the graph has no cycles. Thus, there exists a proper policy μ if and only if each node is connected to the $\mathcal{X}_{\text{goal}}$ with a directed path. Furthermore, an improper policy has finite cost, starting from every initial state, if and only if all the cycles of the corresponding graph have non-negative cost [12]. Convergence of the DP algorithms is proven if the graph is connected and the costs of all its cycles are positive [13].

III. Overview of Dynamic Programming

Dynamic programming solves sequential decision-making problems having a finite number of stages. In terms of DP notation, our system is governed by the equation $x' = f(x, u)$ where the running cost function is defined as $g(x, u) = \texttt{c}(x, f(x, u))$. The objective is to minimize the cumulative cost over a given horizon (see (2)). Given a sequential decision problem of the previous form it is well known that the optimal cost function satisfies the following *Bellman equation*:

$$J^*(x) = \inf_{u \in U(x)} \Big\{ g(x, u) + J^*(f(x, u)) \Big\}, \quad \forall x \in \mathcal{X}. \tag{3}$$

The optimal policy $\mu^* \in \mathcal{M}$ is given by

$$\mu^*(x) \in \operatorname*{arg\,min}_{u \in U(x)} \Big\{ g(x, u) + J^*(f(x, u)) \Big\}, \quad \forall x \in \mathcal{X}. \tag{4}$$

Note that if we are given a policy $\mu \in \mathcal{M}$ (not necessarily optimal) we can compute its cost from

$$J_\mu(x) = g(x, \mu(x)) + J_\mu(f(x, \mu(x))), \quad \forall x \in \mathcal{X}. \tag{5}$$

It follows that $J^*(x) = \inf_{\mu \in \mathcal{M}} J_\mu(x), \quad x \in \mathcal{X}$. By introducing the expression $H(x, u, J) = g(x, u) + J(f(x, u))$ and letting the operator T_μ for a given policy $\mu \in \mathcal{M}$, $(T_\mu J)(x) = H(x, \mu(x), J)$, where $x \in \mathcal{X}$, we can define the Bellman operator T

$$(TJ)(x) = \inf_{u \in U(x)} H(x, u, J) = \inf_{\mu \in \mathcal{M}} (T_\mu J)(x), \; x \in \mathcal{X},$$

which allows us to write the Bellman equation (3) succinctly as $J^* = TJ^*$, and the optimality condition (4) as $T_{\mu^*} J^* = TJ^*$. That is, J^* is the fixed point of the Bellman operator T. In a similar way, J_μ, the cost function of the policy μ is a fixed point of T_μ (see (5)).

There are three different classes of DP algorithms to compute the optimal policy μ^* and the optimal cost function J^*.

a) Value Iteration (VI): This algorithm computes J^* by relaxing Eq. (3), starting with some J^0, and generating a sequence $\big\{T^i J\big\}_{i=0}^{\infty}$ using the iteration $J^{i+1} = TJ^i$. The generated sequence converges to the optimal cost function due to contraction property of the Bellman operator T [13].

b) Policy Iteration (PI): This algorithm starts with an initial policy μ^0 and generates a sequence of policies μ^i by performing Bellman updates. Given the current policy μ^i, the typical iteration is performed in two steps:

i) **Policy evaluation**: compute J_{μ^i} as the unique solution of the equation $J_{\mu^i} = T_{\mu^i} J_{\mu^i}$.
ii) **Policy improvement**: compute a policy μ^{i+1} that satisfies $T_{\mu^{i+1}} J_{\mu^i} = TJ_{\mu^i}$.

c) Optimistic Policy Iteration (O-PI): This algorithm works the same as PI, but differs in the policy evaluation step. Instead of solving the system of linear equations exactly as in the policy evaluation step, it performs an approximate evaluation of the current policy and uses this information in the subsequent policy improvement step.

IV. Proposed Approach

A. From Random Geometric Graphs to DP

The main difference between standard shortest path problems on graphs and sampling-based methods for solving motion planning problems is the fact that in the former case the graph is given a priori, whereas in the latter case the path is constructed on-the-fly by sampling randomly allowable configuration points from $\mathcal{X}_{\text{free}}$ and by constructing the graph $\mathcal{G}$ incrementally, adding one, or more, vertices at each iteration step. Of course, such an iterative construction raises several questions, such as: is the resulting graph connected? under what conditions one can expect that $\mathcal{G}$ is an accurate representation of $\mathcal{X}_{\text{free}}$? how does discretizing the actions/control inputs affects the movement between sampled successor vertices, etc. Some of these questions have been addressed in [5], so we will not elaborate further on the graph construction.

