Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)

Probably Approximately Correct Learning in Stochastic Games with Temporal Logic Specifications

Min Wen
University of Pennsylvania
wenm@seas.upenn.edu

Ufuk Topcu
University of Texas at Austin
utopcu@utexas.edu

Abstract

We consider a controller synthesis problem in turn-based stochastic games with both a qualitative linear temporal logic (LTL) constraint and a quantitative discounted-sum objective. For each case in which the LTL specification is realizable and can be equivalently transformed into a deterministic Buchi automaton, we show that there always exists a *memoryless almost-sure winning* strategy that is ε*-optimal* with respect to the discounted-sum objective for any arbitrary positive ε. Building on the idea of the R-MAX algorithm, we propose a probably approximately correct (PAC) learning algorithm that can learn such a strategy efficiently in an online manner with *a-priori* unknown reward functions and unknown transition distributions. To the best of our knowledge, this is the first result on PAC learning in stochastic games with independent quantitative and qualitative objectives.

1 Introduction

An important goal of reinforcement learning (RL) is to make an agent learn to behave as desired through its experience. For problems modeled as stochastic games, the expectation of the discounted sum of rewards is commonly used as a performance criterion to encode the preference over strategies. It is well known that with discounted-sum objectives, pure memoryless strategies suffice for optimality [Filar and Vrieze, 1996], which significantly simplifies the learning algorithms. However, the discounted-sum objective suffers from several noticeable drawbacks in the task of describing the desired strategies. First, it is not applicable to strategies that require memory. As memoryless strategies are sufficient to achieve optimality, agents lack the incentive to learn the more complicated finite-memory strategies. Second, it cannot restrict behavior during the learning process. With rewards, the agent can only figure out the preferable actions after it actually tries all transitions, even the fatal ones such as crashing into some obstacle, which is obviously unacceptable. Third, there is usually a lack of theoretical proof that any strategy solved with the given reward function is desirable, except in some simple scenarios. For example, multi-dimensional reward functions are generally necessary to represent the conjunction of several requirements, in which case a strategy usually cannot be simultaneously optimized with every single reward. It is hard to know intuitively from the reward function how different the learned strategy is from a desired one.

In order to compensate for these problems, we propose to use linear temporal logic (LTL) specifications to complement the encoding of the desired strategies. Practically, it is relatively straightforward to extract LTL specifications from high-level task requirements in robot planning and control [Kress-Gazit *et al.*, 2007; Smith *et al.*, 2011; Guo *et al.*, 2013; Wolff *et al.*, 2013]. Algorithmically, all LTL formulas can be transformed to deterministic Rabin or parity automata (DRA or DPA), which can be further used to construct product stochastic Rabin or parity games. Strategies synthesized for such product Rabin or parity games are guaranteed to satisfy the corresponding LTL specifications with probability one (i.e. almost surely), treating LTL specifications as 'game rules' that should never be violated. Both the construction of DRA or DPA from LTL formulas and the synthesis can be performed using off-the-shelf tools [Gaiser *et al.*, 2012; Tsai *et al.*, 2013; Klein, 2015; Blahoudek, 2015; Friedmann and Lange, 2009]. Although it has been shown that pure memoryless strategies suffice for almost-sure winning in the product stochastic Rabin or parity games [Chatterjee *et al.*, 2003; 2005a], these strategies use memory in the original stochastic games. In this way, LTL specifications offer a systematic way of designing the memory for the desired strategies. We will show later that with the pre-computation of almost-sure winning regions in the product games, we can keep the agent safe even during the learning procedure.

In this paper we use both discounted rewards and LTL specifications to encode task requirements. In particular, if an LTL specification is realizable and can be transformed into a deterministic Buchi automaton (DBA), we prove the existence of a memoryless strategy which is both almost-sure winning with respect to the Buchi objective and ε-optimal with respect to the discounted-sum objective. We also propose a probably approximately correct (PAC) algorithm to learn such a strategy online when the reward function and the transition distributions are both unknown a priori [Strehl *et al.*, 2009; Fu and Topcu, 2014]. To the best of our knowledge, this is the first PAC learning algorithm for stochastic games with independent quantitative and qualitative objectives.

Related work Strategy synthesis in which perfect knowledge of the game is available and the task involves both qualitative and quantitative objectives has been extensively studied. Examples include mean-payoff parity games [Chatterjee *et al.*, 2005b], energy parity games [Chatterjee and Doyen, 2010], their multi-dimension version [Chatterjee *et al.*, 2012] and stochastic version [Chatterjee *et al.*, 2014]. There are also results on strategy synthesis in stochastic games with total reward constraints and LTL specifications [Chen *et al.*, 2013]. However, only limited work has been done when the transition distributions and the rewards are unknown a priori. This paper is an extension of our previous work [Wen *et al.*, 2015], in which we considered a learning problem with both temporal logic constraints and unknown discounted-sum objectives in deterministic games. Our new algorithm here works with general stochastic games and guarantees to learn a near-optimal strategy with any specified optimality bound.

2 Preliminaries

For any countable set M, let $|M|$ be its cardinality, M^ω be the set of infinite sequences composed of elements in M, and $\mathcal{D}(M)$ be the set of all probability distributions over M.

