Advanced Robotics

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tadr20

Model-based motion planning in POMDPs with temporal logic specifications

Junchao Li, Mingyu Cai, Zhaoan Wang & Shaoping Xiao

To cite this article: Junchao Li, Mingyu Cai, Zhaoan Wang & Shaoping Xiao (2023) Model-based motion planning in POMDPs with temporal logic specifications, Advanced Robotics, 37:14, 871-886, DOI: 10.1080/01691864.2023.2226191

To link to this article: https://doi.org/10.1080/01691864.2023.2226191

ADVANCED ROBOTICS
2023, VOL. 37, NO. 14, 871–886
https://doi.org/10.1080/01691864.2023.2226191

FULL PAPER

Model-based motion planning in POMDPs with temporal logic specifications

Junchao Li[a], Mingyu Cai[b], Zhaoan Wang[a] and Shaoping Xiao[a]

[a]Department of Mechanical Engineering, Iowa Technology Institute, The University of Iowa, Iowa City, IA, USA; [b]Department of Mechanical Engineering, Lehigh University, Bethlehem, PA, USA

ABSTRACT

Partially observable Markov decision processes (POMDPs) have been used as mathematical models for sequential decision-making under uncertain and incomplete information. Since the state space is partially observable in a POMDP, the agent has to make a decision based on the integrated information over the past experiences of actions and observations. This study aims to solve probabilistic motion planning problems in which the agent is assigned a complex task under a partially observable environment. We employ linear temporal logic (LTL) to formulate the complex task and then convert it to a limit-deterministic generalized Büchi automaton (LDGBA). We reformulate the problem as finding an optimal policy on the product of POMDP and LDGBA based on model-checking techniques. This paper adopts and modifies two reinforcement learning (RL) approaches: value iteration and deep Q-learning. Both are model-based because the optimal policy is a function of belief states that need transition and observation probabilities to be updated. We illustrate the applicability of the proposed methods by addressing two simulations, including a grid-world problem with various sizes and a TurtleBot office path planning problem.

ARTICLE HISTORY
Received 15 November 2022
Revised 11 March 2023
Accepted 14 May 2023

KEYWORDS
Partially observable Markov decision process; linear temporal logics; automaton; reinforcement learning; deep q-learning

1. Introduction

Markov decision processes (MDPs) [1] have been widely applied in robotics motion planning, assuming the environment was fully observable. However, in some real-world applications, the agent may not have access to complete or reliable information about the state of the environment. Therefore, partially observable Markov decision processes (POMDPs) [2] need to be adopted. POMDPs are particularly useful when sensors or perception systems are prone to errors or uncertainty. By incorporating the observation probability model into the POMDP framework, it is possible to quantify the effects of perception errors on the agent's decision-making performance. On the other hand, there has been increasing interest in considering complex tasks in robotics motion planning other than simple go-to-goal missions, especially dealing with uncertain and dynamic environments.

Reinforcement learning (RL) methods [3] have been employed to solve robotics motion planning in partially observable environments. Model-based RL approaches assume that the agent knows the probabilistic characteristics of transition and observation. Therefore, a belief state can be updated via the Bayesian theory [2, 4] to denote a probability distribution over all possible states. Consequently, a POMDP problem with the original state space becomes an MDP problem with a corresponding belief space. In this case, the policy is a function of belief states for action selection. One commonly-used approach is the value iteration algorithm to solve POMDP problems as a form of dynamic programing.

Early works aim to determine exact value functions in the belief space for small POMDP problems, e.g. enumeration algorithms [2, 5]. Based on Sondik's one-pass algorithm [5], Cheng [6] proposed a simple linear support algorithm, which started at a random belief state and generated a vector for its value function. Also, the witness algorithm [7] simplified the POMDP problem by considering one action and one observation at a time. Furthermore, Zhang and Liu [8] integrated the enumeration and witness algorithms and proposed a so-called incremental pruning algorithm. However, in large POMDP problems with complex dimensions, finding the optimal policy precisely in the belief space can be computationally intensive. Therefore, point-based value iteration (PBVI) algorithms [9] were proposed for infinite-horizon POMDP problems by approximating optimally reachable belief spaces to address this issue. This approach updated value functions locally for a finite subset of the belief space. One of the point-based solvers [10], SARSOP (Successive Approximations of the Reachable Space under Optimal Policies) [11, 12], could handle POMDP problems with large state spaces.

CONTACT Junchao Li junchao-li@uiowa.edu

Another approach is employing Q-learning to learn the optimal policy on the belief state space of a POMDP. It shall be noted that Q-learning [13] is a model-free approach that allows the agent to learn to act optimally in MDP problems. However, since the belief states need to be calculated from the transition and observation probabilities, this approach is still model-based. On the other hand, because the belief state spaces are usually continuous in POMDP problems, classical Q-learning algorithms such as tabular Q-learning [13] are no longer applicable. Consequently, deep Q-network [14] (DQN) is employed in this paper to map POMDP belief states to state-action values. Some other works [15, 16] have utilized DQN to achieve the optimal policy regarding a sequence of observations. However, they were model-free approaches to POMDPs, which is not our focus in this study.

Many works have been done to consider complex tasks other than simple go-to-goal missions in MDP problems by adopting formal languages [17], such as linear temporal logic (LTL), to formulate user-defined high-level specifications. Then, the LTL formula is commonly converted to an ω-automaton over infinite words with a Büchi or a Rabin acceptance condition [18]. Consequently, robotics motion planning problems can be solved via control synthesis for a product of MDP and automaton with model checking [17]. It has been shown that a limit-deterministic Büchi automaton (LDBA) has more advantages than a deterministic Rabin automaton (DRA). Hahn *et al.* [19] implemented LDBA and DRA in model-free RL with mild restrictions, respectively. They concluded that LDBA was more suitable for both qualitative and quantitative analysis of MDPs under all ω-regular objectives. In another work, Hasanbeig *et al.* [20] stated that converting the LTL specification into an LDBA might result in a smaller automaton state space than a DRA. However, LDBA suffers the sparsity of reward and may highly slow the RL's convergence [21].

Similarly, model checking provides the formal approach to verifying complex task objectives when solving POMDP problems with certain temporal logic constraints. Chatterjee *et al.* [22] studied the undecidability of the qualitative model checking in a POMDP problem with the infinite horizon. They proposed a finite-memory approach for the verification and synthesis of POMDPs with LTL constraints. It relied on exploring the entire belief space and was most suitable to the problems with small state spaces. In other works [23, 24], LTL specifications were converted to a DRA, and then a product of POMDP and DRA was constructed. Specifically, Sharan *et al.* [23] employed finite state controllers to limit the policy search via the value iteration methods. On the other hand, Bouton *et al.* [24] utilized the approximate POMDP solver, SARSOP [11], to search for an optimal policy on the finite belief state space of the product POMDP. Finally, it shall be noted that Wang *et al.* [25] employed LDBA and converted a POMDP to the corresponding belief MDP before building a product of belief MDP and LDBA. However, simulation examples are expected to demonstrate the feasibility of the proposed method.

