An Approximate Dynamic Programming Approach to Multiagent Persistent Monitoring in Stochastic Environments With Temporal Logic Constraints

Kun Deng, Yushan Chen, and Calin Belta, *Fellow, IEEE*

***Abstract*—We consider the problem of generating control policies for a team of robots moving in a stochastic environment. The team is required to achieve an optimal surveillance mission, in which a certain "optimizing proposition" needs to be satisfied infinitely often. In addition, a correctness requirement expressed as a temporal logic formula is imposed. By modeling the robots as game transition systems and the environmental elements as Markov chains, the problem reduces to finding an optimal control policy for a Markov decision process, which also satisfies a temporal logic specification. The existing approaches based on dynamic programming are computationally intensive, thus not feasible for large environments and/or large numbers of robots. We propose an approximate dynamic programming (ADP) framework to obtain suboptimal policies with reduced computational complexity. Specifically, we choose a set of basis functions to approximate the optimal costs and find the best approximation through the least-squares method. We also propose a simulation-based ADP approach to further reduce the computational complexity by employing low-dimensional calculations and simulation samples.**

***Index Terms*—Approximate dynamic programming (ADP), Markov decision process (MDP), multiagent system, simulation-based method, temporal logic constraints.**

Manuscript received March 29, 2016; revised September 6, 2016; accepted February 7, 2016. Date of publication March 6, 2017; date of current version August 28, 2017. This work was supported in part by the Office of Naval Research under Grant MURI N00014-09-1051, the Army Research Office under Grant W911NF-09-1-0088, and the Air Force Office of Scientific Research under Grant YIP FA9550-09-1-0209 at Boston University. This paper was presented in part at the IEEE Conference on Decision and Control, Maui, HI, USA, Dec. 10–13, 2012. Recommended by Associate Editor L. Wu.

K. Deng was with the Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA. He is now with Ford Motor Company, Dearborn, MI 48126 USA (e-mail: kundeng2@gmail.com).

Y. Chen was with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215 USA. She is now with Orbeus Inc., Chicago, IL 60654 USA (e-mail: yushanc@bu.edu).

C. Belta is with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215 USA. (e-mail: cbelta@bu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAC.2017.2678920

I. Introduction

TEMPORAL logics, such as linear temporal logic (LTL) and computation tree logic, have been customarily used in formal verification to specify correctness requirements for computer programs and digital circuits [1]. In recent years, temporal logics have also been used as rich and expressive specification languages for control systems [2], [3]. Most of the existing results focus on finite systems with perfect state information and specifications given in LTL or fragments of LTL. For deterministic transition systems, control strategies from specifications given as LTL formulas over a set of state labels can be found through adaptations of off-the-shelf model checking algorithms [1], [4]–[6]. If the system is nondeterministic, the control problem with LTL specifications can be mapped to a Rabin game [7] in general and to a Büchi [8] or GR(1) game [9], [10] if the specifications are restricted to fragments of LTL. For probabilistic systems, the problem reduces to finding a control policy that maximizes the probability of satisfying an LTL formula [1], [11]–[13].

In many application areas, policies that maximize the satisfaction probability are generally not unique, but rather belong to a family of policies with the same "transient response" that maximizes the probability of reaching a specific set of accepting states [1], [12], [14]. To find an optimal policy that optimizes the long-term behavior of the system after reaching the accepting states, one can consider combining the temporal logic constraints with the optimality requirements. Such a problem was first approached in [15] for a finite weighted deterministic transition system model. More complicated deterministic models with locally sensed rewards in the states of the system were studied in [16]–[18]. For probabilistic systems, the problem was studied in our recent work [11], [19], [20] by modeling the system as a Markov decision process (MDP) under temporal logic constraints. In particular, we formulated an MDP optimization problem in terms of minimizing the average cost per cycle (ACPC), where cycles are defined by successive satisfactions of the optimizing proposition. A product MDP is constructed using an MDP capturing the environment uncertainties and an automaton representing the LTL formulas. To solve this problem, we first transformed this MDP minimizing the

ACPC into another equivalent MDP in terms of minimizing the average cost per stage (ACPS), which is a more standard MDP problem that have been well studied in the literature [21], [22]. Using such equivalence, we successfully developed a dynamic programming (DP) algorithm to synthesize an optimal control policy for the MDP under LTL specification constraints [20].

In this paper, we consider the problem controlling a team of robots to perform a persistent monitoring task in a stochastic environment. The goal is to optimize the long-term behavior of the robot team by minimizing the average time in between consecutive executions of the persistence task. In addition, the team should satisfy correctness requirements expressed as LTL formulas. We only consider a centralized control problem for this multiagent system, i.e., there is effectively a single decision maker knowing the global state information. By modeling the robots as game transition systems and the environments as Markov chains, the problem reduces to finding an optimal control policy for an MDP, which also satisfies a temporal logic specification.

The algorithm presented in [20] can be computationally expensive for problems considered here, particularly for problems with large environments/large numbers of robots/complex temporal logic constraints. First, the constructed MDP can be quite large since it is a product of an MDP capturing environment uncertainties and an automaton representing LTL constraints. Second, the transition probability matrix for the transformed MDP needs to be calculated through the inversion of the product MDP transition matrix [20]. Third, the computational requirements of the DP algorithm can become quite prohibitive for large-size MDPs [21].

Motivated by these limitations, the focus of this paper is to develop efficient algorithms with reduced complexity that provide suboptimal solutions to the MDP problem under LTL constraints. We propose a new ADP framework to approximately solve the MDP considered in this paper. A preliminary version of this paper appeared in conference proceedings [23]. In addition to [23], this paper includes a simulation-based approach together with a theoretical analysis of the convergence results and more technical details. The main contribution of the proposed ADP framework is twofold. First, instead of directly solving the transformed MDP, we transform the Bellman equation into an equivalent but simpler form. In such a way, we eliminate the need to calculate the matrix inversions for the transformed MDP. Then, we approximately solve the transformed Bellman equation through the linear parameterization and least-squares method. In particular, a set of basis functions is chosen based on the Krylov space method [24] to approximate the solution of the transformed Bellman equation. The best approximation parameters are obtained minimizing the least-squares error [21], [25]. Second, to further reduce the complexity of the proposed ADP approach, we propose a simulation-based method to approximate the high-dimensional matrix multiplications involved in obtaining the least-squares parameters [26]–[29]. We also establish the convergence results and analyze the convergence rate of the simulation-based algorithm.

II. Problem Formulation

A. Problem Formulation

In this paper, we consider robot missions requiring infinite executions, such as surveillance, persistent monitoring, or pickup-delivery tasks. LTL offers a formal framework for describing such missions [1]. A detailed description of the syntax and semantics of LTL is beyond the scope of this paper and can be found in [1]. Roughly speaking, an LTL formula is built up from a set of atomic propositions Π, standard Boolean operators $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), $\Rightarrow$ (implication), and temporal operators X (next), U (until), F (eventually), G (always). The semantics of LTL formulas are given over infinite words over 2^{Π}. For example, the specification "monitor infinitely many times, and come back to base every time after monitoring" translates to formula $\phi = \mathsf{GF\,Monitor} \wedge \mathsf{G}(\mathsf{Monitor} \Rightarrow \mathsf{X\,Base})$.

In addition, we define a special task, called "optimizing task," which is required to be executed infinitely often. We want to minimize the time in between two consecutive executions of this task. Specifically, our specification is an LTL formula of the form:

$$\phi = \varphi \wedge \mathsf{GF}\psi \tag{1}$$

where φ can be any LTL formula over Π, and ψ is a boolean combination of atomic propositions in Π. In other words, the satisfaction of ϕ requires that φ is satisfied and ψ is satisfied infinitely often. We use ψ to capture the optimizing task and φ to specify other missions or rules that must be obeyed. Our goal is to control a team of robots to accomplish a mission in the form of (1) and also minimize the time in between two consecutive executions of the optimizing task ψ.

We consider a team of robots moving in an environment consisting of both static and stochastically changing elements. More specifically, we consider an indoor-like environment consisting of rooms (static) and doors (changing and stochastic), which is inspired from [30]. To keep the discussion focused, we summarize the detailed definitions of the environment, door, and robot models as well as control policy in Appendix A. Given the initial room locations of the robots $\{v_k^{in}, k \in \mathcal{R}\}$, the control policies $\{\pi_k, k \in \mathcal{R}\}$, and the door behaviors, we can produce a set of robot paths $\{\mathbb{P}_k, k \in \mathcal{R}\}$, where the next room v_k^{n+1} for robot k is determined by both control μ_k^n and associated door status at time instance $\mathbb{T}_k^n$. Note that the resultant robot paths $\{\mathbb{P}_k, k \in \mathcal{R}\}$ and the corresponding team behaviors of robots are not unique due to the stochasticity of the door behaviors. The probability that the set of team behaviors satisfies an LTL formula is well defined since team behaviors can be modeled as an MDP [5]. Now, we are ready to formulate the problem:

Problem 1: Given a partitioned environment $\mathcal{E}$ [see (28)], a team of robots modeled as game transition systems $\{\mathcal{T}_k, k \in \mathcal{R}\}$ [see (30)], a set of doors modeled as Markov Chains $\{\mathcal{C}_i, i \in \mathcal{D}\}$ [see (29)], and a specification in the form of an LTL formula ϕ [see (1)], synthesize a set of robot control policies $\{\pi_k, k \in \mathcal{R}\}$ for the team, such that the team behaviors generated by $\{\pi_k, k \in \mathcal{R}\}$ satisfy ϕ with the maximum probability and $J(\{v_k^{in}, k \in \mathcal{R}\}) = \limsup_{n\to\infty} E((\mathbb{T}^{\psi}_{\text{team}}(n+1) - \mathbb{T}^{\psi}_{\text{team}}(n))$ is minimized, where $E(\cdot)$ denotes the expectation operator and $\mathbb{T}^{\psi}_{\text{team}}(n)$ stands for the time instance when the optimizing task ψ is executed for the nth time.

B. MDP Construction

We first define a composition of game transition systems modeling robots and Markov chains capturing door behaviors in form of an MDP (see Appendix A for definitions and notations). We start by equipping each robot $k \in \mathcal{R}$ with a clock, which keeps track of the amount of time that the robot has been traveling between robot states in the game transition system $\mathcal{T}_k$. We initiate all the clocks at zeros. Given two room states (v, v') such that $v \to_k v'$, robot k can transit from v to v' only when the current clock value plus 1 is equal to the travel time of this transition. When the clock value is smaller than the required travel time, the robot is in an intermediate state, which means that the robot has left v and is moving toward its target location v'. Thus, for each robot, we insert a new state denoted by $v^{\to v'}, \forall (v, v') \in \to_{\mathcal{E}}$, such that the travel time between v and v' is greater than 1, to represent the intermediate room state between v and v'.

In addition, the MDP also captures how the doors affect the motion of the robotic team. If robot k is at v and plans to move to v', where $\exists i \in \mathcal{D}$ such that $v \to_k q_i^v \to_k v'$ (i.e., v and v' are separated by a door), the next state of robot k is decided by the current status of door i. If door i is open, the robot starts moving from state v to v', and similarly, if $g_k(q_i^v, v') > 1$, the robot transits to the intermediate state $v^{\to v'}$. If the door is closed, the robot stays at the same state. Therefore, the MDP can be seen as a special product of the game transition systems $\mathcal{T}_k$ with the set of clocks and the Markov Chains $\mathcal{C}_i$. Formally, it is defined as follows (An example for constructing MDP is illustrated in Fig. 1.):

$$\mathcal{M}_{\mathcal{G}} = (Q_{\mathcal{G}}, \iota_{\mathcal{G}}, U_{\mathcal{G}}, P_{\mathcal{G}}, \Pi_{\mathcal{G}}, L_{\mathcal{G}}, g_{\mathcal{G}}) \tag{2}$$

where

1) $Q_{\mathcal{G}} \subset \overline{\mathcal{V}}^{|\mathcal{R}|} \times \prod_{i\in\mathcal{D}} S_i \times \mathbb{N}^{|\mathcal{R}|}$ is a set of states, where $\overline{\mathcal{V}} = \mathcal{V} \times Q_{\mathcal{D}} \times \mathcal{V}^{\text{mid}}$ denotes an extended set of room states in $\mathcal{V}$, which is comprised of the rooms, auxiliary door states in $Q_{\mathcal{D}}$ that interact with doors, and intermediate states in $\mathcal{V}^{\text{mid}}$ that account for travel time between rooms; in the following, we denote $q_{\mathcal{G}} = (\mathsf{st}_r, \mathsf{st}_d, \mathsf{clk})$ to represent a state in $Q_{\mathcal{G}}$, where $\mathsf{st}_r = (\overline{v}_1, \ldots, \overline{v}_{|\mathcal{R}|}) \in \overline{\mathcal{V}}^{|\mathcal{R}|}$ is the vector for extended set of states, $\mathsf{st}_d = (s_1, \ldots, s_{|\mathcal{D}|}) \in \prod_{i\in\mathcal{D}} S_i$ is the vector for door states, and $\mathsf{clk} = (\mathsf{clk}_1, \ldots, \mathsf{clk}_{|\mathcal{R}|}) \in \mathbb{N}^{|\mathcal{R}|}$ is the vector for clock values.
2) $\iota_{\mathcal{G}}$ is an initial distribution such that $\iota_{\mathcal{G}}(\mathsf{st}_r, \mathsf{st}_d, \mathsf{clk}) = \prod_{i\in\mathcal{D}} \iota_i(\mathsf{st}_d[i])$, if and only if $\mathsf{st}_r = (v_1^{in}, \ldots, v_{|\mathcal{R}|}^{in})$ and $\mathsf{clk} = (0, 0, \ldots, 0)$, and $\iota_{\mathcal{G}}(\mathsf{st}_r, \mathsf{st}_d, \mathsf{clk}) = 0$ otherwise.
3) $U_{\mathcal{G}}$ is a set of controls and $U_{\mathcal{G}}(\mathsf{st}_r, \mathsf{st}_d, \mathsf{clk}) = (\overline{u}_1, \ldots, \overline{u}_{|\mathcal{R}|})$, where $\overline{u}_k$ represents the control applied to robot $k \in \mathcal{R}$ at extended set of states (if $\overline{v}_k \in \mathcal{V}$ is a room state, we can apply any available control $\overline{u}_k \in U_k(\overline{v}_k)$ for room $\overline{v}_k$; if $\overline{v}_k \in \mathcal{V}^{\text{mid}}$ is an intermediate state, we only apply a dummy control $\overline{u}_k = \epsilon_k$ with no actions).
4) $P_{\mathcal{G}} : Q_{\mathcal{G}} \times U_{\mathcal{G}} \times Q_{\mathcal{G}} \to [0, 1]$ is a transition probability function such that