We first formulate the decision-making problem as a turn-based labeled stochastic game. A *turn-based labeled stochastic game* between the controlled agent (the 'system') and the uncontrolled agent (the 'environment') is defined as a tuple $\mathcal{G} = (S_\mathcal{G}, S_{s,\mathcal{G}}, S_{e,\mathcal{G}}, I_\mathcal{G}, A_\mathcal{G}, T_\mathcal{G}, \mathcal{R}_\mathcal{G}, AP, L_\mathcal{G})$, where $S_\mathcal{G}$ is a finite state space; $S_{s,\mathcal{G}} \subseteq S_\mathcal{G}$ is the set of states at which the system chooses actions, and $S_{e,\mathcal{G}} = S_\mathcal{G} \backslash S_{s,\mathcal{G}}$ is a set of states at which the environment chooses actions; $I_\mathcal{G} \subseteq S_\mathcal{G}$ is a set of initial states; $A_\mathcal{G}$ is a finite action space; $T_\mathcal{G} : S_\mathcal{G} \times A_\mathcal{G} \to \mathcal{D}(S_\mathcal{G})$ is a transition function; $\mathcal{R}_\mathcal{G} : S_\mathcal{G} \times A_\mathcal{G} \times S_\mathcal{G} \to \mathbb{R}^{\geq 0}$ is a non-negative reward function; AP is a set of atomic propositions (Boolean variables); $L_\mathcal{G} : S_\mathcal{G} \to 2^{AP}$ is a labeling function.

LTL specifications put restrictions on the label sequences corresponding to the infinite state sequences of $\mathcal{G}$. Interested readers may refer to [Baier *et al.*, 2008] for the detailed syntax and semantics of LTL. Instead of treating LTL specifications directly as formulas, we translate them into ω-regular automata, or deterministic Buchi automata, to be precise. A *deterministic Buchi automaton (DBA)* is a tuple $\mathcal{A} = (Q_\mathcal{A}, \Sigma_\mathcal{A}, \delta_\mathcal{A}, Q_{0,\mathcal{A}}, F_\mathcal{A})$ where $Q_\mathcal{A}$ is a finite set of states; $\Sigma_\mathcal{A}$ is a finite input alphabet; $\delta_\mathcal{A} : Q_\mathcal{A} \times \Sigma_\mathcal{A} \to Q_\mathcal{A}$ is a transition function; $Q_{0,\mathcal{A}} \subseteq Q_\mathcal{A}$ is a set of initial states; $F_\mathcal{A} \subseteq Q_\mathcal{A}$ is a set of accepting states. A *run* of $\mathcal{A}$ over an input sequence $(p_t)_{t\in\mathbb{N}} \in \Sigma_\mathcal{A}^\omega$ is an infinite sequence $(q_t)_{t\in\mathbb{N}} \in Q_\mathcal{A}^\omega$, where $q_0 \in Q_{0,\mathcal{A}}$ and $q_{t+1} = \delta_\mathcal{A}(q_t, p_t)$ for all $t \in \mathbb{N}$. A run $(q_t)_{t\in\mathbb{N}}$ is *accepted* by $\mathcal{A}$ if $|\{t \in \mathbb{N} : q_t \in F_\mathcal{A}\}| = \infty$. Only a subclass of LTL formulas can be transformed into equivalent DBAs, but this subclass of specifications covers a wide range of requirements in robot planning tasks. For example, $\Box safe\_region$ (always stay in states labeled as 'safe region'), $\Diamond goal$ (eventually reach a state labeled as 'goal'), $\Box(request \to \Diamond response)$ (if a 'request' state is observed, a 'response' state should be visited later), $\Box\Diamond charging$ (always get to the 'charging' state sometime later), $\Box((\neg charging)\,\mathcal{U}\, battery\_low)$ (never charge yourself before the battery gets low), just to name a few.

If $\Sigma_\mathcal{A} = 2^{AP}$, we can compose a *turn-based stochastic Buchi game* $G = (S, S_s, S_e, I, A, T, \mathcal{R}, F)$ of $\mathcal{G}$ and $\mathcal{A}$ with the standard product automata construction:

- $S = S_\mathcal{G} \times Q_\mathcal{A}$ is a finite state space;
- $S_s = S_{s,\mathcal{G}} \times Q_\mathcal{A}$ is the set of system states, and $S_e := S\backslash S_s$ is the set of environment states;
- $I = I_\mathcal{G} \times Q_{0,\mathcal{A}}$ is a set of initial states;
- $A = A_\mathcal{G}$ is a finite action space;
- $T : S \times A \to \mathcal{D}(S)$ is the transition function such that $T((s, q), a)(s', \delta_\mathcal{A}(q, L_\mathcal{G}(s))) = T_\mathcal{G}(s, a)(s')$;
- $\mathcal{R} : S \times A \times S \to \mathbb{R}^{\geq 0}$ is a non-negative reward function such that $\mathcal{R}((s, q), a, (s', q')) = \mathcal{R}_\mathcal{G}(s, a, s')$;
- $F = S_\mathcal{G} \times F_\mathcal{A}$ is a set of accepting states.

The product game G inherits the reward from $\mathcal{G}$ and the winning condition from $\mathcal{A}$. On the reward side, the system is to maximize its future discounted reward, while the environment is to minimize it. On the winning condition side, the system is to win with probability one, regardless of the behavior of the environment. We define the system strategies and then formulate the almost-sure winning objective as well as the discounted-sum objective.

