Relating Reinforcement Learning to Dynamic Programming-Based Planning

Filip V. Georgiev[1], Kalle G. Timperi[1], Başak Sakçak[1,2], Steven M. LaValle[1⋆]

[1] Center for Applied Computing, Faculty of Information Technology and Electrical Engineering, University of Oulu, Finland
[2] Dept. of Advanced Computing Sciences, Maastricht University, the Netherlands

Abstract. This paper bridges some of the gap between optimal planning and reinforcement learning (RL), both of which share roots in dynamic programming (DP) applied to sequential decision making or optimal control. Whereas planning typically favors deterministic models, goal termination, and cost minimization, RL tends to favor stochastic models, infinite-horizon discounting, and reward maximization in addition to learning-related parameters such as the learning rate and greediness factor. A derandomized version of RL is developed, analyzed, and implemented to yield performance comparisons with value iteration and Dijkstra's algorithm using simple planning models. Next, mathematical analysis shows: 1) conditions under which cost minimization and reward maximization are equivalent, 2) conditions for equivalence of single-shot goal termination and infinite-horizon episodic learning, and 3) conditions under which discounting causes goal achievement to fail. The paper then advocates for defining and optimizing *truecost*, rather than inserting arbitrary parameters to guide operations. Performance studies are extended to the stochastic case, using planning-oriented criteria and comparing value iteration to RL with learning rates and greediness factors.

Keywords: planning algorithms, motion planning, reinforcement learning, Q-learning, value iteration, dynamic programming

1 Introduction

Bellman introduced the principle of DP in the 1950s as a way to efficiently compute solutions to sequential decision-making problems [3]. The core idea is the *principle of optimality*, a simple observation that optimal solutions must be composed of optimal parts, thereby leading to a powerful recurrence relation or differential equation that underlies the Hamilton-Jacobi-Bellman (HJB) equation in continuous-time optimal control, value and policy iteration methods for discrete-time problems, and is useful for algorithm design and complexity analysis well beyond sequential decision making. In the context of motion planning,

⋆ This work was supported by a European Research Council Advanced Grant (ERC AdG, ILLUSIVE: Foundations of Perception Engineering, 101020977, Academy of Finland BANG! 363637). (e-mail: firstname.lastname@oulu.fi).

Dijkstra's algorithm, A*-search, D* [14,25], and many others compute optimal plans over grids or more general graphs. Dijkstra-like planning algorithms, built upon value iteration, compute approximately optimal feedback plans over continuous spaces using sampling and interpolation of value functions [31]. All of these methods extend gracefully to settings with either nondeterministic (possibilistic) or stochastic (probabilistic) prediction uncertainty [15].

In the 1990s, Barto and Sutton introduced a powerful paradigm that cleverly interleaves the discovery of the effect of actions with the calculations of optimal values and plans [26]. In preceding works, experimental data was first used to learn or estimate a model of the form $x' = f(x, u)$ for states x, x' and action u (or $p(x' \mid x, u)$ in a stochastic setting), and then the model was used as a module in DP methods. The new paradigm, called *reinforcement learning (RL)*, retains all of the structure of Bellman's original DP equations, while inserting learning or discovery directly into the algorithms for computing optimal plans (or policies). Thus, RL is a fusion of optimal sequential decision-making (control, planning) with machine learning (system estimation, identification).

In recent years, RL has become ubiquitous as an approach to problems in robotics and embodied AI [13,27]. As it is applied to motion planning, it has been useful as a way to generate motion/control primitives to complement planners (e.g., [8]); however, the relationships across the Bellman-inspired family of methods have become increasingly opaque. In non-RL works, a clear cost model is formulated in terms of engineering resources, such as time or energy expended, and plans are selected to globally optimize cumulative cost. In standard RL, a reward model is defined that is biologically inspired and is often shaped or tuned to yield desired outcomes [20]. Furthermore, RL tends to be formulated as an infinite-horizon problem in which the most preferred way to make cumulative rewards finite is via an arbitrary discount factor that has little meaning in an engineering context (notable exceptions exist [19]). In non-RL approaches to planning, the episode usually ends when goals are achieved; even though the number of steps may be unbounded, it is finite and can be efficiently handled by termination actions [15] or terminal states [7,21], thereby preserving the true cost model. Finally, RL is formulated exclusively in a stochastic setting, whereas for non-RL methods there is a simple and clear relationship between corresponding algorithms in deterministic and stochastic settings. Probabilistic modeling is not the 'only' or 'most general' way to handle uncertainties (e.g., [12]).

To address these concerns, we investigate a series of methods along the spectrum from deterministic planning to stochastic RL (to Q-learning [30]). Relationships are established through both theoretical analysis and experimental studies. To this end, this paper is closely related to the previous attempts on unifying RL and DP concepts [6,5]. However, in this work, we focus on the interplay between the cost/reward formulations, analyze the effects of randomization and other practices used in RL on the solution through its relation to DP. Section 2 starts with optimal deterministic planning and develops a comparable derandomized analog of RL that clarifies the issues of whether a model is given and whether one can jump at will between arbitrary states. Computation and convergence

rates are compared over several examples. Section 3 bridges the gap between typical planning and RL formulations through rigorous mathematical analysis of their relationship, with an emphasis on using physically motivated costs in models, as opposed to arbitrary discounts and rewards. Section 4 extends the studies to stochastic state-transition models, which are prevalent for RL. Section 5 summarizes the main conclusions and remaining challenges.

2 Optimal Planning and Deterministic RL

2.1 Optimal planning concepts

Consider a standard optimal planning problem expressed as a six-tuple $\mathcal{P} = (X, U, f, x_1, X_G, \ell)$, in which X is a finite *state space*, U is a finite *action space*, $f : X \times U \to X$ is the *state transition function*, $x_I \in X$ is the *initial state*, $X_G \subset X$ is the *goal set*, $\ell : X \times U \to [0, \infty)$ is the *cost function*. Actually, f is allowed to be a partial function, in which $U(x) \subseteq U$ denotes the actions available from each $x \in X$. A *feedback plan, or policy* is a mapping $\pi : X \to U$, and we use the shorthand $f_\pi(x) := f(x, \pi(x))$ to indicate the next state obtained by applying policy π at state x. A policy π is evaluated from any $x_1 \in X$ using the stage-additive *cost functional*

$$L(\pi, x_1) = \sum_{k=1}^{\infty} \ell(x_k, \pi(x_k)), \tag{1}$$

in which $k \in \mathbb{N}$ indexes over *stages*, and $x_{k+1} = f_\pi(x_k)$. Note that different cost functionals are possible and will be introduced later. To take X_G into account, a special *termination action* $u_T \in U(x_g)$ exists from every $x_g \in X_G$, and when applied at stage k, then $x_{i+1} = x_i = x_g$ for all all $i \geq k$ and $l(x_g, u_T) = 0$. This ensures that (1) is finite if the goal is reachable from x_1, in spite of having an infinite horizon, that is, infinite number of stages. A plan or policy π^* is *optimal* if for all other π and all $x_1 \in X$, $L(\pi^*, x_1) \leq L(\pi, x_1)$. Let $G_\pi : X \to [0, \infty)$ be the *cost-to-go* function for a plan π, in which each $G_\pi(x)$ is the total cost obtained in (1) by applying $\pi(x)$ from state $x_1 = x$. Let $G^* = G_{\pi^*}$ be the *optimal cost to go*. Determining G^* and π^* are closely related because:

