Article

The International Journal of Robotics Research
2015, Vol. 34(10) 1329–1344
© The Author(s) 2015
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0278364915581505
ijr.sagepub.com
SAGE

Temporal logic motion control using actor–critic methods

Jing Wang[1], Xuchu Ding[2], Morteza Lahijanian[3], Ioannis Ch. Paschalidis[1] and Calin A. Belta[1]

Abstract

This paper considers the problem of deploying a robot from a specification given as a temporal logic statement about some properties satisfied by the regions of a large, partitioned environment. We assume that the robot has noisy sensors and actuators and model its motion through the regions of the environment as a Markov decision process (MDP). The robot control problem becomes finding the control policy which maximizes the probability of satisfying the temporal logic task on the MDP. For a large environment, obtaining transition probabilities for each state–action pair, as well as solving the necessary optimization problem for the optimal policy, are computationally intensive. To address these issues, we propose an approximate dynamic programming framework based on a least-squares temporal difference learning method of the actor–critic type. This framework operates on sample paths of the robot and optimizes a randomized control policy with respect to a small set of parameters. The transition probabilities are obtained only when needed. Simulations confirm that convergence of the parameters translates to an approximately optimal policy.

Keywords

Motion planning, Markov decision processes, dynamic programming, actor–critic methods

1. Introduction

One major goal in robot motion planning and control is to specify a mission task in an expressive and high-level language and to convert the task automatically to a control strategy for the robot. The robot is subject to mechanical constraints, actuation and measurement noise, and limited communication and sensing capabilities. The challenge in this area is the development of a computationally efficient framework accommodating both the robot constraints and the uncertainty of the environment, while allowing for a large spectrum of task specifications.

In recent years, temporal logics such as linear temporal logic (LTL) and computation tree logic (CTL) have been promoted as formal task specification languages for robotic applications (Loizou and Kyriakopoulos, 2004; Quottrup et al., 2004; Kress-Gazit et al., 2007; Karaman and Frazzoli, 2009; Wongpiromsarn et al., 2009; Bhatia et al., 2010). They are appealing due to their high expressivity and ability to formally capture informal requirements specified in human language. Moreover, several existing formal verification (Clarke et al., 1999; Baier and Katoen, 2008) and synthesis (Baier and Katoen, 2008; Liu et al., 2013; Luna et al., 2014) tools can be adapted to generate motion plans and provably correct control strategies for the robots.

In this paper, we assume that the motion of the robot in the environment is described by a (finite) Markov decision process (MDP). In this model, the robot can precisely determine its current state (a discrete region in the environment), and by applying an action (corresponding to a motion primitive) enabled at each state, it triggers a transition to an adjacent state (region) with a fixed probability. We are interested in controlling the robot such that it maximizes the probability of satisfying a temporal logic formula over a set of properties satisfied at the states of the MDP. For simplicity, case studies presented in this paper focus on actions which cause some motion or movement of the

[1]Division of System Engineering, Department of Mechanical Engineering, and Department of Electrical and Computer Engineering, Boston University, Boston, MA, USA
[2]Embedded Systems and Networks Group, United Technologies Research Center, East Hartford, CT, USA
[3]Department of Computer Science, Rice University, Houston, TX, USA

Corresponding author:
Ioannis Ch. Paschalidis, Division of System Engineering and Department of Electrical and Computer Engineering, Boston University, 8 Saint Mary's Street, Boston, MA 02215, USA.
Email: yannisp@bu.edu

robot. However, our approach can be used for any generic actions such as "take readings of a sensor" or "wait until batteries are charged." Such actions can be used in conjunction with temporal logic to form specifications such as "take sensor readings in the building only after batteries are charged."

By adapting existing probabilistic model checking (De Alfaro, 1997; Vardi, 1999; Baier and Katoen, 2008) and synthesis (Courcoubetis and Yannakakis, 1990; Baier et al., 2004) algorithms, we (Ding et al., 2011; Lahijanian et al., 2012) and others (Wolff et al., 2012) recently developed such computational frameworks for formulae of LTL and a fragment of probabilistic CTL. If the transition probabilities for each state–action pair of the MDP are known, an optimal control policy can be generated using the above approaches to maximize the satisfaction probability. The transition probabilities can be computed by using a Monte Carlo method and repeated forward simulations.

Transition probabilities of the MDP, however, can be difficult to obtain in practice. Previous approaches assumed exact knowledge (Ding et al., 2011; Lahijanian et al., 2012). Wolff et al. (2012) used an uncertain model of the MDP to produce a robust optimal policy corresponding to the worst-case transition probabilities. In this paper, we assume no prior knowledge of the transition probabilities and they are to be learned or computed via simulations online. In the proposed approach, not all transition probabilities of the MDP are needed. We only obtain the ones needed by the algorithm during on-line execution.

In real applications, the size of the state space of the MDP is usually very large. Our previous approaches are not suitable for large-sized problems due to the following limitations. First, they require transition probabilities for all state–action pairs, which are costly to obtain even if an accurate simulator of the robot in the environment is available. Second, the optimal policy is calculated by solving a large linear programming (LP) problem on the product between the original MDP and a Rabin automaton. The existing LP solvers are not efficient enough and their memory usage increases rapidly with the problem size.

In this paper, we show that approximate dynamic programming (Si, 2004; Bertsekas and Tsitsiklis, 1996) can be effectively used to address the above limitations. For large dynamic programming problems, an approximately optimal solution can be provided using actor–critic algorithms (Barto et al., 1983). By approximating both the policy and state–action value function with a parameterized structure, actor–critic algorithms use much less memory compared with other dynamic programming techniques. In particular, actor–critic algorithms with least squares temporal difference (LSTD) learning have been shown recently to be a powerful tool for large-sized problems (Konda and Tsitsiklis, 2003; Estanjini et al., 2011, 2012). Ding et al. (2011) showed that a motion control problem with temporal logic specifications could be converted to a maximal reachability probability (MRP) problem, i.e. maximizing the probability of reaching a set of states. In Estanjini et al. (2011), we showed that the MRP problem is equivalent to a stochastic shortest path (SSP) problem and proposed an actor–critic method to solve the SSP problem. In Ding et al. (2012), we applied the actor–critic method of Estanjini et al. (2011) to find a control policy that maximizes the probability of satisfying a temporal logic specification. Our proposed algorithm produces a *randomized stationary policy* (RSP), which gives a probability distribution over enabled actions at a state. Our method requires transition probabilities to be generated only along sample paths, and is therefore particularly suitable for robotic applications. To the best of our knowledge, this is the first attempt to combine temporal logic formal synthesis with actor–critic-type methods.

This paper builds on preliminary results previously presented in Estanjini et al. (2011) and Ding et al. (2012) in several ways. First, we provide a proof for the equivalence of the MRP and SSP problems. We also provide a proof of convergence for our actor–critic method on the SSP problem. Compared with Ding et al. (2012), we propose a more accurate *safety score*, which helps simplify the RSP structure. We include a case study with more complex temporal logic specifications and present results showing that the specifications are satisfied in a sample path generated by the actor–critic method. Finally, we analyze the time and memory usage of our actor–critic method for problems with different sizes.

The remainder of the paper is organized as follows. In Section 2, we formulate the motion control problem with temporal logic specifications. Section 3 describes our method to solve this problem. Section 4 illustrates our experiment setups and presents the results accordingly. Section 5 concludes the paper.

Notation

We use bold letters to denote sequences and vectors. Vectors are assumed to be column vectors. Transpose of a vector $\mathbf{y}$ is denoted by $\mathbf{y}'$. Here $|S|$ denotes the cardinality of a set S.

2. Problem formulation

In this paper, we consider a robot moving in an environment partitioned into regions such as the Robotic Indoor Environment (RIDE) (see Figure 1) (Lahijanian et al., 2010). Each region in the environment is associated with a set of properties. Properties can be **Un** for unsafe regions or **Up** for a region where the robot can upload data. We assume that the robot can detect its current region. Moreover, the robot is programmed with a set of motion primitives allowing it to move from a region to an adjacent region. To capture noise in actuation and sensing, we make the natural assumption that, at a given region, a motion primitive designed to take the robot to a specific adjacent region may take the robot to a different adjacent region.

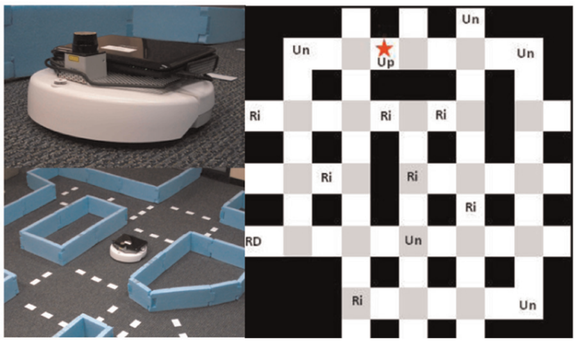

Fig. 1. Robotic Indoor Environment (RIDE) platform. (Left) An iCreate mobile platform moving autonomously through the corridors and intersections of an indoor-like environment. The robot is equipped with a RFID reader that can correctly identify cards placed on the floor and with a laser range finder that is used to implement motion primitives such as `GoLeft` and `GoForward` in an intersection, etc. (Right) An example schematic of the environment. The black blocks represent walls, and the grey and white regions are intersections and corridors, respectively. The labels inside a region represent properties associated with regions, such as **Un** (unsafe regions) and **Ri** (risky regions).

Such a robot model naturally leads to a labeled MDP, which is defined below.

Definition 2.1 (Labeled Markov Decision Process). *A labeled MDP is a tuple $\mathcal{M} = (Q, q_0, U, A, P, \Pi, h)$, where:*

(i) $Q = \{1,\ldots,n\}$ *is a finite set of states;*
(ii) $q_0 \in Q$ *is the initial state;*
(iii) U *is a finite set of actions;*
(iv) A: $Q \to 2^U$ *maps state* $q \in Q$ *to actions enabled at* q*;*
(v) P: $Q \times U \times Q \to [0, 1]$ *is the transition probability function such that for all* $q \in Q$, $\sum_{r \in Q} P(q, u, r) = 1$ *if* $u \in A(q)$, *and* $P(q, u, r) = 0$ *for all* $r \in Q$ *if* $u \notin A(q)$*;*
(vi) Π *is a set of properties;*
(vii) h: $Q \to 2^\Pi$ *is the property map.*

In RIDE platform, each state of the MDP $\mathcal{M}$ modeling the robot in the environment corresponds to an ordered set of regions in the environment, while the actions label the motion primitives that can be applied at a region. In this example, a state of $\mathcal{M}$ is labeled as I_1–C_1, which means that the robot is currently at region C_1, coming from region I_1. Each ordered set of regions corresponds to a recent history of the robot trajectory, and is needed to ensure the Markov property (more details on such MDP abstractions of the robot in the environment can be found in e.g. Lahijanian et al. (2012). The transition probability function P can be obtained through extensive simulations of the robot in the environment. We assume that there exists an accurate simulator that is capable of generating (computing) the transition probability $P(q, u, \cdot)$ for each state–action pair $q \in Q$ and $u \in A(q)$. In our previous work (Lahijanian et al., 2012), we developed such a simulator for the robot shown in Figure 1. More details on the construction of the MDP model for a robot in the RIDE platform are included in Section 4.

A path on $\mathcal{M}$ is a sequence of states $\mathbf{q} = q_0q_1\ldots$ such that for all $k \geq 0$, there exists $u_k \in A(q_k)$ such that $P(q_k, u_k, q_{k+1}) > 0$. Along a path $\mathbf{q} = q_0q_1\ldots$, q_k is said to be the state at time k. The trajectory of the robot in the environment is represented by a path $\mathbf{q}$ on $\mathcal{M}$ (which corresponds to a sequence of regions in the environment). A path $\mathbf{q} = q_1q_2\ldots$ generates a sequence of properties $\mathbf{h}(\mathbf{q}) := o_1o_2\ldots$, where $o_k = h(q_k)$ for all $k \geq 0$. We call $\mathbf{o} = \mathbf{h}(\mathbf{q})$ the word generated by $\mathbf{q}$. A path on $\mathcal{M}$ can be mapped directly to a sequence of region transitions in the environment. Note that, due to how the state is labeled (as the current region label followed by the previous label), the current region-label associated with each state of a path must match the previous region-label of the following state.

Definition 2.2 (Policy). *A control policy for a MDP $\mathcal{M}$ is an infinite sequence $M = \mu_0\mu_1\ldots$, where $\mu_k : Q \times U \to [0, 1]$ is such that $\sum_{u \in A(q)} \mu_k(q, u) = 1$, for all $k \geq 0$ and for all $q \in Q$.*

Namely, at time k, $\mu_k(q,\cdot)$ is a discrete probability distribution over $A(q)$. If $\mu_k = \mu$ for all $k \geq 0$, then $M = \mu\mu\ldots$ is called a *stationary* policy. If for all $k \geq 0$ and for all $q \in Q$, $\mu_k(q, u) = 1$ for some u, then M is *deterministic*; otherwise, M is *randomized*. Given a policy M, we can then generate a sequence of states on $\mathcal{M}$, by applying u_k with probability $\mu_k(q_k, u_k)$ at state q_k for all time k. Such a sequence is called a *path* of $\mathcal{M}$ under policy M.