$$P_{\mathcal{G}}((\mathsf{st}_r, \mathsf{st}_d, \mathsf{clk}), (\overline{u}_1, \ldots, \overline{u}_{|\mathcal{R}|}), (\mathsf{st}'_r, \mathsf{st}'_d, \mathsf{clk}')) = \prod_{i\in\mathcal{D}} P_i(\mathsf{st}_d[i], \mathsf{st}'_d[i])$$

if and only if $\forall k \in \mathcal{R}$, one of the following conditions holds:
   a) $\mathsf{st}_r[k] = v, \mathsf{st}'_r[k] = v', M(v, \overline{u}_k) = v', \mathsf{clk}_k = 0, \mathsf{clk}'_k = 1$, and $\mathcal{F}_{\mathcal{D}}(v, v') = \emptyset$, $g_k(v, v') = 1$ or $\mathcal{F}_{\mathcal{D}}(v, v') = i, L_i^{\mathcal{C}}(\mathsf{st}_d[i]) = o, g_k(q_i^v, v') = 1$;
   b) $\mathsf{st}_r[k] = \mathsf{st}'_r[k] = v$, $M(v, \overline{u}_k) = v$, $\mathsf{clk}_k = 0$, $\mathsf{clk}'_k = 0$, and $\exists v', i$ such that $\mathcal{F}_{\mathcal{D}}(v, v') = i$, $L_i^{\mathcal{C}}(\mathsf{st}_d[i]) = c$;
   c) $\mathsf{st}_r[k] = v$, $\mathsf{st}'_r[k] = v^{\to v'}$, $M(v, \overline{u}_k) = v^{\to v'}$, $\mathsf{clk}_k = 0$, $\mathsf{clk}'_k = 1$, and $\mathcal{F}_{\mathcal{D}}(v, v') = \emptyset$, $g_k(v, v') > 1$ or $\mathcal{F}_{\mathcal{D}}(v, v') = i$, $L_i^{\mathcal{C}}(\mathsf{st}_d[i]) = o$, $g_k(q_i^v, v') > 1$;
   d) $\mathsf{st}_r[k] = \mathsf{st}'_r[k] = v^{\to v'}$, $\overline{u}_k = \epsilon_k$, $1 \le \mathsf{clk}_k < g_k(v, v') - 1$, and $\mathsf{clk}'_k = \mathsf{clk}_k + 1$;
   e) $\mathsf{st}_r[k] = v^{\to v'}$, $\mathsf{st}'_r[k] = v'$, $\overline{u}_k = \epsilon_k$, $clk'_k = 0$, and $\mathcal{F}_{\mathcal{D}}(v, v') = \emptyset$, $g_k(v, v') = clk_k + 1$ or $\mathcal{F}_{\mathcal{D}}(v, v') = i$, $g_k(q_i^v, v') = clk_k + 1$

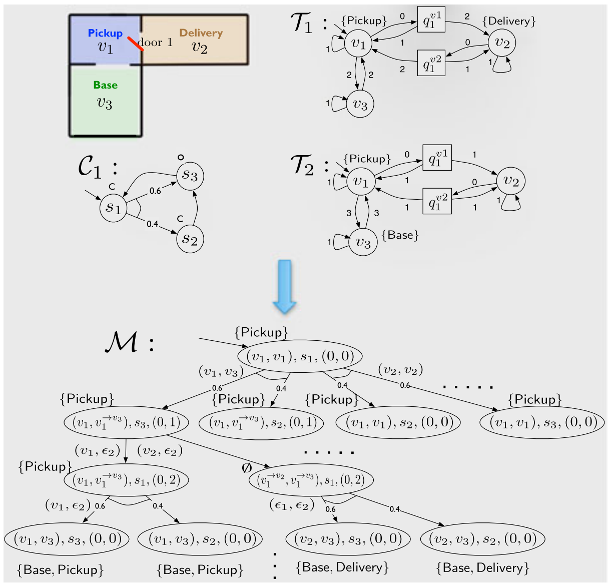


Fig. 1. MDP example for two robots moving in a simple environment with one door. At the initial state $((v_1, v_1), s_1, (0, 0))$, when control (v_2, v_2) is taken, both robots stay at v_1 in the next states (i.e., $((v_1, v_1), s_2, (0, 0))$ and $((v_1, v_1), s_3, (0, 0))$), because the status of door i is c (i.e., door i "closed") at state s_1. If input (v_1, v_3) is chosen at the initial state, the system can transit to state $((v_1, v_1^{\to v_3}), s_3, (0, 1))$ with probability 0.6 since $P_1(s_1, s_3) = 0.6$, $g_1(v_1, v_1) = 1 = \mathsf{clk}_1 + 1$, and $g_2(v_1, v_3) = 3 > \mathsf{clk}_2 + 1$. At state $((v_1, v_1^{\to v_3}), s_3, (0, 1))$, robot 1 can move toward v_2 since the status of door is o at state s_3. Since $g_1(v_1, v_2) = 2 > \mathsf{clk}_1 + 1$, the next move of robot 1 is $v_1^{\to v_2}$ to state $((v_1^{\to v_2}, v_1^{\to v_3}), s_1, (0, 2))$.

and $P_{\mathcal{G}}((\mathsf{st}_r, \mathsf{st}_d, \mathsf{c}lk), (\overline{u}_1, \ldots, \overline{u}_{|\mathcal{R}|}), (\mathsf{st}'_r, \mathsf{st}'_d, \mathsf{c}lk')) = 0$ otherwise.

5) $\Pi_{\mathcal{G}} = \cup_{k \in \mathcal{R}} \Pi_k$ is a set of atomic propositions.
6) $L_{\mathcal{G}} : Q_{\mathcal{G}} \to 2^{\Pi}$ is a labeling function inherited from the game transition system such that $L_{\mathcal{G}}(\mathsf{st}_r, \mathsf{st}_d, \mathsf{c}lk) = \cup_{k \in \mathcal{R}} \{L_k(\mathsf{st}_r[k]) \mid \mathsf{st}_r[k] \in \mathcal{V}\}$.
7) $g_{\mathcal{G}} : Q_{\mathcal{G}} \times U_{\mathcal{G}} \to 1$ is a function that assigns 1 to all transitions.

C. Optimization Problem

As shown in [19], it is sufficient to consider stationary control policies for the constructed MDP $\mathcal{M}_{\mathcal{G}}$. We denote $\mu_{\mathcal{G}} : Q_{\mathcal{G}} \to U_{\mathcal{G}}$ a stationary policy for $\mathcal{M}_{\mathcal{G}}$. Clearly, any stationary policy $\mu_{\mathcal{G}}$ induces a stationary control policy $\mu_k : Q_k \times \prod_{i \in \mathcal{D}} S_i \to U_k$ for each robot $k \in \mathcal{R}$. We denote M_ϕ the set of all policies that maximize the probability of satisfying the LTL formula ϕ. It has been shown that there typically exist many (possibly an infinite number of) policies in the set M_ϕ [12], [14]. Therefore, Problem 1 reduces to finding an optimal stationary policy such that the probability of satisfying ϕ is maximized, and the expected time in between visiting states satisfying ψ is minimized. To formalize this, we let $Q_\psi := \{q \in Q_{\mathcal{G}} \mid \psi \in L_{\mathcal{G}}(q)\}$ denote the set of states where the task ψ to be optimized is feasible. We say that each visit of the MDP path to the set Q_ψ *completes a cycle*. Given an MDP sample path $q^0_{\mathcal{G}} q^1_{\mathcal{G}} \ldots$, we use $C(q^0_{\mathcal{G}}, \ldots, q^N_{\mathcal{G}})$ to denote the cycle index up to stage N, which is defined as the total number of cycles completed at stage N plus 1, i.e., the cycle index starts with 1 at the initial state. We define an ACPC under control policy $\mu_{\mathcal{G}}$ for the MDP $\mathcal{M}_{\mathcal{G}}$:

$$J_{\mu_{\mathcal{G}}} = \limsup_{N \to \infty} \mathsf{E} \left\{ \frac{\sum_{n=0}^{N} g_{\mathcal{G}}(q^n_{\mathcal{G}}, \mu_{\mathcal{G}}(q^n_{\mathcal{G}}))}{C(q^0_{\mathcal{G}} q^1_{\mathcal{G}} \ldots q^N_{\mathcal{G}})} \right\}. \tag{3}$$

Then, Problem 1 reduces to the following ACPC minimization problem.

Problem 2: Find a stationary policy $\mu^\star_{\mathcal{G}}$ that minimizes the ACPC over all policies that maximize the probability of satisfying ϕ, i.e., $\mu^*_{\mathcal{G}} \in \arg\min_{\mu_{\mathcal{G}} \in M_\phi} J_{\mu_{\mathcal{G}}}$, where $J_{\mu_{\mathcal{G}}}$ is defined in (3).

The existing approaches to solve Problem 2 can be found in [1], [11], and [20] and references therein. These approaches generally consist of two major steps.

1) *LTL synthesis step*, in which a policy in M_ϕ is computed. This step proceeds with constructing a product MDP $\mathcal{M}_{\mathcal{P}} = \mathcal{M}_{\mathcal{G}} \times \mathcal{A}_\phi$, where $\mathcal{M}_{\mathcal{G}}$ is the constructed MDP and $\mathcal{A}_\phi$ is a Deterministic Rabin Automaton accepting all and only words satisfying ϕ [1]. Then, a set of accepting maximum end components (AMECs) of $\mathcal{M}_{\mathcal{P}}$ is computed. In fact, an AMEC, denoted as $\mathcal{M}$, is a *communicating* sub-MDP of the MDP $\mathcal{M}_{\mathcal{P}}$. Note that there may exist many AMECs. As stated in [1], a stationary policy on $\mathcal{M}_{\mathcal{P}}$ that can reach an AMEC with the maximum probability induces at least one stationary policy on $\mathcal{M}_{\mathcal{G}}$ that satisfies the LTL formula ϕ with the maximum probability. Such a policy in M_ϕ is in general not unique, and it can be obtained by a variety of algorithms [31].
2) *Optimizing step*, in which an optimal policy in M_ϕ minimizing (3) is obtained through DP. In this step, an optimal stationary policy in M_ϕ minimizing the ACPC cost (3) is computed for each obtained AMEC through the DP algorithm. Note that the optimization is performed on the AMECs instead of the product MDP $\mathcal{M}_{\mathcal{P}}$. This is because given a policy in M_ϕ, the states of $\mathcal{M}_{\mathcal{P}}$ outside AMECs are transient, while the states inside AMECs are long term. Since we aim to optimize the long-term behavior of the system, we only need to solve the optimization problem within each AMEC.

Remark 1: The computational complexity of the LTL synthesis step is bounded above by $O(\tilde{n} \cdot |Q_{\mathcal{P}}|)$, where $Q_{\mathcal{P}}$ is the set of states of $\mathcal{M}_{\mathcal{P}}$ and $\tilde{n}$ denotes the number of transitions in $\mathcal{M}_{\mathcal{P}}$ with nonzero probabilities [31]. Using the algorithm proposed in [32], a lower upper bound $O(\tilde{n} \cdot \min(|Q_{\mathcal{P}}|^{\frac{2}{3}}, \sqrt{\tilde{n}}))$ can be achieved. On the other hand, in the worst-case scenario (the number of states of an AMEC is equal to $|Q_{\mathcal{P}}|$), the computational time for solving DP in the optimizing step is of order $O(|Q_{\mathcal{P}}|^3)$, e.g., using policy iteration (PI) algorithm [20]. Thus, the optimizing step is the computational bottleneck for solving Problem 2. Therefore, in this paper, we focus on developing algorithms to solve the ACPC optimization problem in a more computationally efficient way.

III. Approximate Dynamic Programming Approach

Instead of exactly solving MDPs, numerous methods have been proposed to obtain approximate solutions with reduced complexity, encompassed in a framework known as approximate dynamic programming (ADP) [21], [33], [34]. Approximate policy iteration (API) is the ADP extension of PI. In this section, we present an ADP framework for approximate policy evaluation using simulation-based function approximations. Based on the proposed ADP framework, we develop a computationally appealing algorithm to compute a suboptimal solution to the MDP. We also establish the convergence results of the proposed algorithm.

A. Reformulation of the ACPC Problem

Formally, we consider the following ACPC optimization problem for the optimizing step of Problem 2:

Problem 3: Given an AMEC $\mathcal{M} = (Q, \iota, U, P, \Pi, L, g)$ (a communicating MDP found in LTL synthesis step), and an optimizing task ψ, find a stationary policy $\mu^\star$ that minimizes the cost defined in (3).

In the following, the number of states for $\mathcal{M}$ is denoted as $n = |Q|$. A stationary policy μ on $\mathcal{M}$ induces a finite-state Markov chain, where its set of states is Q and the transition probability from state q to q' is $P(q, \mu(q), q')$. We use P_μ to denote the transition probability matrix:

$$P_\mu(q, q') := P(q, \mu(q), q'). \tag{4}$$

We use g_μ to denote the cost per stage vector:

$$g_\mu(q) := g(q, \mu(q)) \tag{5}$$

which captures the travel time from the state q to another state by taking the control $\mu(q)$. A stationary policy μ is said to be *proper* if, under μ, all initial states have positive probabilities to reach the set Q_ψ in a finite number of stages.