$$\pi^*(x) \in \operatorname*{arg\,min}_{u \in U(x)} \{\ell(x, u) + G^*(x')\}, \tag{2}$$

in which $x' = f(x, u)$ (see [15], Ch. 2). In the case of classical planning, the model is known; thus, π^* and G^* can be computed using value iteration or by Dijkstra's algorithm, in which the latter can be seen an efficient version of value iteration. Whereas Dijkstra's algorithm is typically faster than value iteration, value iteration generalizes more easily to stochastic planning and RL. Value iteration proceeds by successively computing cost-to-go functions for each stage, proceeding backwards through time, until the values stabilize (they do not change). For each $x_k \in X$, the value is calculated as

$$G_k^*(x_k) = \min_{u_k \in U(x_k)} \Big\{ \ell(x_k, u_k) + G_{k+1}^*(x_{k+1}) \Big\}, \tag{3}$$

in which $x_{k+1} = f(x_k, u_k)$. Most RL algorithms essentially create an asynchronous, step-by-step version of (3), in which each step processes a state-action pair (x_k, u_k).

2.2 Handling imperfect models by derandomizing RL

Whereas most RL algorithms are designed considering imperfect models, planning algorithms typically assume a complete model. Therefore, our first task is to carefully relate and compare these. Two critical options emerge: 1) Is the model given to the algorithm? Suppose that in the *model-based* case, $\mathcal{P}$ is completely given to the algorithm. In the *model-free* case, $\mathcal{P}$ is known 'to the gods' but not the algorithm. The robot can only try actions and then sense the next state and incurred cost. 2) Can the robot freely 'jump' between states for planning purposes? *Virtual jumping* occurs in planning by using the model: The algorithm deduces what would happen if action u were applied from state x. By default, we focus on *physical jumping*, in which the robot is moved to a new state.

There are three cases based on the critical options: 1) **Model-based.** If the model is available, then, virtual jumping can be used, allowing algorithms such as Dijkstra or value iteration to solve the optimal planning problem before the robot is ever moved. 2) **Model-free with jumping**. If physical jumping is free and X were known, then the model can be completely discovered by trying every $u \in U(x)$ from every $x \in X$. Suppose that even X is unknown. In this case, we assume sensing is sufficient so that states can be uniquely labeled as they are discovered, and returns to previous states are correctly determined. Even in this extreme scenario, Dijkstra or value iteration could be applied to derive the optimal plan. The difference compared to the first case is that here, jumping must be physical. (If a cost is assigned for physical jumping, then an intriguing new optimization problem emerges.) 3) **Model-free without jumping.** This case most closely matches common RL formulations. The robot must move along one long path to discover an optimal plan. (The directed transition graph underlying $\mathcal{P}$ is required to have every state reachable from every other.)

Note that since the system is deterministic, every state-action pair needs to be considered only once to completely discover f; however, revisits may be needed to fully compute G^*. Furthermore, f can be represented as a directed multigraph with vertices X and labeled edges $U(x)$ for all $x \in X$. This leads to a simple two-phase algorithm: 1) First, 'physically' explore enough to discover the entire graph (jumping not allowed!); several efficient algorithms exist [1,9]. 2) Then, run Dijkstra's algorithm or value iteration to calculate G^* and π^*. We call the resulting algorithms *Model-free Dijkstra* and *Model-free Value Iteration*, respectively. This, however, does not follow the spirit of most RL, which interleaves learning and optimal planning during the entire execution. To move closer to this approach, we introduce a derandomized version of Q-learning, which serves as a gateway between typical planning and RL scenarios.

Let $Q_\pi : X \times U \to \mathbb{R}$ be the *Q-value function*, obtained from (1) by applying u_1 at x_1, and then applying actions $u_i = \pi(x_i)$ for all $i > 1$. Let $Q^* = Q_{\pi^*}$ be

the *optimal Q-value function*. Note that G^* can be computed from Q^* by

$$G^*(x) = \min_{u \in U(x)} \Big\{ Q^*(x,u) \Big\}. \tag{4}$$

Q-learning estimates Q^* through the process of stochastic iteration [26,22]. Using a stochastic state transition model $p(x_{k+1}|x_k, u_k)$ (addressed in Section 4), the estimate $\hat{Q}^*$ of Q^*, is determined iteratively by

$$\hat{Q}^*(x,u) \leftarrow (1-\rho)\hat{Q}^*(x,u) + \rho \left(\ell(x,u) + \min_{u' \in U(x')} \Big\{ \hat{Q}^*(x',u') \Big\} \right), \tag{5}$$

in which x' is obtained by applying u from x in a single step, $\rho \in (0,1)$ is the *learning rate*, and $\leftarrow$ denotes an assignment (see [15], Sec. 10.4). Note that jumping is not required: The minimization over all $u' \in U(x')$ involves looking up internally stored Q-values. The parameter ρ is chosen to be smaller if the amount of uncertainty or instability is higher. If every reachable state-action pair is visited infinitely often, then $\hat{Q}^*$ converges to Q^* in probability [5,29].

In the limiting case of a deterministic system, there is no uncertainty, which would suggest setting $\rho = 1$ to obtain our proposed *derandomized Q-learning*:

$$\hat{Q}^*(x,u) \leftarrow \ell(x,u) + \min_{u' \in U(x')} \Big\{ \hat{Q}^*(x',u') \Big\}. \tag{6}$$

Q-learning is considered *off-policy* in the sense that no particular plan of movement for exploration is assumed, as long as every state-action pair is visited infinitely often. In practice, however, a movement plan is needed. To ensure all state-action pairs are visited infinitely often, two *pure exploration* plans are considered: 1) apply random actions, leading to probabilistic convergence, and 2) apply a universal plan from [28] to obtain deterministic convergence.

Proposition 1. *If every state-action pair (x,u) is visited infinitely often for all $x \in X$ and $u \in U(x)$, and (6) is applied in every step, then after a finite number of iterations, $\hat{Q}^* = Q^*$.*

Proof. Let $Z = X \times U$ be an augmented state space and let $f_Z : Z \times U \to Z$ be the state transition function defined as $f_Z((x,u),u') = (f(x,u),u')$, in which f is the state transition function for the original problem. Rewriting (6) in Z yields $\hat{Q}^*(z) \leftarrow \ell(z) + \min_{u' \in U(f(x,u))}\{\hat{Q}^*(z')\}$, in which $z' = f_Z(z,u')$. Because in Q-learning the values are updated when x is visited and u is applied, this can be seen as a step in an asynchronous value iteration with single state updates in contrast with value iteration which does a complete sweep of all the states. Then, the finite time convergence to the optimal values follows from the finite time convergence of asynchronous value iteration for deterministic shortest-path problems [4].

As is common in RL, we balance exploration vs. exploitation in the form of an *ϵ-greedy* plan/policy. For some fixed value $\epsilon \in [0,1]$, the policy applies the next action in the pure exploration plan with probability ϵ; otherwise, it applies

the minimizing u' in (6). Thus, it is 'greedy' (exploitation) $1 - \epsilon$ of the time on average. Using ϵ-decay was tested, but excluded; see the appendix.

