Dynamic Programming Principles for Sampling-based Motion Planners

Oktay Arslan and **Panagiotis Tsiotras**
Institute for Robotics and Intelligent Machines
Georgia Institute of Technology
Atlanta, Georgia 30332

Abstract

Recent randomized asymptotically optimal motion planners incrementally construct an approximate, discrete, sparse representation of the underlying continuous search space, and use locally optimal solutions, along with a local reassignment of the edges, to obtain optimal solutions encoded in the current graph/tree data structure. Careful analysis reveals that all these algorithms can be interpreted as implementing asynchronous dynamic programming (DP) iterations in the underlying graph. Based on this key insight, we utilize existing DP strategies, such as value iteration (VI) or policy iteration (PI) to solve problems arising in randomized sampled-based motion planning. In this work, we show how DP algorithms can be utilized in the framework of sampling-based algorithms. This connection yields different types of algorithms which have asymptotic optimality guarantees, faster convergence rates to the optimal solution and lend themselves readily to different execution models (sequential, parallel).

Introduction

Sampling-based planners, notably PRM and RRT have proven to be successful in solving many high-dimensional real-world planning problems. These methods are simple to implement, require less memory, provide feasible solutions quickly, and work efficiently in high-dimensional problem, but come with a relaxed notion of completeness, i.e., resolution-completeness. Recently, asymptotically optimal variants, such as the PRM^* and RRT^* algorithms (Karaman and Frazzoli 2011) have been proposed, which aim to equip these algorithms with certain performance guarantees. The RRT^* algorithm, performs a "local rewiring step" after a new vertex is included in the graph in order to improve the current best solution locally. In our previous work (Arslan and Tsiotras 2013) we showed that this rewiring step amounts to locally performing Bellman updates in a neighborhood of the newly included vertex. Furthermore, careful analysis reveals that this rewiring step can be replaced by a different implementation of value iteration in order to achieve faster convergence rates. Based on these insights, a new sampling-based motion planning algorithm, called $\mathrm{RRT}^{\#}$, has been proposed in (Arslan and Tsiotras 2013; Arslan and Tsiotras 2015), which makes novel a connection between random graphs and dynamic programming algorithms.

It is well-known that dynamic programming (DP) provides a simple and intuitive characterization of optimal solutions for sequential decision making problems. Well-studied algorithms are available for solving DP problems, for example, value iteration (VI) or policy iteration (PI). However, these DP algorithms perform more efficiently on discrete state and control spaces, which prevents their application directly on continuous-space/continuous-control robot motion planning problem. On the other hand, a sampling-based algorithm, such as Rapidly-exploring Random Graph (RRG) provides *non-uniform* discretizations of the search space that can be utilized by a DP-like algorithm to extract optimal solutions. Furthermore, owing to the anytime flavor of many DP algorithms, one can guide the graph construction towards favorable regions of the state space, based on the existing solution computed so far. By doing so, one can improve the memory requirements by avoiding exploration of the whole state space, focusing instead on the relevant region at any given query.

From DP to Sampling-based Motion Planners

Dynamic programming provides a powerful framework to solve many types of sequential decision making problems (Bertsekas 2000). It is based on the fact that the optimal cost function

$$J^*(x) = \inf_{\pi \in \Pi} J_\pi(x), \quad x \in X,$$

over the policy space Π consisting of sequences $\{\mu_0, \mu_1, \dots\}$ where $\mu_k : X \mapsto U$ satisfies the *Bellman equation*

$$J^*(x) = \inf_{u \in U(x)} \Big\{ g(x,u) + \alpha J^*(f(x,u)) \Big\}, \tag{1}$$

for all $x \in X$, where $x_{k+1} = f(x_k, u_k)$ is the dynamic model of the process, and $J_\pi(x_0) = \sum_{k=0}^{\infty} \alpha^k g(x_k, \mu_k(x_k))$, is the cost of the policy $\pi = \{\mu_0, \mu_1, \dots\}$, starting from x_0. Once Eq. (1) is solved, the optimal policy μ^* can be obtained from

$$\mu^*(x) \in \arg\min_{u \in U(x)} \Big\{ g(x,u) + \alpha J^*(f(x,u)) \Big\}. \tag{2}$$

for all $x \in X$. One can rewrite Eqs. (1) and (2) succinctly using By defining the operator T_μ

$$(T_\mu J)(x) = H(x, \mu(x), J), \quad x \in X,$$

we can express the Bellman operator T

$$(TJ)(x) = \inf_{\mu \in \mathcal{M}} (T_\mu J)(x), \quad x \in X.$$

and expressing Bellman's equation (1) and the optimality condition (2) in a compact form as

$$J^* = TJ^*, \qquad T_{\mu^*} J^* = TJ^*.$$

There are two prevailing types of DP algorithms for computing the optimal policy μ^* and its cost function J^*.

Value Iteration (VI): This algorithm computes J^* by relaxing Eq. (1), starting from some J^0, and generating the sequence $\{T^k J^0\}$ via $J^{k+1} = TJ^k$. Therefore, this method is an indirect way computing the optimal policy μ^* by using the information of its optimal cost function J^*. The generated sequence converges to the optimal cost function due to contraction property of the Bellman operator T.

Policy Iteration (PI): The algorithm starts with an initial policy μ^0 and generates a sequence of policies μ^k by performing Bellman updates in two steps as follows.

i) **Policy evaluation**: computing J_{μ^k} as the unique solution of the equation $J_{\mu^k} = T_{\mu^k} J_{\mu^k}$.

ii) **Policy improvement**: computing a policy μ^{k+1} that satisfies $T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}$.