A *(randomized) system strategy* is defined as a tuple $\sigma_s = (\sigma_s^m, \rho_s^m, M_s, m_s^0)$, where M_s is a (possibly countably infinite) set of memory states; $m_s^0 \in M_s$ is the initial memory state; $\sigma_s^m : S_s \times M_s \to \mathcal{D}(A)$, and $\rho_s^m : S \times M_s \to M_s$ is the memory update function. If M_s is a singleton, σ_s is a *memoryless* strategy; if M_s is a finite set, σ_s is a *finite-memory strategy*. With a slight abuse of notation, we use $\sigma_s(s, a)$ to represent $\sigma_s^m(s, m_s^0)(a)$ for $s \in S_s$, $a \in A^G(s)$ when σ_s is memoryless. If $|\{s' \in S : \sigma_s(s, m)(s') > 0\}| = 1$ for all $s \in S_s$ and $m \in M_s$, σ_s is a *pure* strategy. An environment strategy $\sigma_e = (\sigma_e^m, \rho_e^m, M_e, m_e^0)$ can be defined analogously.

The Almost-Sure Winning Objective The winning condition for G is defined on its runs. Let $A^G : S \to 2^A\backslash\emptyset$ be a mapping from each state to its available actions in G. For each $s \in S$, $a \in A^G(s)$, let $E^G(s, a) \subseteq S$ be the set of possible successors by taking a at s in G. A *run* $\pi = (s_\pi^{t-1}, a_\pi^t)_{t\in\mathbb{N}^+}$ of G is an infinite sequence of state-action pairs such that for all $t \in \mathbb{N}^+$, $s_\pi^{t-1} \in S$, $a_\pi^t \in A^G(s_\pi^t)$, and $s_\pi^t \in E^G(s_\pi^{t-1}, a_\pi^t)$. π is *winning for the system* with respect to the Buchi condition if and only if F is visited for infinitely many times in π, i.e. $|\{t \in \mathbb{N} : s_\pi^i \in F\}| = \infty$. A system strategy σ_s is *almost-sure winning* at $s \in S$ if a run π with $s_\pi^0 = s$ is winning for the system with probability one when the system takes σ_s, regardless of the environment strategy. The *almost-sure winning region* for the system, denoted by W_{as} (or W_{as}^G to explicitly indicate G), is the set of states at which the system has almost-sure winning strategies. By definition, there are no outgoing transitions from $S_e \bigcap W_{as}$ that leaves W_{as}.

The *almost-sure winning objective* for the system is to always take an almost-sure winning strategy. In other words, the system strategy is supposed to be almost-sure winning at any time in the process of learning.

The Discounted-Sum Objective As common in the RL literature, we use state value functions and action value functions to evaluate system strategies. The *state value function* $V_{\sigma_s} : S \to \mathbb{R}^{\geq 0}$ specifies the worst-case expected discounted reward from each state when the system the strategy σ_s. The *action value function* $Q_{\sigma_s} : S \times A \to \mathbb{R}^{\geq 0}$ shows the worst-case expected discounted reward if the system takes a given action at the current step and follows the strategy σ_s thereafter. A system strategy σ_s is *optimal* if it maximizes the state value functions over all system strategies. When σ_s is an optimal strategy, its state value function and action value function are called the *optimal state value function* and the *optimal action value function*, denoted by V^* and Q^* respectively. V^* and Q^* satisfy the following optimality conditions [Littman, 2001]:

$$V^*(s) = \begin{cases} \max_{a \in A^G(s)} Q^*(s,a), & \text{if } s \in S_s \\ \min_{a \in A^G(s)} Q^*(s,a), & \text{if } s \in S_e \end{cases}$$

$$Q^*(s,a) = \sum_{s' \in E^G(s,a)} T(s,a)(s')\big(\mathcal{R}(s,a,s') + \gamma V^*(s')\big),$$

where $\gamma \in (0,1)$ is a *discount factor*. For any $\varepsilon > 0$, a system strategy σ_s is *ε-optimal at $s \in S$* if $V_{\sigma_s}(s) \geq V^*(s) - \varepsilon$. Given Σ_s as a set of system strategies, a system strategy $\sigma_s \in \Sigma_s$ is *optimal over Σ_s at $s \in S$* if $V_{\sigma_s}(s) \geq \max_{\sigma'_s \in \Sigma_s} V_{\sigma'_s}(s)$, or *$\varepsilon$-optimal over Σ_s at s* if $V_{\sigma_s}(s) \geq \max_{\sigma'_s \in \Sigma_s} V_{\sigma'_s}(s) - \varepsilon$.

The *discounted-sum objective* for the system is to be ε-optimal over all almost-sure winning system strategies at all states that are visited infinitely often. In other words, eventually the worst-case expected reward at any state that will be visited in future is ε-optimal.

3 Problem Formulation

We make the following assumptions in our formulation.

Assumption 1. *Both the environment and the system are aware of their current states, and can observe the reward after a transition is taken. In other words, the game is fully observable for both sides.*

Assumption 2. *The system knows the correct list of all possible successors for all state-action pairs, but does not know the exact transition distributions a priori.*

The knowledge of all existing transitions is critical in order to achieve the almost-sure winning objective. It may seem demanding at the first glance, but it is possible that an almost-sure winning strategy can never be learned from experience without the previous knowledge. For example, even if $s' \in S$ is never witnessed as a successor of a state-action pair (s,a), we cannot confirm that $T(s,a)(s') = 0$. Also, without resetting, there can be a positive probability that the system leaves its almost-sure winning region during the exploration period and cannot guarantee almost-sure winning thereafter.

The third assumption is on the unknown reward. The upper bound R_{max} can be set easily as it is not required to be tight.

Assumption 3. *The reward function is unknown a priori, but is upper bounded by the specified positive number R_{max}.*

With the above assumptions, we formulate our learning problem as follows.