The RRG algorithm [5] constructs random graphs by placing a collection of vertices drawn randomly according to a specified probability distribution. Two points (vertices) are connected if and only if the distance between the two is less than some distance r. It can be shown that the resulting random geometric graph is connected almost surely as long as the connection radius r is strictly greater than a critical value $r^* = \big\{\log(n)/(n\zeta_d)\big\}^d$, where ζ_d is volume of the unit ball in $\mathbb{R}^d$. In the RRG algorithm the connection radius is shrunk as a function of the number of vertices, while still being strictly greater than the critical radius value r^*. By doing so, one is guaranteed to obtain a connected and sparse graph, yet the graph is rich enough to provide asymptotic optimality guarantees, almost surely. In this work, we leverage this nice feature of random geometric graphs to get a consistent discretization of the continuous domain of the robot motion planning problem.

To this end, let $\mathcal{G} = (V, E)$ denote the graph constructed by the RRG algorithm at some iteration, where V and $E \subseteq V \times V$ are finite sets of vertices and edges, respectively. Based on the previous discussion, $\mathcal{G}$ is connected and all edge costs are positive, which implies that the cost of all the cycles in $\mathcal{G}$ are positive. Using the notation introduced in Section II, we can define on this graph the sequential decision system $x' = f(x, u)$ where $x' \in \mathtt{succ}(\mathcal{G}, x)$ and with transition cost $g(x, u) = \mathtt{c}(x, f(x, u))$. Once a policy μ is given (optimal or not), there is a unique $x' \in \mathtt{succ}(\mathcal{G}, x)$ such that $x' = f(x, \mu(x))$, that is, the parent of x. Accordingly, x is the child of x' under the policy μ. Conversely, a parent assignment for each node in $\mathcal{G}$ defines a policy. Note that each node has a single parent under a given policy, but may have multiple children.

In our case, the graph computed by the RRG algorithm is a connected graph by construction, and all edge cost values are positive, which implies that the costs of all its cycles are positive. Therefore, convergence is guaranteed and the resulting optimal policy is proper.

B. DP Algorithms for Sampling-based Planners

The sampling-based motion planner which utilizes VI, i.e., RRT$^\#$, was presented in [6], [10]. The RRT$^\#$ algorithm implements the Gauss-Seidel version of the VI algorithm and provides a sequential implementation. In this work, we follow up on the same idea and propose a sampling-based algorithm which utilizes PI.

The body of the PI-RRT$^\#$ algorithm is given in Algorithm 1. The algorithm initializes the graph with x_{goal} in Line 2 and incrementally builds the graph from x_{goal} toward x_{init}. The algorithm includes a new vertex and a couple new edges into the existing graph at each iteration. If this new information has a potential to improve the existing policy, then, a slightly modified PI algorithm is subsequently called in the Replan procedure. Specifically, and for the sake of numerical efficiency, unlike the standard PI algorithm, policy improvement is performed only for a subset vertices B which have the potential to be part of the optimal solution. Note that $B^{k,i}$ will denote the set of these vertices during the kth iteration and ith policy improvement step. The fact that this modification of the PI still ensures the asymptotic optimality, almost surely, of the proposed PI-RRT$^\#$ algorithm requires extra analysis, which is the topic of Section V.