Often, instead of solving the system of linear of equations exactly in the policy evaluation step, one may perform an approximate evaluation of the current policy and use this information in the subsequent policy improvement step. This variant is called *Optimistic Policy Iteration (PI)* (Bertsekas 2000). Policy iteration may be advantageous over value iteration for some problems, especially because it usually converges in fewer iterations than value iteration. Moreover, the policy improvement step can be computed fast by massive parallelization on Graphics Processing Units (GPUs).

Random Graphs and DP Algorithms

Let $\mathcal{G} = (V, E)$ denote the graph constructed by the RRG algorithm, or one of its variants, at some iteration, where V and $E \subseteq V \times V$ are finite sets of vertices and edges, respectively. We assume a graph having a finite number of vertices and a positive cost $\mathtt{c}(x, x')$ associated with each directed edge (x, x'). Once we efficiently represent the obstacle-free space $\mathcal{X}_{\text{free}}$ by a graph $\mathcal{G}$, the robot motion planning problem transforms into a shortest-path problem on a graph where we can utilize DP algorithms to obtain optimal policies. By letting the dynamic model $x_{k+1} = f(x_k, u_k) = x_k + u_k$, cost function $g(x, u) = \mathtt{c}(x, x+u)$ and control space $U = \cup_{x \in V} U(x)$ where $U(x) = \{u : u = x' - x, \ \forall\, x' \text{ s.t. } (x, x') \in E\}$, we obtain the Bellman equation

$$J^*(x) = \min_{u \in U(x)} \{\mathtt{c}(x, x+u) + J^*(x+u)\}. \qquad (3)$$

Since the the graph computed by the RRG algorithm is a connected graph by construction (Karaman and Frazzoli 2011) and all edge cost values are positive (which implies that the costs of all its cycles are positive) convergence of DP algorithms is guaranteed, and the resulted optimal policy μ is proper.

A sampling-based motion planner that utilizes the VI algorithm, namely the RRT$^\#$ algorithm, has been proposed in (Arslan and Tsiotras 2013). The RRT$^\#$ algorithm implements the Gauss-Seidel version of VI and provides a sequential, asynchronous implementation for solving Bellman's equation on the current graph. Following up on this idea, we propose in this paper a sampling-based algorithm that utilizes PI to solve equation (3). The new algorithm, called PI-RRT$^\#$, is better-suited for massive parallelization, since the policy improvement step can be simultaneously implemented for each vertex.

The details of the PI-RRT$^\#$ implementation are given as follows. Let $\mathcal{G}^n = (V^n, E^n)$ denote the graph constructed after calling the `Extend` procedure at nth iteration (see (Arslan and Tsiotras 2013) for the RRT$^\#$ procedures). The graph $\mathcal{G}^n$ is rooted at the goal region and grows incrementally toward the initial state. Whenever a new vertex is included in the `Extend` procedure, the control from the corresponding state is initialized as $\mu^{n,0}(x_{\text{new}}) = \arg\min_{x \in \mathcal{X}_{\text{near}}} \{\mathtt{c}(x_{\text{new}}, x) + J_{\mu^{n,0}}(x)\}$. The `Replan` procedure is subsequently called to compute the optimal policy for the new graph by using a somewhat modified version of the PI algorithm for DP with initial policy $\mu^{n,0}$. Instead of updating the policy for all vertices, for the existing policy $\mu^{n,k}$, the modified PI algorithm computes a subset of vertices $S^{n,k} = \{\{x, \mathtt{pred}(\mathcal{G}^n, x)\} : x \in V^n \text{ and } \mathtt{h}(x_{\text{init}}, x) + J_{\mu^{n,k}}(x) \leq J_{\mu^{n,k}}(x_{\text{init}})\}$ during each policy evaluation step, where $\mathtt{h}$ is some heuristic, and performs policy improvement only for the vertices in $S^{n,k}$. Note that when the initial state has not been included to the graph, i.e., no feasible solution has been computed yet, the cost value of x_{init} is infinite, which implies that $S^{n,k} = V^n$, and hence policy evaluation and improvement steps are performed for all vertices, as done in the standard PI implementation. Once a feasible path is computed, the set of promising vertices $S^{n,k}$ is computed and the existing policy is update for vertices of $S^{n,k}$. The `Replan` procedure terminates when the policy becomes stationary for all vertices in $S^{n,k}$.

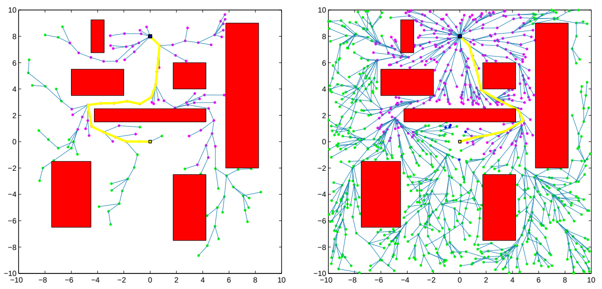

Figure 1: The evolution of the tree computed by the PI-based PI-RRT$^\#$ algorithm. The lowest-cost path is shown in yellow and the set of promising vertices is shown in magenta.

References

[Arslan and Tsiotras 2013] Arslan, O., and Tsiotras, P. 2013. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *IEEE International Conference on Robotics and Automation*, 2413–2420.

[Arslan and Tsiotras 2015] Arslan, O., and Tsiotras, P. 2015. Dynamic programming guided exploration for sampling-based motion planning algorithms. In *IEEE International Conference on Robotics and Automation*.

[Bertsekas 2000] Bertsekas, D. P. 2000. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific.

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