It is shown in [19] that Problem 3 can be equivalently converted to a traditional ACPS problem (see [21]). To achieve this, another new MDP is constructed such that solving the ACPS problem for the new MDP is equivalent to finding a solution to Problem 3. To construct the new MDP, we first denote two $n \times n$ matrices:

$$\overleftarrow{P}_\mu(q,q') = \begin{cases} P_\mu(q,q'), & \text{if } q' \in Q_\psi \\ 0, & \text{otherwise} \end{cases}$$

$$\overrightarrow{P}_\mu(q,q') = \begin{cases} P_\mu(q,q'), & \text{if } q' \notin Q_\psi \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

The matrix $(I - \overrightarrow{P}_\mu)$ is shown to be nonsingular for any proper policy μ (i.e., $(I - \overrightarrow{P}_\mu)^{-1}$ is well defined). Given $\overleftarrow{P}_\mu$ and $\overrightarrow{P}_\mu$, we obtain the transition probability matrix, denoted by $\widetilde{P}_\mu$, and the cost per stage vector, denoted by $\widetilde{g}_\mu$, for the new MDP:

$$\widetilde{P}_\mu := (I - \overrightarrow{P}_\mu)^{-1}\overleftarrow{P}_\mu, \quad \widetilde{g}_\mu := (I - \overrightarrow{P}_\mu)^{-1} g_\mu.$$

As shown in [19], the matrix $\widetilde{P}_\mu$ is also a stochastic matrix. The vector $\widetilde{g}_\mu$ is a weighted cost per stage vector compared to g_μ, where the weight for each state is proportional to the expected number of transitions initiated from the current state to reach the set Q_ψ (the set of states for the task ψ to be optimized is feasible). Given the new MDP with $\widetilde{P}_\mu$ and $\widetilde{g}_\mu$, (3) in Problem 3 is proven to be equal to the following ACPS for the new MDP:

$$J_\mu(\iota) = \lim_{N\to\infty} \mathsf{E}\left\{\frac{\sum_{n=0}^{N} \widetilde{g}(q_n, \mu(q_n))}{N+1}\right\}.$$

The ACPS $J_\mu(\iota)$ does not depend on the initial distribution ι [19]. Thus, we have $J_\mu(\iota) = \lambda_\mu$, where λ_μ is a finite-valued scalar and it is called the *average cost* under policy μ. In addition, a *relative cost* vector h_μ is defined to quantify the total deviation from the average cost:

$$h_\mu := \lim_{N\to\infty} \sum_{k=0}^{N} (\widetilde{P}_\mu^k \widetilde{g}_\mu - \lambda_\mu \mathbf{1}).$$

A key result for the ACPS problem is that, for any policy μ, the average cost λ_μ associated with the relative cost vector h_μ satisfies the Bellman equation:

$$\lambda_\mu \mathbf{1} + h_\mu = \widetilde{g}_\mu + \widetilde{P}_\mu h_\mu. \tag{7}$$

This is a system of n linear equations with $(n+1)$ unknowns. The solution to (7) is unique up to a constant shift for the relative cost vector [21]. The solution to (7) can be made unique by eliminating one degree of freedom, such as adding one more linear equation for h_μ:

$$\mathbf{1}' h_\mu = 0. \tag{8}$$

The unique solution to (7) and (8) can be obtained by the Gauss–Jordan elimination method or by a matrix inversion [23]. We denote $\mu^\star$ as the stationary policy minimizing (3) over all policies on $\mathcal{M}$. Let $\lambda^\star$ and $h^\star$ denote the average cost and relative cost vector corresponding to the policy $\mu^\star$, respectively. One of the traditional methods to find the optimal policy $\mu^\star$ uses the policy iteration algorithm (PIA) [21]. For large state spaces, the PIA is computational intensive. At each iteration of the PIA, the computational complexity is of order $O(n^3)$.

B. Linear Parametric Function Approximation

We employ the function approximation method to approximate the solution to the Bellman equation (7). Using our previous results, we first transform the Bellman equation (7) into an equivalent form that is more computationally convenient.

Lemma 1 (see [19, Prop IV.10]): The Bellman equation (7) can be represented in the following equivalent form:

$$\lambda_\mu \mathbf{1} + h_\mu = g_\mu + P_\mu h_\mu + \lambda_\mu \overrightarrow{P}_\mu \mathbf{1}. \tag{9}$$

Notice that the transformed can help us avoid the calculation of the matrix inversions involved by $\widetilde{P}_\mu$ and $\widetilde{g}_\mu$. Together with (8), (9) can be expressed compactly as

$$A_\mu x_\mu = b_\mu \tag{10}$$

where

$$A_\mu := \begin{bmatrix} I - P_\mu & \mathbf{1} - \overrightarrow{P}_\mu \mathbf{1} \\ \mathbf{1}' & 0 \end{bmatrix}, \; x_\mu := \begin{bmatrix} h_\mu \\ \lambda_\mu \end{bmatrix}, \; b_\mu := \begin{bmatrix} g_\mu \\ 0 \end{bmatrix}. \tag{11}$$

The solution to (10) is unique and can be obtained via the matrix inversion $A_\mu^{-1} b_\mu$, where A_μ is an $(n+1)\times(n+1)$ matrix and b_μ is an $(n+1)\times 1$ vector. The computational complexity to obtain the solution is of order $O(n^3)$ due to the matrix inverse operation. Clearly, the computational complexity mainly comes from the high dimensionality of relative cost vector h_μ. To mitigate the complexity of the problem, we aim to find a lower dimension approximation for h_μ, and a scalar $\lambda \in \mathbb{R}$ to approximate λ_μ. Formally, we approximate h_μ using a linear parametric form $h(r) = \sum_{k=1}^{m} r_k \phi_k$, where r_k is a tunable parameter scalar, ϕ_k is a given *basis function* (a vector with dimension n), and m is a user-defined number to tradeoff between optimality and computational complexity. For a given policy μ, we define a *basis matrix*

$$\Phi_\mu := [\phi_1 \mid \phi_2 \mid \cdots \mid \phi_m]. \tag{12}$$

Then, $h(r)$ can be expressed compactly as $h(r) = \Phi_\mu r$, where $r = [r_1, \ldots, r_m]'$.

Informally speaking, we would like to select independent basis functions to capture the dominated nonlinearities of the relative cost vector h_μ. In this paper, we employ the Krylov subspace method to automatically generate basis functions [24]. As shown in our previous paper [23], a good candidate set of basis functions can be taken as

$$\mathcal{B}_\mu = \{\mathbf{1}, g_\mu, P_\mu g_\mu, \ldots, P_\mu^{\bar{m}} g_\mu, p_\mu, P_\mu p_\mu, \ldots, P_\mu^{\bar{m}} p_\mu\} \tag{13}$$

where $\bar{m}$ is a user-chosen integer. In general, the vectors in the set $\mathcal{B}_\mu$ are not independent to each other. We usually choose $\bar{m}$ large enough, such that we can select m basis functions together with the unit vector $\mathbf{1}$ that are all linear independent by eliminating the dependencies of vectors in the set $\mathcal{B}_\mu$ (e.g., through the Gram–Schmidt process or Householder transformations [35]).

For any given policy μ, by choosing a set of basis functions from $\mathcal{B}_\mu$ [see (13)], we can approximate h_μ with $h(r) = \Phi_\mu r$ and λ_μ with a scalar $\lambda \in \mathbb{R}$, respectively. We denote

$$\Psi_\mu := \begin{bmatrix} \Phi_\mu & \mathbf{0} \\ \mathbf{0}' & 1 \end{bmatrix}, \quad x(r,\lambda) := \Psi_\mu \begin{bmatrix} r \\ \lambda \end{bmatrix} = \begin{bmatrix} h(r) \\ \lambda \end{bmatrix}. \tag{14}$$

The optimal parameter vector $\hat{r}^*_\mu := [r^*_\mu, \lambda^*_\mu]$ can be obtained by minimizing the least-squares error of the Bellman equation (10) [25]: $\min_{r\in\mathbb{R}^m,\lambda\in\mathbb{R}} \|A_\mu x(r,\lambda) - b_\mu\|^2$. By denoting

$$C_\mu := \Psi'_\mu A'_\mu \widehat{\Xi} A_\mu \Psi_\mu, \quad d_\mu := \Psi'_\mu A'_\mu \widehat{\Xi} b_\mu \tag{15}$$

with $\widehat{\Xi} := \mathrm{diag}(1/n, \ldots, 1/n)$ being an $(n+1)\times(n+1)$ diagonal matrix, we can explicitly express the optimal parameter vector as

$$\hat{r}^*_\mu = C^{-1}_\mu d_\mu. \tag{16}$$

Note that C_μ is an $(m+1)\times(m+1)$ matrix and d_μ is an $(m+1)\times 1$ vector. So, the matrix inverse involved in (16) is less demanding than the one involved in obtaining the solution to (10) since we always have $m \le n$. In particular, the computational cost can be largely reduced if one chooses much less number of basis functions than the number of MDP states, i.e., $m \ll n$.

Based on the ADP framework described above, we developed an *approximate policy iteration algorithm* (APIA) in our previous work [23] to compute the suboptimal solution to Problem 3 (see [23, algorithm 1] for details). Note that although the matrices C_μ and d_μ are of low dimensions, the computation of two matrices C_μ and d_μ in (16) still involves matrix multiplications of dimension $(n+1)$. For problems with large n, the explicit computation of C_μ and d_μ may still be very time and memory consuming because of the high-dimensional matrix multiplications. In the remaining of this section, we propose a simulation-based method called simulation-based approximate policy iteration algorithm (S-APIA) to further reduce the computational complexity of the APIA. The main idea is to approximate the high-dimensional matrix multiplications involved in (16) using *simulation samples* and *low-dimensional calculations*; see [26]–[29] for recent results and references therein.

C. Approximation Through Simulation

We first describe a simulation mechanism to approximate C_μ and d_μ. It is based on sampling the Markov chain corresponding to the policy μ. Let $\xi = (1/n, \ldots, 1/n)'$ denote an $n \times 1$ probability vector. Then, $\Xi = \mathrm{diag}(\xi)$ is an $n\times n$ diagonal matrix, and $P_\xi = \mathbf{1}\xi'$ is a special $n\times n$ stochastic matrix with identical rows and columns all as ξ. Using (11), (14), and (15), we can represent C_μ and d_μ equivalently as

$$C_\mu = \begin{bmatrix} C^{11}_\mu & C^{12}_\mu \\ C^{21}_\mu & C^{22}_\mu \end{bmatrix}, \quad d_\mu = \begin{bmatrix} d^1_\mu \\ d^2_\mu \end{bmatrix}$$

where the submatrices $C^{11}_\mu = \Phi'_\mu(I - P'_\mu)\Xi(I - P_\mu)\Phi_\mu + \Phi'_\mu P_\xi \Phi_\mu$, $C^{12}_\mu = \Phi'_\mu(I - P'_\mu)\Xi(I - \overrightarrow{P}_\mu)\mathbf{1}$, $C^{21}_\mu = \mathbf{1}'(I - \overrightarrow{P}'_\mu)\Xi(I - P_\mu)\Phi_\mu$, $C^{22}_\mu = \mathbf{1}'(I - \overrightarrow{P}'_\mu)\Xi(I - \overrightarrow{P}_\mu)\mathbf{1}$, $d^1_\mu = \Phi'_\mu(I - P'_\mu)\Xi g_\mu$, and $d^2_\mu = \mathbf{1}'(I - \overrightarrow{P}'_\mu)\Xi g_\mu$.

Several simulation sequences are generated according to probabilistic mechanisms: 1) *Indices sampling:* Two sequences $\{i_0, i_1, \ldots\}$ and $\{\hat{i}_0, \hat{i}_1, \ldots\}$ are independently generated according to the probability distribution ξ. 2) *Transitions sampling:* Two sequences $\{(i_0, j_0), (i_1, j_1), \ldots\}$ and $\{(i_0, \hat{j}_0), (i_1, \hat{j}_1), \ldots\}$ are independently generated according to the same Markov transition matrix P_μ. After collecting $(t+1)$ samples, we approximate C_μ and d_μ with

$$\bar{C}_t = \begin{bmatrix} \bar{C}^{11}_t & \bar{C}^{12}_t \\ \bar{C}^{21}_t & \bar{C}^{22}_t \end{bmatrix}, \quad \bar{d}_t = \begin{bmatrix} \bar{d}^1_t \\ \bar{d}^2_t \end{bmatrix} \tag{17}$$

where $\bar{C}^{11}_t = C^{11}_t/(t+1)$, $\bar{C}^{12}_t = C^{12}_t/(t+1)$, $\bar{C}^{21}_t = C^{21}_t/(t+1)$, $\bar{C}^{22}_t = C^{22}_t/(t+1)$, $\bar{d}^1_t = d^1_t/(t+1)$, and $\bar{d}^2_t = d^2_t/(t+1)$ with submatrices defined as $C^{11}_t = \sum_{k=0}^t (\phi(i_k) - \phi(j_k))(\phi(i_k) - \phi(\hat{j}_k))' + n\phi(i_k)\phi(\hat{i}_k)'$, $C^{12}_t = \sum_{k=0}^t (\phi(i_k) - \phi(j_k))(1 - \overrightarrow{P}_\mu(i_k, \hat{j}_k)/P_\mu(i_k, \hat{j}_k))$, $C^{21}_t = \sum_{k=0}^t (1 - \overrightarrow{P}_\mu(i_k, j_k)/P_\mu(i_k, j_k))(\phi(i_k) - \phi(\hat{j}_k))'$, $C^{22}_t = \sum_{k=0}^t (1 - \overrightarrow{P}_\mu(i_k, j_k)/P_\mu(i_k, j_k))(1 - \overrightarrow{P}_\mu(i_k, \hat{j}_k)/P_\mu(i_k, \hat{j}_k))$, $d^1_t = \sum_{k=0}^t (\phi(i_k) - \phi(j_k))g(i_k)$, and $d^2_t = \sum_{k=0}^t (1 - \overrightarrow{P}_\mu(i_k, j_k)/P_\mu(i_k, j_k))g(i_k)$.

Note that $\phi(i)$ is an $m\times 1$ vector that denotes the row of Φ_μ corresponding to state i, i.e., $\phi(i)'$ is the ith row of Φ_μ. Thus, at each time step t, above calculations only involve low-dimensional matrix multiplications of dimension m, where m can be chosen much less than n. Following the standard law of large numbers arguments for Markov chains, with probability 1, we have

$$\lim_{t\to\infty} \bar{C}_t = C_\mu, \quad \lim_{t\to\infty} \bar{d}_t = d_\mu. \tag{18}$$

D. Iterative Least-Squares Update

In the last section, we constructed simulation-based estimates $\bar{C}_t$ and $\bar{d}_t$ such that $\bar{C}_t \to C_\mu$ and $\bar{d}_t \to d_\mu$ with probability 1 as $t \to \infty$. The most straightforward way to approximate $\hat{r}^*_\mu$ is based on direct matrix inversion:

$$\bar{r}_t = \bar{C}^{-1}_t \bar{d}_t. \tag{19}$$

However, the iteration $\bar{r}_t$ is highly sensitive to the *simulation noise errors* $(\bar{C}_t - C_\mu)$ and $(\bar{d}_t - d_\mu)$ [26], [36]. The high sensitivity to simulation noise errors could cause serious computational difficulties in obtaining an accurate estimate to $\hat{r}^*_\mu$.