The first contribution of this paper is formulating a complex task via LTL, converting it to a limit-deterministic generalized Büchi automaton (LDGBA), and constructing a product of POMDP with LDGBA. Although the strategy of representing complex tasks by LTL and then automaton is not new, most works focused on MDP instead of POMDP problems. In addition, this study employs LDGBA for the first time in POMDP problems, since it can lead to a smaller product state space [19] than DRA [23, 24] and can address the sparsity of rewards caused by LDBA [21]. The second contribution is to reformulate the original problem of finding a policy that satisfies LTL specifications in a POMDP as finding an optimal policy to maximize the collected reward on the corresponding product of POMDP and LDGBA. Therefore, model-based RL approaches can be employed. In addition to a modified PBVI solver, we propose convolutional neural network (CNN)-enhanced deep Q networks to approximate the state-action value functions at a given belief product state. The last contribution is redesigning the reward function and introducing a frontier set in LDGBA to record non-visited accepting sets so that the proposed methods can efficiently handle surveillance tasks.

Model-predictive Control (MPC), Rapidly-exploring Random Trees (RRT), and Probabilistic Roadmaps (PRM) are classical motion and trajectory planning algorithms. Recently, some works like [26] utilized RRT to map a belief state to possible paths that connect the start state to a goal state for pathfinding. Therefore, RRT (and PRM) can address the observation uncertainty. Although this approach could be extended to solve the product POMDP problems proposed in this study to handle complex tasks, no such works were reported based on the authors' best knowledge. On the other hand, the belief state space in POMDP is infinite. When model-based motion planning algorithms, such as RRT and PRM, search the paths on belief state spaces, they can be computationally expensive. On the other hand, an online MPC solver will be more appropriate when the observation uncertainty is considered. However, the optimal control policy needs to be updated frequently based on the new measurements. This method can be computationally intensive, too. Compared with MPC, RRT, and PRM, one of our approaches utilizes the SARSOP solver,

which explores a belief tree on the optimally reachable belief space to improve computational efficiency.

The organization of this paper is described as follows. Section 2 reviews POMDP problems and state-of-art solutions, including PBVI/SARSOP and Deep Q-learning. Section 3 introduces LTL, LDGBA, and product POMDP. Then, we redefine the POMDP problem with LTL specifications and depict reward design and tracking frontier function. Section 4 proposes the approaches to product POMDP problems and provides detailed algorithms. Finally, two simulations and results are included in Section 5, followed by the conclusion and future works.

2. POMDP and its solution

2.1. POMDP

The POMDP is a mathematical framework to model a problem in which the environment is not fully observable.

Definition 2.1 (POMDP): Considering the action and observation uncertainties and the labels of states, a POMDP can be denoted by a tuple $\mathcal{P} = (S, A, T, s_0, R, O, \Omega, \Pi, L)$, where:

- $S = \{s_1, \ldots, s_n\}$ is a finite set of states.
- $A = \{a_1, \ldots, a_m\}$ is a finite set of actions. Specifically, $A(s)$ represents the set of available actions that can be taken by the agent at state s.
- $T : S \times A \times S \to [0, 1]$ is a function representing the transition probability from state $s \in S$ to state $s' \in S$ after the agent takes action $a \in A(s)$. It satisfies $\sum_{s' \in S} T(s, a, s') = 1$.
- $s_0 \in S$ is the initial state.
- $R : S \to R$ is a reward function, which can also be defined as $R(s, a, s')$ or $R(a, s')$.
- $O = \{o_1, \ldots, o_z\}$ is a finite set of observations, and $O(s)$ denotes the set of possible observations that the agent can acquire at state s.
- $\Omega : S \times A \times O \to [0, 1]$ is a function representing the observation probability that the agent can perceive at state $s' \in S$ after taking action $a \in A(s)$. This function satisfies $\sum_{o \in O} \Omega(s', a, o) = 1$.
- Π is a set of atomic propositions.
- $L : S \to 2^{\Pi}$ is a labeling function, and 2^{Π} is the power set of Π.

At every time step during the learning, the agent is at state $s \in S$ and selects an action $a \in A(s)$, which transits the agent to the next state $s' \in S$. Since we consider the motion uncertainty, the probability of such a transition is $T(s, a, s')$. After the agent reaches the next state (s'), it then can perceive an observation $o \in O(s')$ with the probability of $\Omega(s', a, o)$ to gather the information of this state. On the other hand, after the transition, the agent also collects a reward given by the function $R(s')$. The goal of the agent choosing a sequence of actions is to maximize its expected return, i.e. the accumulated rewards, starting from state s at the current time $t = 0$ as

$$U(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \,\middle|\, s_{t=0} = s\right] \tag{1}$$

where s_t denotes the agent's state at time t. $\gamma \in [0, 1]$ is the discount factor to balance the importance between immediate and future rewards.

In model-based approaches to solving POMDPs, it is common to find an optimal policy in the belief state space instead of the state space defined in the POMDP, i.e. S. A belief state, $b \in B$ where $B = Dist(S)$ is the belief state space, represents the probability distribution over all the possible states $s \in S$. Specifically, $b_t(s)$ denotes the probability of the agent being at state s at time step t. The initial belief state, b_0, depends on the agent's knowledge of its initial state s_0. If the agent is aware of its initial state, $b_0(s_0) = 1$ and $b_0(s) = 0$ for $s \neq s_0$. Otherwise, b_0 is a uniform probability distribution. If the agent's current belief state is $b_t(s)$, after taking action a_t and obtaining observation o_{t+1}, the new belief state $b_{t+1}(s)$ can be updated by Icarte et al. [27]:

$$b_{t+1}(s') \propto \Omega(s', a_t, o_{t+1}) \sum_{s \in S} T(s, a_t, s') b_t(s) \tag{2}$$

Consequently, the belief state holds the experience of a complete history [27], and a policy can map the belief state $b_t \in B$ to the action $a_t \in A$. Then, the expected return in (1) under a policy ξ can be rewritten as

$$U^{\xi}(b) = \mathbb{E}^{\xi}\left[\sum_{t=0}^{\infty} \gamma^t R(b_t) \,\middle|\, b_{t=0} = b\right] \tag{3}$$

where $R(b_t) = \sum_{s \in S} b_t(s) R(s)$ is the reward that the agent can collect depending on the current belief state. Finally, the POMDP problem becomes finding the optimal policies in a corresponding MDP with the belief state space to maximize the expected return, i.e. $\boldsymbol{\xi}^* = \text{argmax}_{\boldsymbol{\xi}} U^{\boldsymbol{\xi}}(b)$.