We require the trajectory of the robot in the environment to satisfy a rich task specification given as an LTL (see, e.g., Baier and Katoen, 2008; Clarke et al., 1999) formula over a set of properties Π. An LTL formula over Π is evaluated over an (infinite) sequence $\mathbf{o} = o_0o_1\ldots$ (e.g. a word generated by a path on $\mathcal{M}$), where $o_k \subseteq \Pi$ for all $k \geq 0$. We denote $\mathbf{o} \models \phi$ if word $\mathbf{o}$ satisfies the LTL formula ϕ, and we say $\mathbf{q}$ satisfies ϕ if $\mathbf{h}(\mathbf{q}) \models \phi$. Roughly, ϕ can be constructed from a set of properties Π, Boolean operators $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), $\to$ (implication), and temporal operators `X` (next), `U` (until), `F` (eventually), `G` (always). A variety of robotic tasks can be easily translated to LTL formulas. For example, the following complex task command in natural language: *"Gather data at locations* **Da** *infinitely often. Only reach a risky region* **Ri** *if valuable data* **VD** *can be gathered, and always avoid unsafe regions* (**Un**)*"* can be translated to the LTL formula:

$$\phi := \mathrm{G}\,\mathrm{F}\,\mathbf{Da} \wedge \mathrm{G}\,(\mathbf{Ri} \to \mathbf{VD}) \wedge \mathrm{G}\,\neg\mathbf{Un}$$

In Ding et al. (2011) (see also Rutten et al., 2004), we consider the following problem.

Problem 2.3. *Given a labeled MDP $\mathcal{M} = (Q, q_0, U, A, P, \Pi, h)$ and a LTL formula ϕ, find a control policy that maximizes the probability of its path satisfying ϕ.*

The probability that paths generated under a policy μ satisfy an LTL formula ϕ is well defined with a suitable measure over the set of all paths generated by μ (Baier and Katoen, 2008).

In Ding et al. (2011), we proposed a computational framework to solve Problem 2.3, by adapting methods from the area of probabilistic model checking (De Alfaro, 1997; Vardi, 1999; Baier and Katoen, 2008). However, this framework relies upon the fact that the transition probabilities are known for all state–action pairs. These transition probabilities are typically not directly available and often computationally expensive to compute. Moreover, even if the transition probabilities are obtained for each state–action pair, this method still requires solving a linear program on the product of the MDP and the automata representing the formula, which can be very large (thousands or even millions of states).

If the exact transition probabilities are not known, $\mathcal{M}$ can be seen as a labeled non-deterministic transition system (NTS) $\mathcal{M}^{\mathcal{N}} = (Q, q_0, U, A, P^{\mathcal{N}}, \Pi, h)$, where $P^{\mathcal{N}}$ is the mapping $Q \times U \times Q \rightarrow \{0,\ 1\}$, such that $P^{\mathcal{N}}(q, u, r) = 1$ indicates that a transition from q to r exists after applying an enabled action $u \in A(q)$ and $P^{\mathcal{N}}(q, u, r) = 0$ indicates that no transition from q to r is possible under u. In many robotic applications, the NTS model $\mathcal{M}^{\mathcal{N}} = (Q, q_0, U, A, P^{\mathcal{N}}, \Pi, h)$ can be quickly constructed for the robot in the environment and a simulator is available to generate actual transition probabilities "on the fly." Motivated by this, we focus on the following problem in this paper.

Problem 2.4. *Given a labeled NTS $\mathcal{M}^{\mathcal{N}} = (Q, q_0, U, A, P^{\mathcal{N}}, \Pi, h)$, an LTL formula ϕ, and an accurate simulator to compute transition probabilities $P(q,\ u,\cdot)$ given a state–action pair $(q,\ u)$, find a control policy that maximizes the probability of its path satisfying ϕ.*

Transition probabilities for all state–action pairs are necessary for exact optimal solution of Problem 2.4. In this paper, we propose an approximate method that only needs transition probabilities for a portion of state–action pairs. Our approach to Problem 2.4 can be summarized as follows. First, we formulate the problem as a MRP problem using $\mathcal{M}^{\mathcal{N}}$ and ϕ (Section 3.1), and convert the MRP problem into a SSP problem. We then use an actor–critic framework to find a randomized policy giving an approximate solution to the SSP problem (Section 3.3). The randomized policy is constructed to be a function of a small set of parameters and we find a policy that is locally optimal with respect to these parameters. The construction of a class of policies suitable for SSP problems is explained in Section 3.4. The algorithmic framework presented in this paper is summarized in Section 3.5.

3. Control synthesis

3.1. Formulation of the MRP problem

The formulation of the MRP problem is based on Ding et al. (2011), De Alfaro (1997), Baier and Katoen (2008) and Vardi (1999), with the exception that the MDP is replaced by the NTS. We start by converting the LTL formula ϕ over Π to a so-called deterministic *Rabin automaton*, which is defined as follows.

Definition 3.1 (Deterministic Rabin automaton). *A deterministic Rabin automaton (DRA) is a tuple $\mathcal{R} = (S, s_0, \Sigma, \delta, F)$, where*:

(i) *S is a finite set of states;*
(ii) *$s_0 \in S$ is the initial state;*
(iii) *Σ is a set of inputs (alphabet);*
(iv) *$\delta : S \times \Sigma \rightarrow S$ is the transition function;*
(v) *$F = \{(L(1),\ K(1)),\ldots,(L(M),\ K(M))\}$ is a set of pairs of sets of states such that $L(i),\ K(i)\ \subseteq S$ for all $i = 1,\ldots,M$.*

A run of a Rabin automaton $\mathcal{R}$, denoted by $\mathbf{r} = s_0 s_1\ldots$, is an infinite sequence of states in $\mathcal{R}$ such that for each $k \geq 0$, $s_{k+1} \in \delta(s_k,\ \alpha)$ for some $\alpha \in \Sigma$. A run $\mathbf{r}$ is *accepting* if there exists a pair $(L,\ K) \in F$ such that $\mathbf{r}$ intersects with L finitely many times and K infinitely many times. For any LTL formula ϕ over Π, one can construct a DRA (which we denote by $\mathcal{R}_\phi$) with input alphabet $\Sigma = 2^{\Pi}$ accepting all and only words over Π that satisfy ϕ (see Gradel et al., 2002).

We then obtain a MDP as the product of a labeled MDP $\mathcal{M}$ and a DRA $\mathcal{R}_\phi$, which captures all paths of $\mathcal{M}$ satisfying ϕ. Note that this product MDP can only be constructed from a MDP and a deterministic automaton, this is why we require a DRA instead of, e.g., a (generally non-deterministic) Büchi automaton (see Baier and Katoen, 2008).

Definition 3.2 (Product MDP). *The product MDP $\mathcal{M} \times \mathcal{R}_\phi$ between a labeled MDP $\mathcal{M} = (Q, q_0, U, A, P, \Pi, h)$ and a DRA $\mathcal{R}_\phi = (S, s_0, 2^{\Pi}, \delta, F)$ is a MDP $\mathcal{P} = (S_{\mathcal{P}}, s_{\mathcal{P}0}, U_{\mathcal{P}}, A_{\mathcal{P}}, P_{\mathcal{P}}, F_{\mathcal{P}})$ where*:

(i) *$S_{\mathcal{P}} = Q \times S$ is a set of states;*
(ii) *$s_{\mathcal{P}0} = (q_0, s_0)$ is the initial state;*
(iii) *$U_{\mathcal{P}} = U$ is a set of actions inherited from $\mathcal{M}$;*
(iv) *$A_{\mathcal{P}}$ is also inherited from $\mathcal{M}$ and $A_{\mathcal{P}}((q,s)) := A(q)$;*
(v) *$P_{\mathcal{P}}$ gives the transition probabilities*

$$P_{\mathcal{P}}((q_1, s_1), u, (q_2, s_2)) = \begin{cases} P(q_1, u, q_2), & \text{if } q_2 = \delta(s_1, h(q_1)), \\ 0, & \text{otherwise} \end{cases}$$

(vi) *$F_{\mathcal{P}} = \{(L_{\mathcal{P}}(1), K_{\mathcal{P}}(1)),\ \ldots,(L_{\mathcal{P}}(M), K_{\mathcal{P}}(M))\}$ is a set of accepting states, where $L_{\mathcal{P}}(i) = Q \times L(i)$, $K_{\mathcal{P}}(i) = Q \times K(i)$, for $i = 1,\ldots,M$.*

The product MDP is constructed such that, given a path $(s_0,\ q_0)(s_1,\ q_1)\ldots$, the corresponding path $s_0 s_1\ldots$ on $\mathcal{M}$ satisfies ϕ if and only if there exists a pair $(L_{\mathcal{P}}, K_{\mathcal{P}}) \in F_{\mathcal{P}}$ satisfying the Rabin acceptance condition, i.e. the set $K_{\mathcal{P}}$ is visited infinitely often and the set $L_{\mathcal{P}}$ is visited finitely often.

We can make a very similar product between a labeled NTS $\mathcal{M}^{\mathcal{N}} = (Q, q_0, U, A, P^{\mathcal{N}}, \Pi, h)$ and $\mathcal{R}_\phi$. This product is also a NTS, which we denote by $\mathcal{P}^{\mathcal{N}} = (S_{\mathcal{P}}, s_{\mathcal{P}0}, U_{\mathcal{P}}, A_{\mathcal{P}}, P^{\mathcal{N}}_{\mathcal{P}}, \Pi, F_{\mathcal{P}}) := \mathcal{M}^{\mathcal{N}} \times \mathcal{R}_\phi$. The definition (and the accepting condition) of $\mathcal{P}^{\mathcal{N}}$ is exactly the same as for the product MDP. The only difference between $\mathcal{P}^{\mathcal{N}}$ and $\mathcal{P}$ is in $P^{\mathcal{N}}_{\mathcal{P}}$, which is either 0 or 1 for every state–action–state tuple.

From the product $\mathcal{P}$ or equivalently $\mathcal{P}^{\mathcal{N}}$, we can proceed to construct the MRP problem. To do so, it is necessary to produce the so-called *accepting maximum end components* (AMECs). An end component is a subset of a MDP (consisting of a subset of states and a subset of enabled actions at each state) such that for each pair of states (*i*, *j*) in $\mathcal{P}$, there is a sequence of actions such that *i* can be reached from *j* with positive probability, and states outside the component cannot be reached. The definition of AMECs is as follows (Baier and Katoen, 2008).

Definition 3.3 (Accepting maximal end components). *Given $(L_{\mathcal{P}}, K_{\mathcal{P}}) \in F_{\mathcal{P}}$, an AMEC of $\mathcal{P}$ is the largest end component containing at least one state in $K_{\mathcal{P}}$ and no state in $L_{\mathcal{P}}$, for a pair $(K_{\mathcal{P}}, L_{\mathcal{P}}) \in F_{\mathcal{P}}$.*

If an execution of a product MDP reaches a state in an AMEC, any state outside the AMEC will not be reachable under any policy. A procedure to obtain all AMECs of a MDP is outlined by Baier and Katoen (2008). This procedure is intended to be used for the product MDP $\mathcal{P}$, but it can be used without modification to find all AMECs associated with $\mathcal{P}$ when $\mathcal{P}^{\mathcal{N}}$ is used instead of $\mathcal{P}$. This is because the information needed to construct the AMECs is the set of all possible state transitions at each state, and this information is already contained in $\mathcal{P}^{\mathcal{N}}$. Note that the computation of AMECs, whose time complexity is quadratic to the size of $\mathcal{P}$ or $\mathcal{P}^{\mathcal{N}}$, cannot be avoided in any method as long as the LTL formula is not co-safe (Baier and Katoen, 2008). As a result, we exclude the time of calculating AMECs when comparing our actor–critic algorithm with alternative methods. For a co-safe LTL formula, this computation cost can be avoided by using deterministic finite automaton (DFA) instead of DRA (Baier and Katoen, 2008).

If we denote by $S^{\star}_{\mathcal{P}}$ as the union of all states in all AMECs associated with $\mathcal{P}$, it has been shown in probabilistic model checking (see e.g. Baier and Katoen, 2008) that maximizing the probability of satisfying the LTL formula is equivalent to a MRP problem whose definition is as follows.

Problem 3.4. *Given a product MDP $\mathcal{P} = (S_{\mathcal{P}}, s_{\mathcal{P}0}, U_{\mathcal{P}}, A_{\mathcal{P}}, P_{\mathcal{P}}, F_{\mathcal{P}})$ and a set of states $S^{\star}_{\mathcal{P}} \subseteq S_{\mathcal{P}}$, find the optimal policy μ that maximizes the probability of reaching the set $S^{\star}_{\mathcal{P}}$.*

If transition probabilities are available for each state–action pair, then Problem 3.4 can be solved by a linear program (see Puterman, 1994; Baier and Katoen, 2008). The resultant optimal policy is deterministic and is a huge table containing optimal controls for each state *s* in $\mathcal{P}$. In this paper, we approximate the optimal policy μ with a parameterized policy $\mu_{\boldsymbol{\theta}}$, which improves computation efficiency by taking advantage of prior knowledge of the policy structure.

By the definition of AMECs, if an execution of the product MDP reaches $S^{\star}_{\mathcal{P}}$, it cannot leave it. We call such a state set *absorbing state set*. Intuitively, the only case when the state does not reach $S^{\star}_{\mathcal{P}}$ is because it is "trapped" in another set of states.

Definition 3.5 (Trap state). *A trap state is a state on product MDP $\mathcal{P}$ that cannot reach $S^{\star}_{\mathcal{P}}$ under any policy.*

We will describe the method to calculate trap states in Section 3.4. Denoting by $\bar{S}^{\star}_{\mathcal{P}}$ the set of all trap states, we present the following theorem.