Algorithm 1: Body of the PI-RRT$^\#$ Algorithm

```
1  PI-RRT#($x_{init}$, $x_{goal}$, $\mathcal{X}$)
2      $V \leftarrow \{x_{goal}\}$; $B \leftarrow V$; $E \leftarrow \emptyset$;
3      $\mathcal{G} \leftarrow (V, E)$;
4      for $k = 1$ to $N$ do
5          $x_{rand} = \mathtt{Sample}(k)$;
6          $(\mathcal{G}, B') \leftarrow \mathtt{Extend}(\mathcal{G}, B, x_{init}, x_{rand})$;
7          if $|B'| > |B|$ then
8              $B \leftarrow \mathtt{Replan}(\mathcal{G}, B', x_{init}, x_{goal})$;
9      $(V, E) \leftarrow \mathcal{G}$; $E' \leftarrow \emptyset$;
10     foreach $x \in V$ do
11         $E' \leftarrow E' \cup \{(x, \mathtt{parent}(x))\}$;
12     return $\mathcal{T} = (V, E')$;
```

The Extend procedure is given in Algorithm 2. If a new vertex is decided for inclusion, its control is initialized by performing policy improvement in Lines 6-14. Then, it is checked in Line 15 if the new vertex has a potential to improve the existing policy where h denotes an admissible heuristic function that computes an estimate of the cost between two given points. If so, it is included to the set of vertices which are selected to perform policy improvement. Such heuristics have been previously used to focus the search of sampling-based planner, see for example [14], [6].

Algorithm 2: Extend Procedure for PI-RRT$^\#$

```
1  Extend($\mathcal{G}$, $B$, $x_{init}$, $x_{rand}$)
2      $(V, E) \leftarrow \mathcal{G}$; $E' \leftarrow \emptyset$;
3      $x_{nearest} = \mathtt{Nearest}(\mathcal{G}, x_{rand})$;
4      $x_{new} = \mathtt{Steer}(x_{rand}, x_{nearest})$;
5      if $\mathtt{ObstacleFree}(x_{new}, x_{nearest})$ then
6          $\mathtt{J}(x_{new}) = \mathtt{c}(x_{new}, x_{nearest}) + \mathtt{J}(x_{nearest})$;
7          $\mathtt{parent}(x_{new}) = x_{nearest}$;
8          $\mathcal{X}_{near} \leftarrow \mathtt{Near}(\mathcal{G}, x_{new}, |V|)$;
9          foreach $x_{near} \in \mathcal{X}_{near}$ do
10             if $\mathtt{ObstacleFree}(x_{new}, x_{near})$ then
11                 if $\mathtt{J}(x_{new}) > \mathtt{c}(x_{new}, x_{near}) + \mathtt{J}(x_{near})$
                   then
12                     $\mathtt{J}(x_{new}) = \mathtt{c}(x_{new}, x_{near}) + \mathtt{J}(x_{near})$;
13                     $\mathtt{parent}(x_{new}) = x_{near}$;
14                 $E' \leftarrow E' \cup \{(x_{new}, x_{near}), (x_{near}, x_{new})\}$;
15         if $\mathtt{h}(x_{init}, \mathtt{parent}(x_{new})) + \mathtt{J}(\mathtt{parent}(x_{new})) <$
           $\mathtt{J}(x_{init})$ then
16             $B \leftarrow B \cup \{x_{new}\}$;
17         $V \leftarrow V \cup \{x_{new}\}$;
18         $E \leftarrow E \cup E'$;
19     $\mathcal{G} \leftarrow (V, E)$;
20     return $(\mathcal{G}, B)$;
```

The Replan procedure which implements the PI algorithm is shown in Algorithm 3. The policy improvement step is performed in Lines 2-9 until the cost of the existing policy becomes almost stationary. Note that the for-loop in the Replan procedure can run in parallel.

The policy evaluation step is implemented in Algorithm 4. Algorithm 4 solves a system of linear equations by exploiting the underlying structure. Simply, the existing policy forms a tree in the current graph and the solution of the system

Algorithm 3: `Replan` Procedure (PI)

```
1  Replan($\mathcal{G}$, $B$, $x_{\rm init}$, $x_{\rm goal}$)
2      Loop
3          foreach $x \in B$ do
4              $J' = \mathtt{J}(x)$;
5              foreach $v \in \mathtt{succ}(\mathcal{G}, x)$ do
6                  if $J' > \mathtt{c}(x, v) + \mathtt{J}(v)$ then
7                      $J' = \mathtt{c}(x, v) + \mathtt{J}(v)$;
8                      $\mathtt{parent}(x) = v$;
9              $\Delta J(x) = \mathtt{J}(x) - J'$;
10         if $\max_{x \in B} \Delta J(x) \leq \epsilon$ then
11             return $B$;
12         $B \leftarrow \mathtt{Evaluate}(\mathcal{G}, x_{\rm init}, x_{\rm goal})$;
```

of linear equations corresponds to the cost of each path connecting vertices to the goal region via edges of the tree. If $x_{\rm init}$ is already in the graph, the algorithm computes the cost of the path between $x_{\rm init}$ and $x_{\rm goal}$ by using queue q. Subsequently, the set of vertices that are promising, i.e., those in the set B and their cost-to-go values are computed by using the cost-to-go value of $x_{\rm init}$.