2.2. PBVI and deep Q-learning

Since the belief state space is infinite and most belief points are very unlikely to be reached, the PBVI algorithms [9] prune away belief points by selecting the best action on every trajectory to generate the next belief point. Specifically, the PBVI algorithms introduce a set

of alpha vectors $\mathcal{V} = \{\alpha_0, \alpha_1, \ldots, \alpha_m\}$ to approximate the value functions as [9]:

$$V(b) = \max_{\alpha \in \mathcal{V}} \sum_{s \in S} \alpha(s) b(s) \tag{4}$$

It shall be noted that each alpha vector is an $|S|$-dimensional hyper-plane and constrained by the boundary of the belief state space, i.e. B. A number of alpha vectors form the approximated value function $V(b)$, which is a piece-wise linear and convex function [9]. Only one alpha vector on each belief point is maintained during the value backup, which is defined as [9]

$$V(b) = R(b) + \gamma \max_{a \in A} \sum_{o \in O} \max_{\alpha \in \mathcal{V}} \sum_{s \in S} \sum_{s' \in S} \Omega(s', a, o) \times T(s, a, s') \alpha(s') b(s) \tag{5}$$

In addition, the PBVI algorithms use a precision parameter, the difference between an upper bound and a lower bound of the value function, to guarantee policy convergence [9].

Another reported approach [14] to POMDP problems is utilizing deep neural networks (DNNs) to approximate the state-action values. Mnih *et al.* [16] firstly introduced Deep Q-learning (DQN) to solve MDP problems. Their DQN architecture consists of two DNNs (called Q-networks), an evaluation Q-network $Q_e(s, a; \theta_e)$ and a target Q-network $Q_t(s, a; \theta_t)$ to keep the learning more stable and effective. During the learning process, ϵ-greedy exploration strategy [3] and experience replay memory [28] are used for the action selection and the collection of the training samples, respectively.

M. Egorov [14] leveraged DQN to solve POMDP problems by mapping POMDP beliefs to the state-action values. Instead of the model-free algorithm commonly used in MDP problems, his approach updates the belief state as the input feature to Q networks. Consequently, this approach is still model-based, and the derived policy is a function of belief states.

3. Linear temporal logic and product POMDP

3.1. Linear temporal logic (LTL)

Linear temporal logic is a high-level language to describe user-specified tasks. Basically, an LTL specification can be formulated inductively via the combination of Boolean operators, such as negation ($\neg$) and conjunction ($\wedge$), and two temporal operators, including next ($\bigcirc$) and until ($\mathcal{U}$). The formula $\bigcirc\phi$ can be read as 'ϕ is true at the next state' while $\phi_1 \mathcal{U} \phi_2$ as 'ϕ_2 is true at some future states and ϕ_1 is true at each state until then.' Consequently, the syntax of an LTL formula is defined inductively as [29]

$$\phi := \text{True} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc\phi \mid \phi_1 \mathcal{U} \phi_2 \tag{6}$$

where $a \in \Pi$ is an atomic proposition. Other common Boolean and temporal operators are derived as follows:

$$\begin{array}{ll} \text{or}: & \phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2) \\ \text{implies}: & \phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2 \\ \text{eventually}: & \Diamond\phi \equiv \text{True}\mathcal{U}\phi \\ \text{always}: & \Box\phi \equiv \neg(\Diamond\neg\phi) \end{array}$$

Let $\models$ denote the satisfaction relationship. The semantics of an LTL formula is interpreted over words, which is an infinite sequence $\boldsymbol{w} = w_0 w_1 \ldots$ with $w_i \in 2^{\Pi}$ for all $i \geq 0$, and defined as:

$$\begin{array}{lcl} \boldsymbol{w} \models \text{True} & & \\ \boldsymbol{w} \models \alpha & \Leftrightarrow & \alpha \in L(\boldsymbol{w}[0]) \\ \boldsymbol{w} \models \phi_1 \wedge \phi_2 & \Leftrightarrow & \boldsymbol{w} \models \phi_1 \text{and} \boldsymbol{w} \models \phi_2 \\ \boldsymbol{w} \models \neg\phi & \Leftrightarrow & \boldsymbol{w} \not\models \phi \\ \boldsymbol{w} \models \bigcirc\phi & \Leftrightarrow & \boldsymbol{w}[1:] \models \phi \\ \boldsymbol{w} \models \phi_1 \mathcal{U} \phi_2 & \Leftrightarrow & \exists t \text{s.t.} \boldsymbol{w}[t:] \models \phi_2, \forall t' \in [0, t), \\ & & \boldsymbol{w}[t':] \models \phi_1 \end{array}$$

3.2. Limit-deterministic generalized Büchi automaton (LDGBA)

Given an LTL that specifies a complex task, its satisfaction can be evaluated by an LDGBA [30].

Definition 3.1 (LDGBA): An LDGBA is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$,where Q is a finite set of states, $\Sigma = 2^{\Pi}$ is a finite alphabet, $\delta: \ Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$ is the transition function, $q_0 \in Q$ is an initial state, and $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_f\}$ is a set of accepting sets with $\mathcal{F}_i \subseteq Q$, $\forall i \in \{1, \ldots, f\}$. The state set Q is partitioned into a deterministic set Q_D and a non-deterministic set Q_N, i.e. $Q_D \cup Q_N = Q$ and $Q_D \cap Q_N = \emptyset$, where

- The state transitions in Q_D are total and restricted within it, i.e. $|\delta(q, \alpha)| = 1$ and $\delta(q, \alpha) \subseteq Q_D$ for every state $q \in Q_D$ and $\alpha \in \Sigma$,
- The ϵ-transition is not allowed in the deterministic set, i.e. for any $q \in Q_D$, $\delta(q, \epsilon) = \emptyset$,
- The ϵ-transitions are only defined for state transitions from Q_N to Q_D, which do not consume the input alphabet, and
- The accepting sets are only in the deterministic set, i.e. $\mathcal{F}_i \subseteq Q_D$ for every $\mathcal{F}_i \in \mathcal{F}$.

A run of an LDGBA, subject to an input word $\boldsymbol{w} = w_0 w_1 \ldots$, can be represented as $\boldsymbol{q} = q_0 q_1 \ldots$, and $\inf(\boldsymbol{q})$ represents the infinite portion of $\boldsymbol{q}$. If there exists $\inf(\boldsymbol{q}) \cap \mathcal{F}_i \neq \emptyset$, $\forall i \in \{1, \ldots f\}$, we say that $\boldsymbol{q}$ satisfies

the LDGBA acceptance condition. In other words, the LDGBA accepts the word $\boldsymbol{w}$. We recommend Owl [18] to readers for more details about automaton generation. Consequently, this study aims to solve the POMDP problems with LTL specifications defined below.

Problem 3.1: Given a POMDP with its belief state space B that can be derived via (2) and a task expressed as an LTL formula. The objective is to find a policy $\xi^*(b)$ that can complete the task by satisfying the acceptance condition of the LTL-induced LDGBA.

3.3. Product POMDP

Therefore, we introduce a framework for solving a POMDP problem by exploiting the fact that an LTL formula can be transformed into an LDGBA representing the task variables and safety constraints of the POMDP. The problem of satisfying a given LTL objective ϕ in a POMDP $\mathcal{P}$ can be reduced to the problem of satisfying a repeated reachability (Büchi) objective ϕ_B in the product POMDP.