Problem 1. *Given a turn-based stochastic Buchi game $G^{in} = (S^{in}, S^{in}_s, S^{in}_e, I^{in}, A^{in}, T^{in}, \mathcal{R}^{in}, F^{in})$ satisfying Assumptions 1 - 3, a discount factor $\gamma \in (0,1)$ and suboptimality bound $\varepsilon > 0$, learn a memoryless system strategy $\sigma_{s,\varepsilon}$ that satisfies both the almost-sure winning objective and the discounted-sum objective.*

4 Main Approach

We now develop the main algorithm to solve Problem 1. The first step is to compute the almost-sure winning region W^{in}_{as} for the system. By definition of almost-sure winning regions, the system can only win almost-surely if it always stays within W^{in}_{as}. The definition guarantees that for any $s \in S^{in}_s \bigcap W^{in}_{as}$, there exists an action $a \in A^{G^{in}}(s)$ such that $E^{G^{in}}(s,a) \subseteq W^{in}_{as}$; for any $s \in S^{in}_e \bigcap W^{in}_{as}$, $E^{G^{in}}(s,a) \subseteq W^{in}_{as}$ for all $a \in A^{G^{in}}(s)$. In other words, the system has a strategy to force the environment to stay within W^{in}_{as} once it is entered. Therefore we can construct a new game G from G^{in} as follows to ensure that the visited states are always restricted to W^{in}_{as} in G. $G = G^{in} \restriction W^{in}_{as} = (S, S_s, S_e, I, A, T, \mathcal{R}, F)$ such that:

Algorithm 1 HatGame and RecoverHatStrategy

```
1:  function HatGame($G$, $Q^*$, $\varepsilon_1$, $p_{\varepsilon_1}$)
2:      For $i \in \{1,2\}$, $S^i_s \leftarrow \{s^i | s \in S_s\}$.
3:      Define $B : S^1_s \bigcup S^2_s \to S_s$ such that for all $s^i \in S^i_s$
        and $i \in \{1,2\}$, $B(s^i) = s$ holds.
4:      $\hat{S}_s \leftarrow S_s \bigcup S^1_s \bigcup S^2_s$, $\hat{S} \leftarrow \hat{S}_s \bigcup S_e$.
5:      Define $A^*_{\varepsilon_1} : S_s \to 2^A \backslash \emptyset$ such that for all $s \in$
        $S_s$, $A^*_{\varepsilon_1}(s) = \{a \in A^G(s) | \max_{a' \in A^G(s)} Q(s,a') -$
        $Q(s,a) \leq \varepsilon_1\}$.
6:      $\hat{A} \leftarrow A \bigcup \{\hat{a}\}$.
7:      Define the set of available actions for all $s \in \hat{S}$:
        $A^{\hat{G}}(s) = \begin{cases} A^G(s) & \text{if } s \in S_e, \\ \{\hat{a}\} & \text{if } s \in S_s, \\ A^G(B(s)) & \text{if } s \in S^1_s, \\ A^*_{\varepsilon_1}(B(s)) & \text{if } s \in S^2_s. \end{cases}$
8:      Define the transition function for all
        $s, s' \in \hat{S}$ and $a \in A^{\hat{G}}(s)$: $\hat{T}(s,a)(s') =$
        $\begin{cases} p_{\varepsilon_1} & \text{if } s \in S_s, s' = s^1, \\ 1 - p_{\varepsilon_1} & \text{if } s \in S_s, s' = s^2, \\ T(s,a)(s') & \text{if } s \in S_e, s' \in S, \\ T(B(s),a)(s') & \text{if } s \in S^1_s \bigcup S^2_s, s' \in S. \end{cases}$
9:      return $\hat{G} = (\hat{S}, \hat{S}_s, \hat{S}_e, I, \hat{T}, \mathcal{R}, F)$.
10: end function
11: function RecoverHatStrategy($\hat{G}$, $\hat{\sigma}_s$)
12:     For all $s \in S_s$ and $a \in A^{\bar{G}}(s)$, $\bar{\sigma}_s(s,a) \leftarrow$
        $\hat{T}(s,\hat{a})(s^1)\hat{\sigma}_s(s^1,a) + \hat{T}(s,\hat{a})(s^2)\hat{\sigma}_s(s^2,a)$, where $\hat{T}$ is
        the transition function of $\hat{G}$.
13:     return $\bar{\sigma}_s$.
14: end function
```

- $S = W_{as}^{in}$, $S_s = S_s^{in} \bigcap W_{as}^{in}$, $S_e = S_e^{in} \bigcap W_{as}^{in}$, $I = I^{in} \bigcap W_{as}^{in}$, and $F = F^{in} \bigcap W_{as}^{in}$.
- For all $s \in S_s$, $A^G(s) = \{a \in A^{G^{in}}(s) : E^{G^{in}}(s,a) \subseteq W_{as}^{in}\}$; for all $s \in S_e$, $A^G(s) = A^{G^{in}}(s)$.
- For all transition (s,a,s') in G, $T(s,a)(s') = T^{G^{in}}(s,a)(s')$, $\mathcal{R}(s,a,s') = \mathcal{R}^{in}(s,a,s')$.

The set of almost-sure winning system strategies in G^{in} is a subset of the system strategies in G. Therefore if a system strategy σ_s is ε-optimal at $s \in S$, it is also ε-optimal over all almost-sure winning strategies at s in G^{in}. So we transform Problem 1 into learning a memoryless almost-sure winning system strategy that is ε-optimal in G.