We follow another common RL practice, which is to allow periodic jumping back to the initial state. Thus, the execution is divided into *episodes*, in which each one corresponds to applying an action sequence from the initial state until either X_G or a maximum iteration limit is reached (to escape greediness-induced traps). This issue does not arise in value iteration because the state becomes frozen once X_G is reached and u_T is applied. The relationship between these two settings is analyzed in Section 3.3.

2.3 Computational comparisons

We compared Q-learning and the model-free versions of Dijkstra's algorithm, value iteration, and asynchronous value iteration (see Sec. 2.2), over several planning problems encoded as grids with standard 4-neighbors. Higher dimensions and continuous problems are avoided here because our aim is to improve understanding and we believe the key issues identified in our studies extend to higher dimensional lattices [16] and continuous problems involving DP with interpolation [17]. The experiments were conducted using Python[3]. They were run on a desktop computer with a Ryzen 7 3700X CPU (3.6GHz)[4]. Figure 1 shows a representative sample of the problems we investigated; many more results appear in the appendix.

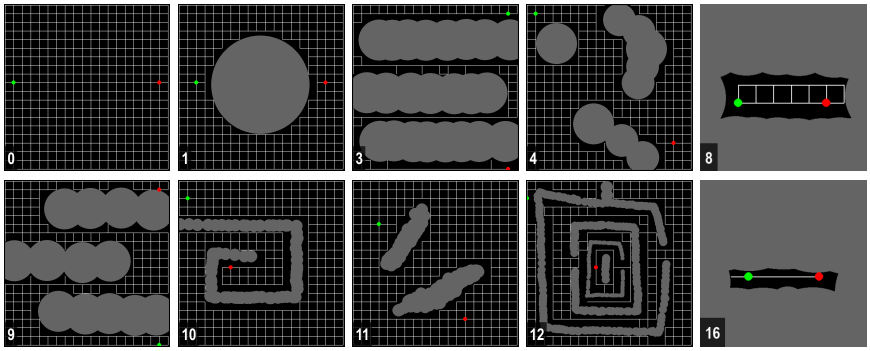

Fig. 1. Ten representative problems (enumerated from top-left to bottom-right as 0, 1, 3, 4, 8, 9, 10, 11, 12, 16). The green dot is the initial state. The red dot is the goal state. The white segments connect the states and can be viewed as the actions allowed by the robot. All actions have the same step cost. The gray areas are obstacles.

Run times are shown, which closely correlate with number of actions taken (these appear in the appendix). We ran each algorithm 100 times and report the mean and standard deviation. Unless otherwise stated, the number of episodes used for Q-learning is set to 1000, and the number of steps per episode is set to

[3] The codebase can be found here.

[4] A computer with an i5-12400F CPU (2.50GHz) was used for some of the stochastic experiments ($\gamma \leq 0.9$).

3000 to be sufficiently longer than any optimal path. These were chosen through trial and error to optimize Q-learning performance. Each run was terminated either after convergence or an iteration limit was reached. We considered three questions: (1) Was a path from the initial state to a goal state found? (2) Was the path found optimal (is the cost-to-go value from the initial state optimal)? (3) Is the cost-to-go value for each state in the state space optimal?

The first question corresponds to a feasible planning solution: An action sequence that leads to the goal. Figure 2 shows how varying ϵ affects the time in which the goal state is first discovered. When all state-action pairs (except the goal) are initialized to the same value, a greedy action would be chosen arbitrarily (depending on which action is first in its enumeration) because all neighboring values are equal. Later, as the cost of any action that has been chosen in previous iterations increases relative to the other available actions, unexplored actions would be favored. This happens until the goal is discovered and better values are propagated backwards, causing actions making progress towards a lower cost solution to be preferred. For problems in which there are less states and more obstacles (e.g., Problem 12), the goal is discovered more quickly with smaller values for ϵ. For Problem 0, since the whole state space is available and the goal is relatively far away, a purely greedy approach ($\epsilon = 0$) takes more time to reach it. However, a random policy ($\epsilon = 1$) is quicker because the robot is not hindered by any obstacles.

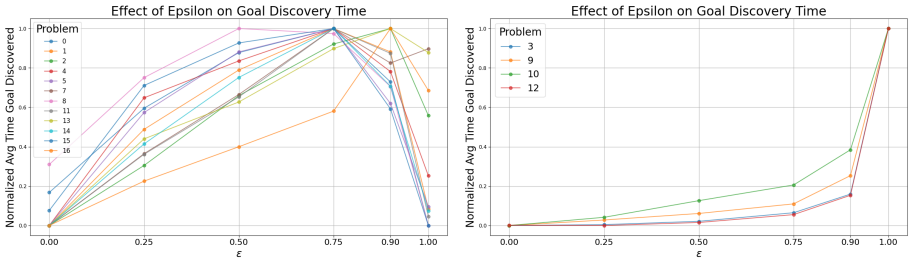

Fig. 2. The effect of decreasing greediness in a policy when applied on different problems.

To address questions (2) and (3) above, we define convergence for Q-learning as convergence to the optimal cost-to-go values computed by a model-based value iteration. Note that Q^* can be obtained from G^* using (4) for this comparison. Table 1 shows the performance of the model-free versions of Dijkstra's algorithm and value iteration, and that of Q-learning with varying ϵ for a representative problem. We also include a deterministic version of Q-learning, labeled as '(π)', that uses digits of $\pi = 3.14\ldots$ base four, to select actions [28]. We show the average run time of the algorithm, how often the values for the whole state space converge (frequency of convergence), and the time it takes for the value of the initial state to converge.

Unsurprisingly, the model-free Dijkstra algorithm has the fastest run time, followed by model-free asynchronous and then synchronous value iteration (these run times include graph exploration). The fastest version of Q-learning is purely

Table 1. Performance of Q-learning and model-free approaches on a deterministic problem (Problem 10). Value iteration shortened to VI.

Algorithm (Problem 10)	Run time (mean ± std)	Convergence	Optimal Initial Cost-to-Go Time (mean ± std)
Q-learning ($\epsilon = 0$)	0.33383 ± 0.0025	0.0 %	0.1720 ± 0.0016
Q-learning ($\epsilon = 0.25$)	0.44088 ± 0.0030	0.0 %	0.2043 ± 0.0041
Q-learning ($\epsilon = 0.5$)	0.64394 ± 0.0045	0.0 %	0.2517 ± 0.0066
Q-learning ($\epsilon = 0.75$)	1.08726 ± 0.2099	76.0 %	0.3502 ± 0.0129
Q-learning ($\epsilon = 0.9$)	0.61889 ± 0.0559	100.0 %	0.4858 ± 0.0239
Q-learning ($\epsilon = 1$)	1.88356 ± 0.6740	100.0 %	1.2836 ± 0.3801
Q-learning (π) ($\epsilon = 1$)	2.02001 ± 0.4996	100.0 %	1.3425 ± 0.2770
Model-free Dijkstra	0.00248 ± 0.0006	N/A	N/A
Model-free Asynchronous VI	0.04920 ± 0.0004	N/A	N/A
Model-free VI	0.05585 ± 0.0007	N/A	N/A

greedy ($\epsilon = 0$). As a result, we can see that the more randomness we introduce in the form of increasing ϵ, the slower the algorithm runs. However, there is a reverse relationship on the convergence rate: lower ϵ decreases the frequency of convergence. This aligns with intuition because less exploration will prevent states from being visited sufficiently often with high probability. However, as ϵ increases, both the convergence frequency and run time increase. Interestingly, for lower ϵ values, the optimal cost-to-go value from the initial state converges faster. Based on this observation, if the objective is to simply find an optimal path between the initial and the goal states when the system is deterministic, picking $\epsilon = 0$ is often the best choice.