Definition 3.2 (Product POMDP): The product POMDP $\mathcal{P}^\times = \mathcal{P} \times \mathcal{A}$ of a POMDP $\mathcal{P} = (S, A, T, s_0, R, O, \Omega, \Pi, L)$ and an LDGBA $\mathcal{A} = (Q, \Sigma, \delta, q_0, \mathcal{F})$ is defined as a tuple $\mathcal{P}^\times = (S^\times, A^\times, T^\times, s_0^\times, R^\times, O, \Omega^\times, \mathcal{F}^\times)$ where:

- $S^\times = S \times Q$ is the finite set of labeled states, i.e. $s^\times = \langle s, q\rangle \in S^\times$ where $s \in S$ and $q \in Q$.
- $A^\times = A \cup \{\epsilon\}$ is the set of actions.
- $T^\times = S^\times \times A^\times \times S^\times \to [0, 1]$ is the transition function, specifically,

$$T^\times\left(s^\times, a^\times, s^{\times\prime}\right) = \begin{cases} T(s,a,s') & q' = \delta\left(q, l\right), \\ & l \in L(s') \text{ and } a \in A \\ 1 & a^\times \in \{\epsilon\} \text{ and} \\ & q' \in \delta(q, \epsilon) \text{ and} \\ & s' = s, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

- $s_0^\times = \langle s_0, q_0\rangle \in S^\times$ is the initial state, where $s_0 \in S$ and $q_0 \in Q$.
- $R^\times = S^\times \to R$ is the reward function of the product state $\langle s, q\rangle$.

$$R^\times(s^\times) = \begin{cases} R(s) & l \in L(s), q' = \delta\left(q, l\right) \in \mathcal{F}_i, \text{ and} \\ & \mathcal{F}_i \in \mathcal{F} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

- $\Omega^\times = S^\times \times A^\times \times O \to [0, 1]$ is the observation function

$$\Omega^\times(s^{\times\prime}, a^\times, o) = \Omega(s', a^\times, o) \tag{9}$$

where $s' \in S$ and $a^\times \in A$. On the other hand, if $a^\times \in \{\epsilon\}$, the agent stays at the same s, i.e. $s' = s$, and the belief state won't be updated.

- $\mathcal{F}^\times = \{\mathcal{F}_1^\times, \mathcal{F}_2^\times, \ldots, \mathcal{F}_f^\times\}$ is the set of accepting sets, where $\mathcal{F}_i^\times = \{\langle s, q\rangle | s \in S; q \in \mathcal{F}_i\}, i = 1, \ldots f$.

A random path $(s_0, q_0)(s_1, q_1)\ldots$ of the product POMDP corresponds uniquely to the combination of a path $s_0, s_1 \ldots$ of the POMDP and a path $q_0, q_1 \ldots$ of the LDGBA. On the other hand, the belief product state denoted as $b^\times \in B^\times$, where $B^\times = Dist(S^\times)$, represents the probability distribution over all the possible product POMDP states. Similarly, the new belief product state can be updated by the formula derived from (2):

$$b_{t+1}^\times(s^{\times\prime}) \propto \Omega^\times(s^{\times\prime}, a_t^\times, o_{t+1}) \sum_{s^\times \in S^\times} T^\times(s^\times, a_t^\times, s^{\times\prime}) \times b_t^\times(s^\times) \tag{10}$$

It is noted that the product POMDP shares the same observation space of POMDP. The initial belief state is $b_0^\times \in B^\times$. Considering the initial automaton state is q_0, $b_0^\times$ is defined as:

$$b_0^\times(s, q) = \begin{cases} b_0(s) & q = q_0 \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

The expected return under a policy $\xi^\times$ can be written as below, similar to (1).

$$U^{\xi^\times}(b^\times) = \mathbb{E}^{\xi^\times}\left[\sum_{t=0}^{\infty} \gamma^t R(b_t^\times) \middle| b_{t=0}^\times = b^\times\right] \tag{12}$$

where the reward function remains as

$$R(b_t^\times) = \sum_{s^\times \in S^\times} b_t^\times(s^\times) R(s^\times). \tag{13}$$

The constructed product POMDP $\mathcal{P}^\times$ can be interpreted as the POMDP $\mathcal{P}$ with the augmented state space to account for the temporal logic specifications represented by LDGBA $\mathcal{A}$. All the feasible paths on $\mathcal{P}^\times$ share the intersections between all the accessible paths over $\mathcal{P}$ and all words accepted by $\mathcal{A}$. Specifically, a path $\sigma^{\xi^\times} = (s_0, q_0)(s_1, q_1)\ldots$, generated by a random policy $\xi^\times$ on the product POMDP $\mathcal{P}^\times$, is accepted if $\inf(\sigma^{\xi^\times}) \cap \mathcal{F}_i^\times \neq \emptyset, \forall i = 1, \ldots f$, where $\mathcal{F}^\times$ captures the acceptance conditions of LDGBA $\mathcal{A}$.

The belief state $b_t(s)$ represents the probability distribution of the agent in POMDP state $s \in S$ given the history up to time t. b_{t+1} is then updated from the previous belief state b_t, the executed action a_t, and the resulting observation o_{t+1} by (2) for all $s' \in S$. Based on that, $b_t \in B$ also has the Markovian property. Therefore, finding the

optimal policy as a function of belief states in a POMDP problem is equivalent to solving an MDP problem with a continuous belief state space. Moreover, according to previous studies on MDP problems with LTL specifications [29, 31–33], the optimal policy $\xi^{\times *}(b^{\times})$ on the product POMDP $\mathcal{P}^{\times}$ is also the optimal policy $\xi^{*}(b)$ on the POMDP $\mathcal{P}$ satisfying LTL specifications. Consequently, Problem 3.1 can be reformulated as follows.

Problem 3.2: Given a product POMDP defined in Section 3.3 as $\mathcal{P}^{\times} = \mathcal{P} \times \mathcal{A}$ of a POMDP $\mathcal{P}$ and an LDGBA $\mathcal{A}$ generated from LTL specifications ϕ, the corresponding belief state space can be derived via (10). The objective is to find a policy $\xi^{\times *}(b^{\times})$ so that the expected return is maximized.

3.4. Reward redesign or tracking-frontier function

When directly applying the constructed LDGBA in solving a product POMDP problem defined in Problem 3.2, it may fail to find the deterministic policy that satisfies the LTL specifications [20, 21]. For example, to complete the task that an agent shall visit states labeled with 'a' and 'b' once at a time for infinitely many times, the LTL formula can be written as

$$\phi = (\square\diamond a \wedge \square\diamond b) \wedge \square\neg c \tag{14}$$

Rabinizer 4 [18] is used to convert the LTL formula into an LDGBA. We augment the accepting states in order to separate 'a' and 'b' transitions. In addition, we simplify the automaton by only keeping single labeled transitions, which is sufficient to explain this example. The full automaton after states augmentation is shown in Figure 1, and the set of accepting sets is $\mathcal{F} = \{\{q_0\}, \{q_1\}\}$. Theoretically, this automaton may accept a word of $(b^*a^*)^{\omega}$ where $*\omega * \omega$ matche the preceding character(s) finite times and infinite times. However, since the typical reward design in (8) depends on the product states corresponding to the LDGBA accepting states, the agent may tend to keep visiting one of the labeled states infinitely many times to collect more rewards. Consequently, the specified task cannot be accomplished.