Theorem 3.6. *The set $\bar{S}^{\star}_{\mathcal{P}}$ is an absorbing state set of product MDP $\mathcal{P}$. Furthermore, in product MDP $\mathcal{P}$, there is no absorbing state set in states $S_{\mathcal{P}}/(\bar{S}^{\star}_{\mathcal{P}} \cup S^{\star}_{\mathcal{P}})$.*

Proof. Assume that there is a state $s^A_{\mathcal{P}} \in \bar{S}^{\star}_{\mathcal{P}}$ and a state $s^B_{\mathcal{P}} \notin \bar{S}^{\star}_{\mathcal{P}}$ such that $s^B_{\mathcal{P}}$ is accessible from $s^A_{\mathcal{P}}$ under a certain policy. By definition of $\bar{S}^{\star}_{\mathcal{P}}$, $S^{\star}_{\mathcal{P}}$ should not be accessible from $s^A_{\mathcal{P}}$. However, $s^{\star}_{\mathcal{P}}$ is accessible from $s^B_{\mathcal{P}}$, thus, it is accessible from $s^A_{\mathcal{P}}$. This contradiction shows that no state out of $\bar{S}^{\star}_{\mathcal{P}}$ is accessible from any state in $\bar{S}^{\star}_{\mathcal{P}}$, thus, $\bar{S}^{\star}_{\mathcal{P}}$ is an absorbing state set.

Suppose also that there exists some other absorbing state set that does not intersect with either $S^{\star}_{\mathcal{P}}$ or $\bar{S}^{\star}_{\mathcal{P}}$. By definition, states in such an absorbing state state should be trap states, thus, this set should be a subset of $\bar{S}^{\star}_{\mathcal{P}}$, which brings us to a contradiction. ■

Remark 3.7. *Theorem 3.6 suggests that Problem 3.4 is equivalent to the problem of minimizing the probabilities of reaching the set $\bar{S}^{\star}_{\mathcal{P}}$.*

Remark 3.8. *It is only necessary to find the optimal policy for states not in the set $S^{\star}_{\mathcal{P}}$. This is because, by construction, there exists a policy inside any AMEC that almost surely satisfies the LTL formula ϕ by reaching a state in $K_{\mathcal{P}}$ infinitely often. This policy can be obtained by simply choosing an action (among the subset of actions retained by the AMEC) at each state randomly, i.e. a trivial randomized stationary policy exists that almost surely satisfies ϕ.*

3.2. Conversion from MRP to a SSP problem

Although a linear program can be used to solve Problem 3.4, it has high time and memory complexity. In particular, it requires transition probabilities for all states of the product MDP. In many situations, we only have a simulator to generate the transition probabilities on the fly, and due to the large state space, we want to avoid simulating transitions at all states.

The so-called approximate dynamic programming (also known as neuro-dynamic programming or reinforcement learning) can help solve these issues (Bertsekas, 1995; Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998). In order to apply techniques in approximate dynamic programming, we need to convert the MRP problem into a stochastic shortest path (SSP) problem. We define the following new MDP based on the product MDP.

Definition 3.9 (SSP MDP). *Given the product MDP $\mathcal{P} = (S_\mathcal{P}, s_{\mathcal{P}0}, U_\mathcal{P}, A_\mathcal{P}, P_\mathcal{P}, F_\mathcal{P})$ and a set of states $S^\star_\mathcal{P} \subseteq S_\mathcal{P}$, define a new MDP $\widetilde{\mathcal{P}} = (\tilde{S}_\mathcal{P}, \tilde{s}_{\mathcal{P}0}, \tilde{U}_\mathcal{P}, \tilde{A}_\mathcal{P}, \tilde{P}_\mathcal{P})$, where*:

(i) $\tilde{S}_\mathcal{P} = (S_\mathcal{P} \setminus S^\star_\mathcal{P}) \cup \{s^\star_\mathcal{P}\}$, *where $s^\star_\mathcal{P}$ is a "dummy" terminal state;*
(ii) $\tilde{s}_{\mathcal{P}0} = s_{\mathcal{P}0}$ *(without loss of generality, we exclude the trivial case where $s_{\mathcal{P}0} \in S^\star_\mathcal{P}$)*;
(iii) $\tilde{U}_\mathcal{P} = U_\mathcal{P}$;
(iv) $\tilde{A}_\mathcal{P}(s_\mathcal{P}) = A_\mathcal{P}(s_\mathcal{P})$ *for all $s_\mathcal{P} \in S_\mathcal{P}$, and for the dummy state we set $\tilde{A}_\mathcal{P}(s^\star_\mathcal{P}) = \tilde{U}_\mathcal{P}$;*
(v) *The transition probabilities is redefined as follows*

$$\tilde{P}_\mathcal{P}(s_\mathcal{P}, u, \tilde{s}_\mathcal{P}) = \begin{cases} \sum_{s \in S^\star_\mathcal{P}} P_\mathcal{P}(s_\mathcal{P}, u, s), & \text{if } \tilde{s}_\mathcal{P} = s^\star_\mathcal{P}, \\ P_\mathcal{P}(s_\mathcal{P}, u, \tilde{s}_\mathcal{P}), & \text{if } \tilde{s}_\mathcal{P} \in S_\mathcal{P} \setminus S^\star_\mathcal{P}, \end{cases}$$

for all $s_\mathcal{P} \in S_\mathcal{P} \setminus (S^\star_\mathcal{P} \cup \bar{S}^\star_\mathcal{P})$ and $u \in \tilde{U}_\mathcal{P}$. Moreover, for all $s_\mathcal{P} \in \bar{S}^\star_\mathcal{P}$ and $u \in \tilde{U}_\mathcal{P}$, we set $\tilde{P}_\mathcal{P}(s^\star_\mathcal{P}, u, s^\star_\mathcal{P}) = 1$ and $\tilde{P}_\mathcal{P}(s_\mathcal{P}, u, s_{\mathcal{P}0}) = 1$;

For all $s_\mathcal{P} \in \tilde{S}_\mathcal{P}$ and $u \in \tilde{U}_\mathcal{P}$, we define a one-step cost function $\tilde{g}_\mathcal{P}(s_\mathcal{P}, u) = 1$ if $s_\mathcal{P} \in \bar{S}^\star_\mathcal{P}$, and $\tilde{g}_\mathcal{P}(s_\mathcal{P}, u) = 0$ otherwise. Then the SSP problem is defined as follows.

Problem 3.10. *Given a SSP MDP $\widetilde{\mathcal{P}} = (\tilde{S}_\mathcal{P}, \tilde{s}_{\mathcal{P}0}, \tilde{U}_\mathcal{P}, \tilde{A}_\mathcal{P}, \tilde{P}_\mathcal{P})$ and a one-step cost function $\tilde{g}_\mathcal{P}$, find a policy μ to minimize the expected total cost*

$$\bar{\alpha}_\mu = E\left\{ \sum_{k=0}^{T^\star} \tilde{g}_\mathcal{P}(x_k, u_k) \right\} \tag{1}$$

where $T^\star$is the first time when $s^\star_\mathcal{P}$ is reached, and x_k and u_k are the state and the action at time k, respectively.

Remark 3.11. *In product MDP $\mathcal{P}$, both $S^\star_\mathcal{P}$ and $\bar{S}^\star_\mathcal{P}$ are absorbing state sets. In contrast, in SSP MDP $\widetilde{\mathcal{P}}$, $s^\star_\mathcal{P}$ is the only absorbing state; whenever the state reaches $\bar{S}^\star_\mathcal{P}$, it returns to the initial state $\tilde{s}_{\mathcal{P}0}$ (as if the process restarts). The expected total cost $\bar{\alpha}_\mu$ in (1) is the expected total number of falls into $\bar{S}^\star_\mathcal{P}$ before reaching $s^\star_\mathcal{P}$ in SSP MDP $\widetilde{\mathcal{P}}$.*

Let $\mathrm{R}^\mathcal{P}_\mu$ be the reachability probability in Problem 3.4 under policy μ. The following lemma presents the relationship between $\mathrm{R}^\mathcal{P}_\mu$ and the expected total cost $\bar{\alpha}_\mu$ defined in (1).

Lemma 3.12. *For any RSP μ, we have $\mathrm{R}^\mathcal{P}_\mu = 1/\left(\bar{\alpha}_\mu + 1\right)$.*

Proof. According to the definition of the cost function, we know that $\bar{\alpha}_\mu$ is the expected number of times when states in $\bar{S}^\star_\mathcal{P}$ are visited before the termination state $s^\star_\mathcal{P}$ is reached in SSP MDP $\widetilde{\mathcal{P}}$. From the construction of $\widetilde{\mathcal{P}}$, reaching $s^\star_\mathcal{P}$ in $\widetilde{\mathcal{P}}$ is equivalent to reaching one of the goal states $S^\star_\mathcal{P}$ in $\mathcal{P}$. On the other hand, in the Markov chain generated by applying policy μ to $\mathcal{P}$, the states $S^\star_\mathcal{P}$ and $\bar{S}^\star_\mathcal{P}$ are the only absorbing state sets, and all other states are transient. Thus, the probability of visiting a state in $\bar{S}^\star_\mathcal{P}$ from $s_{\mathcal{P}0}$ on $\mathcal{P}$ is $1 - \mathrm{R}^\mathcal{P}_\mu$, which is the same as the probability of visiting $\bar{S}^\star_\mathcal{P}$ for each run of $\widetilde{\mathcal{P}}$, due to the construction of transition probabilities in Definition 3.9. We can now consider a geometric distribution where the probability of success is $\mathrm{R}^\mathcal{P}_\mu$. Because $\bar{\alpha}_\mu$ is the expected number of times when a state in $\bar{S}^\star_\mathcal{P}$ is visited before $s^\star_\mathcal{P}$ is reached, this is the same as the expected number of failures of Bernoulli trails (with probability of success being $\mathrm{R}^\mathcal{P}_\mu$) before a success. This implies $\bar{\alpha}_\mu = \frac{1-\mathrm{R}^\mathcal{P}_\mu}{\mathrm{R}^\mathcal{P}_\mu}$.

Lemma 3.12 states that the policy minimizing (1) for Problem 3.10 with MDP $\widetilde{\mathcal{P}}$ and the termination state $s^\star_\mathcal{P}$ is a policy maximizing the probability of reaching the set $S^\star_\mathcal{P}$ on $\mathcal{P}$, i.e. a solution to Problem 3.4.

Problem 3.10 can also be constructed from the NTS $\mathcal{P}^\mathcal{N}$. In this case we obtain an NTS $\widetilde{\mathcal{P}}^\mathcal{N}(\tilde{S}_\mathcal{P}, \tilde{s}_{\mathcal{P}0}, \tilde{U}_\mathcal{P}, \tilde{A}_\mathcal{P}, \tilde{P}^\mathcal{N}_\mathcal{P})$, using the exact same construction as Definition 3.9, except for the definition of $\tilde{P}^\mathcal{N}_\mathcal{P}$. The transition function $\tilde{P}^\mathcal{N}_\mathcal{P}(s_\mathcal{P}, u, \tilde{s}_\mathcal{P})$ is instead defined as

$$\tilde{P}^\mathcal{N}_\mathcal{P}(s_\mathcal{P}, u, \tilde{s}_\mathcal{P}) = \begin{cases} \max_{s \in S^\star_\mathcal{P}} P^\mathcal{N}_\mathcal{P}(s_\mathcal{P}, u, s), & \text{if } \tilde{s}_\mathcal{P} = s^\star_\mathcal{P} \\ P^\mathcal{N}_\mathcal{P}(s_\mathcal{P}, u, \tilde{s}_\mathcal{P}), & \text{if } \tilde{s}_\mathcal{P} \in S_\mathcal{P} \setminus S^\star_\mathcal{P} \end{cases}$$

for all $s_\mathcal{P} \in S_\mathcal{P} \setminus (S^\star_\mathcal{P} \cup \bar{S}^\star_\mathcal{P})$ and $u \in \tilde{U}_\mathcal{P}$. Moreover, for all $s_\mathcal{P} \in \bar{S}^\star_\mathcal{P}$ and $u \in \tilde{U}_\mathcal{P}$, we set $\tilde{P}^\mathcal{N}_\mathcal{P}(s^\star_\mathcal{P}, u, s^\star_\mathcal{P}) = 1$ and $\tilde{P}^\mathcal{N}_\mathcal{P}(s_\mathcal{P}, u, s_{\mathcal{P}0}) = 1$.

In many robotic applications, the NTS $\widetilde{\mathcal{P}}^\mathcal{N}$ can be quickly constructed for the robot in the environment and a simulator is available to generate "transition probabilities" on the fly. In our method, it is sufficient to have the NTS $\widetilde{\mathcal{P}}^\mathcal{N}$ and a simulator to calculate transition probabilities.

3.3. LSTD actor–critic method

In this section, we propose an action-critic method that obtains an RSP (see Definition 2.2) $M = \mu_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}\ldots$, where $\mu_{\boldsymbol{\theta}}(x, u)$ is a function of the state–action pair (x, u) and $\theta \in \mathbb{R}^n$, which is a vector of parameters. For convenience, we denote an RSP $\mu_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}\ldots$ simply by $\mu_{\boldsymbol{\theta}}$. In addition, we denote the expected total cost defined in (1) by $\bar{\alpha}(\theta)$. In this section we assume the RSP $\mu_{\boldsymbol{\theta}}(q, u)$ to be given, and we will describe in Section 3.4 how to design a suitable RSP. Similar to Konda (2002), we also assume that either $\inf_{\boldsymbol{\theta}}$ $\mu_{\boldsymbol{\theta}} > 0$ or $\mu_{\boldsymbol{\theta}} \equiv 0$, for all $\boldsymbol{\theta}$.