Algorithm 4: `Evaluate` Procedure (PI)

```
1  Evaluate($\mathcal{G}$, $x_{\rm init}$, $x_{\rm goal}$)
2      $(V, E) \leftarrow \mathcal{G}$;
3      if $x_{\rm init} \in V$ then
4          $x = x_{\rm init}$;
5          while $x \neq x_{\rm goal}$ do
6              $q.push\_front(x)$;
7              $x = \mathtt{parent}(x)$;
8          $\mathtt{J}(x_{\rm goal}) = 0$;
9          while $q.empty()$ do
10             $x = q.pop\_front()$;
11             $\mathtt{J}(x) = \mathtt{c}(x, \mathtt{parent}(x)) + \mathtt{J}(\mathtt{parent}(x))$;
12     else
13         $\mathtt{J}(x_{\rm init}) = \infty$;
14     $B \leftarrow \{x_{\rm goal}\}$;
15     $q.push\_back(x_{\rm goal})$;
16     while $q.nonempty()$ do
17         $x = q.pop\_front()$;
18         if $\mathtt{h}(x_{\rm init}, x) + \mathtt{J}(x) < \mathtt{J}(x_{\rm init})$ then
19             foreach $v \in \mathtt{pred}(\mathcal{G}, x)$ do
20                 if $\mathtt{parent}(v) = x$ then
21                     $\mathtt{J}(v) = \mathtt{c}(v, x) + \mathtt{J}(x)$;
22                 $B \leftarrow B \cup \{v\}$;
23                 $q.push\_back(v)$;
24     return $B$;
```

V. Theoretical Analysis

The main purpose of this section is to show that the proposed PI-RRT$^\#$ algorithm inherits the nice properties of the RRG and RRT* algorithms and thus it is asymptotically optimal, almost surely. Note that the result follows trivially if policy iteration is performed on all the vertices of the current graph $\mathcal{G}^k$ at the kth iteration of the PI-RRT$^\#$ algorithm. However, for the sake of numerical efficiency, we wish to preform policy improvement only on those vertices that have the potential of being part of the optimal solution, and only those. This will require a more detailed analysis, since we only have an estimate of this set of (so-called promising) vertices. The basic idea of the proof is based on the fact that each policy improvement progresses sequentially and computes best paths of length one, then of length two, then of length three, and so on. This allows us to keep track of all promising vertices that can be part of the optimal path of increasing lengths (e.g., increasing number of path edges) as we start from the goal vertex and move back towards the start vertex. The proof is rather long and thus is split in a sequence of several lemmas.