The model-free version of Dijkstra's algorithm is on average 134.65 times faster and uses 22.88 times fewer actions than Q-learning with $\epsilon = 0$. If the objective is not only to find an optimal path from the initial state, but to find the optimal cost-to-go values for all states, we choose $\epsilon = 0.9$, since this is the value for which Q-learning converges consistently with the lowest run time. In this case, model-free Dijkstra's algorithm is 249.62 times faster and uses 46.50 times fewer actions, which is an even bigger difference in performance. We observed that the relative performance among the different Q-learning variants and the model-free approaches is consistent across all problems studied, although the magnitude of the performance differences varies.

3 Analysis of Cost/Reward Models

This section brings the planning-oriented model of Section 2 closer to the models typically used in RL, which involve rewards, discounting, and episodes. We mathematically analyze and experimentally study the effect of each, while remaining in a deterministic setting to improve clarity; Section 4 will then consider the case of stochastic systems.

3.1 Cost and reward models are equivalent, almost

Engineers have tended to minimize *cost* whereas AI researchers, especially in RL, have preferred to maximize *reward*. The former usually involves physical quantities, such as time, energy, distance, or perhaps money in a business setting. The latter takes inspiration from biology by imagining the robot or agent as being conditioned by Pavlovian rewards. As shown below, these formulations are equivalent at a general level; however, the problem with biologically inspired reward is that it tends to lead to the heuristic invention of reward functions to 'motivate' the robot to act in preconceived ways [20]. We therefore introduce the idea of *truecost*, in which the aim is to formulate costs that directly map to the physics of a given system (or perhaps money in a business setting). Ideally, costs or rewards should not be tweaked until the algorithm does what is was supposed to do. Of course, we can imagine *truereward*, but we emphasize that it should be more concrete, such as profit (revenue minus cost) in a business setting, rather than a vague bio-inspired motivator.

It is generally known that replacing the costs in a problem formulation by the corresponding rewards (multiplying the costs by -1) and switching from minimization to maximization results in an equivalent problem formulation in terms of the optimal policies. We show that this is generally true for any cost functional that is linear in terms of the sequence of stepwise costs. Note that this criterion covers all the standard cost formulations, including total cost, discounted cost, asymptotic average cost and the expected values of these in the stochastic formulations.

Proposition 2. *Let $P = (X, U, f, x_1, X_G, \ell)$ be a planning problem with a cost functional $L_c : U^{\mathbb{N}} \times X \to \mathbb{R}$. Assume that the cost-functional can be written as $L_c(\tilde{u}, x_1) = \mathcal{L}\big(h(\tilde{u}, x_1)\big)$ in which $h : U^{\mathbb{N}} \times X \to \mathbb{R}^{\mathbb{N}}$ maps a path onto the corresponding sequence of stepwise (immediate) costs and $\mathcal{L} : \mathbb{R}^{\mathbb{N}} \to \mathbb{R}$ is a linear operator. Then a policy π attains a minimum for L_c if and only if it attains a maximum for the operator L_r defined by $L_r(\tilde{u}, x_1) = \mathcal{L}\big(-h(\tilde{u}, x_1)\big)$.*

Proof. For $x \in X$, let $\tilde{u}^* \in U^{\mathbb{N}}$ be the action sequence corresponding to following an L_c-optimal policy π^* from x. Then $L_c(\tilde{u}^*, x) \leq L_c(\tilde{u}, x)$ for all $\tilde{u} \neq \tilde{u}^*$ so that

$$L_r(\tilde{u}^*, x) = \mathcal{L}\big(-h(\tilde{u}^*, x)\big) \geq \mathcal{L}\big(-h(\tilde{u}, x)\big) = L_r(\tilde{u}, x)$$

for all $\tilde{u} \neq \tilde{u}^*$. This means that applying $\tilde{u}^*$ from x corresponds to maximizing the reward functional L_r, in other words, following an L_r-optimal policy. The implication in the other direction follows similarly, and since x was arbitrary, the claim follows. □

3.2 The dangers of discounting

Infinite horizon decision making problems tend to yield diverging sums of costs or rewards. If there is a goal, as in the case of planning, then the simplest way to fix it is to use a termination action as introduced in Section 2: No more

costs/rewards accumulate after termination, thereby forcing (1) to be finite if the goal is reachable. The preferred way in RL is to use discounted cost/reward, which is a kind of mathematical hack dating back to Bellman [3]. For any $\alpha \in (0,1)$, the following sum is finite:

$$L_\alpha(\pi, x_1) = \lim_{K\to\infty} \left(\sum_{k=1}^{K} \alpha^{k-1} \ell(x_k, u_k) \right). \tag{7}$$

This implies that future costs/rewards are considered less valuable than current ones and it might be fine in highly unpredictable settings such as financial markets, but for robotics this causes a large and dangerous deviation from truecost. In the following, we discuss this aspect of discounting.

Let $\tilde{u} = (u_1, \ldots, u_{N-1})$ be a sequence of actions. A sequence of states $(x_1, \ldots, x_N)$ in which $x_{k+1} = f(x_k, u_k)$ for $k = 1, \ldots, N-1$ is a *cycle*, if $x_1 = x_N$ and $x_k \neq x_1$ for $k = 2, \ldots, N-1$. We denote by $C(\tilde{u}, x)$ a cycle beginning at x corresponding to $\tilde{u}$. If $\tilde{u}$ is given by a state-feedback policy $\pi : X \to U$, we denote a cycle starting at x by $C_\pi(x)$. For the following proposition, suppose ℓ is a strictly positive cost function other than $\ell(x, u_T)$ which is 0 for any x. Without loss of generality, we assume that there is a single goal state x_G.