We first consider the two objectives in Problem 1 when the reward functions and transition distributions are known. Without the Buchi objective, the discounted-sum objective degenerates to learning a system strategy that is ε-optimal at the infinitely-often-visited states, which can be solved with some slight modification of the R-max algorithm [Brafman and Tennenholtz, 2003]. However, R-max is not applicable to the normal discounted-sum objective, as the Buchi condition is given independently from the reward function, and there is no standard parameterization of the space of all almost-sure winning strategies. This difficulty is common for all RL algorithms that were designed to optimize a reward function.

To avoid this difficulty, we consider the discounted-sum objective first. For discounted rewards, a memoryless system strategy σ_s is optimal if and only if $Q_{\sigma_s}(s,a) = V^*(s)$ holds for all $s \in S_s$ and $a \in \{a' \in A^G(s) : \sigma_s(s,a') > 0\}$ [Filar and Vrieze, 1996]. In other words, σ_s is optimal when it only takes the *optimal actions*. We construct a game G' from G such that the two games are identical except that all actions available to the system states in G' are optimal actions in G. Thus any system strategy in G' is optimal in G. If $W_{as}^{G'}$ coincides with W_{as}^G, there exists a memoryless almost-sure winning strategy σ_s in G', which is also almost-sure winning in G as there are no restrictions to the environment in G'. Then σ_s is an optimal almost-sure winning system strategy in G.

However, $W_{as}^{G'}$ may be a proper subset of W_{as}^G, as the system has fewer actions available in G'. If such is the case, we would have to allow the system to take some suboptimal actions in G in order to preserve the almost-sure winning region. Intuitively, if a system strategy σ_s' specifies only a small probability to the suboptimal actions, V_{σ_s} can still be near-optimal. The following lemma shows that there exists such a σ_s' that is almost-sure winning in G.

Lemma 1. *Let $G^{in} = (S^{in}, S_s^{in}, S_e^{in}, I^{in}, A^{in}, T^{in}, F^{in})$ be a turn-based Buchi game, and W_{as}^{in} be the almost-sure winning region of G^{in}. Let $G = G^{in} \restriction W_{as}^{in} = (S, S_s, S_e, I, A, T, F)$. Then a memoryless system strategy σ_s such that $\sigma_s(s,a) > 0$ for all $s \in S_s$, $a \in A^G(s)$ is almost-sure winning in G.*

To bound the probability of taking suboptimal actions, we construct a game $\hat{G} =$ **HatGame**$(G, Q^*, \varepsilon_1, p_{\varepsilon_1})$ as in Algorithm 1. For each system state s, we have an additional action $\hat{a}$ and two additional system states s^1 and s^2 in $\hat{G}$ such that $\hat{a}$ is the unique available action at s and

Algorithm 2 Overall algorithm for Problem 1

```
Input: A turn-based Buchi game $G^{in} = (S^{in}, S_s^{in}, S_e^{in}, I^{in}, A^{in}, T^{in}, \mathcal{R}^{in}, F^{in})$ satisfying
    Assumptions 1 - 3, reward upper bound $R_{max} > 0$, a discount factor $\gamma \in (0,1)$, a suboptimality bound $\varepsilon > 0$,
    a confidence bound $\delta_c \in (0,1)$.
 1: Compute the almost-sure winning region $W_{as}^{in}$ and a memoryless almost-sure winning strategy $\sigma_s$ for the system in $G^{in}$.
 2: $G := G^{in} \restriction W_{as}^{in} = (S, S_s, S_e, I, A, T, \mathcal{R}, F)$.
 3: Initialize the game model $\bar{G} \leftarrow (S, S_s, S_e, I, A, \bar{T}, \bar{\mathcal{R}}, F)$ such that $\bar{G}$ shares the
    same set of transitions with $G$ and all transition distributions in $\bar{T}$ are uniform; $\bar{\mathcal{R}}(s,a,s') \leftarrow \frac{R_{max}}{1-\gamma}$ for all
    transition $(s,a,s')$.
 4: $\sigma_{s,\varepsilon} \leftarrow \sigma_s$.
 5: For all existing transition $(s,a,s')$ in $\bar{G}$, $k(s,a,s') \leftarrow 0$, $L(s,a) \leftarrow 0$, $\bar{Q}^*(s,a) \leftarrow \frac{R_{max}}{(1-\gamma)^2}$.
 6: $\delta \leftarrow \frac{\varepsilon(1-\gamma)^2 \log(\gamma)}{6R_{max}|S| \log(\varepsilon(1-\gamma)^2/6R_{max})}$, $K \leftarrow \frac{1}{2\delta^2} \log \frac{4|A||S|^2}{\delta_c}$.
 7: $\varepsilon_1 \leftarrow \frac{\varepsilon}{12}$, $p_{\varepsilon_1} \leftarrow \frac{\varepsilon(1-\gamma)^2}{12R_{max} - \varepsilon(1-\gamma)^2}$.
 8: while TRUE do
 9:     if $s \in S_s$ then
10:         Take $\sigma_{s,\varepsilon}$ for one step.
11:     else
12:         Environment takes a transition.
13:     end if
14:     Observe the transition $(s,a,s')$ and the reward $r$.
15:     if $L(s,a) = 0$ then
16:         $k(s,a,s') \leftarrow k(s,a,s') + 1$, $\bar{\mathcal{R}}(s,a,s') \leftarrow r$.
17:         if $\sum_{s' \in S} k(s,a,s') \geq K$ or $|E^{\bar{G}}(s,a)| = 1$ then
18:             $L(s,a) \leftarrow 1$.
19:             For all $s' \in S$, $\bar{T}(s,a)(s') \leftarrow \frac{k(s,a,s')}{\sum_{s' \in S} k(s,a,s')}$.
20:             Update the optimal action value function $\bar{Q}^*$.
21:             Construct $\hat{G} \leftarrow$ HatGame$(\bar{G}, \bar{Q}^*, \varepsilon_1, p_{\varepsilon_1})$.
22:             Compute a memoryless almost-sure winning strategy $\hat{\sigma}_s$ for the system in $\hat{G}$.
23:             $\sigma_{s,\varepsilon} \leftarrow$ RecoverHatStrategy$(\hat{G}, \hat{\sigma}_s)$.
24:         end if
25:     end if
26: end while
```