To this end, let $\mathcal{G}^k = (V^k, E^k)$ denote the graph at the end of the kth iteration of the PI-RRT$^\#$ algorithm. Given a vertex $x \in V^k$, let the control set $U^k(x)$ be divided into two disjoint sets $U^{k,*}(x)$ and $U^{k,\prime}(x)$ and let the successor set $\mathtt{succ}(\mathcal{G}^k, x)$ also be divided accordingly into two disjoint sets $S^{k,*}(x)$ and $S^{k,\prime}(x)$, as follows: $U^k(x) = U^{k,*}(x) \cup U^{k,\prime}(x)$ where $U^{k,*}(x) = \arg\min_{u \in U^k(x)} H(x, u, J^{k,*})$ and $U^{k,\prime}(x) = U^k(x) \setminus U^{k,*}(x)$ and $\mathtt{succ}(\mathcal{G}^k, x) = S^{k,*}(x) \cup S^{k,\prime}(x)$ where $S^{k,*}(x) = \{x^* \in V^k : \exists u^* \in U^{k,*}(x) \text{ s.t. } x^* = f(x, u^*)\}$ and $S^{k,\prime}(x) = \mathtt{succ}(\mathcal{G}^k, x) \setminus S^{k,*}(x)$. Let $\mathcal{M}^k$ denote the set of all policies at the end of the kth iteration of the PI-RRT$^\#$ algorithm, that is, let $\mathcal{M}^k = \{\mu : \mu(x) \in U^k(x), \forall x \in V^k\}$. Given a policy $\mu \in \mathcal{M}^k$ and an initial state x, let $\sigma^\mu(x)$ denote the path resulting from executing the policy μ starting at x. That is, $\sigma^\mu(x) = (x_0, x_1, \ldots, x_N)$ such that $\sigma_0^\mu(x) = x$ and $\sigma_N^\mu(x) \in \mathcal{X}_{\rm goal}$ for $N > 0$, where $\sigma_j^\mu(x)$ is the jth element of $\sigma^\mu(x)$. By definition, $x_{j+1} = f(x_j, \mu(x_j))$ for $j = 0, 1, \ldots, N-1$. Let now $\Sigma^k(x)$ be the set of all paths rooted at x and let $\Sigma^{k,*}(x)$ denote the set of all lowest-cost paths rooted at x that reach the goal region at the kth iteration of the algorithm, that is, Note that the set $\Sigma^{k,*}(x)$ may contain more than a single path. Finally, let $N^k(x)$ denote the shortest path length in $\Sigma^{k,*}(x)$, that is, $N^k(x) = \min_{\sigma^\mu(x) \in \Sigma^{k,*}(x)} \mathtt{len}(\sigma^\mu(x))$.

Let us define the following sets for a given policy $\mu^{k,i} \in \mathcal{M}^k$ and its corresponding value function $J_{\mu^{k,i}}$ at the end of the ith policy improvement step and at the kth iteration of the PI-RRT$^\#$ algorithm:

a) The set of vertices in $\mathcal{G}^k$ whose optimal cost value is less than that of $x_{\rm init}$, $V^k_{\rm prom} = \{x \in V^k : J^{k,*}(x) < J^{k,*}(x_{\rm init})\}$. This is the set of promising vertices.
b) The set of promising vertices in $\mathcal{G}^k$, whose optimal cost value is achieved by executing the policy $\mu^{k,i}$ at the ith policy iteration step $O^{k,i} = \{x \in V^k : J_{\mu^{k,i}}(x) = J^{k,*}(x) < J^{k,*}(x_{\rm init})\}$.
c) The set of vertices in $\mathcal{G}^k$ that can be connected to the goal region at iteration k with an optimal path of length less than or equal to ℓ $L^{k,\ell} = \{x \in V^k : \exists \sigma^\mu(x) \in \Sigma^{k,*}(x) \text{ s.t. } N^k(x) \leq \ell\}$.
d) The set of promising vertices in $\mathcal{G}^k$ that are connected to the goal region via optimal paths whose length is less than or equal to ℓ $P^{k,\ell} = L^{k,\ell} \cap V^k_{\rm prom}$.
e) The set of vertices that are selected for a Bellman update during the beginning of the ith policy improve-

ment $B^{k,i} = \{\overline{\texttt{pred}}(\mathcal{G}^k, x) : x \in V^k,\ J_{\mu^{k,i}}(x) < J_{\mu^{k,i}}(x_{\rm init})\}$.

Note from d) that the set of promising vertices that can be connected to the goal region via optimal paths whose length is exactly $\ell+1$ is given by $\partial P^{k,\ell} = P^{k,\ell+1} \setminus P^{k,\ell}$. It should also be clear from these definitions that $O^{k,i} \subseteq B^{k,i}$ for all $k = 1, 2, \ldots$ and $i = 0, 1, 2, \ldots$.

We have the following results, whose proofs are omitted for the sake of brevity, but can be found in [15] and [10].