Proposition 3. *Given a planning problem $\mathcal{P} = (X, U, f, x_I, x_G, \ell)$, suppose there exists a policy π_G for which $L(\pi_G, x_I)$, the cost according to the cost functional in* (1)*, is finite. Furthermore, suppose there exists a π_C such that the path $(x_1 = x_I, \pi_C(x_1), x_2, \pi_C(x_2), \ldots)$ contains a cycle $C_\pi(x)$, in which $x = x_m$, for some m. Then, there exist ℓ and α such that $L(\pi^*_\alpha, x_1) = \infty$, in which π^*_α is an optimal policy minimizing the cost functional defined in* (7)*.*

Proof. Without loss of generality, suppose that the cycle given by π_C starts at x_I, that is, $C_{\pi_C}(x_I) = (x_1 = x_I, x_2, \ldots, x_{N+1} = x_I)$ for some N and suppose that C_{π_C} does not include x_G. Since we are considering an infinite-horizon setting, this cycle will be executed infinitely many times. Let $\bar{c} = c_1 + \alpha c_2 + \alpha^2 c_3 + \ldots \alpha^{N-1} c_N$ be the cost that accumulates going through this cycle once, in which $c_i = \ell(x_i, \pi_C(x_i))$, $i = 1, \ldots, N$. Let $\bar{\alpha} := \alpha^N$. The discounted cost associated with π_C is $L_\alpha(\pi_C, x_I) = \lim_{K\to\infty} \sum_{k=0}^{K} (\bar{\alpha})^k \bar{c} = \frac{\bar{c}}{1-\bar{\alpha}}$. Let π^*_G be a policy that minimizes (7) among the policies that terminate at goal. The total discounted cost for π^*_G is $L_\alpha(\pi^*_G, x_I) = \lim_{K\to\infty} \sum_{k=1}^{K} \alpha^{k-1} \ell(x_k, \pi^*_G(x_k)) = \sum_{k=1}^{M} \alpha^{k-1} \ell(x_k, \pi^*_G(x_k))$, since the cost of termination action u_T, applied at the goal state $x_k = x_G$ for $k > M-1$ for some M, is 0. Denote by π^*_α, an optimal policy minimizing (7). Then, $L_\alpha(\pi^*_\alpha, x_I) \leq L_\alpha(\pi_C, x_I)$ and $L_\alpha(\pi^*_\alpha, x_I) \leq L_\alpha(\pi^*_G, x_I)$. If $L_\alpha(\pi^*_\alpha, x_I) = \frac{\bar{c}}{1-\bar{\alpha}} < \sum_{k=1}^{K-1} \alpha^{k-1} \ell(x_k, \pi^*_G(x_k)) = L_\alpha(\pi^*_G, x_I)$, π^*_α must contain a cycle. Then, we can pick ℓ and α such that this inequality is satisfied. In that case, $L(\pi^*_\alpha, x_I) = \infty$ since it contains a cycle.

To illustrate the issue that an optimal solution to a discounted cost problem might miss the goal even if the goal is reachable, consider the planning problem with $X = \{0, 1, \ldots, 5\}$, $x_G = 5$, $U = \{-1, 1\}$, $f(x, u) = \min(5, \max(0, x + u))$,

and $\ell(x,u) = x+1$. The only policy that reaches the goal is π_G for which $\pi_G(x) = 1$ for any $x \in X \setminus \{5\}$ and $\pi_G(5) = u_T$. For this policy, the truecost from $x_1 = 0$ is $L(\pi_G, 0) = 15$. For the discounted formulation, let $\pi_C(x) = -1$ for any $x \in X \setminus \{5\}$ and $\pi_C(5) = u_T$. Then, if α is selected so that $1/(1-\alpha) < \sum_{i=1}^{5} \alpha^{i-1}(i)$, an optimal policy π^*_α for the discounted cost formulation will have a cycle and miss the goal.

A more optimistic interpretation of this result is that a careful choice of a discount factor could prevent from such danger. However, we highlight that discounting is a heuristic that is as problematic as using potential functions for motion planning and becoming trapped in local minima [2]. One could try to pick α as close to one as possible, hoping to fend off the problem of ignoring the goal. An alternative way to ensure that the cost remains bounded over an infinite horizon is to optimize the average cost/reward per stage, which is also traced back to Bellman [3], and has been more recently suggested for RL [23,18]. The relationship between this and termination actions is the basis of Section 3.3.

3.3 Episodic equivalences

Imagine that the robot must solve the same tasks over and over, forever. Each time it arrives in the goal, it gets magically teleported back to the initial state and must get to the goal again; this is an example of physical jumping from Section 2. This leads to an infinite-horizon problem that fits the model of Section 2 but is closer to the common RL formulation that relies upon repeated *episodes* [26]. In this section, we establish an equivalence between the two formulations. This is worth considering because the resulting insight extends to problems in which different goal requests may be made in each episode (see, for example, [24]).

Consider the following two problem formulations:

(i) Let $P_{\text{unsp}} = (X, U, f, x_I, X_G, \ell)$ be an *unspecified-horizon* problem. We assume there exists a termination action u_T for which the terminal cost is $\ell(x_G, u_T) = 0$. The cost functional L is given by (1).

(ii) Let $P_{\text{inf}} = (X, U, f, x_I, X_G, \ell)$ be an *infinite-horizon* problem. The cost functional $L_{\text{av}} : U^{\mathbb{N}} \to \mathbb{R}$ is defined as the asymptotic average

$$L_{\text{av}}(\tilde{u}, x_1) := \lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} \ell(x_k, u_k). \tag{8}$$

Instead of setting $\ell(x_G, u_T) = 0$ as in (i), we assume that the system resets to x_I whenever it reaches x_G and incurs a negative cost (a bonus) $M < 0$.

We characterize the planning problems for which the above definitions (i) and (ii) share an optimal policy. For P_{unsp} there exists an optimal state-feedback policy π^*_{u}, which takes the robot from any initial state to the termination state with minimal cost. In P_{inf} there is no termination state, so any optimal policy leads the robot to enter a cycle with the lowest asymptotic average cost and to loop there indefinitely. Due to the reset, every cycle containing x_G must also

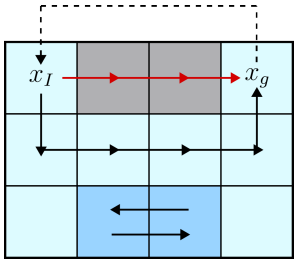

Fig. 3. The difference between a single trial (unspecified horizon) versus infinite-horizon formulation. The optimal path for a single trial is indicated with black arrows from x_I to x_G. A more costly, but shorter, path is indicated with red arrows. In the infinite-horizon formulation, a shortcut (dashed arrow) takes the robot from x_G and returns it to x_I. The dark blue squares indicate a region in state space where the robot could loop indefinitely with negligible average cost per step without ever reaching the goal.

contain x_I. It follows then from the optimality of π_{u}^* that the lowest-cost cycle containing x_G is $C_{\pi_{\mathrm{u}}^*}(x_I)$. However, there may exist other cycles that do not contain x_G and that have a lower asymptotic cost than $L_{\mathrm{av}}(C_{\pi_{\mathrm{u}}^*}(x_I))$. In such cases π_{u}^* fails to be optimal for P_{inf}. This can be remedied by tuning the bonus parameter M so that its cumulative effect overcomes the difference. However, changing the magnitude of M may also have the effect that some cycle $C(\tilde{u}, x_I) \neq C_{\pi_{\mathrm{u}}^*}(x_I)$ containing x_G may become optimal under the asymptotic average-cost formulation due to the resetting. See Figure 3.3 for an illustration of the two types of cycles. The conclusion is that the formulations P_{unsp} and P_{inf} are not generally equivalent, but may be so if certain conditions regarding the lengths and average costs of cycles in P_{inf} are satisfied. This is made precise in the following.