To address the above issue for surveillance tasks, we redesign the reward function as below by adding a constraint to (8) such that the agent can visit the accepting sets repeatedly.

$$R^{\times}(s^{\times}) = \begin{cases} R(s) & l \in L(s), q' = \delta\left(q, l\right) \in \mathcal{F}_i, \mathcal{F}_i \in \mathcal{F}, \\ & \text{and } q' \neq q \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

where $q \neq q'$ prevents the repeated transitions, which lead to the same automaton accepting state by removing the rewards on the associated labeled POMDP states.

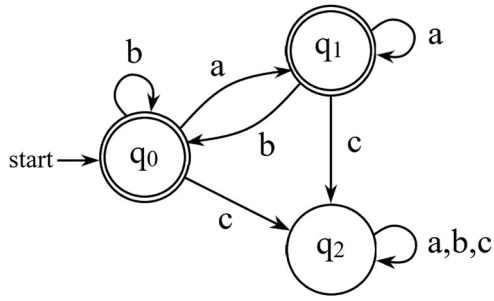

Figure 1. an LDGBA automaton.

After applying this constraint to the reward function, the derived optimal policy satisfies the desired task specification. We provide the simulation of this example with the model-based SARSOP solver on the POMDP environment in Section 5.1.

In addition, we introduce another approach that implements a frontier set $\mathcal{T}$ to keep track of non-visited accepting sets based on the previous work by M. Cai [32]. In most cases, $\mathcal{T}$ is initialized as $\mathcal{F}$. If a state of one or more accepting sets has been visited, those sets will be removed from the frontier set $\mathcal{T}$. Mathematically, the tracking-frontier function $\mathcal{T}_{\mathcal{F}}$ is defined and updated as [32]:

$$\mathcal{T}_{\mathcal{F}}\left(q, \mathcal{T}\right) = \begin{cases} \mathcal{T} \setminus \mathcal{F}_i, & \text{if } q \in \mathcal{F}_i \text{ and } \mathcal{F}_i \in \mathcal{T}, \\ \mathcal{F} \setminus \mathcal{F}_i, & \text{if } q \in \mathcal{F}_i \text{ and } \mathcal{T} \setminus \mathcal{F}_i = \emptyset, \\ \mathcal{T}, & \text{otherwise.} \end{cases} \tag{16}$$

Once the frontier set $\mathcal{T}$ becomes empty, it will be reset as $\mathcal{F}$ if the specification requires the infinite visits of all accepting sets. Consequently, the accepted words in Figure 1 can be $(ab)^*$, $(ba)^*$. It shall be noted that the visited LDGBA accepting sets are removed from the frontier set $\mathcal{T}$, and at the same time, the rewards on the associated labeled states are disabled. Therefore, it prevents the repeated visiting of the same automaton state. Furthermore, the original definition of reward in (8) is redefined as

$$R^{\times}(s^{\times}) = \begin{cases} R(s) & l \in L(s), q' = \delta\left(q, l\right) \in \mathcal{F}_i, \text{ and} \\ & \mathcal{F}_i \in \mathcal{T} \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

The above tracking-frontier function $\mathcal{T}_{\mathcal{F}}$ can be revised according to different task specifications. For example, additional constraints can be added to $\mathcal{T}_{\mathcal{F}}$ that only removes $\mathcal{F}_i$ in order, forcing the agent to visit the labeled states in a specific sequence. This approach is not easily directly implemented in the SARSOP solver but in DQN. Therefore, we only applied the frontier set to DQN in this study.

4. Problem solutions

4.1. Point-based value iteration

A product POMDP can be viewed as a POMDP but simultaneously satisfies the task constraints provided by the LTL-induced automaton. The transitions in the product POMDP are restricted and prevented by the automaton transitions. Therefore, the proposed product POMDP problem can be solved by the methods that have been successfully applied to POMDP problems. It shall be noted that the state space is extended due to the product of POMDP and LDGBA.

One of our approaches uses SARSOP [11] to approximate the optimal value functions on product POMDP. To achieve this, SARSOP introduces a lower-bound target level L and an upper-bound target level U. A target gap size ε between L and U at $b_0^\times$ is initialized. When the sampling process follows the belief tree $T_\mathcal{R}$, the target levels, i.e. L and U, are updated as L_t and U_t where t is the depth of the node in the tree. The sampling path will be terminated once the gap size between L_t and U_t for all the leaves in $T_\mathcal{R}$ reaches $\gamma^{-t}\varepsilon$ where γ is the discount factor [11].

To update the lower and upper-bound target levels (L_t and U_t) from L_{t-1} and U_{t-1}, the following equation is utilized first to calculate the lower bound of the optimal Q value [11] once an action is selected.

$$\underline{Q}(b^\times, a^\times) = R(b^\times) + \gamma \sum_{o\in O} P(o|b^\times, a^\times)\underline{V}(b^{\times\prime}) \quad (18)$$

where $\underline{V}$, obtained by the fixed-action policy [34], is the lower bound of the optimal value function V^* at $b^{\times\prime}$. The upper bound of the optimal Q value, $\overline{Q}$, is obtained similar to (18), and its upper bound $\overline{V}$ can be acquired by sawtooth approximation [34]. In addition, $P(o|b^\times, a^\times)$ is defined as [10]

$$P(o|b^\times, a^\times) = \sum_{s^{\times\prime}\in S^\times} \Omega^\times(s^{\times\prime}, a^\times, o) \times \sum_{s^\times\in S^\times} T^\times(s^\times, a^\times, s^{\times\prime}) b^\times(s^\times) \quad (19)$$

Then, $\underline{Q}$ is used to find an intermediate lower-bound target level L' as the maximum value between L_{t-1} and $\underline{Q}$. Similarly, an intermediate upper-bound target level $\overline{U}'$ is the maximum value between U_{t-1} and $\underline{Q} + \gamma^{-t}\varepsilon$. Finally, the target level L_t for the next belief node $b^{\times\prime}$ can be calculated, and it is needed for $\underline{Q}$ to achieve its target level L'. Similarly, U_t is acquired by computing $\overline{Q}$ with U' [11].

Next, the standard backup process for α-vectors in Γ is performed with the value function approximation. The last step is the pruning process [11] in which the sub-optimal belief nodes and α-vectors are pruned away. Finally, a more strict requirement of dominance check, called δ-dominance [11], is used to eliminate the sub-optimal α-vectors. Algorithm 1 describes the implementation of this approach to determine a set of α vectors, Γ, which is used to approximate the optimal value functions of the product POMDP $\mathcal{P}^\times$.

Algorithm 1 Point-based Model Checking (SARSOP) on POMDPs with LTL specifications.