Lemma 1 *The sequence $O^{k,i}$ generated by the policy iteration step of the PI-RRT$^{\#}$ algorithm is non-decreasing, that is, $O^{k,i} \subseteq O^{k,i+1}$ for all $i = 0, 1, \ldots$. Similarly, the sequence $L^{k,\ell}$ is non-decreasing, that is, $L^{k,\ell} \subseteq L^{k,\ell+1}$ for $\ell = 0, 1, \ldots$. Furthermore, for all $x \in \partial L^{k,\ell} = L^{k,\ell+1} \setminus L^{k,\ell}$, there exists $x^* \in L^{k,\ell} \cap S^{k,*}(x)$. Finally, the sequence $P^{k,\ell}$ is non-decreasing, that is, $P^{k,\ell} \subseteq P^{k,\ell+1}$ for $\ell = 0, 1, \ldots$. Furthermore, for all $x \in \partial P^{k,\ell} = P^{k,\ell+1} \setminus P^{k,\ell}$, there exists $x^* \in P^{k,\ell} \cap S^{k,*}(x)$.*

Lemma 2 *Let the policy $\mu^{k,i}$ and its corresponding cost function $J_{\mu^{k,i}}$, and assume that $P^{k,\ell} \subseteq O^{k,i}$. Then $\partial P^{k,\ell} \subseteq B^{k,i}$, which implies that $P^{k,\ell+1} \subseteq B^{k,i}$ before the beginning of the $(i+1)$th policy improvement step. Furthermore, $P^{k,\ell+1} \subseteq O^{k,i+1}$ after the $(i+1)$th policy improvement step.*

Lemma 3 *All vertices whose optimal cost value is less than that of $x_{\rm init}$, and which are part of an optimal path from $x_{\rm init}$ to $\mathcal{X}_{\rm goal}$ whose length is less than or equal to i, achieve their optimal cost value at the end of the ith policy improvement step, that is, $P^{k,i} \subseteq O^{k,i}$ for $i = 0, 1, \ldots$ when using policy $\mu^{k,i}$.*

Theorem 1 (Optimality of Each Iteration) *The optimal action and the optimal cost value for the initial vertex is achieved when the* `Replan` *procedure of the* PI-RRT$^{\#}$ *algorithm terminates after a finite number of policy improvement steps.*

Proof: (Sketch) This theorem is proven by using the fact that termination of the `Replan` procedure is guaranteed since the policy space is finite. The optimality result can easily be derived for the cases when the `Replan` procedure terminates by performing a number of policy improvement steps that happens to be more than the number of segments of the optimal path connecting $x_{\rm init}$ to the goal region. For earlier termination, it is shown that the optimal cost function value and policy for $x_{\rm init}$ had already been computed. ■

The previous theorem states that when the `Replan` procedure terminates at the beginning of the $N^k(x_{\rm init})$th policy improvement step, it has already computed the optimal action and cost function value for $x_{\rm init}$. So the `Replan` procedure always returns the optimal solution if it terminates after a finite number of iterations. The next theorem states that this is always the case.

Theorem 2 (Termination of `Replan` Procedure) *Let $\mathcal{G}^k = (V^k, E^k)$ be the graph built at the end of kth iteration of the* PI-RRT$^{\#}$ *algorithm. Then, the* `Replan` *procedure of the* PI-RRT$^{\#}$ *algorithm terminates after at most $(\overline{N}^k + 2)$ policy improvement steps, where $\overline{N}^k = \max_{x \in V^k_{\rm prom}} N^k(x)$ and $\overline{V}^k_{\rm prom} = \{\overline{\texttt{pred}}(\mathcal{G}^k, x) : x \in V^k_{\rm prom}\}$.*

Proof: (Sketch) This theorem is proven by using the fact that the sequence $B^{k,i}$ with respect to i has a limiting super set, i.e., the set of promising vertices and their predecessors. Then, it is shown that the termination condition of he `Replan` procedure holds when the optimal cost function value and policy for all vertices in the limiting set of $B^{k,i}$ are computed, which takes at most as many policy improvement steps as the number of segments of the longest optimal path connecting vertices in the limiting set of $B^{k,i}$ to the goal region. ■

Theorem 3 (Asymptotic Optimality of PI-RRT$^{\#}$) *Let $\mathcal{G}^k = (V^k, E^k)$ be the graph built at the end of the kth iteration of the PI-*RRT$^{\#}$ *algorithm and let N^k is maximum number of policy improvement steps performed at the k iteration. As $k \to \infty$, the policy $\mu^{k,N^k}(x_{\rm init})$ and its corresponding cost function $J_{\mu^{k,N^k}}(x_{\rm init})$, converge to the optimal policy $\mu^*(x_{\rm init})$ and corresponding optimal cost function $J_{\mu^*}(x_{\rm init})$ with probability one.*