$E^{\hat{G}}(s,\hat{a}) = \{s^1, s^2\}$. At s^1 the system can take all actions in $A^G(s)$; at s^2 the system can only take the optimal actions. The transition probability from s to s^1 is confined to be a small number p_ε. For each memoryless system strategy $\hat{\sigma}_s$ in $\hat{G}$, we can construct a memoryless system strategy $\sigma_s =$ **RecoverHatStrategy**$(\hat{G}, \hat{\sigma}_s)$ in G. As expected, such σ_s is near-optimal in G, as shown in Lemma 2.

Lemma 2. *Let G and $\hat{G}$ be two turn-based Buchi games such that $\hat{G} =$ **HatGame**$(G, Q^*, \varepsilon_1, p_{\varepsilon_1})$, where Q^* is the optimal action value function of G, and $\varepsilon_1 > 0, p_{\varepsilon_1} \in (0,1)$ are constants. The reward function $\mathcal{R}$ in G is bounded by $\frac{R_{max}}{1-\gamma}$, where $\gamma \in (0,1)$ is the discount factor. If $\hat{\sigma}_s$ is a strategy* *for the system in $\hat{G}$, then σ_s =* ***RecoverHatStrategy***$(\hat{G}, \hat{\sigma}_s)$ *is* $\left(\varepsilon_1(1-p_{\varepsilon_1}) + p_{\varepsilon_1}\frac{R_{max}}{(1-\gamma)^2}\right)$*-optimal in G.*

The following lemma guarantees that the strategies σ_s and $\hat{\sigma}_s$ can only be almost-sure winning simultaneously.

Lemma 3. *The games G, $\hat{G}$ and the strategies σ_s, $\hat{\sigma}_s$ are defined as in Lemma 2. Then $\hat{\sigma}_s$ is almost-sure winning for the system in $\hat{G}$ if and only if the constructed σ_s is almost-sure winning for the system in G.*

Therefore, given the optimal action value function Q^*, we can solve a memoryless ε-optimal almost-sure winning strategy in G as follows: first properly construct a game $\hat{G}$, then synthesize a memoryless almost-sure winning strategy $\hat{\sigma}_s$ in $\hat{G}$ (whose existence is shown in Lemma 1), and eventually recover a memoryless strategy σ_s in G from $\hat{\sigma}_s$. The constructed game $\hat{G}$ changes when the optimal actions identified by Q^* changes. But the recovered strategy σ_s is always almost-sure winning in G, no matter what Q^* is.

Now we return to Problem 1, where both the reward function and the transition distributions are unknown. As in the R-max algorithm, we refine an optimistically-initialized game model $\bar{G}$ in an online manner. With the help of Algorithm 1, we can update the system strategy $\sigma_{s,\varepsilon}$ every time a new transition is learned, and always keep $\sigma_{s,\varepsilon}$ to be both almost-sure winning and ε-optimal with respect to its current model $\bar{G}$. The overall algorithm is shown in Algorithm 2, and Theorem 1 follows from an argument similar to the correctness proof of the R-max algorithm. A complete proof can be found in the technical version at https://www.ae.utexas.edu/facultysites/topcu/archive/ijcai16.html.

Theorem 1. *Let G^{in} be a turn-based Buchi game, R_{max} be a positive upper bound of the reward, $\gamma \in (0,1)$ be a discount factor, $\varepsilon > 0$ be a suboptimality bound and $1-\delta_c \in (0,1)$ be a confidence lower bound, as given in the input of Algorithm 2. The system strategy $\sigma_{s,\varepsilon}$ in Algorithm 2 is always memoryless and almost-sure winning. With probability no less than $1-\delta_c$, by taking $\sigma_{s,\varepsilon}$, the future discounted reward from the current state s is at least $V^*(s) - \varepsilon$, except for some number of steps polynomial in $|S|, |A|, \frac{1}{\varepsilon}$ and $\frac{1}{\delta_c}$.*

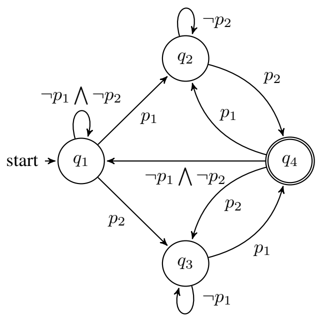

Figure 1: A DBA constructed for the example. p_1 stands for the lower left block, and p_2 stands for the upper right block.

5 Example

We show the usage of our algorithm with a robot motion planning problem which involves simultaneous resource collection and surveillance. This example was run on a laptop with an 8 Intel(R) Core(TM) 2.40GHz CPU and 8 GB memory.