Given an RSP $\mu_{\boldsymbol{\theta}}$, we apply an iterative procedure, i.e. an actor–critic method, to obtain a policy that locally minimizes the cost function (1) by simulating sample paths on $\widetilde{\mathcal{P}}$. Each sample path on $\widetilde{\mathcal{P}}$ starts at $\tilde{s}_{\mathcal{P}0}$ and ends when the termination state $s^\star_\mathcal{P}$ is reached. Since the probabilities are

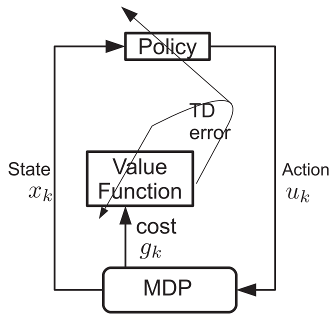

Fig. 2. Illustration of actor–critic methods (adapted from Konda, 2002).

needed only along the sample path, we do not require the MDP $\tilde{\mathcal{P}}$, but only $\tilde{\mathcal{P}}^{\mathcal{N}}$.

As suggested by the name, actor–critic algorithms have two learning units, an actor and a critic, interacting with each other and with the MDP during the iterations of the algorithms. At each iteration, the critic observes the state and the one-step cost from the MDP and uses the observed information to update a value function. The value function is then used to update the RSP; the actor generates the action based on the RSP and applies the action to the MDP (cf. Figure 2). The algorithm stops when the gradient of $\bar{\alpha}(\theta)$ is small enough (i.e. $\boldsymbol{\theta}$ is locally optimal).

Consider the SSP MDP $\tilde{\mathcal{P}}$. Let k denote time, $x_k \in \tilde{S}_{\mathcal{P}}$ and $u_k \in \tilde{A}_{\mathcal{P}}(x_k)$ be the state and the action taken at time k. Under a fixed policy $\mu_{\boldsymbol{\theta}}$, $\{x_k\}$ and $\{x_k, u_k\}$ are Markov chains with stationary distributions. We denote these stationary distributions as $\pi_{\boldsymbol{\theta}}(x)$ and $\eta_{\boldsymbol{\theta}}(x, u)$, respectively.

Let $P_{\boldsymbol{\theta}}$ be an operator to take expectation after one transition, namely, for a function $f(x, u)$,

$$(P_{\boldsymbol{\theta}} f)(x, u) = \sum_{j \in \tilde{S}_{\mathcal{P}}} \sum_{\nu \in \tilde{A}_{\mathcal{P}}(j)} \tilde{P}_{\mathcal{P}}(x, u, j) \mu_{\boldsymbol{\theta}}(j, \nu) f(j, \nu)$$

Define the function $Q_{\boldsymbol{\theta}}$ to be the function that satisfies the following Poisson equation:

$$Q_{\boldsymbol{\theta}}(x, u) = g_{\mathcal{P}}(x, u) + (P_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}})(x, u) \quad (2)$$

Here $Q_{\boldsymbol{\theta}}(x, u)$ can be interpreted as the expected total future cost after applying action u at state x and is named as the state–action value function (Sutton and Barto, 1998; Estanjini et al., 2011). Let

$$\psi_{\boldsymbol{\theta}}(x, u) = \nabla_{\boldsymbol{\theta}} \ln(\mu_{\boldsymbol{\theta}}(x, u)) \quad (3)$$

where $\psi_{\boldsymbol{\theta}}(x, u) = 0$ when x, u are such that $\mu_{\boldsymbol{\theta}}(x, u) \equiv 0$, for all $\boldsymbol{\theta}$. We assume that $\psi_{\boldsymbol{\theta}}(x, u)$ is bounded and continuously differentiable. For all states $x \in \tilde{S}_{\mathcal{P}}$ and actions $u \in \tilde{A}_{\mathcal{P}}(x)$, $\psi_{\boldsymbol{\theta}}(x, u)$ is an n-dimensional vector, where n is the dimensionality of $\boldsymbol{\theta}$. We write $\psi_{\boldsymbol{\theta}}(x, u) = (\psi^1_{\boldsymbol{\theta}}(x, u), \ldots, \psi^n_{\boldsymbol{\theta}}(x, u))$. Let $\langle \cdot, \cdot \rangle$ be an inner product operator defined as follows:

$$\nabla \langle f_1, f_2 \rangle = \sum_{x \in \tilde{S}_{\mathcal{P}}} \sum_{u \in \tilde{A}_{\mathcal{P}}(x)} \eta_{\boldsymbol{\theta}}(x, u) f_1(x, u) f_2(x, u) \quad (4)$$

where $f_1(x, u)$ and $f_2(x, u)$ are two functions. It has been proved that the gradient of the expected total cost $\bar{\alpha}(\boldsymbol{\theta})$ is equal to (Konda, 2002)

$$\nabla \bar{\alpha}(\boldsymbol{\theta}) = \langle Q_{\boldsymbol{\theta}}, \psi_{\boldsymbol{\theta}} \rangle \quad (5)$$

Instead of storing $Q_{\boldsymbol{\theta}}$ explicitly, which is a huge table, we approximate $Q_{\boldsymbol{\theta}}$ with a linear architecture of the following form

$$Q^{\mathbf{r}}_{\boldsymbol{\theta}}(x, u) = \psi'_{\boldsymbol{\theta}}(x, u)\mathbf{r}, \quad \mathbf{r} \in \mathbb{R}^n \quad (6)$$

Let $\|\cdot\|$ be the norm induced by the inner product operator $\langle \cdot, \cdot \rangle$, i.e. $\|f\|^2 = \langle f, f \rangle$. The optimal coefficient $\mathbf{r}^\star$ should satisfy

$$\mathbf{r}^\star = \arg\min_{\mathbf{r}} \left\| Q_{\boldsymbol{\theta}} - Q^{\mathbf{r}}_{\boldsymbol{\theta}} \right\| \quad (7)$$

Temporal difference algorithms can be leveraged to learn the optimal coefficient $\mathbf{r}^\star$. As a property of inner products, the linear approximation in (6) does not change the estimate of the gradient $\nabla \bar{\alpha}(\boldsymbol{\theta})$ if the optimal coefficient $\mathbf{r}^\star$ in (7) is used. Furthermore, the linear approximation reduces the complexity of learning from the space $\mathbb{R}^{|\mathbb{X}||\mathbb{U}|}$ to the space $\mathbb{R}^n$, where n is the dimensionality of $\boldsymbol{\theta}$ (Konda, 2002; Estanjini et al., 2012).

In this paper, we present an actor–critic algorithm that uses LSTD learning to estimate $\mathbf{r}^\star$. We summarize the algorithm, which is referred to as LSTD actor–critic algorithm, in Algorithm 1, and we note that it does not depend on the form of RSP $\mu_{\boldsymbol{\theta}}$. In each iteration, we first compute the transition probabilities $\tilde{P}_{\mathcal{P}}(x_k, u_k, \cdot)$ using a simulator (step 3 of Algorithm 1). Note that in contrast to the linear programming-based method of Ding et al. (2011), the transition probabilities are generated only when needed. This approach is suitable for problems with a large state space and strict memory requirements. Next, we obtain the simulated next state x_{k+1} based on the transition probabilities and obtain an action u_{k+1} based on the current RSP $\mu_{\boldsymbol{\theta}_k}$ (step 4 and 5 of Algorithm 1). Then, we update our estimate of $\mathbf{r}$ in the critic step (step 6 of Algorithm 1), and update our estimate of $\boldsymbol{\theta}$ in the actor step (step 7 of Algorithm 1).

In the critic step, $\mathbf{z}_k \in \mathbb{R}^n$ represents Sutton's eligibility trace (Sutton and Barto, 1998; Konda, 2002); $\mathbf{b}_k \in \mathbb{R}^n$ maintains a sample estimate for one-step cost; $\mathbf{A}_k \in \mathbb{R}^{n \times n}$ is a sample estimate for the matrix formed by $\mathbf{z}_k(\psi_{\boldsymbol{\theta}_k}(x_{k+1}, u_{k+1}) - \psi_{\boldsymbol{\theta}_k}(x_k, u_k))$. Here $\mathbf{r}_k$ is a least-squares estimate of $\mathbf{r}$.

In the actor step, $\mathbf{r}'_k \psi_{\boldsymbol{\theta}_k}(x_{k+1}, u_{k+1}) \psi_{\boldsymbol{\theta}_k}(x_{k+1}, u_{k+1})$ is a sample estimate of the gradient $\nabla \bar{\alpha}(\boldsymbol{\theta})$ (cf. (5)). The actor step is simply a gradient descent update. The role of $\Gamma(\mathbf{r})$ is mainly to keep the actor updates bounded, and we can for instance use $\Gamma(\mathbf{r}) = \min\left(\frac{D}{\|r\|}, 1\right)$ for some $D > 0$.

In Algorithm 1, $\{\gamma_k\}$ controls the critic step-size, while $\{\beta_k\}$ control the actor step-size together. We leave the proof of convergence for Algorithm 1 as well as the requirements for the step-sizes to the appendix.

Algorithm 1. LSTD actor–critic algorithm for Problem 3.10.

```
Input: The NTS $\widetilde{\mathcal{P}}^{\mathcal{N}}(\tilde{S}_{\mathcal{P}}, \tilde{s}_{\mathcal{P}0}, \tilde{U}_{\mathcal{P}}, \tilde{A}_{\mathcal{P}}, \tilde{P}^{\mathcal{N}}_{\mathcal{P}}, g_{\mathcal{P}})$ with the terminal state $s^{\star}_{\mathcal{P}}$, the RSP $\mu_{\boldsymbol{\theta}}$, and a computation tool to obtain $\tilde{P}_{\mathcal{P}}(s_{\mathcal{P}}, u, \cdot)$ for a given $(s_{\mathcal{P}}, u)$ state–action pair.
1: Initialization: Set all entries in $\mathbf{z}_0$, $\mathbf{b}_0$ and $\mathbf{r}_0$ to zeros. Set $\mathbf{A}_0$ to identity matrix. Let $\boldsymbol{\theta}_0$ take some initial value. Set initial state $x_0 := \tilde{s}_{\mathcal{P}0}$. Obtain action $u_0$ using the RSP $\mu_{\theta_0}$.
2: repeat
3:     Compute the transition probabilities $\tilde{P}_{\mathcal{P}}(x_k, u_k, \cdot)$.
4:     Obtain the simulated subsequent state $x_{k+1}$ using the transition probabilities $\tilde{P}_{\mathcal{P}}(x_k, u_k, \cdot)$. If $x_k = s^{\star}_{\mathcal{P}}$, set $x_{k+1} := x_0$.
5:     Obtain action $u_{k+1}$ using the RSP $\mu_{\theta_k}$
6:     Critic Update:
           $\mathbf{z}_{k+1} = \lambda \mathbf{z}_k + \psi_{\theta_k}(x_k, u_k)$
           $\mathbf{b}_{k+1} = \mathbf{b}_k + \gamma_k(\tilde{g}_{\mathcal{P}}(x_k, u_k)\mathbf{z}_k - \mathbf{b}_k)$
           $\mathbf{A}_{k+1} = \mathbf{A}_k + \gamma_k(\mathbf{z}_k(\psi'_{\theta_k}(x_{k+1}, u_{k+1}) - \psi'_{\theta_k}(x_k, u_k)) - \mathbf{A}_k)$
           $\mathbf{r}_{k+1} = -\mathbf{A}_k^{-1}\mathbf{b}_k$
7:     Actor Update:
           $\theta_{k+1} = \theta_k - \beta_k \Gamma(\mathbf{r}_k)\mathbf{r}'_k \psi_{\theta_k}(x_{k+1}, u_{k+1})\psi_{\theta_k}(x_{k+1}, u_{k+1})$
8: until $\|\nabla\bar{\alpha}(\theta_k)\| \le \epsilon$ for some given $\epsilon$.
```

Algorithm 1 learns the critic parameters using a LSTD method, which has been shown to be superior to other stochastic learning methods in terms of the convergence rate (Konda and Tsitsiklis, 2003; Boyan, 1999). Estanjini et al. (2012) proposed and established the convergence of a LSTD actor–critic method similar to Algorithm 1 for problems of minimizing *expected average costs*. In comparison, the goal of the Problem 3.10 in this paper is to minimize an *expected total cost* (cf. (1)). The output of Algorithm 1 is a (locally) optimal policy parameter $\boldsymbol{\theta}^{\star}$. The corresponding policy $\mu_{\boldsymbol{\theta}\star}$ (locally) maximizes the satisfaction probability of the RSP (12).

Compared with some existing efforts of applying approximate dynamic programming techniques in robot motion control, the actor–critic algorithm used in this paper is very suitable for large-scale problems. For example, the method in (Fu and Topcu, 2014) is based on value iteration. Value iteration is a classical dynamic programming technique, but it is suitable only for MDPs with small state spaces because of its high time and memory complexity (Bertsekas, 1995). In addition, value iteration requires transition probabilities at all states. The method in (Sadigh et al., 2014) is also based on temporal difference learning, however, it uses a lookup-table representation of the value function, i.e. it explicitly stores the expected future cost for each state. Again memory requirements can be enormous for large-sized problems. In contrast, in our actor–critic algorithm, only the coefficients $\mathbf{r}$ need to be stored since the value function $Q_{\boldsymbol{\theta}}(x, u)$ is expressed as linear combination of some basis functions. The problems considered in both Fu and Topcu (2014) and Sadigh et al. (2014) are much smaller than ours. It is worth noting that in the context of approximate dynamic programming, the poor scalability of value iteration and lookup-table-based temporal difference methods is an important motivation for actor–critic methods (Bertsekas and Tsitsiklis, 1996; Konda, 2002).