For an action sequence $\tilde{u}_k := (u_1, \ldots, u_{k-1})$, denote by $\Phi\big(x, \tilde{u}_k\big)$ the path $(x_1, \ldots, x_k)$ in which $x_1 = x$ and $x_{j+1} = f(x_j, u_j)$ for all $j = 1, \ldots, k-1$. Denote by $\varphi(x, \tilde{u}_k)$ the last state in $\Phi(x, \tilde{u}_k)$. Let $U_G := \{\tilde{u} : \varphi(x_I, \tilde{u}) = x_G\}$ be the set of action sequences that take the robot from x_I to x_G. The policies for P_{inf} can be divided into those that correspond to applying some $\tilde{u} \in U_G$ from x_I, and those that tend towards some other cycle that does not contain x_G. For any policy π that corresponds to looping an action sequence $\tilde{u} \in U_G$ from the initial state x_I, the asymptotic average cost is given by

$$L_{\mathrm{av}}(\tilde{u}, x_I) = \frac{M + L(\tilde{u}, x_I)}{S(\tilde{u})} = \frac{1}{S(\tilde{u})}\left(M + \sum_{k=1}^{S(\tilde{u})-1} \ell\big(x_k, f(x_k, u_k)\big)\right) \tag{9}$$

in which $S(\tilde{u})$ denotes the length of the path $\Phi\big(x_I, \tilde{u}\big)$. Let $\tilde{u}^* \in U_G$ be the action sequence corresponding to applying the optimal policy for P_{unsp} at x_I. Define for each $\tilde{u} \in U_G$ the value

$$\mathcal{A}(\tilde{u}) := \left(1 - \frac{S(\tilde{u})}{S(\tilde{u}^*)}\right)\left(\frac{S(\tilde{u})L(\tilde{u}^*, x_I)}{S(\tilde{u}^*)L(\tilde{u}, x_I)} - 1\right) \tag{10}$$

and the sets $\mathcal{R}^- := \big\{\tilde{u} : S(\tilde{u}) < S(\tilde{u}^*)\big\}$, $\mathcal{R}^+ := \big\{\tilde{u} : S(\tilde{u}) > S(\tilde{u})^*\big\}$. Furthermore, let $\hat{C} = C(\tilde{u}, x_1)$ be the cycle which has, among all the cycles that do not contain x_G, the smallest asymptotic average cost

$$L_{\mathrm{av}}(\hat{C}) = L_{\mathrm{av}}(\tilde{u}, x_1) = \frac{L(\tilde{u}, x_1)}{S(\tilde{u}) - 1}. \tag{11}$$

Proposition 4. *Let π^* be an optimal policy for P_{unsp}. Then π^* is also an optimal policy for P_{inf} with reset bonus $M < 0$ if and only if M satisfies*

$$a \leq M \leq \min\big\{b, S(\tilde{u}^*)L_{av}(\hat{C}) - L(\tilde{u}^*, x_I)\big\},$$

in which $\tilde{u}^*$ is the action sequence corresponding to applying π^* from x_I to reach x_G, $\hat{C}$ is as in (11)*, $\mathcal{A}(\tilde{u})$ is as in* (10)*, and*

$$a := \max_{\tilde{u}\in\mathcal{R}^-}\{\mathcal{A}(\tilde{u})\}, \quad \textit{and} \quad b := \min_{\tilde{u}\in\mathcal{R}^+}\{\mathcal{A}(\tilde{u})\}.$$

Proof. For π^* to be optimal for P_{inf}, the asymptotic average cost $L_{\text{av}}(\tilde{u}^*, x_I) = \big(M + L(\tilde{u}^*, x_I)\big)/S(\tilde{u}^*)$ must attain the lowest value among both the action sequences $\tilde{u} \in U_G$ and $\tilde{u} \notin U_G$. The second condition holds if and only if the cost $L_{\text{av}}(\tilde{u}^*, x_I)$ is less than $L_{\text{av}}(\hat{C})$, which can be expressed as

$$M < S(\tilde{u})L_{\text{av}}(\hat{C}) - L(\tilde{u}, x_I).$$

For $\tilde{u}^*$ to yield the lowest asymptotic average cost among the action sequences $\tilde{u} \in U_G$, it needs to satisfy the inequality

$$\frac{L(\tilde{u}^*, x_I) + M}{S(\tilde{u}^*)} \leq \frac{L(\tilde{u}, x_I) + M}{S(\tilde{u})} \tag{12}$$

for every $\tilde{u} \in U_G$. This is equivalent to $M \leq \mathcal{A}(\tilde{u})$ for $\tilde{u} \in \mathcal{R}^+$ and to $M \geq \mathcal{A}(\tilde{u})$ for $\tilde{u} \in \mathcal{R}^-$. Thus, for $\tilde{u}^*$ to satisfy (12) relative to all $\tilde{u} \in U_G$ is equivalent to M satisfying the double inequality $\max_{\tilde{u}\in\mathcal{R}^-}\{\mathcal{A}(\tilde{u})\} \leq M \leq \min_{\tilde{u}\in\mathcal{R}^+}\{\mathcal{A}(\tilde{u})\}$. □

4 From Deterministic to Stochastic Models

4.1 Basic assumptions and approach

The concepts from Sections 2 and 3 extend gracefully to the stochastic setting by replacing f in $\mathcal{P}$ with a probabilistic transition model $P(x_{k+1}|x_k, u_k)$, which denotes the probability that x_{k+1} is reached if u_k is applied from x_k. This causes paths taken during execution to be unpredictable, resulting in G^* and Q^* representing *expected* costs. Furthermore, every state-action pair should be visited an infinite number of times to learn P, rather than once to learn f from the deterministic setting. Thus, the method from Section 2 of quickly learning f first, and then applying Dijkstra's algorithm is no longer applicable. At best, one could try to estimate the transition probabilities to some level of statistical confidence and then apply stochastic value iteration to obtain G^* and π^*. For comparative purposes, we will consider the case of stochastic value iteration applied to a perfectly learned transition model. However, most of our studies in this section are based on Q-learning.

In our implementations, the transition model $P(x_{k+1}|x_k, u_k)$ is defined to be a direct extension of f from Section 2, in terms of a *predictability factor*, $\gamma \in (0, 1]$, that controls the amount of entropy or prediction uncertainty. From

each x_k, consider applying u_k to get $x_{k+1} = f(x_k, u_k)$. For the stochastic version with predictability factor γ, we define $P(x'_{k+1}|x_k, u_k) = \gamma$ if $x'_{k+1} = x_{k+1}$; otherwise, the remaining probability mass $1 - \gamma$ is spread uniformly as if nature causes the remaining actions (excluding u_k) to be chosen from $U(x_k)$ (including a termination action that holds the state constant). If $\gamma = 1$, then the deterministic model can essentially be simulated. If $\gamma = 1/2$, then on average, half of the time the robot moves in the direction it was commanded, and for the other half it effectively executes an alternative action randomly.