Proof: The graph $\mathcal{G}^k = (V^k, E^k)$ is constructed by the RRG algorithm at the beginning of kth iteration. In the PI-RRT$^{\#}$ algorithm, the optimal cost function value of $x_{\rm init}$ with respect to $\mathcal{G}^k$ is computed during the `Replan` procedure at the end of kth iteration, that is, $J_{\mu^{k,N^k}}(x_{\rm init}) = J^{k,*}(x_{\rm init})$ and $\mu^{k,N^k}(x_{\rm init}) = \mu^{k,*}(x_{\rm init})$. Since the RRG algorithm is asymptotically optimal with probability one, $\mathcal{G}^k$ will encode, almost surely, the optimal path between $x_{\rm init}$ and goal region as $k \to \infty$. This implies that $J_{\mu^{k,N^k}}(x_{\rm init}) = J^{k,*}(x_{\rm init}) \to J^*(x_{\rm init})$ and $\mu^{k,N^k}(x_{\rm init}) = \mu^{k,*}(x_{\rm init}) \to \mu^*(x_{\rm init})$ with probability one. ■

VI. Numerical Simulations

We have implemented both the baseline RRT$^{\#}$ and PI-RRT$^{\#}$ algorithms in MATLAB and performed Monte Carlo simulations on shortest path planning problems in two different 2D environments. The initial and goal points are shown in Figure 1 in yellow and dark blue squares, respectively. The obstacles are shown in red and the best path computed during each iteration is shown in yellow.

The results were averaged over 100 trials and each trial was run for 10,000 iterations. We computed the total time required to complete a trial and measured the time spent on non-planning (sampling, extension, etc.) and planning related procedures of the algorithms, separately since the two algorithms should differ only in their planning part and we would like to confirm this. The growth of the tree is shown in Figure 1. At each iteration, a subset of promising vertices is determined during the policy evaluation step and policy improvement is performed only for these vertices. The promising vertices are shown in magenta in Figure 1.

The average time spent for non-planning (left) and planning (right) related procedures in the RRT$^{\#}$ and PI-RRT$^{\#}$ algorithms are shown in blue and red colors, respectively, in Figure 2. As seen in the left plot, PI-RRT$^{\#}$ is slightly

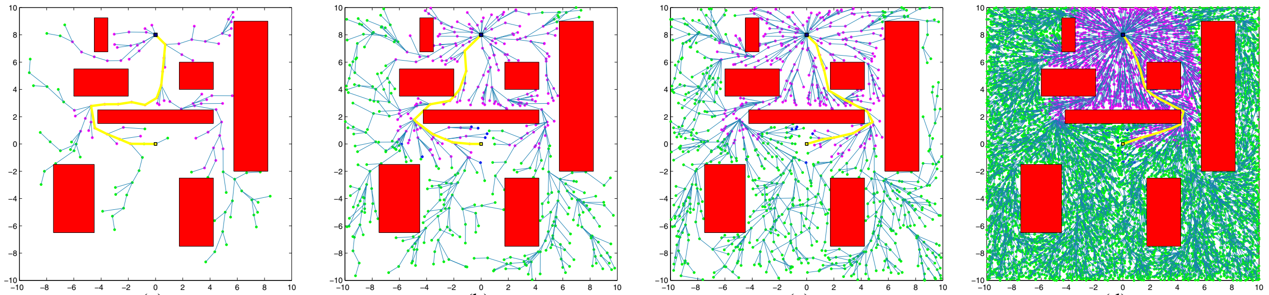
Fig. 1: The evolution of the tree computed by PI-RRT$^{\#}$ algorithm is shown in (a)-(d). The configuration of the trees (a), is at 200 iterations, (b) is at 600 iterations, (c) is at 1,000 iterations, and (d) is at 10,000 iterations.

faster than RRT$^{\#}$, especially when adding a new vertex to the graph. Since there is no priority queue in the PI-RRT$^{\#}$ algorithm, it is much cheaper to include a new vertex, and there is no need for ordering. As seen in the right figure, the relation between time and iteration is linear when the number of iterations becomes large in the log-log scale plot, which implies a polynomial relationship, i.e., $t(n) = cn$, where $t(n)$ is the time required to complete n iterations. The fitted time-iteration lines (dashed) for RRT$^{\#}$ and PI-RRT$^{\#}$ are shown in magenta and green, respectively. In our implementation, one processor was used to perform policy improvement due to simplicity. However, policy improvement step can be done in parallel. One can then divide the set of promising vertices into disjoint sets and assign each of the subsets to a different processor.