In the following, we describe a simulation-based recursive algorithm, which can take advantage of *a priori* knowledge of $\hat{r}^*_\mu$ and reduce the sensitivity to simulation noise errors. Instead of using (19), we estimate $\hat{r}^*_\mu$ iteratively by solving a *regularized least-squares* problem:

$$\hat{r}_{t+1} = \arg\min_{\hat{r}} \left\{ (\bar{C}_t \hat{r} - d_t)' \Sigma^{-1} (\bar{C}_t \hat{r} - d_t) + \beta \|\hat{r} - \hat{r}_t\|^2 \right\} \tag{20}$$

where $\hat{r}_t$ is the step-t simulation-based approximation to $\hat{r}^*_\mu$, $\Sigma \succ 0$ is a positive-definite matrix, and $\beta > 0$ is a positive scalar. In the objective of (20), the first term $(\bar{C}_t\hat{r} - \bar{d}_t)'\Sigma^{-1}(\bar{C}_t\hat{r} - \bar{d}_t)$ quantifies the approximation quality of $\hat{r}$. The second term $\beta\|\hat{r} - \hat{r}_t\|^2$ has the effect of biasing the estimate $\hat{r}_{t+1}$ toward the *a priori* estimate $\hat{r}_t$. The user-specific matrix Σ can be chosen to reduce the effect of the near-singularity of $\bar{C}_t$, while the scalar β can be chosen to regularize the bias to *a priori* estimate [26].

By setting the gradient of the objective in (20) to zero, we obtain the solution to (20) in a closed iterative update form

$$\hat{r}_{t+1} = \hat{r}_t - \bar{G}_t(\bar{C}_t\hat{r}_t - \bar{d}_t) \tag{21}$$

where $\bar{G}_t$ is defined as

$$\bar{G}_t := (\bar{C}_t'\Sigma^{-1}\bar{C}_t + \beta I)^{-1}\bar{C}_t'\Sigma^{-1}. \tag{22}$$

Given the condition that scalar $\beta > 0$ and matrix $\Sigma \succ 0$, the matrix $\bar{G}_t$ is well defined with nonsingular matrix inversions. Denote the deterministic counterpart of $\bar{G}_t$ as

$$G_\mu := (C_\mu'\Sigma^{-1}C_\mu + \beta I)^{-1}C_\mu'\Sigma^{-1}. \tag{23}$$

Using (18), with probability 1, we have

$$\lim_{t\to\infty} \bar{G}_t = G_\mu. \tag{24}$$

E. Convergence Analysis

To perform the convergence analysis for the iteration (21), we first consider its deterministic counterpart

$$\hat{r}_{t+1} = \hat{r}_t - G_\mu(C_\mu\hat{r}_t - d_\mu). \tag{25}$$

Clearly, the iteration (25) is convergent if the mapping $(I - G_\mu C_\mu)$ is a contraction with respect to a certain matrix norm (or equivalently all its eigenvalues are strictly within the unit circle). In the following lemma, we show that the mapping $(I - G_\mu C_\mu)$ is indeed a contraction. The proof is given in Appendix B.

Lemma 2: Let C_μ be defined in (15) and G_μ be defined in (23). For any scalar $\beta > 0$ and any matrix $\Sigma \succ 0$, the mapping $(I - G_\mu C_\mu)$ is a contraction with the operator norm $\|I - G_\mu C_\mu\| = \beta/(\beta + \lambda_n) < 1$, where $\lambda_n > 0$ denotes the smallest positive eigenvalue of the matrix $C_\mu'\Sigma^{-1}C_\mu$.

We represent the iteration (21) equivalently as a combination of (25) with a stochastic noise term:

$$\hat{r}_{t+1} = \hat{r}_t - G_\mu(C_\mu\hat{r}_t - d_\mu) - (\bar{\Upsilon}_t\hat{r}_t - \bar{\upsilon}_t) \tag{26}$$

where $\bar{\Upsilon}_t$ and $\bar{\upsilon}_t$ are the stochastic noises defined as $\bar{\Upsilon}_t = \bar{G}_t\bar{C}_t - G_\mu C_\mu$ and $\bar{\upsilon}_t = \bar{G}_t\bar{d}_t - G_\mu d_\mu$. According to (18) and (24), with probability 1, we have $\lim_{t\to\infty}\bar{\Upsilon}_t = 0$ and $\lim_{t\to\infty}\bar{\upsilon}_t = 0$. The following proposition presents the convergence result for the iteration (21). The proof appears in Appendix C.

Proposition 1: Let $\bar{C}_t$, $\bar{d}_t$, and $\bar{G}_t$ be given in (17) and (22), respectively. For any scalar $\beta > 0$ and any matrix $\Sigma \succ 0$, the vector $\hat{r}_t$ is updated using (21). Let $\hat{r}^*_\mu$ be defined in (16). Then, with probability 1, we have $\hat{r}_t \to \hat{r}^*_\mu$ as $t \to \infty$.

We also study the rate of convergence for the iteration (21). We will show that the rate of convergence has two time scales: the fast time scale at which $\hat{r}_t$ tracks $\bar{r}_t = \bar{C}_t^{-1}\bar{d}_t$, and the slow time scale at which $\bar{C}_t$, $\bar{d}_t$, and $\bar{G}_t$ converge to C_μ, d_μ, and G_μ, respectively. The two-time-scale convergence rate has been vastly observed in the literature [26], [27]. The difference between $\hat{r}_{t+1}$ and $\bar{r}_{t+1}$ can be represented as

$$\hat{r}_{t+1} - \bar{r}_{t+1} = (I - \bar{G}_t\bar{C}_t)(\hat{r}_t - \bar{r}_t) + (I - \bar{G}_t\bar{C}_t)(\bar{r}_t - \bar{r}_{t+1}). \tag{27}$$

In what follows, we present the convergence rate analysis in two steps. We first show that the distance between $\hat{r}_t$ and $\bar{r}_t$ shrinks at the rate of $O(1/t)$. The proof is given in Appendix D.

Proposition 2: The sequence of random variables $t(\hat{r}_t - \bar{r}_t)$ is bounded with probability 1.

The results of [27] and [28] show that $\bar{r}_t$ converges to the limit $\hat{r}^*_\mu$ at the rate of $O(1/\sqrt{t})$. It then follows from Proposition 2 that $\hat{r}_t$ also converges to the limit $\hat{r}^*_\mu$ at the rate of $O(1/\sqrt{t})$. Moreover, the error $(\hat{r}_t - \hat{r}^*_\mu)$ is normally distributed as a consequence of the central limit theorem. The proof is given in Appendix E.

Proposition 3: For any given initial condition $\hat{r}_0$, the sequence of random variables $\sqrt{t}(\hat{r}_t - \hat{r}^*_\mu)$ converges in distribution to a Gaussian random vector $\mathcal{N}(0, \Sigma_c)$ as $t \to \infty$, where Σ_c is some finite covariance matrix.

F. Simulation-Based Approximate Dynamic Programming Algorithm

So far, all basis functions are assumed to be generated from the set $\mathcal{B}_\mu$ constructed from high-dimensional matrix powers and multiplications $P^m_\mu p_\mu$ (or $P^m_\mu g_\mu$). In fact, basis functions can also be generated through simulation samples of associated Markov chains [37]. We simply denote $\widehat{\Phi}_{\mu,t}$ and $\widehat{\mathcal{B}}_{\mu,t}$ as simulation-based counterpart of the candidate set (12) and basis matrix (13), respectively. All analyses and proofs presented in previous sections are all carried through due to the ergodicity theory of Markov chain [37]. We omit the details here for the sake of space. In Algorithm 1, we present an S-APIA to obtain suboptimal solutions to Problem 3.

Remark 2: The S-APIA is a simulation-based ADP algorithm generalized from its deterministic counterpart APIA proposed in our previous work [23]. Compared to the PIA (traditional DP algorithm, optimal method) and the APIA (deterministic ADP algorithm, suboptimal method), the S-APIA is a suboptimal method with much lower computational complexity due to the involvement of simulation samples and low-dimensional matrix calculations. For each iteration of the S-APIA, the computational complexity is of order $O(m^3Tn)$, where m is the number of independent basis functions, n is the number of MDP states, and T is the number of simulation samples. Recall that the complexity of the APIA is of order $O(mn^2)$ and the PIA is of order $O(n^3)$. For large-scale MDPs, we have $m \ll n$ and $T \ll n$, which implies that the complexity of the simulation-based suboptimal method S-APIA is *linear*, the deterministic suboptimal method APIA is *quadratic*, and the optimal method PIA is *cubic*, with respect to n. Thus, the S-APIA is the one with the lowest computational complexity, but it only guarantees to generate suboptimal policies and it may require more iterations than the PIA and the APIA before the termination of the algorithm. The approximation accuracy decides the suboptimality

Algorithm 1: Simulation-Based Approximate Policy Iteration Algorithm.

```
Input: MDP $\mathcal{M} = (Q, \iota, U, P, \Pi, L, g)$, integer $\bar{m}$, number
    of simulation samples $T$, scalar parameter $\beta$, and
    positive matrix $\Sigma$
Output: Suboptimal policy $\hat{\mu}^*_{\bar{m},T}$ and average-relative cost
    pair $(\hat{\lambda}^*_{\bar{m},T}, \hat{h}^*_{\bar{m},T})$
 1: Initialize a proper policy $\mu^0$ and a vector $\hat{r}_0$, and set
    $k = 0$
 2: repeat
 3:     Construct transition matrix $P_{\mu^k}$, cost per stage vector
        $g_{\mu^k}$, and auxiliary matrix $\overrightarrow{P}_{\mu^k}$ using (4)–(6),
        respectively
 4:     Generate an index sequence $\{\tilde{i}_t\}_{t=0}^T$ sampled
        according to $\tilde{i}_{t+1} \sim P_{\mu^k}(\tilde{i}_t, \cdot)$
 5:     Construct a candidate set of basis functions $\widehat{\mathcal{B}}_{\mu^k,T}$,
        obtain $m$ independent basis functions from $\mathcal{B}_{\mu^k}$, and
        construct a basis matrix $\widehat{\Phi}_{\mu^k,T}$
 6:     Independently generate two sequences $\{i_t\}_{t=0}^T$ and
        $\{\hat{i}_t\}_{t=0}^T$ according to the probability distribution $\xi$
 7:     Independently generate two sequences $\{(i_t, j_t)\}_{t=0}^T$
        and $\{(i_t, \hat{j}_t)\}_{t=0}^T$ according to transition matrix $P_\mu$
 8:     Iteratively compute matrices $\{\bar{C}_t\}_{t=0}^T$ and $\{\bar{d}_t\}_{t=0}^T$
        using (17) with basis matrix $\widehat{\Phi}_{\mu^k,T}$, and sample
        sequences $\{i_t\}_{t=0}^T$, $\{\hat{i}_t\}_{t=0}^T$, $\{(i_t, j_t)\}_{t=0}^T$, and
        $\{(i_t, \hat{j}_t)\}_{t=0}^T$
 9:     Iteratively compute vectors $\{\hat{r}_t\}_{t=0}^T$ through the
        iteration (21)
10:     Obtain parameter vector $r^*_{\mu^k} = [\hat{r}_T(1), \ldots, \hat{r}_T(m)]'$
        and average cost $\lambda^*_{\mu^k} = \hat{r}_T(m+1)$
11:     Compute relative cost $h^*_{\mu^k} = (\widehat{\Phi}_{\mu^k,T}) r^*_{\mu^k}$
12:     Find updated policy $\mu^{k+1} \in \arg\min_\mu [g_\mu +$
        $P_\mu h^*_{\mu^k} + \lambda^*_{\mu^k} \overrightarrow{P}_\mu \mathbf{1}]$
13:     Set $k \leftarrow k + 1$
14: until $\mu^{k+1} = \mu^k$
15: return $\hat{\mu}^*_{\bar{m},T} := \mu^k$, $\hat{\lambda}^*_{\bar{m},T} := \lambda^*_{\mu^k}$, and $\hat{h}^*_{\bar{m},T} := h^*_{\mu^k}$
```

of the S-APIA solution. For each iteration of the S-APIA, if $\hat{\mu}^*_{\bar{m},T} = h^\star_{\mu^k}$ and $\hat{\lambda}^*_{\bar{m},T} = \lambda^\star_{\mu^k}$, $\forall \mu^k$ $(k \geq 0)$, then the suboptimal policy $\hat{h}^*_{\bar{m},T}$ returned by the S-APIA is in fact an optimal policy to Problem 3. More rigorous results on optimality conditions and known issues of simulation-based ADP algorithms can be found in [21] and references therein.

G. Related Work

The ADP algorithms proposed in this paper are related to the research work in the field of control and machine learning [21], [33], [34], [38]. The overall ADP framework adopted here follows the state-of-the-art API algorithms that employ the linear parameterization and simulation samples to approximate the value function [29], e.g., the actor critic algorithm [27], the LSTD algorithm [36], [39], and LSPE algorithm [26], [40], [41]. But the ways that we use to construct the basis functions and develop the simulated-based algorithm are different. In traditional API algorithms, the transition probabilities of states are explicitly known *a priori* or can be directly learned from sampled data [21], [34]. However, for our problem, the transition probabilities of the transformed MDP need to be calculated from the high-dimensional matrix inversion of the original MDP. Thus, the benefits of the traditional API algorithm as well as its known varieties cannot be fully appreciated for our problem. Motivated by the MDP transformation used in the time aggregation method [42], the ADP framework proposed in this paper transforms the Bellman equation into an equivalent form. In such a way, we eliminate the need to calculate the matrix inversions for the transformed MDP.

The computational complexity of the simulation-based algorithm is of order $O(n)$, where n is the number of states. This algorithm may become intractable for very large n. This is a fundamental difficulty for many other related algorithms aiming to reduce the complexity for solving MDPs [21], [34]. Compared to other traditional approaches, the proposed algorithm can be easily extended to a more computationally efficient version using parallel computation, since the associated simulations are easily parallelizable (see Section III-C). Besides the parallel computation, there also exist other effective methods to further reduce the complexity of the proposed algorithm [21], [34], [43], such as the decentralized method (see Appendix F), the learning-based method, and the asynchronous update method. This will be a subject of our future work.

IV. Case Study

The algorithmic framework developed in this paper was implemented in MATLAB and used in conjunction with a simulator to demonstrate the motion of a robotic team in the stochastic environment shown in Fig. 2. The performance and complexity of two algorithms (PIA and S-APIA) are compared for computing control policies of a case study. A computer with 2.00-GHz CPU and with 3-GB RAM was used to generate the simulation results.