Another critical parameter is the learning rate ρ, which appears in (5). Imagine implementing a complementary filter that uses ρ to interpolate between a noisy data source and the current best estimate. If the data is perfectly reliable, as in the deterministic case of Section 2, then we can set $\rho = 1$. As the predictability decreases (lower γ), we would expect to use lower values for ρ so that the noise is attenuated and the estimate becomes more stable. Choosing a good value of ρ is a delicate art. We considered multiple values in our experiments, and furthermore developed an adaptive tuning method for ρ so that it decreases during execution according to the formula

$$\rho(x, u) = \frac{1}{n(x, u)^\omega}, \tag{13}$$

in which $n(x, u)$ is the number of times that a state-action pair has been visited and ω is an arbitrary parameter (this idea was introduced in [10] and used in the Double Q-learning approach [11]). In this approach, there is a separate $\rho(x, u)$ for each state-action pair, which is decreased as visits accumulate.

4.2 Computational comparisons

The experimental setup outlined in Section 2.3 is reused with a few minor changes. In cases where $\gamma \leq 0.9$ we use 10 samples instead of 100. We increased the number of episodes to 3000 to improve the convergence frequency, that is, the percentage of times the values converge. We investigated performance by trading off three parameters: learning rate ρ, greedy parameter ϵ, and predictability factor γ. We studied convergence rates in comparison to the true optimal cost to go, obtained from stochastic value iteration, and flagged cases in which the Q-values are within 10% of optimal to identify approximate successes.

For cases of high γ (more predictable), we investigated combinations of $\epsilon, \rho \in \{0.5, 0.7, 0.9, 0.99\}$. The ρ values were chosen after careful investigation and observing that for $\rho < 0.5$ and $\rho > 0.99$, the results were not more informative. We also found that increasing ρ decreases run time. In Table 2, which combines the data for Problems 1 and 10, note that for $\gamma = 0.999$ there is less benefit of being greedy in comparison to the results of Section 2.3. Although choosing $\epsilon = 0$ still leads to a better or equal run time to $\epsilon = 1$, we see that $\epsilon = 0.9$ has a much better performance than both, while also converging. This appears to be due the fact that nondeterminism throws the state away from greedy routes, thereby requiring more exploration to get values to converge.

Table 2. Performance of Q-learning and DP methods for Problems 1 and 10 for stochastic problem with $\gamma = 0.999$. Note the addition of two columns: how often the value for the initial state converges; the shortest and longest path achieved over all samples.

Algorithm ($\gamma = 0.999$)	Run time (mean ± std)	Convergence	Optimal Initial Cost2Go Time (mean ± std)	Optimal Initial Cost2Go Convergence	Shortest/Longest Path
(Pr 1) Q-learning ($\rho = 0.99, \epsilon = 0$)	0.40677 ± 0.0041	0.0%	0.0888 ± 0.0008	100.0%	28/32
(Pr 1) Q-learning ($\rho = 0.99, \epsilon = 0.25$)	0.52970 ± 0.0064	0.0%	0.1006 ± 0.0024	100.0%	28/29
(Pr 1) Q-learning ($\rho = 0.99, \epsilon = 0.5$)	0.76335 ± 0.0084	0.0%	0.1144 ± 0.0024	100.0%	28/30
(Pr 1) Q-learning ($\rho = 0.99, \epsilon = 0.75$)	0.52489 ± 0.1930	100.0%	0.1516 ± 0.0045	100.0%	28/30
(Pr 1) Q-learning ($\rho = 0.99, \epsilon = 0.9$)	0.27856 ± 0.0271	100.0%	0.2044 ± 0.0109	100.0%	28/29
(Pr 1) Q-learning ($\rho = 0.99, \epsilon = 1$)	0.40572 ± 0.0491	100.0%	0.3512 ± 0.0450	100.0%	28/30
(Pr 1) Async Value Iteration	0.12869 ± 0.0009	N/A	N/A	N/A	28/29
(Pr 1) Value Iteration	0.17870 ± 0.0024	N/A	N/A	N/A	28/30
(Pr 10) Q-learning ($\rho = 0.99, \epsilon = 0$)	0.94648 ± 0.0174	0.0%	0.2290 ± 0.0314	100.0%	64/27731
(Pr 10) Q-learning ($\rho = 0.99, \epsilon = 0.25$)	1.24982 ± 0.0203	0.0%	0.2513 ± 0.0056	100.0%	64/66
(Pr 10) Q-learning ($\rho = 0.99, \epsilon = 0.5$)	1.91614 ± 0.0188	0.0%	0.3029 ± 0.0058	100.0%	64/67
(Pr 10) Q-learning ($\rho = 0.99, \epsilon = 0.75$)	0.97765 ± 0.2121	100.0%	0.4122 ± 0.0107	100.0%	64/66
(Pr 10) Q-learning ($\rho = 0.99, \epsilon = 0.9$)	0.65742 ± 0.0572	100.0%	0.5611 ± 0.0225	100.0%	64/66
(Pr 10) Q-learning ($\rho = 0.99, \epsilon = 1$)	2.26297 ± 0.7142	100.0%	1.6362 ± 0.4014	100.0%	64/66
(Pr 10) Async Value Iteration	0.21926 ± 0.0162	N/A	N/A	N/A	64/66
(Pr 10) Value Iteration	0.35463 ± 0.0252	N/A	N/A	N/A	64/66

Furthermore, we can see that DP methods converge about two orders of magnitude more quickly than RL, which reflects the price of learning on the fly. Similar trends were observed in other problems; the appendix contains many more. It also reports statistics for the goal discovery time, which are similar to the deterministic case, but grow with increasing ρ.

Lowering γ to 0.99 prevents convergence, even if we increase the episode count to 5000 or 10000. In fact, further investigation reveals that the average distance between the optimal values and the values obtained with Q-learning remains nearly constant as the number of episodes increase when $\epsilon = 1$. This suggests that convergence happens on a state-space level, but to values different than the optimal ones. However, the initial cost-to-go values still converge for some parameter combinations. Table 3 shows when this happens for Problems 1 and 10. The general observation is that a lower ρ yields more reliable convergence because it is more robust to unpredictability, but requires more iterations. As γ is lowered, then ρ should be lowered accordingly, although the best combination is found only by trial and error. It is worthy noting that a policy that leads the robot to the goal in a nearly optimal manner can be computed even if the Q-values from the initial state do not converge, as seen in the configuration with $\rho = \epsilon = 0.5$ for Problem 1. Interestingly, decreasing γ, we observe, for Problems 1 and 11, that the run time of value iteration becomes closer to that of greedy Q-learning with a small ρ.

To study global state-space convergence to optimal values, we investigated the effects of a small ρ on simpler problems (i.e., Problem 8 and Problem 16). For $\gamma = 0.9$, a random policy ($\epsilon = 1$) with $\rho = 0.1$ converges for both problems, whereas for Problem 16 even a greedy policy ($\epsilon = 0$) with the same ρ value suffices. However, this approach is not sufficient for $\gamma = 0.7$. Instead, global convergence with $\epsilon = 1$ occurs only by decaying ρ as described in (13), using $\omega = 0.7$. As state-action visits increase, ρ decays at a rate controlled by ω,

Table 3. Performance of Q-learning and DP methods for Problems 1 and 10 for stochastic problem with $\gamma = 0.99$.