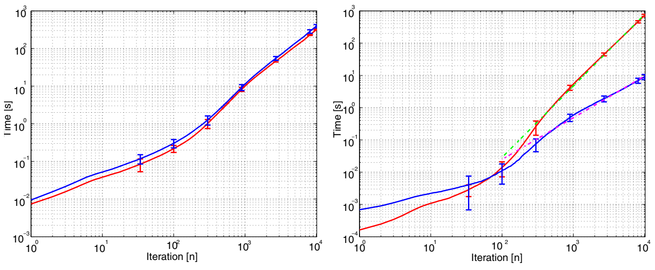

Fig. 2: The time required for non-planning (left) and planning (right) procedures to complete a certain number of iterations for the first problem set. The time curve for RRT$^{\#}$ and PI-RRT$^{\#}$ are shown in blue and red, respectively. Vertical bars denote standard deviation averaged over 100 trials.

VII. Conclusion

We show that a connection between DP and RRGs may yield sampling-based motion planning algorithms that utilize ideas from dynamic programming. These algorithms ensure asymptotic optimality (with probability one) as the number of samples tends to infinity. Use of policy iteration, instead of value iteration during the exploitation step, may offer several advantages, such as completely parallel implementation, avoidance of sorting and maintaining a queue of all sampled vertices in the graph, etc. We have implemented these ideas in the replanning step of the RRT$^{\#}$ algorithm. The proposed PI-RRT$^{\#}$ algorithm can be massively parallelized, which can be exploited by leveraging the recent advances in the computational power of GPUs. This is part of ongoing work.

Acknowledgement: Support for this work has been provided by ARO MURI award W911NF-11-1-0046 and ONR award N00014-13-1-0563.

References

[1] J. H. Reif, "Complexity of the movers problem and generalizations extended abstract," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 421–427, 1979.

[2] S. M. Lavalle and K. J. J., "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2001.

[3] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research (IJRR)*, vol. 20, no. 5, pp. 378–400, 2001.

[4] S. M. LaValle, *Planning Algorithms*. New York: Cambridge University Press, 2006.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[6] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *IEEE International Conference on Robotics and Automation*, (Karlsrühe, Germany), pp. 2413–2420, May 6–10 2013.

[7] O. Arslan and P. Tsiotras, "Dynamic programming guided exploration for sampling-based motion planning algorithms," in *IEEE International Conference on Robotics and Automation*, (Seattle, WA), pp. 4819–4826, May 26–29 2015.

[8] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International Journal of Robotics Research*, pp. 883–921, 2015.

[9] M. Otte and E. Frazzoli, "RRT$^{\text{X}}$: Real-time motion planning/replanning for environments with unpredictable obstacles," in *Algorithmic Foundations of Robotics XI*, pp. 461–478, Springer, 2015.

[10] O. Arslan, *Machine Learning and Dynamic Programming Algorithms for Motion Planning and Control*. PhD Thesis, Georgia Institute of Technology, 2015.

[11] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *IEEE International Conference on Robotics and Automation*, (Seattle, WA), pp. 867–875, May 26–29 2015.

[12] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific Belmont, MA, 2000.

[13] D. Bertsekas, *Abstract Dynamic Programming*. Belmont, Massachusetts: Athena Scientific, 2013.

[14] M. Otte and N. Correll, "C-FOREST: Parallel shortest-path planning with super linear speedup," *IEEE Transactions on Robotics*, vol. 29, pp. 798–806, June 2013.

[15] O. Arslan and P. Tsiotras, "Incremental sampling-based motion planners using policy iteration methods," 2016. http://arxiv.org/abs/1609.05960.