Theorem 3.13. *[Actor convergence] For the LSTD actor–critic algorithm (Algorithm 1) with some step-size sequence $\{\beta_k\}$ satisfying*

$$\sum_k \beta_k = \infty, \quad \sum_k \beta_k^2 < \infty, \quad \lim_{k\to\infty} \frac{\beta_k}{\gamma_k} = 0 \tag{8}$$

and for any $\epsilon > 0$, there exists some λ sufficiently close to 1, such that $\liminf_{k\to\infty}\|\nabla\bar{\alpha}(\boldsymbol{\theta}_k)\| < \varepsilon$ w.p.1. That is, $\boldsymbol{\theta}_k$ visits an arbitrary neighborhood of a stationary point infinitely often.

Proof. See the Appendix.

3.4. Designing a RSP

In this section we describe a RSP suitable to be used in Algorithm 1 for Problem 3.10. We first describe some "features", i.e. progress and safety scores for each state. Then we define some "desirability degrees" for controls in each state and propose an RSP based on the Boltzmann distribution.

We define the *overlay graph* $G_o = (V_o, E_o)$ for NTS $\widetilde{\mathcal{P}}^{\mathcal{N}}$ as an unweighted graph whose vertex set $V_0 = \tilde{S}_{\mathcal{P}}$. For any pair of states $i, j \in \tilde{S}_{\mathcal{P}}$, we have $(i, j) \in E_o$ if $\tilde{P}^{\mathcal{N}}_{\mathcal{P}}(i, u, j) = 1$ for some $u \in A_{\mathcal{P}}(i)$, i.e. j is reachable from i in one step in the NTS. Let $d_g(i)$ be the shortest distance from state i to the "dummy" terminal state $s^{\star}_{\mathcal{P}}$ in G_o. The distance $d_g(\cdot)$ can be efficiently calculated using breath-first search (BFS) with $O(|V_o| + |E_o|)$ time complexity and $O(|V_o|)$ space complexity (Cormen et al., 2001).

Recall that the trap states (cf. Def. 3.5) are states on $\widetilde{\mathcal{P}}^{\mathcal{N}}$ that cannot reach $s^{\star}_{\mathcal{P}}$ under any policy, hence, they have infinite distance $d_g(\cdot)$ to $s^{\star}_{\mathcal{P}}$. In order to obtain the trap state set $\bar{S}^{\star}_{\mathcal{P}}$, we need to calculate the distance $d_g(i)$ for all states i and let $\bar{S}^{\star}_{\mathcal{P}} = \{i : d_g(i) = \infty\}$. Note the LP method described in Ding et al. (2011) also needs to calculate $\bar{S}^{\star}_{\mathcal{P}}$ by calculating $d_g(\cdot)$ first.

We define the *progress score* of a state $i \in \tilde{S}_{\mathcal{P}}$ as

$$\text{prog}(i) := -d_g(i) \tag{9}$$

Larger *progress score* means the state is closer to $s^{\star}_{\mathcal{P}}$ in G_o, thus, it is more likely to hit $s^{\star}_{\mathcal{P}}$ in the near future.

Let μ_{null} be a "null policy" in which each action is chosen with equal probability. If we fix the policy to be this "null policy," the process $\{x_k, u_k\}$ becomes a Markov chain. In the following part, we calculate the "r-step transition probabilities" of this Markov chain, where r is a predefined parameter representing sensing range.

Suppose $\tilde{P}^{(r)}_{\mathcal{P}}(i,j)$ is the probability from state i to state j in r steps without reaching any state in $\bar{S}^{\star}_{\mathcal{P}}$ under μ_{null}. Here $\tilde{P}^{(r)}_{\mathcal{P}}(i, u, j)$ can be calculated recursively. For all i, j, u and $m = 2,\ldots,r$,

$$\tilde{P}^{(m)}_{\mathcal{P}}(i,j) = \sum_{x \in \tilde{S}_{\mathcal{P}}} \tilde{P}^{(m-1)}_{\mathcal{P}}(i,x)\tilde{P}^{(1)}(x,j)$$

where

$$\tilde{P}^{(1)}(i,j) = \begin{cases} 1, & \text{if } i \in \bar{S}^{\star}_{\mathcal{P}} \text{ and } i = j \\ 0, & \text{if } i \in \bar{S}^{\star}_{\mathcal{P}} \text{ and } i \neq j \\ \frac{1}{|\tilde{U}_{\mathcal{P}}|}\sum_{u \in \tilde{U}_{\mathcal{P}}} \tilde{P}_{\mathcal{P}}(i,u,j), & \text{otherwise} \end{cases}$$

Here $\tilde{P}^{(1)}_{\mathcal{P}}(i,j)$ is the one-step transition probability from state i to state j under μ_{null}. Define the *safety score* for state i as

$$\text{safe}(i) := \sum_j \tilde{P}^{(r)}_{\mathcal{P}}(i,j) I(j) \tag{10}$$

where $I(j)$ is an indicator function such that $I(i) = 1$ if and only if $i \in \tilde{S}_{\mathcal{P}} \setminus \bar{S}^{\star}_{\mathcal{P}}$ and $I(i) = 0$ otherwise. The *safety score* of a state is the probability of hitting trap states $\bar{S}^{\star}_{\mathcal{P}}$ in the following r steps under μ_{null}. Thus, a higher safety score for the current state implies that it is less likely to reach $\bar{S}^{\star}_{\mathcal{P}}$ in the near future.

Compared with the *safety score* defined in Ding et al. (2012), equation (10) is much more accurate. The *safety score* in Ding et al. (2012) is the proportion of non-trap states in the neighborhood of a state, which is problematic in many cases. For example, for a state that has only one trap state in its neighborhood but has a large transition probability to this trap state, it will have a high *safety score* according to Ding et al. (2012) despite the fact that it is quite unsafe.

The *desirability degree* of an action u is defined as

$$\begin{aligned} a(\boldsymbol{\theta}, i, u) = \theta_1\Big(\sum_j \tilde{P}_{\mathcal{P}}(i,u,j)\text{prog}(j) - \text{prog}(i)\Big) \\ + \theta_2\Big(\sum_j \tilde{P}_{\mathcal{P}}(i,u,j)\text{safe}(j) - \text{safe}(i)\Big) \end{aligned} \tag{11}$$

which is a weighted sum of two contributing terms, and $\boldsymbol{\theta} = (\theta_1, \theta_2)$ is the vector of corresponding weights. The first term is based on the influence of this action on improving progress, and the second term is based on its influence on improving safety.

Our RSP is constructed using the Boltzmann distribution. The probability of taking action u at state i is defined as

$$\mu_{\boldsymbol{\theta}}(i,u) = \frac{\exp(a(\boldsymbol{\theta}, i, u)/T)}{\sum_{u \in \tilde{U}_{\mathcal{P}}} \exp(a(\boldsymbol{\theta}, i, u)/T)} \tag{12}$$

where T is the temperature of the Boltzmann distribution.

As a comparison, and to alleviate the influence of the poor *safety score* we discussed earlier, the RSP in Ding et al. (2012)"looks multiple steps ahead," i.e. considers all possible sequences of actions in the several following steps. Although such an RSP helps performance, its computation cost is quite high. For a state that is visited multiple times, the procedure of "looking multiple steps ahead" needs to be performed at every visit. To address this problem, we simplify the RSP structure and incorporate the "look ahead" information into the *safety score* in (10), which can be reused at every subsequent visit. Intuitively, the *safety score* in (10) can be viewed as some information left in a state by a "vanguard" robot that uses policy μ_{null} and looks r steps ahead at every state.

There is a well-known tradeoff between *exploitation* and *exploration* in designing RSPs (Sutton and Barto, 1998). A RSP will have higher *exploitation* if it is more greedy, i.e. it is more likely to only pick the action with the highest desirability degree. However, in each step, the *exploration* of undesirable actions is necessary because they may be desirable in the long run. High *exploitation* and low *exploration* may result in a sub-optimal solution. On the contrary, low *exploitation* and high *exploration* may reduce the convergence rate of the actor–critic algorithm. Based on the Boltzmann distribution, our RSP defined in (12) is flexible because tuning T in (12) can effectively adjust the degree of *exploration*. High-temperature results in more *exploration*

Algorithm 2. Overall algorithm providing a solution to Problem 2.4.

```
Input: A labeled NTS $\mathcal{M}^{\mathcal{N}} = (Q, q_0, U, A, P^{\mathcal{N}}, \Pi, h)$ modeling a robot in a partitioned environment, LTL formula $\phi$ over $\Pi$, and a
       simulator to compute $P(q, u, \cdot)$ given a state–action pair $(q, u)$.
  1: Translate the LTL formula $\phi$ to a DRA $\mathcal{R}_{\varphi}$.
  2: Generate the product NTS $\mathcal{P}^{\mathcal{N}} = \mathcal{M}^{\mathcal{N}} \times \mathcal{R}_{\varphi}$.
  3: Find the union of all AMECs $S^{\star}_{\mathcal{P}}$ associated with $\mathcal{P}^{\mathcal{N}}$.
  4: Convert from a MRP to a SSP and generate $\widetilde{\mathcal{P}}^{\mathcal{N}}$.
  5: Obtain the RSP $\mu_{\boldsymbol{\theta}}$ with $\widetilde{\mathcal{P}}^{\mathcal{N}}$.
  6: Execute Algorithm 1 with $\widetilde{\mathcal{P}}^{\mathcal{N}}$ and $\mu_{\boldsymbol{\theta}}$ as inputs until $\left\|\nabla\bar{\alpha}(\theta^{\star})\right\| \leq \epsilon$ for a $\boldsymbol{\theta}^{\star}$ and a given $\epsilon$.
Output: RSP $\mu_{\boldsymbol{\theta}}$ and $\boldsymbol{\theta}^{\star}$ locally maximizing the probability of satisfying  with respect to $\boldsymbol{\theta}$ up to a threshold $\epsilon$.
```

and vice versa. A large T also makes the RSP more random while a small T makes the RSP more deterministic.

3.5. Overall algorithm

We now connect all of the pieces together and present the overall algorithm giving a solution to Problem 2.4.

Proposition 3.14. *For an LTL formula ϕ, let R^{ϕ}_{θ} be the satisfaction probability of RSP $\mu_{\boldsymbol{\theta}}$ for ϕ. For any given ϵ, Algorithm 2 returns a locally optimal $\boldsymbol{\theta}^{\star}$ such that $\left\|\nabla R^{\mathcal{P}}_{\boldsymbol{\theta}^{\star}}\right\| \leq \epsilon$.*

Proof. Denote by $\mathcal{P}_{\phi}$ the product MDP for ϕ. For any $\boldsymbol{\theta}$ and RSP $\mu_{\boldsymbol{\theta}}$, let $R^{\mathcal{P}_{\phi}}_{\boldsymbol{\theta}}$ be the reachability probability in Problem 3.4 and $\bar{\alpha}(\boldsymbol{\theta})$ be the expected total cost in Problem 3.10. Let $\boldsymbol{\theta}^{\star}$ be the outcome of Algorithm 1 for Problem 3.10; then $\left\|\nabla\bar{\alpha}(\boldsymbol{\theta}^{\star})\right\| \leq \epsilon$ for a given ϵ. In addition, we have $\bar{\alpha}(\boldsymbol{\theta}^{\star}) = 1/R^{\mathcal{P}_{\phi}}_{\boldsymbol{\theta}^{\star}} - 1$ according to Lemma 3.12 and $R^{\phi}_{\boldsymbol{\theta}^{\star}} = R^{\mathcal{P}_{\phi}}_{\boldsymbol{\theta}^{\star}} \leq 1$ by definition. As a result, $\left\|\nabla R^{\phi}_{\boldsymbol{\theta}^{\star}}\right\| \leq (R^{\phi}_{\boldsymbol{\theta}^{\star}})^2\epsilon \leq \epsilon$. ■

4. Simulation results

We test the algorithms proposed in this paper through simulation in the RIDE environment (as shown in Figure 1). The transition probabilities are computed by an accurate simulator of RIDE as needed (cf. Section 4.2). We compare the results of our actor–critic method with the results of the method in Ding et al. (2011), which is referred to as the LP method. We analyze the movements of robots under the policy calculated by our actor–critic method in a 21×21 scenario. We also analyze the increase of time and memory usages of both methods when the problem size increases.

4.1. Environment

In this case study, we consider environments with topologies such as that shown in Figure 3. Such environments are made of square blocks formed by corridors and intersections. In the case shown in Figure 3, the corridors (C_1, C_2, …, C_{164}) are of one- or three-unit lengths. The three-

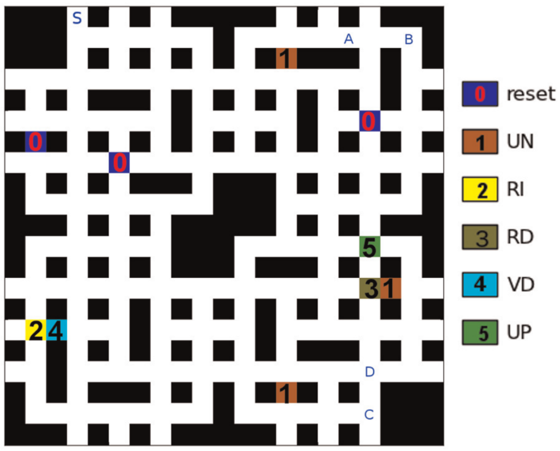

Fig. 3. An example schematic representation of an environment with intersections and corridors. The black blocks represent walls, and the white regions are intersection and corridors. There are five properties of interest in the regions indicated with **VD** = *ValuableData*, **RD** = *RegularData*, **Up** = *Upload*, **Ri** = *Risky*, and **Un** = *Unsafe*. There are also some reset states in the scene. Regions with properties of interests are plotted in corresponding colors. The correspondence of properties and colors are shown in the legend. A is a corridor with length one. In our settings, corridors connected by two-way intersections form a longer corridor. B marks an L-shape corridor formed by two 1-block corridors that are connected by a two-way intersection. C and D are examples of three- and four-way intersections. The initial position of the robot is marked with a blue **S**.

unit corridors are used to build corners in the environment. The intersections (I_1, I_2, …, I_{84}) are of two types, three-way and four-way. The black regions in this figure represent the walls of the environment. Note that there is always a corridor between two intersections. Thus, we can recursively build larger scenes by concatenating smaller scenes and merging the one-length corridors on the borders.