A. Setup of Case Study

We consider two robots and assume that Robot 1 moves twice as fast as Robot 2. We consider a persistent surveillance task, where they are required to monitor rooms V1 and V3 and then return to Base to report the collected information. In other words, the robots should occupy rooms V1 and V3 at the same time and then return to Base together, infinitely often. During their motion in the environment, Robots 1 and 2 should always avoid regions Unsafe1 and Unsafe2, respectively. To specify this task, we define a set of atomic propositions in the form $\Pi = \{\mathsf{Base1}, \mathsf{Base2}, \mathsf{M\_V1}, \mathsf{M\_V3}, \mathsf{Unsafe1}, \mathsf{Unsafe2}\}$ and assign the atomic propositions to the robots as follows: $\Pi_1 = \{\mathsf{Base1}, \mathsf{M\_V1}, \mathsf{M\_V3}, \mathsf{Unsafe1}\}$ and $\Pi_2 = \{\mathsf{Base2}, \mathsf{M\_V1}, \mathsf{M\_V3}, \mathsf{Unsafe2}\}$. The labeling functions for Robots 1 and 2 are defined as follows: $L_1(\mathsf{V1}) = L_2(\mathsf{V1}) = \{\mathsf{M\_V1}\}$, $L_1(\mathsf{V3}) = L_2(\mathsf{V3}) = \{\mathsf{M\_V3}\}$, $L_1(\mathsf{V6}) = \{\mathsf{Unsafe1}\}$, $L_2(\mathsf{V5}) = \{\mathsf{Unsafe2}\}$, $L_1(\mathsf{V7}) = \{\mathsf{Base1}\}$, and $L_2(\mathsf{V7}) = \{\mathsf{Base2}\}$.

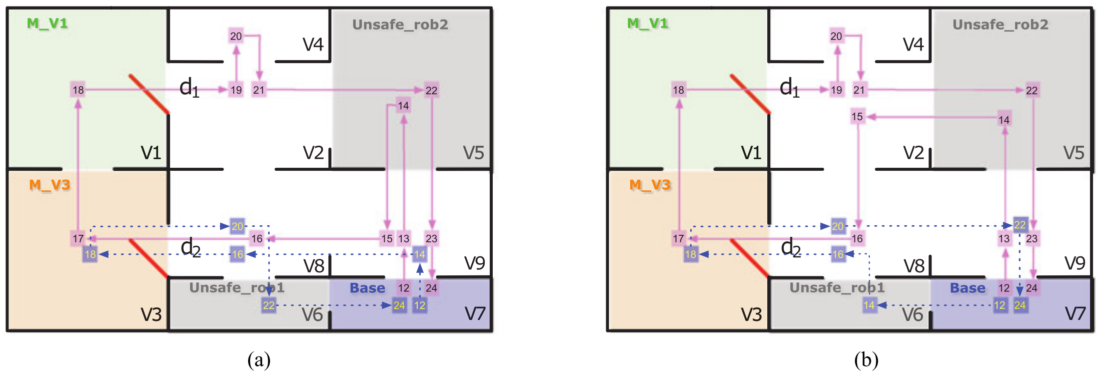


Fig. 2. Typical simulated paths of two robots that employ the PIA optimal policy $\mu^\star$ in (a) and the S-APIA suboptimal policy $\mu^*_{\bar{m},T}$ in (b) for the case study from time step 12 to 24, where the pink solid line is the simulated path for Robot 1 and the blue-dashed line for Robot 2. The numbers show the time instances when robots are at the corresponding locations. At time step 12, both robots are at room V7. After two time steps, robot 1 moves to V5, while Robot 2 moves to V9 (or V6) under optimal policy (or suboptimal policy). Note that robot 2 is slower than robot 1. At time step 18, the constraint M_V1 $\wedge$ M_V3 is satisfied by the team under both polices; therefore, none of the robots can visit V1 or V3 before they both reach the base simultaneously.

Our goal is to minimize the expected time in between the robots' simultaneous visits to V1 and V3. Therefore, the optimizing task ψ is M_V1 $\wedge$ M_V3. The specification is given as follows: $\phi = \mathsf{G}(\neg\text{Unsafe1}) \wedge \mathsf{G}(\neg\text{Unsafe2}) \wedge \mathsf{GF}(\text{Base1} \wedge \text{Base2}) \wedge \mathsf{G}((\text{M\_V1} \wedge \text{M\_V3}) \rightarrow \mathsf{X}\,((\neg\text{M\_V1} \wedge \neg\text{M\_V3})\,\mathsf{U}\;(\text{Base1} \wedge \text{Base2}))) \wedge \mathsf{GF}(\text{M\_V1} \wedge \text{M\_V3})$. GF (Base1 $\wedge$ Base2) ensures that both robots visit the base simultaneously. G($\neg$Unsafe1) and G($\neg$Unsafe2) specify that the robots do not visit the unsafe regions. G((M_V1 $\wedge$ M_V3) $\rightarrow$ X (($\neg$M_V1 $\wedge$ $\neg$M_V3) U (Base1 $\wedge$ Base2))) ensures that after monitoring both V1 and V3, none of them are visited again before the robots visit the base together.

B. Algorithms for Computing Optimal Policies

To compute optimal policies $\{\pi_k, k \in \mathcal{R}\}$ satisfying ϕ with the maximum probability, we first construct an MDP $\mathcal{M}_\mathcal{G}$ using (2) and translate the LTL formula ϕ to a DRA $\mathcal{A}_\phi$ using the software tool $ltl2dstar$ [44]. Then, we compute a product MDP $\mathcal{M}_\mathcal{P} = \mathcal{M}_\mathcal{G} \times \mathcal{A}_\phi$ such that the constructed $\mathcal{M}_\mathcal{G}$ and $\mathcal{M}_\mathcal{P}$ have 2575 and 33 475 states, respectively. A procedure described in [1] is employed to obtain all AMECs for the product MDP $\mathcal{M}_\mathcal{P}$. We found only one AMEC (i.e., a communicating MDP denoted by $\mathcal{M}$) with $n = 4102$ states in the product MDP $\mathcal{M}_\mathcal{P}$. In the following, we compare the performance and complexity of two algorithms (PIA and S-APIA) to find (sub)optimal robot policies given MDP $\mathcal{M}$ and optimizing task ψ.

1) PIA: The PIA proposed in [19] is taken as a baseline (optimal) algorithm for comparisons purpose. By employing the PIA for solving the MDP $\mathcal{M}$, we obtain the optimal control policy $\mu^\star$ and the average-relative cost pair $(\lambda^\star, h^\star)$. For the case study considered here, the optimal average cost $\lambda^\star = 12.9794$ and computation time for running the PIA is around 519.45 s. The optimal policy $\mu^\star$ is deployed for two robots in the simulation environment. In Fig. 2(a), we depict typical simulated paths for two robots that employ the optimal policy $\mu^\star$ from time step 12 to 24.

2) S-APIA: To reduce the computational complexity of the PIA, in this paper, we developed the S-APIA to compute the suboptimal control polices (Algorithms 1; see Section III-F for details). The S-APIA constructs a simulation-based iterative update $\hat{r}_t$ to approximate the least-squares solution $\hat{r}^*_\mu$, where t denotes the index of simulation samples (see (16) and (21) for more details).

To study the behavior of the simulation-based approximations, we consider an arbitrarily chosen control policy $\bar{\mu}$ for one iteration of the S-APIA. We first compute $\hat{r}^*_{\bar{\mu}}$ using the matrix inversion formula (16) with directly computed matrices $C_{\bar{\mu}}$ and $d_{\bar{\mu}}$. Then, we compute $\hat{r}_t$ using the iteration (21) with matrices $\bar{C}_t$ and $\bar{d}_t$ generated from simulation samples. In Fig. 3, we depict semilog plots of approximation errors $\|\bar{C}_t - C_{\bar{\mu}}\|$ and $\|\bar{d}_t - d_{\bar{\mu}}\|$ for $t = 0, 1, \ldots, 10000$. We observe from Fig. 3 that $\bar{C}_t$ and $\bar{d}_t$ converge to $C_{\bar{\mu}}$ and $d_{\bar{\mu}}$, respectively, as $t \rightarrow \infty$. This demonstrates the theoretical results shown in (18). In Fig. 3(c), we also depict a semilog plot of approximation error $\|\hat{r}_t - \hat{r}^*_{\bar{\mu}}\|$ and the curve $\sigma_0/\sqrt{t}$ with respect to t, where σ_0 is a scalar denoting the initial condition. We observe that $\hat{r}_t$ converges to the limit $\hat{r}^*_{\bar{\mu}}$ at the rate of $O(1/\sqrt{t})$. This demonstrates the theoretical results shown in Propositions 1 and 3.

Let $\bar{m}$ denote the number of basis functions and T denote the number of simulation samples for the S-APIA. By selecting different values of $\bar{m}$ and T, we can trade off the optimality and complexity of the S-APIA. By employing Algorithm 1 for solving the MDP $\mathcal{M}$, we obtain the suboptimal policy $\hat{\mu}^*_{\bar{m},T}$ and the average-relative cost pair $(\hat{\lambda}^*_{\bar{m},T}, \hat{h}^*_{\bar{m},T})$. Similarly, to quantify the optimality of the S-APIA, we compute the cost errors $||\hat{\lambda}^*_{\bar{m},T} - \lambda^\star||$ and $||\hat{h}^*_{\bar{m},T} - h^\star||$ for $\bar{m} = 1, 2, \ldots, 30$ and $T = 500, 1000, \ldots, 8000$ (see Fig. 4(a) and (b) for illustrations). In Fig. 4(c), we depict the computation time for running the S-APIA with respect to different $\bar{m}$ and T. We observe from Fig. 4 that the cost error decreases, while the computation time increases as the number of basis functions $\bar{m}$ or the number of simulation samples T increases. We also observe from Fig. 4(a) and (b) that the optimality of S-APIA mainly depends on the

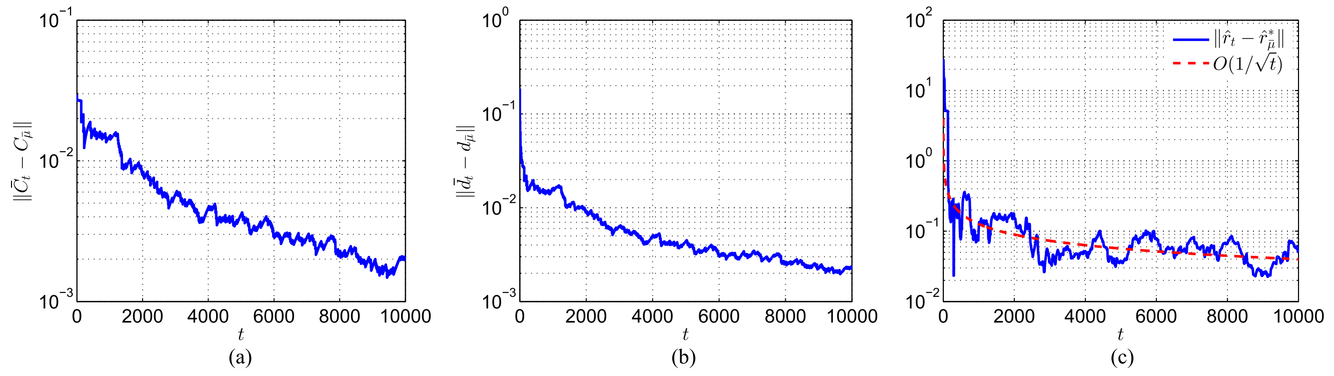


Fig. 3. Semilog plots of approximation errors (a) $\|\bar{C}_t - C_{\bar{\mu}}\|$, (b)$\|\bar{d}_t - d_{\bar{\mu}}\|$, and (c) $\|\hat{r}_t - \hat{r}^*_{\bar{\mu}}\|$ with respect to the simulation sample index t.

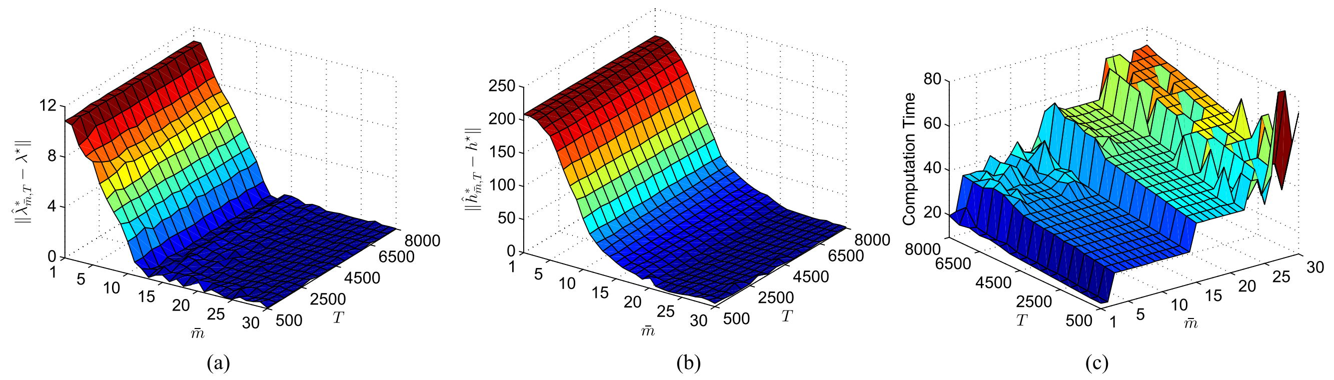


Fig. 4. (a) Average cost error $\|\hat{\lambda}^*_{\bar{m},T} - \lambda^\star\|$, (b) relative cost error $\|\hat{h}^*_{\bar{m},T} - h^\star\|$, and (c) computation time (seconds) for running the S-APIA with respect to number of basis functions $\bar{m}$ and number of simulation samples T, where $(\hat{\lambda}^*_{\bar{m},T}, \hat{h}^*_{\bar{m},T})$ is obtained from the S-APIA and $(\lambda^\star, h^\star)$ is obtained from the PIA.

number of basis functions $\bar{m}$ but not very sensitive to the number of simulation samples, i.e., the cost error largely decreases as $\bar{m}$ increases but only slightly decrease as T increases. For the case study considered here, even with a small T (e.g., $T \approx 1000 - 2000$), the S-APIA can achieve comparably good performance with a large T (e.g., $T > 5000$).