We first introduce the turn-based game and task requirements. The system moves in a 3-by-3 grid world, and it has to move to an adjacent block if the current state is a system state. The environment is a signal light that indicates the dangerous area in the world at the current step which needs to be monitored closely. If the environment light is on, the upper left four blocks are dangerous; otherwise, the lower right four blocks are dangerous. The environment can arbitrarily decide the status of the light in the next step if the current state is an environment state. Furthermore, the lower left block and the upper right block are labeled as post offices. For ease of demonstration, we assume that all transitions are deterministic, i.e. $|E^G(s,a)| = 1$ for all state-action pair (s,a).

We want to learn a strategy for the system to both patrol the dangerous areas and persistently visit the two post offices. We first interpret the task requirements as a almost-sure winning objective and a discounted-sum objective, encode them as inputs to our algorithm, and then show the results.

Almost-Sure Winning Objective The task of visiting the two post offices can be expressed by the four-state DBA in Figure 1. The initial state is q_1, and the set of accepting states is $\{q_4\}$. The upper right block is labeled by 'p_1' ('post office #1') and the lower left block is labeled by 'p_2' ('post office #2'). We show that the Buchi condition is satisfied if and only if both p_1 and p_2 are visited infinitely often. Starting from the initial state, the system transits to q_2 if it visits p_1, or transits to q_3 if it visits p_2. If it visits neither of them, it stays at q_1. From q_2 and q_3, the system should visit the other post office (p_2 for q_2 and p_1 for q_3) in order to enter q_4. q_4 has the same outgoing transitions as q_1. The transitions show that one new visit to q_4 requires at least one new visit to p_1 and one new visit to p_2. Therefore to satisfy the Buchi condition, i.e. to visit q_4 infinitely often, the system has to visit p_1 and p_2 infinitely often. All initial states are with DBA state q_1.

The Discounted-Sum Objective The task of monitoring the dangerous area is interpreted as a discounted-sum objective. A reward function is designed to encourage the system to patrol the dangerous area. The system will be rewarded by 1 in the following cases: (1) when the system transits into the dangerous area; (2) when the light is on and the system moves counterclockwise in the dangerous area; (3) when the light is off and the system moves clockwise in the dangerous area. For all other system transitions and all environment transitions, there is no reward. Throughout this example, the discount factor γ is 0.6, and $R_{max} = 1$.

The system does not know this reward function ahead of time, but eventually manages to learn a strategy with optimal worst-case discounted reward. As shown in Figure 2, the system learns to approach the area specified by the environment as soon as possible and then move in the corresponding direction to maximize the reward.

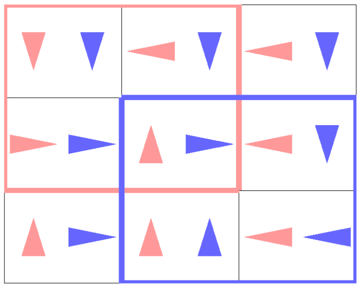

Figure 2: The optimal strategy for the system with only the discounted reward. The pink and blue squares represent the dangerous areas when the light is on and off. The triangles show the optimal transition directions from each block (pink ones for light on, blue ones for light off).

We get the product of the original turn-based game and the DBA, which results in a turn-based Buchi game G^{in}. The almost-sure winning region W_{as}^{in} and a memoryless almost-sure winning strategy σ_s are computed with the off-the-shelf tool PGSolver [Friedmann and Lange, 2015]. It turns out that W_{as}^{in} is the whole state space, and σ_s is illustrated in Figure 3. The suboptimality bound ε is set to be 0.0001. To output the learned strategy in a timely manner, we added a terminating condition to the while loop in Algorithm 2 such that the algorithm stops if there are no updates in the last 10,000 steps.

Upon termination, the learned strategy for the system is shown in Figure 4. The strategy is randomized and allows two actions at each system state, one with probability $(1 - p_{\varepsilon_1})$ and the other with probability p_{ε_1}, represented by the big triangles and small triangles respectively. The worst-case value functions for the learned strategy, the initial almost-sure winning strategy, and an optimal strategy are shown in Figure 5. These value functions are evaluated with the true reward function, and thus are not accessible to the system. It can be found that the value of the learned strategy is much better than that of the initial strategy, although it is not at all close to the optimal strategy. In Figure 4, we marked all states where the learned strategy is ε-optimal with yellow background. It turns out that all system states are marked with yellow, i.e. are ε-optimal, except those that are not reachable from the initial states. The product Buchi game has 144 states, 72 system states and 192 transitions. On average of ten repetitive experiments, the algorithm terminates at 58.05 seconds with the last update occurs at 29.77 seconds.

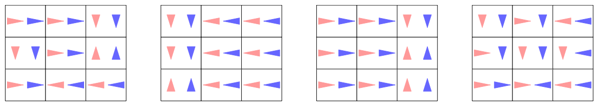

Figure 3: The initial almost-sure winning strategy σ_s. From left to right, the four figures show the system strategy with DBA state q_1 to q_4. In each figure, the pink and blue triangles point to the transition directions at each block when the light is on and off respectively.

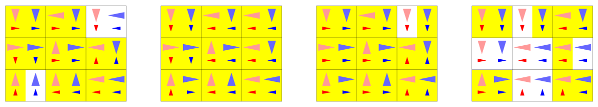

Figure 4: The learned almost-sure winning strategy $\sigma_{s,\varepsilon}$ upon the algorithm termination as no updates occurred in the past 10,000 steps. From left to right, the four figures show the system strategy with DBA state q_1 to q_4. In each figure, the pink and blue triangles point to the transition directions at each block when the light is on and off respectively. Big triangles represent actions with probability $(1 - p_{\varepsilon_1})$, and small triangles represent actions with probability p_{ε_1}. Triangles with yellow background are ε-optimal over all almost-sure winning system strategies.