There are five properties of interest associated with the regions of the environment. These properties are: **VD** = *ValuableData* (regions containing valuable data to be collected), **RD** = *RegularData* (regions containing regular data to be collected), **Up** = *Upload* (regions where data can be uploaded), **Ri** = *Risky* (regions that could pose a threat to the robot), and **Un** = *Unsafe* (regions that are unsafe for the robot). If the robot reaches an unsafe state, it will break down and will not finish the task. There are also some **reset** states in the scene. Whenever a robot reaches a **reset** state, it will be removed from the scene. A mission ends when the robot reaches a reset state, no matter whether the task specification has already been met or not. Our goal is to find a control strategy for the robot to maximize the probability of finishing a task specified as an LTL formula over the set of properties before the mission ends. Note that while **VD**, **RD**, **Up**, **Ri**, and **Un** are properties that comprise the LTL formula, **reset** states only trigger the end of a mission and should not appear in the LTL formula.

4.2. Construction of the MDP model

Let x_k, y_k be the position of the robot at time k. Denote by v_k and ω_k the speed and the heading of the robot, respectively. Then the movement of the robot can be described using the following unicycle model.

$$\begin{aligned} x_{k+1} &= x_k + v_k \cos(\omega_k) + N(0, \sigma_x^2) \\ y_{k+1} &= y_k + v_k \sin(\omega_k) + N(0, \sigma_y^2) \\ v_{k+1} &= v_k + N(0, \sigma_v^2) \\ \omega_{k+1} &= \omega_k + \rho(\omega_d - \omega_k) + N(0, \sigma_\omega^2) \end{aligned}$$

where ω_d is the desired heading, ρ is the angular feedback coefficient, and σ_ω determines the noise actuation noise. Here σ_v affects the speed of the robot, while σ_x and σ_y characterize the roughness of the surface in x and y directions, respectively.

The robot is equipped with a set of feedback control primitives (actions): `GoLeft`, `GoForward`, `GoRight`, and `GoBackward`. Due to the presence of noise in the actuators and sensors, however, the resulting motion may be different than intended. Thus, the outcome of each control primitive is characterized probabilistically.

To create a MDP model of the robot in RIDE, we define each state of the MDP as a collection of two adjacent regions (a corridor and an intersection). The first region in the pair represents the last location of the robot, and the second region in the pair represents the current location. For instance, the pairs C_1–I_2 and I_3–C_4 are two states of the MDP. If the robot is in C_1–I_2 at time k, then it is in intersection I_2 at time k and was in corridor C_1 at time $k-1$. If the robot is in I_3–C_4 at time k, then it is in corridor C_4 at time k and was in intersection I_3 at time $k-1$.

Through this pairing of regions, it was shown that the Markov property (i.e. the result of an action at a state depends only on the current state) can be achieved for the motion of the robot in RIDE (Lahijanian et al., 2012). The resulting MDP has 608 states. The set of actions available at a state is the set of controllers available at the second region corresponding to the state. For example, when in state C_1–I_2 only those actions from region I_2 are allowed. Each state of the MDP whose second region satisfies a property in Π is mapped to that property.

To obtain accurate transition probabilities, we use the unicycle model described earlier to simulate transition probabilities of the labeled MDP. Note that each MDP state is a tuple of two regions, the second one is the current region and the first one is the previous region. To simulate transition probabilities of an MDP state to its neighboring states, we put the robot in the center of the second region (current region), and the initial heading ω_1 is determined by the relative position of the previous region and the current region. A simulation ends if the robot moves out of the current region to a new region. The new MDP state will have the current region as the first element and the new region as the second element. For each action available in each MDP state, we performed a total of 1000 Monte Carlo simulations, and use the normalized frequency as our estimate of the transition probabilities. For example, for state C_1–I_2 and action `GoLeft`, if the robot goes to C_2, C_3, C_4 for 450, 200, 350 times, respectively, then under action `GoLeft`, the transition probabilities from state C_1–I_2 to state I_2–C_2, I_2–C_3, I_2–C_4 are 0.45, 0.2, 0.35, respectively.

4.3. Task specification and results

We consider the following mission, composed of three subtasks (with no restriction on the order):

- reach a location with *ValuableData* (**VD**), then reach *Upload* (**Up**);
- (repeat twice) reach a location with *RegularData* (**RD**), then reach *Upload* (**Up**); with the following requirements:
- always avoid *Unsafe* (**Un**) regions;
- do not reach *Risky* (**Ri**) regions unless directly preceding a location with *ValuableData* (**VD**);
- after getting either *ValuableData* (**VD**) or *RegularData* (**RD**), go to *Upload* (**Up**) before going for another *ValuableData* (**VD**) or *RegularData* (**RD**).

The above task specification can be translated to the LTL formula:

$$\begin{aligned} \phi := &\, \mathrm{F}\,\mathbf{VD} \wedge \mathrm{F}\,(\mathbf{RD} \wedge \mathrm{X}\,\mathrm{F}\,\mathbf{RD}) \\ &\wedge \mathrm{G}\,\neg\mathbf{Un} \\ &\wedge \mathrm{G}\,(\mathbf{Ri} \rightarrow \mathrm{X}\,\mathbf{VD}) \\ &\wedge \mathrm{G}\,(\mathbf{VD} \vee \mathbf{RD} \rightarrow \mathrm{X}\,(\neg(\mathbf{VD} \vee \mathbf{RD})\,\mathrm{U}\,\mathbf{UP})) \end{aligned} \tag{13}$$

Note that in (13), the first line corresponds to the mission specification, and the rest correspond to the mission requirements as stated above.

We now show results of applying our algorithm for two scenarios, a 21×21 grid and a large 81×81 grid. We use the computational frameworks described in this paper to find the control strategy maximizing the probabilities of satisfying the specification. For the 21×21 grid, we visualize a movement of the robot under the policy calculated by our actor–critic method and verify that all task

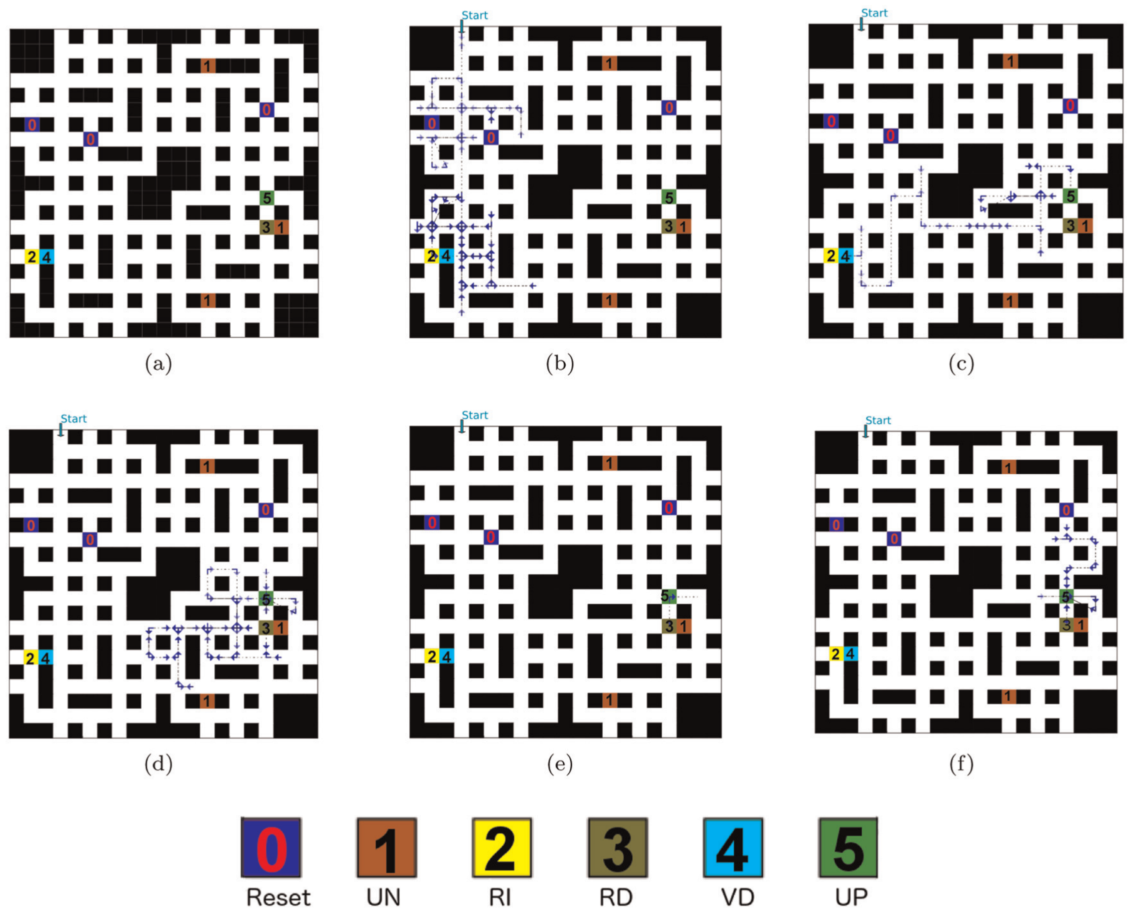

Fig. 4. (a) Schematics of the 21 × 21 grid. An example trace: (b) the robot moves from the start point and obtains valuable data (**VD**) after crossing the Risky (**Ri**) state; (c) the robot uploads (**Up**) the valuable data; (d) the robot picks up the first regular data (**RD**); (e) the robot uploads (**Up**) the first regular data; (f) the robot picks up and uploads the second regular data.

specifications are satisfied. We also evaluate the time and memory usage of our algorithm with the LP method in the 21 × 21 grid and the 81 × 81 grid, respectively. Compared with the LP method, our algorithm uses much less time and memory for the 81 × 81 grid scenario.

4.3.1. Results of a 21 × 21 grid scenario. In this scenario, we use the scene settings described in Section 4.1. Figure 4(a) plots the properties of the states in the 21 × 21 grid using different colors (the legend shows the correspondence of the properties and the colors). Figure 4 also shows an example trace, i.e. a sequence of MDP states generated by the actor–critic algorithm, to verify that the robot motion is indeed correct and satisfies the mission specification and requirements. For convenience of visualization, we divide the example trace into five segments, each segment verifying a subtask in our LTL formula. In Figure 4(b)–(f), paths of the robot are plotted as blue lines, and the turning directions of the robot are plotted as arrows. The transparency of the arrows are associated with the timestamps of the turns. Turns with smaller timestamps are plotted more transparently while turns with larger timestamps are plotted less transparently. We map each MDP state to the current region of the robot. For example, the MDP state R_1–I_3 corresponds to region I_3 in the scene. The robot first moves from starting point to pick up the valuable data (**VD**) after crossing the Risky (**Ri**) state (Figure 4(b)). After that, the robot uploads (**Up**) the valuable data (Figure 4(c)). Later, the robot picks up the first regular data (**RD**) (Figure 4(d)) and uploads (**Up**) the first regular data (Figure 4(e)). The robot eventually picks up and uploads the second regular data (Figure 4(f)).

In this case, the product MDP contains 22,533 states. There are 1085 “goal” states and 10,582 “trap” states as defined in Definition 3.9. Note that in order to solve the probability exactly using the LP method, we need to compute transition probabilities for all 90,132 state–action pairs. In our method, we only compute the transition probabilities along the sample path, which only consists of a very small portion of the state space. By caching the transition probabilities of previous visited states, we can furthermore reduce the computation cost for generating transition probabilities. We observe that less than 7000 transition probabilities are needed during the simulation of the actor–critic algorithm.

The result of the LP method is a deterministic solution, which has been proved to be the exact optimal solution over

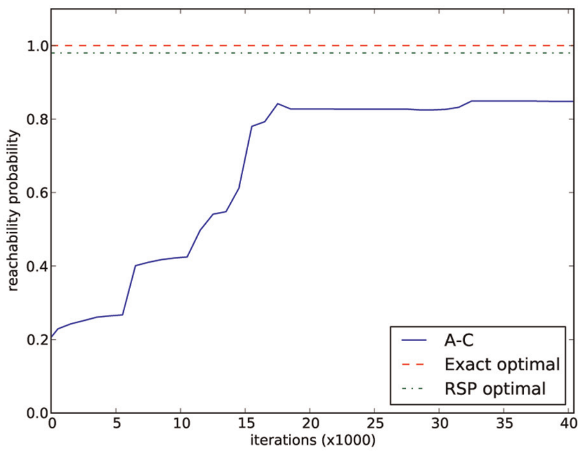

Fig. 5. Reachability probability for the RSP as a function of the number of iterations applying the proposed algorithm. The exact solution (maximum probability of satisfying the specification) is 1. The RSP optimal solution is 0.98.