Using Fig. 4, one can easily make a tradeoff between optimality and complexity of the S-APIA by properly choosing $\bar{m}$ and T. For example, if we choose $\bar{m} = 21$ and $T = 1500$ for running the S-APIA (simulation-based algorithm) to obtain a suboptimal policy $\mu^*_{\bar{m},T}$, the average cost error $\|\hat{\lambda}^*_{\bar{m},T} - \lambda^\star\| = 0.0195$, the relative cost error $\|\hat{h}^*_{\bar{m},T} - h^\star\| = 1.1908$, and the computation time is around 38.59 s [see Fig. 4(a)–(c)]. Recall that, by running the PIA (optimal algorithm) to obtain the optimal policy $\mu^\star$, the average cost $\lambda^\star = 12.9794$ and the computation time is around 519.45 s. Thus, one can run the S-APIA with $\bar{m} = 21$ and $T = 1500$ to obtain a fairly good suboptimal policy $\mu^*_{\bar{m},T}$ to approximate $\mu^\star$ with a largely reduced computational complexity (i.e., the S-APIA achieves less than 0.5% approximation error on average cost and more than 90% reduction on computation time compared to the PIA).

In Fig. 2(b), we also depict typical simulated paths for two robots that employ the suboptimal policy $\mu^*_{\bar{m},T}$ from time step 12 to 24, where the simulation is set up the same as the optimal policy $\mu^\star$ for comparison purpose [see Fig. 2(a)]. We observe from Fig. 2 that the simulated paths generated by employing $\mu^*_{\bar{m},T}$ and $\mu^\star$ are similar with differences coming from alternative control/motion actions taken by the suboptimal policy $\mu^*_{\bar{m},T}$. For short time steps considered here, the average costs corresponding to the simulated paths deployed using both polices are the same, and all temporal logic constraints are satisfied by both policies. For long run of the simulation, the average cost associated with the optimal policy $\mu^\star$ is slightly better/smaller than the one with the suboptimal policy $\mu^*_{\bar{m},T}$, which is consistent with our observation that $\mu^*_{\bar{m},T}$ achieves less than 0.5% approximation error on average cost compared to $\mu^\star$.

V. Conclusion

In this paper, we considered the problem of generating control control policies for a team of robots in a stochastic environment to complete an optimal surveillance mission. We modeled the robots as game transition systems and the environmental elements as labeled Markov Chains. The problem reduced to finding an optimal control policy of an MDP while satisfying a temporal logic specification. We presented an ADP framework to generate (sub)optimal control policies. Compared to the existing approaches based on DP, the proposed ADP framework allows us to trade off between the solution optimality and computational complexity for obtaining the control policies in a suboptimal manner. Two algorithms were developed to implement the proposed ADP framework. The effectiveness and efficiency of the proposed ADP approaches are demonstrated using a case study in simulation environment.

Appendix A
Environment, Door, and Robot Models

Definition 1 (Environment model): The *environment* is modeled as a tuple

$$\mathcal{E} = (\mathcal{V}, \rightarrow_{\mathcal{E}}, \Pi, L_{\mathcal{E}}) \tag{28}$$

where $\mathcal{V}$ is a set of room states, $\rightarrow_{\mathcal{E}} \subseteq \mathcal{V} \times \mathcal{V}$ is an adjacency relation of the rooms, Π is a set of atomic propositions, and $L_{\mathcal{E}} : \mathcal{V} \rightarrow 2^{\Pi}$ is a labeling function with $L_{\mathcal{E}}(v)$ representing the set of atomic propositions that hold true in room v.

An atomic proposition $\alpha \in \Pi$ can be used to represent a service request occurring in the environment (e.g., v needs to be monitored), or a property of a location (e.g., v is unsafe). We denote $\mathcal{D}$ as a set of doors located in the environment $\mathcal{E}$. To capture the door locations, we define a partial function $\mathcal{F}_{\mathcal{D}} : \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{D} \cup \{\Delta\}$, where $\mathcal{F}_{\mathcal{D}}$ is defined for all $(v, v') \in \rightarrow_{\mathcal{E}}$ such that $\mathcal{F}_{\mathcal{D}}(v, v') = i$ means that the adjacent rooms v and v' are separated by door i, whereas $\mathcal{F}_{\mathcal{D}}(v, v') = \{\Delta\}$ means that there exists no door in between v and v'. We assume that the doors behave independently of each other, and each door changes its open or closed status in a stochastic manner modeled as a labeled discrete-time Markov chain (DTMC).

Definition 2 (Door model): Each *door* $i \in \mathcal{D}$ is modeled as a labeled DTMC

$$\mathcal{C}_i = (S_i, \iota_i, P_i, L_i) \tag{29}$$

where S_i is a set of door states, $\iota_i : S_i \rightarrow [0, 1]$ is an initial distribution with $\sum_{s \in S_i} \iota_i(s) = 1$, $P_i : S_i \times S_i \rightarrow [0, 1]$ is a transition probability function such that $\sum_{s' \in S_i} P_i(s, s') = 1$ for any $s \in S_i$, and $L_i^{\mathcal{C}} : S_i \rightarrow \Omega$ is a labeling function with $\Omega = \{o, c\}$ being a status set with $\{o\}$ standing for "open" and $\{c\}$ for "closed."

An example of door model $\mathcal{C}_1$ is shown in Fig. 1(top left). There are three door states $S_1 = \{s_1, s_2, s_3\}$ for door 1, where the states evolve according to a DTMC. The transition probabilities are shown on top of the corresponding arrow and omitted when the probabilities are 1. The incoming arrow indicates the initial state. The status corresponding each state is given by $L_1^{\mathcal{C}}(s_1) = L_1^{\mathcal{C}}(s_2) = c$ and $L_1^{\mathcal{C}}(s_3) = o$.

We consider a team of robots moving in the environment $\mathcal{E}$, whose motions are restricted by their motion primitives (controls) and doors' statuses. We denote $\mathcal{R}$ as a set for robots. For any $k \in \mathcal{R}$ and $v \in \mathcal{V}$, we denote $U_k(v)$ as a finite set of available motion controls at room v for robot k, where a motion function $M : \mathcal{V} \times U_k \rightarrow \mathcal{V}$ is defined such that for $(v, v') \in \rightarrow_{\mathcal{E}}$, $M(v, u) = v'$ represents that robot k can move to room v' by applying the motion control $u \in U_k(v)$ at room v. If there is no door separating two adjacent rooms v and v', robot k in room v can take a control u such that $M(v, u) = v'$ to moves from v to v'. If there is a door separating v and v', then robot k needs to interact with the door for moving. For any door $i \in \mathcal{D}$, if it separates rooms v and v', we introduce a set of *auxiliary door states* associated with door i denoted by $Q_i^d = \{q_i^v, q_i^{v'} \mid i = \mathcal{F}_{\mathcal{D}}(v, v'),\ v, v' \in \mathcal{V}\}$. For example, to move from room v to an adjacent room v', which are separated by door i, the robot k takes two steps: the first step is that the robot k moves from v to q_i^v by taking a control u such that $M(v, u) = q_i^v$; the second step is that the door status decides the next room for the robot k such that it moves from q_i^v to the next room v' if the door i is open or it moves from q_i^v back to the room v if the door i is closed. We denoted $Q_{\mathcal{D}} = \prod_{i \in \mathcal{D}} Q_i^d$ as a set of all auxiliary door states. We model each robot $k \in \mathcal{R}$ as a game transition system, denoted by $\mathcal{T}_k$.

Definition 3 (Robot model): Each robot $k \in \mathcal{R}$ is modeled as a game transition system

$$\mathcal{T}_k = (Q_k, v_k^{in}, U_k, \rightarrow_k, \Pi_k, L_k, g_k) \tag{30}$$

where $Q_k = \mathcal{V} \times Q_{\mathcal{D}}$ is a finite set of robot states; $v_k^{in} \in \mathcal{V}$ is an initial state of the robot, representing its initial position; U_k is a set of controls; $\rightarrow_k \subseteq Q_k \times Q_k$ is a transition relation such that for $v, v' \in \mathcal{V}$ and $u \in U_k(v)$, we have 1) $(v, v') \in \rightarrow_k$ iff $M(v, u) = v'$ and $\mathcal{F}_{\mathcal{D}}(v, v') = \{\Delta\}$, and 2) $(v, q_i^v) \in \rightarrow_k$, $(q_i^v, v) \in \rightarrow_k$, and $(q_i^v, v') \in \rightarrow_k$ iff $M(v, u) = v'$ and $\mathcal{F}_{\mathcal{D}}(v, v') = i$; $\Pi_k \subseteq \Pi$ is a set of atomic propositions; $L_k : \mathcal{V} \rightarrow 2^{\Pi_k}$ is a labeling function over the states of the robot such that $L_k(v) = L_{\mathcal{E}}(v)$; and $g_k : \rightarrow_k \rightarrow \mathbb{N}$ is a weight function that assigns a nonnegative integer that captures the travel time between of each transition.

The states of $\mathcal{T}_k$ is partitioned in two subsets: the set $\mathcal{V}$ of room states, at which robots take controls, and the set $Q_{\mathcal{D}}$ of auxiliary door states, at which robots interact with doors. A *run* of $\mathcal{T}_k$ is an infinite sequence $\{q^n\}_{n \geq 0}$ such that $q^0 = v_l^{in}$ and $q^n \rightarrow_k q^{n+1}$, $\forall n \geq 0$. By removing all door states from the run of $\mathcal{T}_k$, we obtain an infinite path of the robot, denoted by $\mathbb{P}_k = \{v_k^n\}_{n \geq 0}$. A path of the robot $\mathbb{P}_k$ generates an *output word* (behavior) of the robot, denoted by $\mathbb{O}_k = \{L_k(v_k^n)\}_{n \geq 0}$, and an infinite sequence of time instances $\mathbb{T}_k = \{\mathbb{T}_k^n\}_{n \geq 0}$ such that $L_k(v_k^n)$ is satisfied at time instance $\mathbb{T}_k^n$. A history-dependent *control policy* for robot $k \in \mathcal{R}$ is defined as an infinite sequence $\pi_k = \{\mu_k^n\}_{n \geq 0}$, where $\mu_k^n : (Q_k)^n \times \prod_{i \in \mathcal{D}} (S_i)^n \rightarrow U_k$. If $\mu_k^n = \mu_k$ for all $k \geq 0$, then it is a stationary control policy, and we simply denote $\pi_k = \mu_k : Q_k \times \prod_{i \in \mathcal{D}} S_i \rightarrow U_k$ as the stationary policy for robot k.

Appendix B
Proof of Lemma 2

Since the matrix C_μ is invertible, the matrix $C_\mu' \Sigma^{-1} C_\mu$ is positive definite with eigenvalues $\lambda_1 \geq \ldots \geq \lambda_n > 0$. Let singular value decomposition of $C_\mu' \Sigma^{-1} C_\mu$ be $U_\mu \Lambda_\mu U_\mu'$, where $\Lambda_\mu = \operatorname{diag}\{\lambda_1, \ldots, \lambda_n\}$ and U_μ is a unitary matrix (i.e., $U_\mu' U_\mu = U_\mu U_\mu' = I$). Then, we have

$$\begin{aligned} I - G_\mu C_\mu &= I - U_\mu (\Lambda_\mu + \beta I)^{-1} U_\mu' U_\mu \Lambda_\mu U_\mu' \\ &= U_\mu (\Lambda_\mu + \beta I)^{-1} \beta I U_\mu'. \end{aligned}$$

It follows that the eigenvalues of $(I - G_\mu C_\mu)$ are $\beta/(\beta + \lambda_i)$ for $i = 1, \ldots, n$, which all lie in the interval $(0, 1)$. Note that $(I - G_\mu C_\mu)$ is a symmetric matrix. So the operator norm of

$(I - G_\mu C_\mu)$ is corresponding to its largest eigenvalue, i.e., $\|I - G_\mu C_\mu\| = \beta/(\beta + \lambda_n)$. Since both β and λ_n are positive scalars, we have $\|I - G_\mu C_\mu\| < 1$. Thus, we conclude the proof that the mapping $(I - G_\mu C_\mu)$ is a contraction.

Appendix C
Proof of Proposition 1

Using (26), we have the following representation:

$$\hat{r}_{t+1} - \hat{r}^*_\mu = (I - G_\mu C_\mu - \bar{\Upsilon}_t)(\hat{r}_t - \hat{r}^*_\mu) - (\bar{\Upsilon}_t \hat{r}^*_\mu - \bar{\upsilon}_t). \tag{31}$$

As shown in Lemma 2, we know the mapping $(I - G_\mu C_\mu)$ is a contraction with $\|I - G_\mu C_\mu\| < 1$. For any sample trajectory such that $\bar{\Upsilon}_t \to 0$, there exists $\bar{t}$ such that, for all $t \geq \bar{t}$, we have $\|I - G_\mu C_\mu - \bar{\Upsilon}_t\| < (1-\epsilon)$ for some positive ϵ. Thus, from (31), for all $t \geq \bar{t}$, we have

$$\|\hat{r}_{t+1} - \hat{r}^*_\mu\| \leq (1-\epsilon)\|\hat{r}_t - \hat{r}^*_\mu\| + \|\bar{\Upsilon}_t \hat{r}^*_\mu - \bar{\upsilon}_t\|. \tag{32}$$

For all sample trajectories such that $\bar{\Upsilon}_t \to 0$ and $\bar{\upsilon}_t \to 0$, the stochastic noise term $(\bar{\Upsilon}_t \hat{r}^*_\mu - \bar{\upsilon}_t)$ diminishes to 0 (so that $\|\bar{\Upsilon}_t \hat{r}^*_\mu - \bar{\upsilon}_t\| \to 0$). For $\|\hat{r}_t - \hat{r}^*_\mu\| \geq 2\|\bar{\Upsilon}_t \hat{r}^*_\mu - \bar{\upsilon}_t\|/\epsilon$, according to (32), we have $\|\hat{r}_{t+1} - \hat{r}^*_\mu\| \leq (1-\epsilon/2)\|\hat{r}_t - \hat{r}^*_\mu\|$, which implies $\|\hat{r}_{t+1} - \hat{r}^*_\mu\|$ monotonically decreases to $2\|\bar{\Upsilon}_t \hat{r}^*_\mu - \bar{\upsilon}_t\|/\epsilon$. For $\|\hat{r}_t - \hat{r}^*_\mu\| \leq 2\|\bar{\Upsilon}_t \hat{r}^*_\mu - \bar{\upsilon}_t\|/\epsilon$, we have $\|\hat{r}_t - \hat{r}^*_\mu\| \to 0$ since the right-hand side of the inequality diminishes to 0 as $t \to \infty$. Note that the set of all these sample trajectories has probability 1. We conclude the proof that $\hat{r}_t \to \hat{r}^*_\mu$ with probability 1.