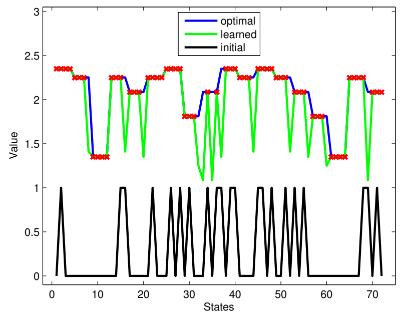

Figure 5: Comparison of the value function of the initial almost-sure winning strategy, the learned strategy and an optimal strategy (which may not be almost-sure winning) for all system states. The red crosses mark all the strongly connected components in which there is at least one state whose value is learned to be ε-optimal.

6 Conclusion

We studied the strategy synthesis in turn-based stochastic games with both qualitative Buchi objectives and quantitative discounted-sum objectives. A PAC learning algorithm is proposed to learn a memoryless ε-optimal almost-sure winning strategy when the reward function and transition distributions are unknown a priori. We also demonstrated the algorithm on an example involving resource-collection and surveillance tasks.

Acknowledgments

This work was partially supported by awards AFRL FA8650-15-C-2546, ONR N000141310778, ARO W911NF-15-1-0592, NSF 1550212 and DARPA W911NF-16-1-0001.

References

[Baier *et al.*, 2008] Christel Baier, Joost-Pieter Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

[Blahoudek, 2015] Fanda Blahoudek. Ltl3dra - ltl to deterministic rabin automata translator based on ltl3ba, 2015.

[Brafman and Tennenholtz, 2003] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

[Chatterjee and Doyen, 2010] Krishnendu Chatterjee and Laurent Doyen. Energy parity games. In *Automata, Languages and Programming*, pages 599–610. Springer, 2010.

[Chatterjee *et al.*, 2003] Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A Henzinger. Simple stochastic parity games. In *Computer Science Logic*, pages 100–113. Springer, 2003.

[Chatterjee *et al.*, 2005a] Krishnendu Chatterjee, Luca De Alfaro, and Thomas A Henzinger. *The complexity of stochastic Rabin and Streett games*. Springer, 2005.

[Chatterjee *et al.*, 2005b] Krishnendu Chatterjee, Thomas A Henzinger, and Marcin Jurdzinski. Mean-payoff parity games. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 178–187. IEEE, 2005.

[Chatterjee *et al.*, 2012] Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy synthesis for multi-dimensional quantitative objectives. In *CONCUR 2012–Concurrency Theory*, pages 115–131. Springer, 2012.

[Chatterjee *et al.*, 2014] Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Youssouf Oualhadj. Perfect-information stochastic mean-payoff parity games. In *Foundations of Software Science and Computation Structures*, pages 210–225. Springer, 2014.

[Chen *et al.*, 2013] Taolue Chen, Marta Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *Quantitative Evaluation of Systems*, pages 322–337. Springer, 2013.

[Filar and Vrieze, 1996] Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer-Verlag New York, Inc., 1996.

[Friedmann and Lange, 2009] Oliver Friedmann and Martin Lange. The pgsolver collection of parity game solvers. *University of Munich*, 2009.

[Friedmann and Lange, 2015] Oliver Friedmann and Martin Lange. tcsprojects/pgsolver, 2015.

[Fu and Topcu, 2014] Jie Fu and Ufuk Topcu. Probably approximately correct mdp learning and control with temporal logic constraints. *arXiv preprint arXiv:1404.7073*, 2014.

[Gaiser *et al.*, 2012] Andreas Gaiser, Jan Křetínskỳ, and Javier Esparza. Rabinizer: Small deterministic automata for ltl (f, g). In *Automated Technology for Verification and Analysis*, pages 72–76. Springer, 2012.

[Guo *et al.*, 2013] Meng Guo, Karl H Johansson, and Dimos V Dimarogonas. Revising motion planning under linear temporal logic specifications in partially known workspaces. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5025–5032. IEEE, 2013.

[Klein, 2015] Joachim Klein. Ltl2dstar - ltl to deterministic streett and rabin automata, 2015.

[Kress-Gazit *et al.*, 2007] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Where's waldo? sensor-based temporal logic motion planning. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3116–3121. IEEE, 2007.

[Littman, 2001] Michael L Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.

[Smith *et al.*, 2011] Stephen L Smith, Jana Tumova, Calin Belta, and Daniela Rus. Optimal path planning for surveillance with temporal logic constraints. *The International Journal of Robotics Research*, page 0278364911417911, 2011.

[Strehl *et al.*, 2009] Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *The Journal of Machine Learning Research*, 10:2413–2444, 2009.

[Tsai *et al.*, 2013] Ming-Hsien Tsai, Yih-Kuen Tsay, and Yu-Shiang Hwang. Goal for games, omega-automata, and logics. In *Computer Aided Verification*, pages 883–889. Springer, 2013.

[Wen *et al.*, 2015] Min Wen, Rudiger Ehlers, and Ufuk Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4983–4990. IEEE, 2015.

[Wolff *et al.*, 2013] Eric M Wolff, Ufuk Topcu, and Richard M Murray. Efficient reactive controller synthesis for a fragment of linear temporal logic. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5033–5040. IEEE, 2013.