Require: LTL formula ϕ, POMDP $\mathcal{P}$, precision ε
Convert ϕ to LDGBA Automaton $\mathcal{A}$,
Construct The product POMDPs $\mathcal{P}^\times = \mathcal{P} \times \mathcal{A}$
Initialize the initial belief $b_0^\times \in B^\times$ as the root of the search tree $T_\mathcal{R}$. Set upper-bound target level U and lower-bound target level L.
Initialize the set of α vectors Γ, upper and lower bounds $\overline{V}$ and $\underline{V}$ of V^*. $\hat{V}$ is set as the prediction of V^*. Sample the initial points ($T_\mathcal{R}$, Γ, $b_0^\times$, L, U, ε, t=1), where t is the time step.
while Termination is not satisfied **do**
  for Sampling points($T_\mathcal{R}$, Γ, $b^\times$, L, U, ε, t) **do**:
    if $\hat{V} \le L$ and $\overline{V} \le max\{U, \underline{V}(b) + \gamma^{-t}\varepsilon\}$ **then**:
      Return
    else
      Update the L_t and U_t.
      Compute the next belief $b^{\times\prime}$ by (10).
      Sample points ($T_\mathcal{R}$, Γ, $b^{\times\prime}$, L_t, U_t, ε, $t+1$).
    end if
  end for
  Perform α-vector backup at the belief node $b^\times$ of $T_\mathcal{R}$ by (5).
  if $\overline{Q}(b^\times, a^\times) < \underline{Q}(b^\times, a^{\times\prime})$ **then**
    Prune away all the points from $b^\times$ taken $a^\times$ in $T_\mathcal{R}$.
  end if
  if α_1 δ-dominates α_2 **then**
    Prune away α_2.
  end if
end while
Return Γ.

4.2. Deep Q-learning

Another approach for solving product POMDP problems is deep Q-learning (DQN). This approach employs neural networks to map a belief product state to the corresponding state-action values, i.e. Q values, for the agent choosing the best action. Compared to the SARSOP solver, DQN requires more computation time since

it uses Monte Carlo simulation. However, it is easier to implement the tracking-frontier function (16) $\mathcal{T}_{\mathcal{F}}(q, \mathcal{T})$ in DQN than in the SARSOP solver. As stated in Section 3.4, the LTL-induced LDGBA with the frontier set $\mathcal{T}$ can handle more complex tasks without adding extra computational complexity [32] to the original derived automaton. In this study, the tracking-frontier function that can record the visited or non-visited accepting sets in each round is implemented in DQN.

Figure 2 illustrates the Q networks' architecture, similar to a convolutional neural network (CNN). The Q networks consist of 'Convolutional' layers, 'Flatten' layers, 'Fully-Connected' layers with linear activation functions at the end to generate the outputs as the approximated Q values of individual actions corresponding to the input belief state. In this study, a considered POMDP has a two-dimensional state space, and the LTL-induced LDGBA state space provides an additional dimension. Consequently, a belief state of the product POMDP is a three-dimensional array as the input to Q networks. Compared with artificial neural networks (ANNs), CNNs can automatically detect important features without any human supervision. In our Q network architecture, there are no 'maxpooling' layers because every belief point is important, and any missing information may cause a dramatic accuracy drop in the trained network.

In DQN, two identical Q networks are initialized: the evaluation network (Q_e) and the target network (Q_t). The target network is utilized for the next action selection and Q value prediction. The evaluation network is trained every M steps by a number (i.e. batch size) of experiences $\langle b_i^{\times}, a_i^{\times}, r_i, b_{i+1}^{\times}\rangle$. After every K steps, the target network is updated by copying the weight coefficients of the evaluation networks. Using two neural networks can prevent the bootstrapping of the DQN with a single neural network. In the so-called model replay, once the random sample set $U(D)$ is collected, the new Q value is computed by the equation as [16]

$$\begin{aligned} Q_{\text{new}}(b_i^{\times}, a_i^{\times}) = {} & Q_e(b_i^{\times}, a_i^{\times}; \theta_e) \\ & + \alpha \left[r_i + \gamma \max_{a_{i+1}^{\times} \in A^{\times}} Q_t(b_{i+1}^{\times}, a_{i+1}^{\times}; \theta_t) \right. \\ & \left. - Q_e(b_i^{\times}, a_i^{\times}; \theta_e) \right] \end{aligned} \tag{20}$$

Algorithm 2 demonstrates the procedure of applying deep Q-learning with LDGBA and tracking-frontier function to approximate the Q value function of POMDP with temporal logic specifications.

Algorithm 2 Deep Q-Learning for product POMDP problems.

Require: LTL formula ϕ, POMDP $\mathcal{P}$
Convert ϕ to an LDGBA $\mathcal{A}$, initialize the frontier set $\mathcal{T} = \mathcal{F}$
Construct the product POMDP $\mathcal{P}^{\times} = \mathcal{P} \times \mathcal{A}$
Initialize the evaluation network Q_e, the target network Q_t, the learning rate α, the discount factor γ, the number of episodes E, the number of steps N, the batch size M, and the Q_t update steps K.
while The current episode e in E **do**
  Randomly select a start state $s_0^{\times}$.
  Initialize the belief state $b_0^{\times}$
  while The current step i in N **do**
    Select and perform an action $a_i^{\times}$.
    After perceiving an observation, update the belief state $b_{i+1}^{\times}$ via (10).
    Update $\mathcal{T}$ and the reward function by Equations (16) and (17).
    Calculate the rewards r_i by (13).
    Store transition $\langle b_i^{\times}, a_i^{\times}, r_i, b_{i+1}^{\times}\rangle$ in the replay memory D.
    if $i > 0$ and $i\%M$=0 **then**
      Randomly select the samples U(D) with length as M.
      while every $\langle b_i^{\times}, a_i^{\times}, r_i, b_{i+1}^{\times}\rangle$ in U(D) **do**
        compute Q_{new} by (20).
      end while
      Train Q_e by the set of Q_{new}.
    end if
    if $i > 0$ and $i\%K$=0 **then**
      Pass the weights of Q_e to Q_t.
    end if
  end while
end while
Training end and save the evaluation network Q_e

5. Simulation results

We evaluate our methodologies on two simulations with the discrete POMDP state spaces. We first carry out the simulations over a partially observable grid world. The product POMDP models are generated in Python 3.9 with Rabinizer 4 [35] and then solved by Julia SARSOP solver [11] and DQN (via Python), respectively. Furthermore, the grid-world simulation is scaled to different sizes of state and observation spaces to evaluate the scalability of the value iteration approach via SARSOP. Then, we apply the value iteration approach to a more realistic office scenario with TurtleBot2 via Pybullet 3.0 [36]. All simulations are completed on a desktop with a 3.20 GHz

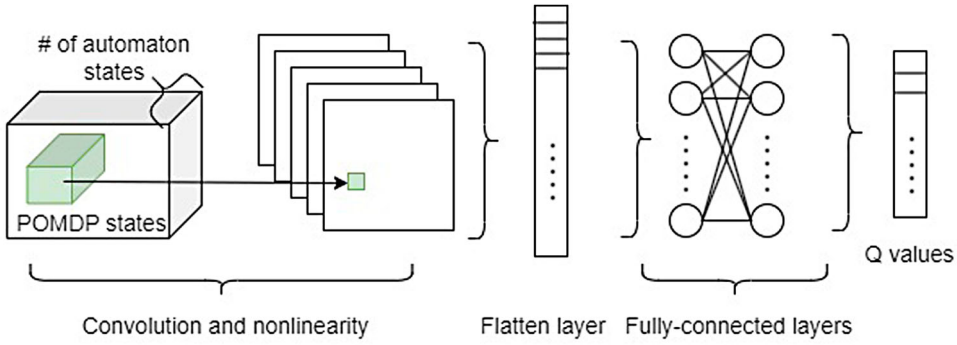

Figure 2. The Q network architecture with convolution and fully-Connected Layers.