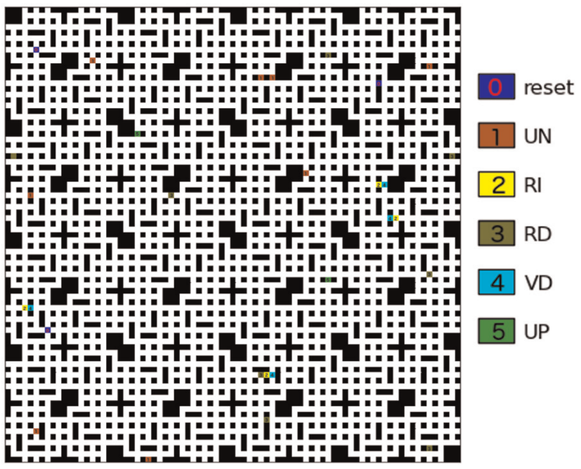

Fig. 6. Schematics of the 81×81 grid. The states with properties are plotted using corresponding color (see the legend for the correspondence of properties and colors).

all policies. The satisfaction probability is 100%. Our actor–critic algorithm optimizes $\boldsymbol{\theta}$ over a set of parameterized RSPs to a local optimal solution. Note that $\boldsymbol{\theta} = (\theta_1, \theta_2)$ is a vector of parameters, where θ_1 and θ_2 are the weights for progress and safety, respectively (cf. 11). If we assume some range for these weights, we can discretize them (e.g. by dividing their range into equal bins) and thus assume that $\boldsymbol{\theta}$ takes values in a discrete set. We could use a brute force method to calculate the optimal solution within all possible values of the discretized $\boldsymbol{\theta}$, which is referred to as *RSP optimal solution*. When discretization is fine enough, we could treat the result of the brute force method as the global optimal solution within the set of parameterized RSPs. In our case, we choose the difference of two consecutive discretized values for both θ_1 and θ_2 to be 0.01. The *RSP optimal solution* for the 21×21 grid scenario has 98% probability of satisfying the specification.

The difference between the exact optimal solution and the *RSP optimal solution* characterizes the expressivity of the RSP structure we used. The difference of the result of the actor–critic algorithm and the *RSP optimal solution* characterizes the effectiveness of the actor–critic algorithm in terms of optimizing the RSP.

The graph of the convergence of the actor–critic solution is shown in Figure 5. The parameters for this examples are: $\lambda = 0.9$, and initial $\boldsymbol{\theta} = (2.85, 100)$. The sensing radius r in the RSP is 2. The actor–critic algorithm converges after 20,000 iterations to an 85% reachability probability. Its difference to exact and RSP optimal solutions is 15% and 13%, respectively.

4.3.2. Results of a 81×81 grid scenario. The 81×81 grid is shown in Figure 6. The grid is created by concatenating 16 smaller 21×21 grids and merging the borders between the adjacent small grids. The states with properties are generated randomly. In this scenario, the product MDP contains 359,973 states. There are 19,135 "goal" states and

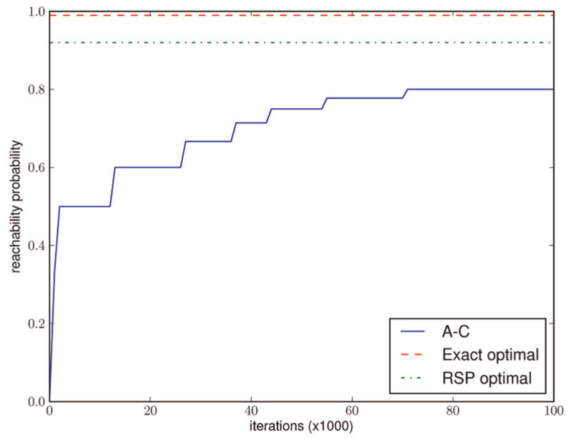

Fig. 7. Reachability probability for the RSP as a function of the number of iterations for the large environment. The exact solution (maximal probability of satisfying the specification) is 0.99. The RSP optimal solution is 0.92.

166,128 "trap" states as defined in Definition 3.9. By applying the LP method, we calculate the exact optimal solution and found a maximum satisfaction probability of 99%. The RSP optimal solution leads to satisfaction probability of 92%.

The initial $\boldsymbol{\theta} = (1, 110)$. The sensing radius r in the RSP is 2. The graph of the convergence of the actor–critic solution is shown in Figure 7. The actor–critic algorithm converges to 80% after 68,000 iterations. Its difference to exact and RSP optimal solutions is 19% and 12%, respectively.

In Problem 3.10, the period cost is zero unless the states reaches $\bar{S}^*_{\mathcal{P}}$, in which case the cost is 1. As a result, when $\bar{S}^*_{\mathcal{P}}$ is reached, the algorithm will receive a strong penalty (bad feedback) for the current policy and make a significant policy update, which results in the jumps of reachability probabilities seen in Figure 7.

Finally, in order to demonstrate the usefulness of our approach, in Table 1 we compare the computation time and

Table 1. Comparison between our method and exact methods.

Grid size	CPU time (A–C)	CPU time (LP)	Memory (A–C)	Memory (LP)
21 × 21	25 s	6.3 s	~ 0.05 G	~ 0.10 G
81 × 81	56 s	110.4 s	~ 0.22 G	~ 1.5 G

memory needed to execute our algorithm versus using the CPLEX LP solver. Both algorithms are run on a machine with 2.50 GHz 4-core Intel i5 CPU and 2 GB memory. Note that the time in the table does not include the time of calculating AMECs. For the 21×21 grid, the LP method is faster. The actor–critic algorithm needs considerable number of iterations to learn the correct parameters, even if the problem size is small, which makes it unfavorable for small problems. However, the time required for the LP method increases rapidly when the problem size increases, while the time for the actor–critic method only increases modestly. In the 81×81 grid, the LP method takes about 2 times more CPU time than our algorithm. The actor–critic method also uses much less memory than the LP method in both scenarios because only a small percentage of transition probabilities needs to be stored. For the 21×21 grid, this percentage is only around 7.6%. Although the actor–critic algorithm also needs to store some additional information (progress and safety scores), the overall memory usage remains substantially smaller. The LP method uses 15 times more memory in the 81×81 scenario than in the 21×21 scenario. At the same time, the memory usage of our method only increases by four times. These results show that the compared with the LP method, the actor–critic method is suitable for large-scale problems because of its reduced time and space complexity.

5. Conclusions and future work

We have presented a framework that brings together an approximate dynamic programming computational method of the actor–critic type, with formal control synthesis for MDPs from temporal logic specifications. We have shown that this approach is particularly suitable for problems where the transition probabilities of the MDP are difficult or computationally expensive to compute, such as for many robotic applications. We have demonstrated that this approach effectively finds an approximate optimal policy within a class of randomized stationary polices maximizing the probability of satisfying the temporal logic formula. Because our experiment setup is based on the RIDE platform, the results in this paper only focus on maze-like environments. However, the techniques presented in this paper, including the conversion from MRP to SSP and the LSTD actor–critic algorithm, are general and should work beyond RIDE as long as a labeled MDP can be properly defined in the environment. Future work can evaluate our approach in more realistic environments such as city roads. Another interesting direction to pursue would be to extend our approach to multi-robot teams.

Funding

This research was partially supported by the NSF (grant numbers CNS-1239021 and IIS-1237022), by the ARO (grant numbers W911NF-11-1-0227 and W911NF-12-1-0390), and by the ONR (grant numbers N00014-09-1051 and N00014-10-1-0952).

References

Baier C, Größer M, Leucker M, Bollig B and Ciesinski F (2004) Controller synthesis for probabilistic systems. In: Levy JJ, Mayr EW and Mitchell JC (eds) *Exploring New Frontiers of Theoretical Informatics*. Springer, pp. 493–506.

Baier C and Katoen JP (2008) *Principles of Model Checking*. Cambridge, MA: MIT Press, pp. 2620–2649.

Barto A, Sutton R and Anderson C (1983) Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13: 834–846.

Bertsekas D and Tsitsiklis J (1996) *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Bertsekas DP (1995) *Dynamic Programming and Optimal Control*, vol. I and II. Belmont, MA: Athena Scientific.

Bhatia A, Kavraki LE and Vardi MY (2010) Sampling-based motion planning with temporal goals. In: *IEEE international conference on robotics and automation (ICRA)*, pp. 2689–2696. IEEE.

Boyan JA (1999) Least-squares temporal difference learning. In: *ICML*, pp. 49–56.

Clarke EM, Grumberg O and Peled D (1999) *Model Checking*. Cambridge, MA: MIT Press.

Cormen TH, Leiserson CE, Rivest RL, et al. (2001) *Introduction to Algorithms*, volume 2. Cambridge, MA: MIT Press.

Courcoubetis C and Yannakakis M (1990) Markov decision processes and regular events. In: *Automata, languages and programming*, New York, pp. 336–349. Springer.

De Alfaro L (1997) *Formal verification of probabilistic systems*. PhD Thesis, Stanford University.

Ding XC, Smith SL, Belta C and Rus D (2011) LTL control in uncertain environments with probabilistic satisfaction guarantees. In: *18th IFAC world congress*, pp. 3515–3520.

Ding XC, Wang J, Lahijanian M, Paschalidis IC and Belta CA (2012) Temporal logic motion control using actor–critic methods. In: *2012 IEEE international conference on robotics and automation (ICRA)*, pp. 4687–4692. IEEE.

Estanjini RM, Ding XC, Lahijanian M, Wang J, Belta CA and Paschalidis IC (2011) Least squares temporal difference actor–critic methods with applications to robot motion control. In: *2011 50th IEEE conference on decision and control and European control conference (CDC-ECC)*, pp. 704–709. IEEE.

Estanjini RM, Li K and Paschalidis IC (2012) A least squares temporal difference actor–critic algorithm with applications to warehouse management. *Naval Research Logistics* 59(3–4): 197–211.

Fu J and Topcu U (2014) Probably approximately correct MDP learning and control with temporal logic constraints. *Proceedings of robotics: science and systems*.

Gradel E, Thomas W and Wilke T (2002) *Automata, Logics, and Infinite Games: A guide to current research* (*Lecture Notes in Computer Science*, vol. 2500) New York: Springer.

Karaman S and Frazzoli E (2009) Sampling-based motion planning with deterministic μ-calculus specifications. In: *Proceedings of the 48th IEEE conference on decision and control, 2009 held jointly with the 2009 28th Chinese control conference (CDC/CCC 2009)*, pp. 2222–2229. IEEE.

Konda VR and Tsitsiklis JN (2003) On actor–critic algorithms. *SIAM Journal on Control and Optimization* 42(4): 1143–1166.

Konda VR (2002) *Actor–critic algorithms*. PhD Thesis, Massachusetts Institute of Technology.

Kress-Gazit H, Fainekos GE and Pappas GJ (2007) Where's Waldo? Sensor-based temporal logic motion planning. In: *2007 IEEE international conference on robotics and automation*, pp. 3116–3121. IEEE.

Lahijanian M, Andersson SB and Belta C (2012) Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transactions on Robotics* 28(2): 396–409.

Lahijanian M, Wasniewski J, Andersson SB and Belta C (2010) Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In: *Proceedings of IEEE international conference on robotics and automation (ICRA)*, Anchorage, AK, USA, pp. 3227–3232.

Liu J, Ozay N, Topcu U and Murray RM (2013) Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Transactions on Automatic Control* 58(7): 1771–1785.

Loizou SG and Kyriakopoulos KJ (2004) Automatic synthesis of multi-agent motion tasks based on LTL specifications. In: *43rd IEEE conference on decision and control, 2004 (CDC)*, vol. 1, pp. 153–158. IEEE.

Luna R, Lahijanian M, Moll M and Kavraki LE (2014) Optimal and efficient stochastic motion planning in partially-known environments. In: *Twenty-eighth AAAI conference on artificial intelligence (AAAI 2014)*, Quebec City, Canada.

Puterman ML (1994) *Markov decision processes: discrete stochastic dynamic programming* (*Wiley Series in Probability and Statistics*, Vol. 414). New York: John Wiley & Sons.

Quottrup MM, Bak T and Zamanabadi R (2004) Multi-robot planning: A timed automata approach. In: *2004 IEEE International Conference on Robotics and Automation*, vol. 5, pp. 4417–4422. IEEE.

Rutten JJ, Panangaden P and Van Breugel F (2004) *Mathematical techniques for analyzing concurrent and probabilistic systems*. Providence, RI: American Mathematical Society.

Sadigh D, Kim ES, Coogan S, Sastry SS and Seshia SA (2014) A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In: *Proceedings of the 53rd IEEE conference on decision and control (CDC)*.

Si J (2004) *Handbook of learning and approximate dynamic programming*, vol. 2. New York: Wiley-IEEE Press.

Sutton RS and Barto AG (1998) *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Vardi MY (1999) Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In: *Formal Methods for Real-Time and Probabilistic Systems*. New York: Springer, pp. 265–276.

Wolff EM, Topcu U and Murray RM (2012) Robust control of uncertain Markov decision processes with temporal logic specifications. In: *2012 IEEE 51st annual conference on decision and control (CDC)*, pp. 3372–3379. IEEE.

Wongpiromsarn T, Topcu U and Murray RM (2009) Receding horizon temporal logic planning for dynamical systems. In: *Proceedings of the 48th IEEE conference on decision and control, 2009 held jointly with the 2009 28th Chinese control conference (CDC/CCC 2009)*, pp. 5997–6004. IEEE.

Appendix: Convergence of the LSTD actor–critic algorithm

We first cite the theory of *linear stochastic approximation driven by a slowly varying Markov chain* (Konda, 2002) (with simplifications).