Appendix D
Proof of Proposition 2

The proof of Proposition 2 is based on the analysis of the iteration (26). To find the shrinking rate of $\hat{r}_t$ to $\bar{r}_t$, we need to bound the convergence rate of the difference of term $(\bar{r}_t - \bar{r}_{t+1})$. The following lemma shows that the norm of $(\bar{r}_t - \bar{r}_{t+1})$ changes at the rate of $O(1/t)$. The proof appears at the end of this section.

Lemma 3: Consider a convergent sample path such that $\bar{C}_t$, $\bar{d}_t$, and $\bar{G}_t$ converge to C_μ, d_μ, and G_μ, respectively. Then, there exists a constant C such that $\|\bar{r}_{t+1} - \bar{r}_t\| \leq C/t$ for all t sufficiently large.

Proof of Proposition 2: We consider a convergent sample path such that $\bar{C}_t$, $\bar{d}_t$, and $\bar{G}_t$ converge to C_μ, d_μ, and G_μ, respectively. Clearly, as shown in (18) and (24), all such sample paths form a set of probability 1, and both $\hat{r}_t$ and $\bar{r}_t$ converge to $\hat{r}^*_\mu$. Thus, the sequence of random variables $t(\hat{r}_t - \bar{r}_t)$ is bounded for any finite t. Then we only need to consider the situation when t is sufficiently large.

Using Lemma 2, we know that $\|I - G_\mu C_\mu\| < 1$. Then, there exists a scalar $\gamma \in (0,1)$ such that, for all t sufficiently large

$$\|I - \bar{G}_t \bar{C}_t\| \leq \gamma. \tag{33}$$

Using (27), we have $\|\hat{r}_{t+1} - \bar{r}_{t+1}\| \leq \|I - \bar{G}_t \bar{C}_t\|\, \|\hat{r}_t - \bar{r}_t\| + \|I - \bar{G}_t \bar{C}_t\|\ \ \|\bar{r}_t - \bar{r}_{t+1}\|$. Applying Lemma 3 and (33), we obtain, for all t sufficiently large,

$$\|\hat{r}_{t+1} - \bar{r}_{t+1}\| \leq \gamma\|\hat{r}_t - \bar{r}_t\| + \frac{\gamma C}{t}. \tag{34}$$

Denote $\bar{t} = \gamma/(1-\gamma) + 1$ and $\bar{\gamma} = \gamma(\bar{t}+1)/\bar{t}$. It is clear that $\bar{t} < \infty$ since $\gamma < 1$. For all $t \geq \bar{t}$, we have $\gamma(t+1)/t \leq \bar{\gamma} < 1$. Using (34), we have, for all $t \geq \bar{t}$,

$$\begin{aligned}(t+1)\|\hat{r}_{t+1} - \bar{r}_{t+1}\| &\leq \gamma\frac{t+1}{t}t\|\hat{r}_t - \bar{r}_t\| + \gamma\frac{t+1}{t}C \\ &\leq \bar{\gamma}t\|\hat{r}_t - \bar{r}_t\| + \bar{\gamma}C. \end{aligned}\tag{35}$$

By recursively applying (35), we have, for all $t \geq \bar{t}$,

$$\begin{aligned}&(t+1)\|\hat{r}_{t+1} - \bar{r}_{t+1}\| \\ &\quad\leq \bar{\gamma}^{(t-\bar{t}+1)}\bar{t}\|\hat{r}_{\bar{t}} - \bar{r}_{\bar{t}}\| + \bar{\gamma}C + \cdots + \bar{\gamma}^{(t-\bar{t}+1)}C \\ &\quad\leq \bar{t}\|\hat{r}_{\bar{t}} - \bar{r}_{\bar{t}}\| + \frac{\bar{\gamma}C}{1-\bar{\gamma}}.\end{aligned}$$

We conclude the proof by noting that $\bar{t}\|\hat{r}_{\bar{t}} - \bar{r}_{\bar{t}}\| + \bar{\gamma}C/(1-\bar{\gamma})$ is always finite. ■

Proof of Lemma 3: Clearly, along such a convergent sample path, both $\hat{r}_t$ and $\bar{r}_t$ converge to $\hat{r}^*_\mu$. By the definition of $\bar{r}_t$, we have

$$\begin{aligned}\|\bar{r}_{t+1} - \bar{r}_t\| &= \|\bar{C}_t^{-1}\bar{d}_t - \bar{C}_{t-1}^{-1}\bar{d}_{t-1}\| \\ &\leq \|\bar{C}_t^{-1} - \bar{C}_{t-1}^{-1}\|\,\|\bar{d}_t\| + \|\bar{C}_{t-1}^{-1}\|\,\|\bar{d}_t - \bar{d}_{t-1}\|.\end{aligned}\tag{36}$$

Since $\bar{C}_t \to C_\mu$ and $\bar{d}_t \to d_\mu$, then $\|\bar{C}_{t-1}^{-1}\| \to \|C_\mu^{-1}\|$ and $\|\bar{d}_t\| \to \|d_\mu\|$. Thus, for sufficiently large t, there exist some constants B_1 and B_2 such that

$$\|\bar{C}_{t-1}^{-1}\| \leq C_1, \quad \|\bar{d}_t\| \leq C_2. \tag{37}$$

Using recursive representation, we can express $\bar{d}_t$ as

$$\bar{d}_t = \bar{d}_{t-1} - \frac{1}{t+1}\bar{d}_{t-1} + \frac{1}{t+1}\begin{bmatrix}(\phi(i_t) - \phi(j_t))g(i_t) \\ (1 - \overrightarrow{P}_\mu(i_t,j_t)/P_\mu(i_t,j_t))g(i_t)\end{bmatrix}.$$

Then, for sufficiently large t, we have

$$\|\bar{d}_t - \bar{d}_{t-1}\| \leq \frac{C_3}{t} \tag{38}$$

for some constant C_3 (since $\bar{d}_t, \phi(i_t), \phi(j_t)\, g(i_t)$, and $\overrightarrow{P}_\mu(i_t,j_t)$ are all bounded for all t, and $P_\mu(i_t,j_t))g(i_t)$ are bounded and nonzero for all t). Using definition of $\bar{C}_t$ [see (17)], we have

$$\begin{aligned}\|\bar{C}_t^{-1} - \bar{C}_{t-1}^{-1}\| &= \|C_t^{-1} + t(C_t^{-1} - C_{t-1}^{-1})\| \\ &\leq \|C_t^{-1}\| + t\|C_t^{-1} - C_{t-1}^{-1}\|.\end{aligned}\tag{39}$$

Using recursive representation, we can express C_t as

$$C_t = C_{t-1} + u(i_t,j_t)u(i_t,\hat{j}_t)' + v(i_t)v(\hat{i}_t)' \tag{40}$$

where the vector functions $u(\cdot,\cdot)$ and $v(\cdot)$ are defined as

$$u(i,j) = \begin{bmatrix}\phi(i) - \phi(j) \\ 1 - \overrightarrow{P}_\mu(i,j)/P_\mu(i,j)\end{bmatrix}, \quad v(i) = \sqrt{n}\begin{bmatrix}\phi(i) \\ 0\end{bmatrix}.$$

Note that the vectors $u(i_t, j_t)$, $u(i_t, \hat{j}_t)$, $v(i_t)$, and $v(\hat{i}_t)$ are all bounded for all $t \geq 0$.

By applying Sherman–Morisson formula for matrix inversion to (40), we obtain

$$C_t^{-1} = \widehat{C}_{t-1}^{-1} - \frac{\widehat{C}_{t-1}^{-1} v(i_t) v(\hat{i}_t)' \widehat{C}_{t-1}^{-1}}{1 + v(\hat{i}_t)' \widehat{C}_{t-1}^{-1} v(i_t)} \tag{41}$$

where $\widehat{C}_{t-1}$ is defined as

$$\widehat{C}_{t-1} = C_{t-1} + u(i_t, j_t) u(i_t, \hat{j}_t)'. \tag{42}$$

By applying the Sherman–Morisson formula again to (42), we obtain

$$\widehat{C}_{t-1}^{-1} = C_{t-1}^{-1} - \frac{C_{t-1}^{-1} u(i_t, j_t) u(i_t, \hat{j}_t)' C_{t-1}^{-1}}{1 + u(i_t, \hat{j}_t)' C_{t-1}^{-1} u(i_t, j_t)}. \tag{43}$$

Note that $C_{t-1} = t\bar{C}_{t-1}$. Then, with probability 1, we have

$$\lim_{t\to\infty} tC_{t-1}^{-1} = \lim_{t\to\infty} t(t\bar{C}_{t-1})^{-1} = \lim_{t\to\infty} \bar{C}_{t-1}^{-1} = C_\mu^{-1}. \tag{44}$$

Denote $\widehat{C_{t-1}^{-1}} = t\widehat{C}_{t-1}^{-1}$. Using (43) and (44), we have

$$\begin{aligned} \lim_{t\to\infty} \widehat{C_{t-1}^{-1}} &= \lim_{t\to\infty} \left(tC_{t-1}^{-1} - \frac{tC_{t-1}^{-1} u(i_t, j_t) u(i_t, \hat{j}_t)' tC_{t-1}^{-1}}{t + u(i_t, \hat{j}_t)' tC_{t-1}^{-1} u(i_t, j_t)} \right) \\ &= C_\mu^{-1}. \end{aligned} \tag{45}$$

Substituting (43) into (41), and using (44) and (45), we have

$$\begin{aligned} & t\|C_t^{-1} - C_{t-1}^{-1}\| \\ &= t \left\| \frac{C_{t-1}^{-1} u(i_t, j_t) u(i_t, \hat{j}_t)' C_{t-1}^{-1}}{1 + u(i_t, \hat{j}_t)' C_{t-1}^{-1} u(i_t, j_t)} + \frac{\widehat{C}_{t-1}^{-1} v(i_t) v(\hat{i}_t)' \widehat{C}_{t-1}^{-1}}{1 + v(\hat{i}_t)' \widehat{C}_{t-1}^{-1} v(i_t)} \right\| \\ &\leq \left\| \frac{\bar{C}_{t-1}^{-1} u(i_t, j_t) u(i_t, \hat{j}_t)' \bar{C}_{t-1}^{-1}}{t + u(i_t, \hat{j}_t)' \bar{C}_{t-1}^{-1} u(i_t, j_t)} \right\| + \left\| \frac{\widehat{C_{t-1}^{-1}} v(i_t) v(\hat{i}_t)' \widehat{C_{t-1}^{-1}}}{t + v(\hat{i}_t)' \widehat{C_{t-1}^{-1}} v(i_t)} \right\| \\ &\leq \frac{C_4}{t} \end{aligned} \tag{46}$$

for some constants C_4 and t sufficiently large. Substituting (46) into (39), we have, for t sufficiently large,

$$\|\bar{C}_t^{-1} - \bar{C}_{t-1}^{-1}\| \leq \frac{1}{t+1} \|\bar{C}_t^{-1}\| + \frac{C_4}{t} \leq \frac{C_1}{t} + \frac{C_4}{t}. \tag{47}$$

Substituting (37), (38), and (47) into (36), we conclude the proof. ■

Appendix E
Proof of Proposition 3

Consider a convergent sample path such that $\bar{C}_t$, $\bar{d}_t$, and $\bar{G}_t$ converge to C_μ, d_μ, and G_μ, respectively. As shown in (18) and (24), all such sample paths form a set of probability 1, and $\hat{r}_t$ and $\bar{r}_t$ converge to $\hat{r}_\mu^*$. Using (19) and (21), we can write

$$\begin{aligned} & \sqrt{t+1}(\hat{r}_{t+1} - \hat{r}_\mu^*) \\ &\quad = \sqrt{t+1}(I - \bar{G}_t \bar{C}_t)(\hat{r}_t - \bar{r}_{t+1}) + \sqrt{t+1}(\bar{r}_{t+1} - \hat{r}_\mu^*). \end{aligned} \tag{48}$$

Using Lemma 2, we know that $\|I - G_\mu C_\mu\| < 1$. Then, there exists a scalar $\gamma \in (0, 1)$ such that $\|I - \bar{G}_t \bar{C}_t\| \leq \gamma$ for t sufficiently large. Thus, for t sufficiently large, we have

$$\begin{aligned} & \|\sqrt{t+1}(I - \bar{G}_t \bar{C}_t)(\hat{r}_t - \bar{r}_{t+1})\| \\ &\qquad \leq \sqrt{t+1}\|(I - \bar{G}_t \bar{C}_t)\|(\|\hat{r}_t - \bar{r}_t\| + \|\bar{r}_t - \bar{r}_{t+1}\|) \\ &\qquad \leq \gamma\sqrt{t+1}\left(\frac{\bar{C}}{t} + \frac{C}{t}\right) \end{aligned} \tag{49}$$

where we obtain (49) using Lemma 3 and Proposition 2 for some finite constants C and $\bar{C}$. Thus, with probability 1, we have

$$\sqrt{t+1}(I - \bar{G}_t \bar{C}_t)(\hat{r}_t - r_{t+1}) \to 0, \quad \text{as } t \to \infty. \tag{50}$$

A convergence rate analysis of $\bar{r}_t$ is provided by Konda [27]. As shown in [27, Th. 6.3], the sequence of random variables $\sqrt{t+1}(\bar{r}_{t+1} - \hat{r}_\mu^*)$ converges in distribution to $\mathcal{N}(0, \Sigma_c)$ as $t \to \infty$. The covariance matrix Σ_c can be computed as $C_\mu^{-1}\Gamma(C_\mu')^{-1}$, where Γ is the covariance matrix of the Gaussian distribution to which $\sqrt{t}(\bar{C}_t \hat{r}_\mu^* - \bar{d}_t)$ converges in distribution. Combining this with (48) and (50), we conclude the proof.

Appendix F
Decentralized Extension to the S-APIA

The S-APIA (see Algorithm 1) proposed in Section III is a centralized algorithm. The computational complexity of S-APIA is of order $O(n)$, where n is the number of MDP states (see Remark 2). In the following, we extend the S-APIP to a decentralized version such that the computational complexity of the algorithm does not depend on the number of MDP states.