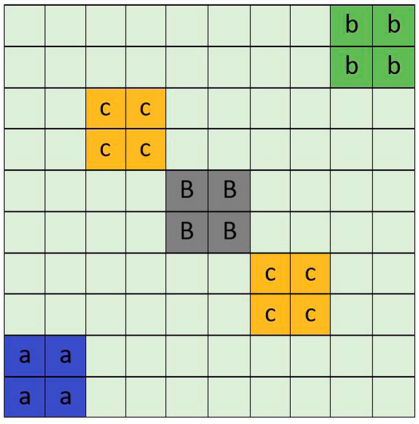

Figure 3. A 10 × 10 grid-world workspace.

eight-core CPU and 32GB of RAM. Some source codes and supplementary materials are provided.[1]

5.1. Grid-world simulation by SARSOP

Here considers a grid-world workspace with a size of 10 × 10, as shown in Figure 3. The labeled cells are marked with different colors representing different areas of interest. For example, 'a' and 'b' are labeled on the goal states that an agent may visit multiple times according to the specified task. 'c' is the label on the trapping states where an agent cannot escape once entered. 'B' represents the block states, i.e. obstacles, that the agent is not able to pass.

The agent, e.g. the mobile robot, in the grid world can take four actions at each state: *up*, *left*, *down*, and *right*. After taking each action, the agent can move towards the desired state with a probability of 0.9 and move sideways with an equally weighted probability distribution otherwise. If the next state is outside the grid world or the obstacle, the agent will stay at the current state. In addition, the agent can observe the current state with a probability of 0.9 and adjacent states with a total probability of 0.1 uniformly distributed. The obstacles (i.e. states labeled with 'B') cannot be observed. The discount factor γ is all set as 0.95.

In the grid world, the agent is required to visit states labeled 'a' and 'b' once at a time for infinitely many times, as stated in Section 3.4. The LTL formula is written as (14), and the induced LDGBA is shown in Figure 1. The POMDP environment has 100 states, and its corresponding product POMDP consists of 300 states. The set of accepting sets of the derived LDGBA is $\mathcal{F} = \{\{q_0\}, \{q_1\}\}$, and the standard rewards in (8) are assigned with the labeled states transitions to q_0 and q_1.

We first use the SARSOP solver to obtain the optimal value functions of belief states and then derive the optimal policy. Figure 4(a) illustrates an induced path from a start state, marked as a large purple solid circle. The agent moves to 'a' straightly, and the LDGBA state transits from q_0 to q_1. In Figure 4(b), it can be seen that the agent keeps visiting states 'a' since q_1 has a recurrent transition by consuming the symbol 'a' in the LDGBA (Figure 1). The agent must visit 'a' on q_1 infinite times to maximize the collected reward. Therefore, the specified task cannot be accomplished.

To address the above issue discussed in Section 3.4, we apply the redesigned reward in (15) by only assigning rewards to the transitions that lead to the accepting states from a different automaton state. Then, after solving the problem again via SARSOP and obtaining the optimal policy, we generate another path, shown in Figure 5. It illustrates a successful run in which the agent visits states 'a' then 'b' and keeps visiting them alternatively for infinite times.

Starting from the initial state, shown in Figure 5(a), the agent moves left straightly to reach the area labeled 'a', which leads the automaton transition from q_0 to q_1. A bend in the route indicates the agent pauses there for an extra time due to the action uncertainty. Figure 5(b) shows the induced path on the POMDP at automaton state q_1. The agent leaves area 'a' and moves right then up to reach the green area labeled as 'b'. Up to this point, the agent successfully completes one round, and

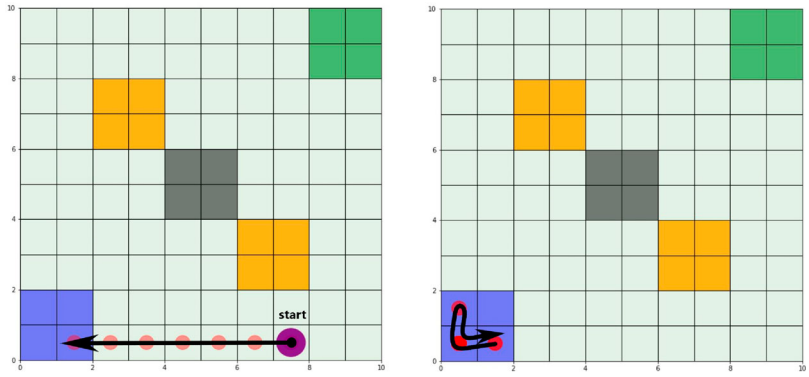

Figure 4. An induced path from the optimal policy via SARSOP using the standard reward design.

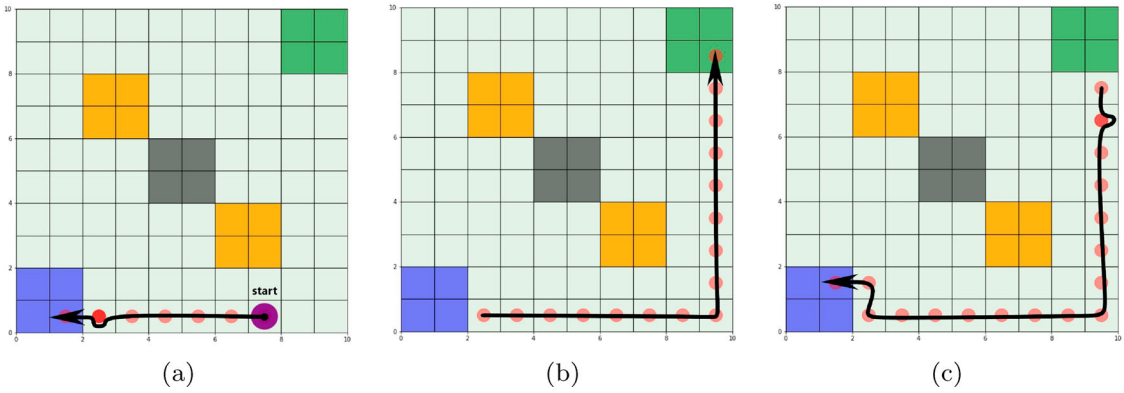

Figure 5. An induced path from the optimal policy via SARSOP using the redesigned reward.

Table 1. Simulation results for various scale workspaces.