Let $\{\mathbf{y}_k\}$ be a finite Markov chain whose transition probabilities $p_{\boldsymbol{\theta}}(\cdot|\cdot)$ depend on a parameter $\boldsymbol{\theta} \in \mathbb{R}^n$. Consider a generic iteration of the form

$$\mathbf{s}_{k+1} = \mathbf{s}_k + \gamma_k(\mathbf{h}_{\boldsymbol{\theta}_k}(\mathbf{y}_{k+1}) - \mathbf{G}_{\boldsymbol{\theta}_k}(\mathbf{y}_{k+1})\mathbf{s}_k) + \gamma_k \Xi_k \mathbf{s}_k \quad (14)$$

where $\mathbf{s}_k \in \mathbb{R}^m$, and $\mathbf{h}_\theta(\cdot) \in \mathbb{R}^m, \mathbf{G}_\theta(\cdot) \in \mathbb{R}^{m\times m}$ are $\boldsymbol{\theta}$-parameterized vector and matrix functions, respectively. We first present the following conditions.

Conditions 5.1. *(1) The sequence $\{\gamma_k\}$ is deterministic, non-increasing, and*

$$\sum_k \gamma_k = \infty, \quad \sum_k \gamma_k^2 < \infty$$

(2) The random sequence $\{\boldsymbol{\theta}_k\}$ satisfies $\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\| \leq \beta_k H_k$ for some process $\{H_k\}$ with bounded moments, where $\{\beta_k\}$ is a deterministic sequence such that

$$\sum_k \left(\frac{\beta_k}{\gamma_k}\right)^d < \infty \quad \textit{for some } d>0$$

(3) Ξ_k is an $m \times m$-matrix valued martingale difference with bounded moments.

(4) For each $\boldsymbol{\theta}$, there exist $\bar{\mathbf{h}}(\boldsymbol{\theta}) \in \mathbb{R}^m$, $\overline{\mathbf{G}}(\boldsymbol{\theta}) \in \mathbb{R}^{m\times m}$, and corresponding m-vector and $m \times m$-matrix functions $\widehat{\mathbf{h}}_{\boldsymbol{\theta}}(\cdot)$, $\widehat{\mathbf{G}}_{\boldsymbol{\theta}}(\cdot)$ that satisfy the Poisson equation. That is, for each $\mathbf{y}$,

$$\widehat{\mathbf{h}}_{\boldsymbol{\theta}}(\mathbf{y}) = \mathbf{h}_{\boldsymbol{\theta}}(\mathbf{y}) - \bar{\mathbf{h}}(\boldsymbol{\theta}) + \sum_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{y})\widehat{\mathbf{h}}_{\boldsymbol{\theta}}(\mathbf{z})$$
$$\widehat{\mathbf{G}}_{\boldsymbol{\theta}}(\mathbf{y}) = \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{y}) - \overline{\mathbf{G}}(\boldsymbol{\theta}) + \sum_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{y})\widehat{\mathbf{G}}_{\boldsymbol{\theta}}(\mathbf{z})$$

Denote by $E_{\boldsymbol{\theta}}[\cdot]$ the expectation with respect to the stationary distribution of the finite Markov chain $\{\mathbf{y}_k\}$, then $\overline{\mathbf{G}}(\boldsymbol{\theta}_k) = E_{\boldsymbol{\theta}}[\mathbf{G}_{\boldsymbol{\theta}_k}(\mathbf{y})]$ and $\bar{\mathbf{h}}(\boldsymbol{\theta}_k) = E_{\boldsymbol{\theta}}[\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{y})]$.

(5) For some constant C and for all $\boldsymbol{\theta}$, we have $\max(\|\bar{\mathbf{h}}(\boldsymbol{\theta})\|, \|\overline{\mathbf{G}}(\boldsymbol{\theta})\|) \leq C$.

(6) For any $d > 0$, there exists $C_d > 0$ such that $\sup_k \mathbf{E}\left[\|\mathbf{f}_{\boldsymbol{\theta}_k}(\mathbf{y}_k)\|^d\right] \leq C_d$,, where $\boldsymbol{f}_{\boldsymbol{\theta}}(\cdot)$ represents any of the functions $\widehat{\mathbf{h}}_{\boldsymbol{\theta}}(\cdot)$, $\boldsymbol{h}_{\boldsymbol{\theta}}(\cdot)$, $\widehat{\mathbf{G}}_{\boldsymbol{\theta}}(\cdot)$ and $\boldsymbol{G}_{\boldsymbol{\theta}}(\cdot)$.

(7) For some constant $C > 0$ and for all $\boldsymbol{\theta}, \overline{\boldsymbol{\theta}} \in \mathbb{R}^n$, $\max(\|\bar{\mathbf{h}}(\boldsymbol{\theta}) - \bar{\mathbf{h}}(\overline{\boldsymbol{\theta}})\|, \|\overline{\mathbf{G}}(\boldsymbol{\theta}) - \overline{\mathbf{G}}(\overline{\boldsymbol{\theta}})\|) \leq C\|\boldsymbol{\theta} - \overline{\boldsymbol{\theta}}\|$.

(8) There exists a positive measurable function $C(\cdot)$ *such that for every* $d > 0$, $\sup_k \mathbf{E}[C(\mathbf{y}_k)^d] < \infty$, *and* $\|\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{y}) - \mathbf{f}_{\bar{\boldsymbol{\theta}}}(\mathbf{y})\| \le C(\mathbf{y})\|\boldsymbol{\theta} - \bar{\boldsymbol{\theta}}\|$.

(9) There exists $a > 0$ *such that for all* $\mathbf{s} \in \mathbb{R}^m$ *and* $\boldsymbol{\theta} \in \mathbb{R}^n$

$$\mathbf{s}'\overline{\mathbf{G}}(\boldsymbol{\theta})\mathbf{s} \ge a\|\mathbf{s}\|^2$$

It has been shown in Konda (2002) that the critic in (14) converges if Conditions 5.1(1)–(9) are met.

Theorem 5.2. *[Convergence of linear stochastic approximation] If Conditions 5.1(1)–(9) are satisfied, then*

$$\lim_{k\to\infty} |\overline{\mathbf{G}}(\boldsymbol{\theta}_k)\mathbf{s}_k - \bar{\mathbf{h}}(\boldsymbol{\theta}_k)| = 0 \tag{15}$$

Proof. See Chapter 3 of Konda (2002).

Based on Theorem 5.2, we present the convergence of the critic in Algorithm 1 in the following theorem.

Theorem 5.3. *[Critic convergence] For the LSTD actor–critic method described in Algorithm 1 with some deterministic and non-increasing step-size sequences* $\{\beta_k\}$, $\{\gamma_k\}$ *satisfying (8), the sequence* $\mathbf{s}_k$ *is bounded, and*

$$\lim_{k\to\infty} |\mathbf{b}_k - E_{\boldsymbol{\theta}}[g(x,u)\mathbf{z})]| = 0$$
$$\lim_{k\to\infty} |\mathbf{v}(\mathbf{A}_k) - E_{\boldsymbol{\theta}}[\mathbf{v}\big(\mathbf{z}((P_{\boldsymbol{\theta}}\psi'_{\boldsymbol{\theta}})(x,u) - \psi'_{\boldsymbol{\theta}}(x,u))\big)]| = 0$$

where $E_{\boldsymbol{\theta}}[\cdot]$ *is the expectation with respect to the stationary distribution of the Markov chain* $\{x_k, u_k, z_k\}$, *and for any matrix* $\mathbf{A}$, $\mathbf{v}(\mathbf{A})$ *is a column vector that stacks all row vectors of* $\mathbf{A}$ *(also written as column vectors).*

Proof. Simple algebra suggests that the critic update in Algorithm 1 can be written in the form of (14) with

$$\mathbf{s}_k = \begin{bmatrix} \mathbf{b}_k \\ \mathbf{v}(\mathbf{A}_k) \\ 1 \end{bmatrix}, \qquad \mathbf{y}_k = (x_k, u_k, \mathbf{z}_k)$$
$$\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{y}) = \begin{bmatrix} g(x,u)\mathbf{z} \\ \mathbf{v}(\mathbf{z}((P_{\boldsymbol{\theta}}\psi'_{\boldsymbol{\theta}})(x,u) - \psi'_{\boldsymbol{\theta}}(x,u))) \\ 1 \end{bmatrix} \tag{16}$$
$$\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{y}) = [\,\mathbf{I}\,]$$
$$\Xi_k = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & D_k \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{A}_k$, $\mathbf{b}_k$, x_k, u_k, and $\mathbf{z}_k$ are the iterates defined in Algorithm 1, $\mathbf{y} = (x, u, \mathbf{z})$ denotes a value of the triplet $\mathbf{y}_k$, and $D_k = \mathbf{v}(\mathbf{z}_k(\psi'_{\boldsymbol{\theta}_k}(x_{k+1}, u_{k+1}) - (P_{\boldsymbol{\theta}}\psi_{\boldsymbol{\theta}})'(x_k, u_k)))$.

The step-sizes γ_k and β_k in Algorithm 1 correspond exactly to the γ_k and β_k in Conditions 5.1(1) and (2), respectively. For MDPs with finite state and action space, Conditions 5.1(1) and (2) reduces to (8).

A direct yet verbose way to prove the theorem is to verify Conditions 5.1(3)–(9). However, a comparison with the convergence proof for the TD(λ) critic in Konda and Tsitsiklis (2003) gives a simpler proof. Let

$$\mathbf{F}_{\boldsymbol{\theta}}(\mathbf{y}) = \mathbf{z}(\psi'_{\boldsymbol{\theta}}(x,u) - (P_{\boldsymbol{\theta}}\psi_{\boldsymbol{\theta}})'(x,u))$$

While proving the convergence of TD(λ) critic operating concurrently with the actor, Konda and Tsitsiklis (2003) showed that

$$\widetilde{\mathbf{h}}_{\boldsymbol{\theta}}(\mathbf{y}) = \begin{bmatrix} \tilde{h}^{(1)}_{\boldsymbol{\theta}}(\mathbf{y}) \\ \widetilde{\mathbf{h}}^{(2)}_{\boldsymbol{\theta}}(\mathbf{y}) \end{bmatrix} = \begin{bmatrix} Mg(x,u) \\ g(x,u)\mathbf{z} \end{bmatrix} \tag{17}$$

$$\widetilde{\mathbf{G}}_{\boldsymbol{\theta}}(\mathbf{y}) = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{z}/M & \mathbf{F}_{\boldsymbol{\theta}}(\mathbf{y}) \end{bmatrix} \tag{18}$$

and

$$\widetilde{\Xi}_k = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{v}\Big(\mathbf{z}_k(\psi'_{\boldsymbol{\theta}_k}(x_{k+1}, u_{k+1}) - (P_{\boldsymbol{\theta}}\psi_{\boldsymbol{\theta}})'(x_k, u_k))\Big) \end{bmatrix} \tag{19}$$

satisfy Conditions 5.1(3)–(8). Then M in (17) and (18) is an arbitrary (large) positive constant whose role is to facilitate the convergence proof. In our case, equation (16) can be rewritten as

$$\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{y}) = \begin{bmatrix} \widetilde{\mathbf{h}}^{(2)}_{\boldsymbol{\theta}}(\mathbf{y}) \\ \mathbf{v}(\mathbf{F}_{\boldsymbol{\theta}}(\mathbf{y})) \\ 1 \end{bmatrix}, \quad \mathbf{G}_{\boldsymbol{\theta}}(\mathbf{y}) = [\,\mathbf{I}\,], \quad \Xi_k = \begin{bmatrix} \widetilde{\Xi}_k \\ \mathbf{0} \end{bmatrix} \tag{20}$$

Note that although $\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{y})$, $\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{y})$, and Ξ_k in (20) are very different from $\widetilde{\mathbf{h}}_{\boldsymbol{\theta}}(\mathbf{y})$, $\widetilde{\mathbf{G}}_{\boldsymbol{\theta}}(\mathbf{y})$, and $\widetilde{\Xi}_k$ in (17), (18) and (19), they involve the same quantities and both in a linear fashion. So, $\mathbf{h}_{\boldsymbol{\theta}}(\cdot)$,$\mathbf{G}_{\boldsymbol{\theta}}(\cdot)$ and Ξ_k also satisfy Conditions 5.1(3)–(8). Meanwhile, the step-size $\{\gamma_k\}$ satisfies Condition 5.1(1), and the step-size $\{\beta_k\}$ satisfies (8) (which is as explained above implies Condition 5.1(2)). Now, only Condition 5.1(9) remains to be checked. Here $\mathbf{G}_{\boldsymbol{\theta}}(\mathbf{y})$ is the identity matrix, hence positive definite. Since all conditions are satisfied, equation (15) holds here. The theorem can be proved by substituting (16) to (15) and using the fact that $\overline{\mathbf{G}}(\boldsymbol{\theta}_k) = E_{\boldsymbol{\theta}}[\mathbf{G}_{\boldsymbol{\theta}_k}(\mathbf{y})]$ and $\bar{\mathbf{h}}(\boldsymbol{\theta}_k) = E_{\boldsymbol{\theta}}[\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{y})]$. ■

Theorem 5.3 states that the critic step in Algorithm 1 can converge to the optimal $\mathbf{r}$. In addition, note that λ in Step 6 of Algorithm 1 is a decay factor used in Sutton's eligibility trace, and has been applied in other approximate dynamic programming methods such as TD(λ) (Sutton and Barto, 1998). Then the following theorem establishes the convergence for the actor step of Algorithm 1.

Proof of Theorem 3.13

Proof. Compared with Konda and Tsitsiklis (2003), the actor update is the same and we only change the critic update. The convergence of the critic update is established in Theorem 5.3. The theorem can be proved by setting $\boldsymbol{\phi}_{\boldsymbol{\theta}} = \boldsymbol{\psi}_{\boldsymbol{\theta}}$ and following the proof in Section 6 of Konda and Tsitsiklis (2003). ■