For the multiagent persistent monitoring problem, the state transitions of the constructed MDP only occur between the neighboring states, which is due to the geometry constraints of the environment model and the motion constraints of the robot model [20], [30]. Denote $\mathcal{N}(i)$ the set of neighboring states of the state i and we assume that there exists a finite number $\bar{N}$ such that $|\mathcal{N}(i)| \leq \bar{N}$ for $i = 1, \ldots, n$. For the considered multiagent problem, we have $\bar{N} \ll n$, that is, the maximum number of neighboring states is much less than the number of total MDP states.

The centralized computations involved in the S-APIA mainly come from the model construction and policy update steps in the API approach (see lines 3 and 12 in Algorithm 1). In the model construction step, we construct the full transition matrix and cost vector for the entire state space. In the policy update step, the improved policy is obtained by minimizing the evaluated cost vector for the entire space. Using the neighboring-transition structure of the considered MDP, we propose a distributed way in the following to generate simulation samples and update the policy only using neighboring transition probabilities and cost values.

The main idea of the distributed algorithm is to obtain $\bar{C}_t$ and $\bar{d}_t$ by averaging samples collected using the controls corresponding to the (approximately) improved policy. In the following, we briefly presented the main changes of the distributed version compared to the original S-APIA algorithm. Given the generated simulation transitions, we update the parameter vector $\hat{r}_t$ using the iteration

$$\hat{r}_{t+1} = \hat{r}_t - \gamma \bar{G}_t(\bar{C}_t \hat{r}_t - \bar{d}_t) \tag{51}$$

where $\gamma \in (0,1]$ is a stepsize, $\bar{G}_t$ is defined the same as (22), and $\bar{C}_t$ and $\bar{d}_t$ are defined the same as (17) but with all submatrices defined equivalently in the following iterative form: $\bar{C}_t^{11} = \frac{t}{t+1}\bar{C}_{t-1}^{11} + \frac{1}{t+1}((\phi(i_t) - \phi(j_t))(\phi(i_t) - \phi(\hat{j}_t))' + n\phi(i_t)\phi(\hat{i}_t)')$, $\bar{C}_t^{12} = \frac{t}{t+1}\bar{C}_{t-1}^{12} + \frac{1}{t+1}(\phi(i_t) - \phi(j_t))(1 - \overrightarrow{P}_\mu(i_t, \hat{j}_t)/P_\mu(i_t, \hat{j}_t))$, $\bar{C}_t^{21} = \frac{t}{t+1}\bar{C}_{t-1}^{21} + \frac{1}{t+1}(1 - \overrightarrow{P}_\mu(i_t, j_t)/P_\mu(i_t, j_t))(\phi(i_t) - \phi(\hat{j}_t))'$, $\bar{C}_t^{22} = \frac{t}{t+1}\bar{C}_{t-1}^{22} + \frac{1}{t+1}(1 - \overrightarrow{P}_\mu(i_t, j_t)/P_\mu(i_t, j_t))(1 - \overrightarrow{P}_\mu(i_t, \hat{j}_t)/P_\mu(i_t, \hat{j}_t))$, $\bar{d}_t^1 = \frac{t}{t+1}\bar{d}_{t-1}^1 + \frac{1}{t+1}(\phi(i_t) - \phi(j_t))g(i_t)$, and $\bar{d}_t^2 = \frac{t}{t+1}\bar{d}_{t-1}^2 + \frac{1}{t+1}(1 - \overrightarrow{P}_\mu(i_t, j_t)/P_\mu(i_t, j_t))g(i_t)$.

In the centralized algorithm, each policy is evaluated with a very large number of samples using the full transition matrix and cost vector, and the improved policy is obtained for the entire state space during each iteration of policy improvement. In the distributed algorithm, the simulated transitions are generated using a policy that is updated every few samples using neighboring transition probabilities and cost values, and the policy is only updated for the index state that has been generated. In the extreme case of a single sample between policies, when the next index state i_{t+1} is generated according to distribution ξ, we only need to update the policy associated with the state i_{t+1} as

$$\mu^{t+1}(i_{t+1}) = \arg\min_{u \in U(i_{t+1})} \sum_{j \in \mathcal{N}(i_{t+1})} (g(i_{t+1}, u) + p(i_{t+1}, u, j)\phi(j)' r_{t+1} + \lambda_{t+1} \overrightarrow{p}(i_{t+1}, u, j))$$

where recall that $\mathcal{N}(i_{t+1})$ denotes the set of neighboring states of i_{t+1}, and $\hat{r}_{t+1} = [r'_{t+1}, \lambda_{t+1}]'$ is updated using the iteration (51). Then, we generate the state transitions (i_{t+1}, j_{t+1}) and $(i_{t+1}, \hat{j}_{t+1})$ independently using the updated policy $\mu^{t+1}(i_{t+1})$. The generated simulation transitions will be used to update the parameter vector $\hat{r}_{t+1}$ in the next iteration and so on.

Compared to the centralized algorithm, the distributed one presented above is an iterative algorithm with each iteration only depending on neighboring transition probabilities and cost values. The policy is updated every few samples only to the states that have been processed. In such a way, the computational complexity of the distributed algorithm does not depend on the number of MDP states. Generally, in distributed algorithm, a substantial number of samples may need to be collected with the same policy before switching policies, in order to reduce the variance of $\bar{C}_t$ and $\bar{d}_t$. The convergence results of the distributed algorithm will be studied in our future work.

References

[1] C. Baier, J. P. Katoen, and K. G. Larsen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[2] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 2689–2696.

[3] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Trans. Autom. Control*, vol. 57, no. 11, pp. 2817–2830, Nov. 2012.

[4] M. Antoniotti and B. Mishra, "Discrete event models + temporal logic = supervisory controller: Automatic synthesis of locomotion controllers," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1995, pp. 1441–1446.

[5] J. Klein and C. Baier, "Experiments with deterministic ω-automata for formulas of linear temporal logic," *Theor. Comput. Sci.*, vol. 363, no. 2, pp. 182–195, 2006.

[6] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Trans. Autom. Control*, vol. 53, no. 1, pp. 287–297, Feb. 2008.

[7] J. Tumova, B. Yordanov, C. Belta, I. Cerna, and J. Barnat, "A symbolic approach to controlling piecewise affine systems," in *Proc. IEEE Conf. Decision Control*, 2010, pp. 4230–4235.

[8] M. Kloetzer and C. Belta, "Dealing with non-determinism in symbolic control," in *Hybrid Systems: Computation and Control*. Berlin, Germany: Springer-Verlag, 2008, pp. 287–300.

[9] H. Kress-Gazit, D. C. Conner, H. Choset, A. A. Rizzi, and G. J. Pappas, "Courteous cars," *IEEE Robot. Autom. Mag.*, vol. 15, no. 1, pp. 30–38, Mar. 2008.

[10] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning for dynamical systems," in *Proc. IEEE Conf. Decision Control*, 2009, pp. 5997–6004.

[11] X. Ding, S. L. Smith, C. Belta, and D. Rus, "LTL control in uncertain environments with probabilistic satisfaction guarantees," in *Proc. IFAC World Congr.*, 2011, pp. 3515–3520.

[12] M. Y. Vardi, "Probabilistic linear-time model checking: An overview of the automata-theoretic approach," in *Formal Methods for Real-Time and Probabilistic Systems (ser. Lecture Notes in Computer Science)*, vol. 1601. Berlin, Germany: Springer, 1999, pp. 265–276.

[13] C. Baier, M. Grober, M. Leucker, B. Bollig, and F. Ciesinski, "Controller synthesis for probabilistic systems," in *Exploring New Frontiers of Theoretical Informatics*. Berlin, Germany: Springer, 2004, pp. 493–506.

[14] L. D. Alfaro, "Formal verification of probabilistic systems," Ph.D. dissertation, Stanford Univ., Stanford, CA, USA, 1997.

[15] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *Int. J. Robot. Res.*, vol. 30, no. 14, pp. 1695–1708, 2011.

[16] M. Svorenova, J. Tumova, J. Barnat, and I. Cerna, "Attraction-based receding horizon path planning with temporal logic constraints," in *Proc. IEEE Conf. Decision Control*, Maui, HI, USA, 2012, pp. 6749–6754.

[17] M. Svorenova, I. Cerna, and C. Belta, "Optimal receding horizon control for finite deterministic systems with temporal logic constraints," in *Proc. Amer. Control Conf.*, 2013, pp. 4399–4404.

[18] X. Ding, M. Lazar, and C. Belta, "LTL receding horizon control for finite deterministic systems," *Automatica*, vol. 50, no. 2, pp. 399–408, 2014.

[19] X. Ding, S. L. Smith, C. Belta, and D. Rus, "MDP optimal control under temporal logic constraints," in *Proc. IEEE Conf. Decision Control*, 2011, pp. 532–538.

[20] X. Ding, S. L. Smith, C. Belta, and D. Rus, "Optimal control of Markov decision processes with linear temporal logic constraints," *IEEE Trans. Autom. Control*, vol. 59, no. 5, pp. 1244–1257, May 2014.

[21] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, vol. II, 4th ed. Belmont, MA, USA: Athena Scientific, 2012.

[22] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2005.

[23] Y. Chen, K. Deng, and C. Belta, "Multi-agent persistent monitoring in stochastic environments with temporal logic constraints," in *Proc. IEEE Conf. Decision Control*, 2012, pp. 2801–2806.

[24] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: SIAM, 2003.

[25] G. J. Gordon, "Stable function approximation in dynamic programming," in *Proc. Int. Conf. Mach. Learn.*, 1995, pp. 261–268.

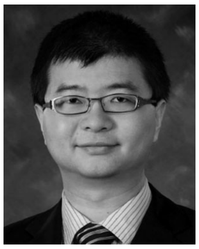
[26] A. Nedic and D. P. Bertsekas, "Least squares policy evaluation algorithms with linear function approximation," *Discrete Event Dyn. Syst.*, vol. 13, nos. 1/2, pp. 79–110, 2003.

[27] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," *SIAM J. Control Optim.*, vol. 42, no. 4, pp. 1143–1166, 2003.

[28] H. Yu and D. P. Bertsekas, "Convergence results for some temporal difference methods based on least squares," *IEEE Trans. Autom. Control*, vol. 54, no. 7, pp. 1515–1531, Jul. 2009.

[29] D. P. Bertsekas, "Approximate policy iteration: A survey and some new methods," *J. Control Theory Appl.*, vol. 9, no. 3, pp. 310–335, 2011.

[30] Y. Chen, J. Tumova, and C. Belta, “Temporal logic robot control based on automata learning of environmental dynamics,” *Int. J. Robot. Res.*, vol. 32, no. 5, pp. 547–565, 2013.
[31] C. Courcoubetis and M. Yannakakis, “The complexity of probabilistic verification,” *J. ACM*, vol. 42, no. 4 pp. 857–907, 1995.
[32] K. Chatterjee and M. Henzinger, “Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification,” in *Proc. Annu. ACM-SIAM Symp. Discrete Algorithms*, 2011, pp. 1318–1336.
[33] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA, USA: Athena Scientific, 1996.
[34] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Hoboken, NJ, USA: Wiley, 2011.
[35] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*. Philadelphia, PA, USA: SIAM, 1997.
[36] S. J. Bradtke and A. G. Barto, “Linear least-squares algorithms for temporal difference learning,” *Mach. Learn.*, vol. 22, no. 1–3, pp. 33–57, 1996.
[37] P. Bremaud, *Markov Chains—Gibbs Fields, Monte Carlo Simulation, and Queues*. New York, NY, USA: Springer, 1999.
[38] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[39] J. Boyan, “Technical update: Least-squares temporal difference learning,” *Mach. Learn.*, vol. 49, pp. 233–246, 2002.
[40] D. P. Bertsekas and S. Ioffe, “Temporal differences-based policy iteration and applications in neuro-dynamic programming,” Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep. LIDS-P-2349, 1996.
[41] D. P. Bertsekas, V. Borkar, and A. Nedic, “Improved temporal difference methods with linear function approximation,” in *Learning and Approximate Dynamic Programming*. Piscataway, NJ, USA: IEEE Press, 2004.
[42] X. Cao, Z. Ren, S. Bhatnagar, M. Fu, and S. Marcus, “A time aggregation approach to Markov decision processes,” *Automatica*, vol. 38, no. 6, pp. 929–943, 2002.
[43] H. Chang, J. Hu, M. Fu, and S. Marcus, *Simulation-Based Algorithms for Markov Decision Processes*. New York, NY, USA: Springer, 2007.
[44] J. Klein, *ltl2dstar—LTL to deterministic Streett and Rabin automata*, 2007. [Online]. Available: http://www.ltl2dstar.de/



Kun Deng received the B.E. degree in automatic control and the M.S. degree in mechanical engineering from Tsinghua University, Beijing, China, in 2005 and 2007, respectively, and the M.S. degree in mathematics and the Ph.D. degree in mechanical engineering from the University of Illinois at Urbana–Champaign, Urbana, IL, USA, in 2010 and 2012, respectively.

He is currently a Research Engineer with Ford Motor Company, Dearborn, MI, USA. Prior to joining Ford, he was a Project Engineer with Whirlpool Corporation. His research interests include model reduction, control and optimization, and system identification.

Yushan Chen received the Ph.D. degree in computer engineering from Boston University, Boston, MA, USA, in 2013.

She is the Director of Product at Orbeus Inc., Chicago, IL, USA. Her research interests include multiagent control, formal synthesis, machine learning, and applications in robotics and computer vision.

Dr. Chen received the Best Student Paper Award at the International Symposium on Distributed Autonomous Robotic Systems in 2010.

Calin Belta (M'03–SM'11–F'17) received the B.Sc. and M.Sc. degrees in control and computer science from the Technical University of Iasi, Iasi, Romania, in 1995 and 1996, respectively, the M.Sc. degree in electrical engineering from Louisiana State University, Baton Rouge, LA, USA, in 1999, and the M.Sc. and Ph.D. degrees in mechanical engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 2001 and 2003, respectively.

He is currently Professor with the Department of Mechanical Engineering, Boston University (BU), Boston, MA, USA, where he holds the Tegan family Distinguished Faculty Fellowship. He is the Director of the BU Robotics Laboratory and is also affiliated with the Department of Electrical and Computer Engineering, the Division of Systems Engineering, the Center for Information and Systems Engineering, and the Bioinformatics Program. His research interests include dynamics and control theory, with particular emphasis on hybrid and cyber-physical systems, formal synthesis and verification, and applications in robotics and systems biology.

Prof. Belta received the Air Force Office of Scientific Research Young Investigator Award and the National Science Foundation CAREER Award.