Algorithm ($\gamma = 0.99$)	Run time (mean ± std)	Convergence	Optimal Initial Cost2Go Time (mean ± std)	Optimal Initial Cost2Go Convergence	Shortest/Longest Path
(Pr 1) Q-learning ($\rho = 0.5, \epsilon = 0$)	0.44775 ± 0.0094	0.0%	0.1815 ± 0.0082	100.0%	28/34
(Pr 1) Q-learning ($\rho = 0.99, \epsilon = 0$)	0.42034 ± 0.0090	0.0%	0.2078 ± 0.0750	86.0%	28/2205
(Pr 1) Q-learning ($\rho = 0.5, \epsilon = 0.5$)	0.82059 ± 0.0169	0.0%	0.5143 ± 0.1522	32.0%	28/31
(Pr 1) Q-learning ($\rho = 0.99, \epsilon = 0.5$)	0.77585 ± 0.0109	0.0%	nan ± nan	0.0%	28/1474
(Pr 1) Q-learning ($\rho = 0.5, \epsilon = 1$)	15.46274 ± 0.1914	0.0%	5.2842 ± 3.5571	90.0%	28/33
(Pr 1) Q-learning ($\rho = 0.99, \epsilon = 1$)	15.39603 ± 0.1846	0.0%	nan ± nan	0.0%	28/32
(Pr 1) Async Value Iteration	0.15656 ± 0.0014	N/A	N/A	N/A	28/33
(Pr 1) Value Iteration	0.22934 ± 0.0020	N/A	N/A	N/A	28/31
(Pr 10) Q-learning ($\rho = 0.5, \epsilon = 0$)	1.09601 ± 0.0226	0.0%	0.4524 ± 0.0133	100.0%	64/322
(Pr 10) Q-learning ($\rho = 0.99, \epsilon = 0$)	1.07131 ± 0.0178	0.0%	0.2651 ± 0.0000	1.0%	66/2596
(Pr 10) Q-learning ($\rho = 0.5, \epsilon = 0.5$)	2.08346 ± 0.0157	0.0%	0.7959 ± 0.2102	100.0%	64/68
(Pr 10) Q-learning ($\rho = 0.99, \epsilon = 0.5$)	1.99043 ± 0.0217	0.0%	0.6288 ± 0.3471	99.0%	64/3689
(Pr 10) Q-learning ($\rho = 0.5, \epsilon = 1$)	30.93903 ± 0.2059	0.0%	3.5574 ± 1.1094	100.0%	64/70
(Pr 10) Q-learning ($\rho = 0.99, \epsilon = 1$)	30.94423 ± 0.1896	0.0%	2.4456 ± 1.0780	100.0%	64/610
(Pr 10) Async Value Iteration	0.24640 ± 0.0036	N/A	N/A	N/A	64/69
(Pr 10) Value Iteration	0.40702 ± 0.0028	N/A	N/A	N/A	64/70

stabilizing the estimate. We observe global convergence for Problem 16 with $\gamma = 0.5$ using the same approach. We were unable to get global convergence in Problem 8 by any choice of ω.

5 Conclusions

Our work has brought planning and reinforcement learning into closer alignment so that approaches, assumptions, and models across both can be compared and be better understood. Through the introduction of a fully deterministic version of RL, we established its convergence and studied its performance relative to dynamic-programming based planning. Through analysis of cost models, we have warned against using discounted cost, as is popular in RL; instead, we have advocated for truecost, in which the costs/rewards translate directly into physical or monetary values, rather than using them as a way to tune performance. We have also established equivalences of maximizing rewards and minimizing costs, and episodic models to single-shot goal satisfaction models. These provide motivation to use undiscounted cost models and termination actions for RL for solving goal-oriented tasks. We applied these models in the stochastic system setting and showed how they extend naturally from the deterministic case, but inherit unique RL-oriented challenges, such as greediness and learning rate parameter tuning, which warrant further study. We are currently extending the mathematical analysis of cost-reward and episodic equivalences to the stochastic case.

Acknowledgments We thank Markku Suomalainen for helpful feedback and discussions.

References

1. S. Albers and M. T. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.
2. J. Barraquand, B. Langlois, and J. C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, & Cybernetics*, 22(2):224–241, 1992.
3. R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.
4. D. P. Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120, 1983.
5. D. P. Bertsekas. *Reinforcement learning and optimal control*, volume 1. Athena Scientific, 2019.
6. D. P. Bertsekas. Model predictive control and reinforcement learning: A unified framework based on dynamic programming. *IFAC-PapersOnLine*, 58(18):363–383, 2024. 8th IFAC Conference on Nonlinear Model Predictive Control NMPC 2024.
7. D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.
8. H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust. Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies. *IEEE Robotics and Automation Letters*, 4:4298–4305, 2019.
9. G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Map validation and self-location in a graph-like world. In *Proceedings AAAI National Conference on Artificial Intelligence*, pages 1648–1653, 1993.
10. E. Even-Dar and Y. Mansour. *Learning Rates for Q-Learning*, page 589–604. Springer Berlin Heidelberg, 2001.
11. H. Hasselt. Double q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.
12. R. Kalman. Randomness reexamined. *Modeling, Identification and Control*, 15(3):141–151, 1994.
13. J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.
14. S. Koenig and M. Likhachev. D^* lite. In *Proceedings AAAI National Conference on Artificial Intelligence*, pages 476–483, 2002.
15. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Also available at http://lavalle.pl/planning/.
16. S. M. LaValle, M. S. Branicky, and S. R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*, 23(7/8):673–692, July/August 2004.
17. S. M. LaValle and P. Konkimalla. Algorithms for computing numerical optimal feedback motion strategies. *International Journal of Robotics Research*, 20(9):729–752, September 2001.
18. S. Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1–3):159–195, 1996.
19. A. Naik, R. Shariff, N. Yasui, H. Yao, and R. S. Sutton. Discounted reinforcement learning is not an optimization problem, 2019. arXiv, 1910.02140.
20. A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, page 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

21. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, April 1994.
22. H. Robbins and S Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, September 1951.
23. A. Schwartz. *A Reinforcement Learning Method for Maximizing Undiscounted Rewards*, page 298–305. Elsevier, 1993.
24. R. Sharma, S. M. LaValle, and S. A. Hutchinson. Optimizing robot motion strategies for assembly with stochastic models of the assembly process. *IEEE Trans. on Robotics and Automation*, 12(2):160–174, April 1996.
25. A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3310–3317, 1994.
26. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
27. C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone. Deep reinforcement learning for robotics: A survey of real-world successes. *Annual Review Control, Robotics, and Autonomous Systems*, 8:153–188, 2025.
28. K. G. Timperi, A. J. LaValle, and S. M. LaValle. Universal plans: One action sequence to solve them all! In N. Amato, K. Driggs-Campbell, E. Chinwe, M. Morales, and J. M. O'Kane, editors, *Algorithmic Foundations of Robotics, XVI*. Springer-Verlag, Berlin, 2025.
29. C. J. Watkins and P Dayan. Technical note: Q-learning. *Machine Learning*, 8(3–4):279–292, May 1992.
30. Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.
31. D. Yershov and S. M. LaValle. Simplicial Dijkstra and A* algorithms for optimal feedback planning. In *Proceedings IEEE International Conference on Intelligent Robots and Systems*, 2011.