Workspace size(cells)	POMDP states	product states	computing time(seconds)
10 × 10	100	300	4.67
20 × 20	400	1200	358.97
40 × 40	1600	4800	23,649.83

the automaton state transits back to q_0. In Figure 5(c), the agent moves from 'b' to 'a' again to start another round.

We also tested the algorithm on the same grid-world problem with various workspace sizes to show the computational complexity. The simulation results are listed in Table 1. It shows the times in seconds used to learn optimal policies.

A similar scalability study was performed in [24], in which the LTL formula was converted to DRA. They investigated grid worlds with state spaces up to 10 by 10, and the simulation time for the size of 10 by 10 is comparable to the one in Table 1.

5.2. Grid-world simulation by DQN

We tested another model-based approach using DQN with the tracking-frontier function to the same grid-world problem. The input shape for Q networks is (10 × 10 × 3). The dimensions correspond to the size of a belief product state, including the row and column numbers of the workspace in the POMDP and the number of automaton states of LDGBA, respectively.

The Q network architecture is shown in Figure 2. In this study, two convolutional layers extract the features from the input belief state. One layer uses eight (4 × 4) filters with stride (2, 2), and the other uses 16 (2 × 2) filters with stride (1, 1). The outputs are converted to a 1D array by a 'Flatten' layer. Then, two fully-connected layers with 128 and 64 neurons are used to predict the corresponding Q values. The Rectified Linear Unit (i.e. ReLU) is utilized as the activation function in the Q networks. The learning process includes 500 steps per episode for 8,000 episodes. Two Q networks, the evaluation network Q_e and the target network Q_t, are randomly initialized. The training batch size for the evaluation network is 32, and the target network is updated by copying the weight coefficients of Q_e every 50 steps. Since DQN uses Monte Carlo simulation, it is computationally intensive. The computing time for this simulation is 49450.4041 seconds.

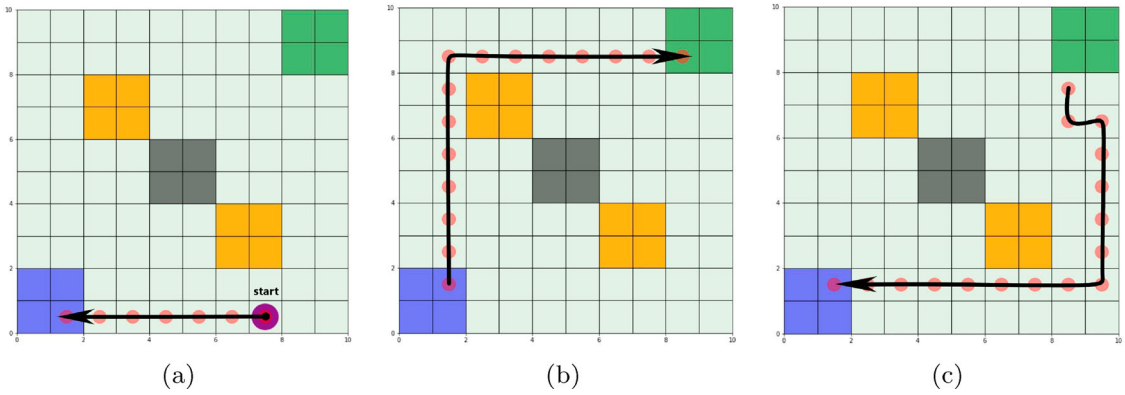

Figure 6. An induced path from the optimal policy via DQN with the tracking-frontier function.

The frontier set $\mathcal{T}$ is initialized as $\{\{q_0\}, \{q_1\}\}$. The rewards are assigned on the states labeled either 'a' or 'b', resulting in the automaton transitions to q_0 or q_1, respectively. Once the automaton state in an accepting set of $\mathcal{T}$ is visited, this accepting set will be removed from $\mathcal{T}$, and the corresponding reward is disabled too, as indicated in (16). After learning, the evaluation Q network can predict the optimal Q values for a given belief state. Consequently, the optimal policy can be derived. Figure 6 shows an induced path resulting in a successful run in that the agent recurrently visited states 'a' and 'b' once at a time for infinitely times. It shall be noted that the generated path differs from the one in Figure 5 because of action uncertainty.

5.3. PyBullet TurtleBot simulations

This section considers an office environment generated from PyBullet3.0 [36], a virtual robotic platform, as shown in Figure 7. First, we map this virtual office environment to a grid world to learn the optimal policy. Then, the policy is applied to TurtleBot2 in PyBullet, where we can implement the robotic dynamics and a PID controller to ensure the robot follows the path generated from the derived policy.

The environment has four office rooms denoted as 'a', 'b', 'c', and 'd', the storage room as 'S', the printer's room as 'Print', and the supply station for the robot to recharge noted as 'Sply'. two big windows (facing west) are located at offices 'a' and 'd'. In addition, the robot can also observe 'wall', 'hallway', and 'door'. The sensor and actuator uncertainties can be modeled as the observation and action probabilities in the POMDP abstraction, as discussed in [37, 38]. We consider two different observation settings in this example. We also assume that the TurtleBot can successfully follow its navigation controller by moving in the intended direction, with a probability of 0.9. Otherwise, the agent will accidentally move toward any other direction with the same probability (the total probability is 0.1). Moving toward the wall will keep the robot staying at the same location.

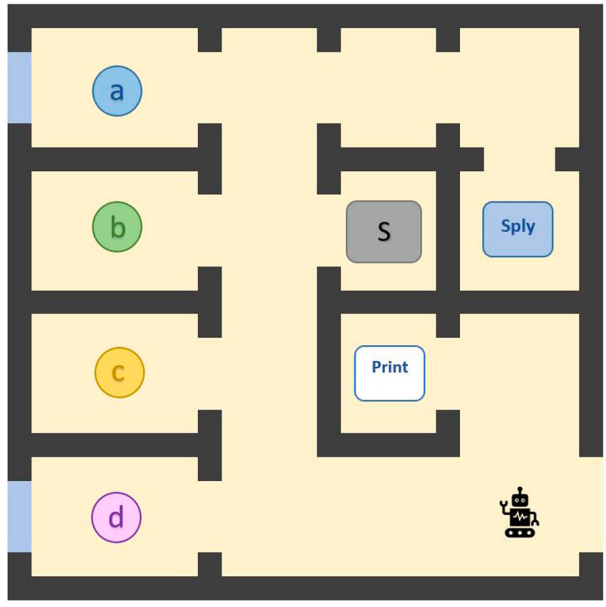

Figure 7. The office environment.

The office space is divided into a 4 by 4 grid with 16 states. There are two big windows (facing west) located at offices 'a' and 'd' that the TurtleBot (i.e. the agent) can sense by its embedded sensors. In addition, the agent can also observe 'wall', 'hallway', and 'door'. Given a specified task, we can formulate this office scenario as a product POMDP problem and apply the SARSOP solver to obtain the optimal policy. Two different observation settings are considered in this study.

5.3.1. Observation of the surroundings

In this setting, the agent can observe the surrounding information of the current state in all four directions without a specific order. There are a total of 8 observations, and each state has only one observation. For

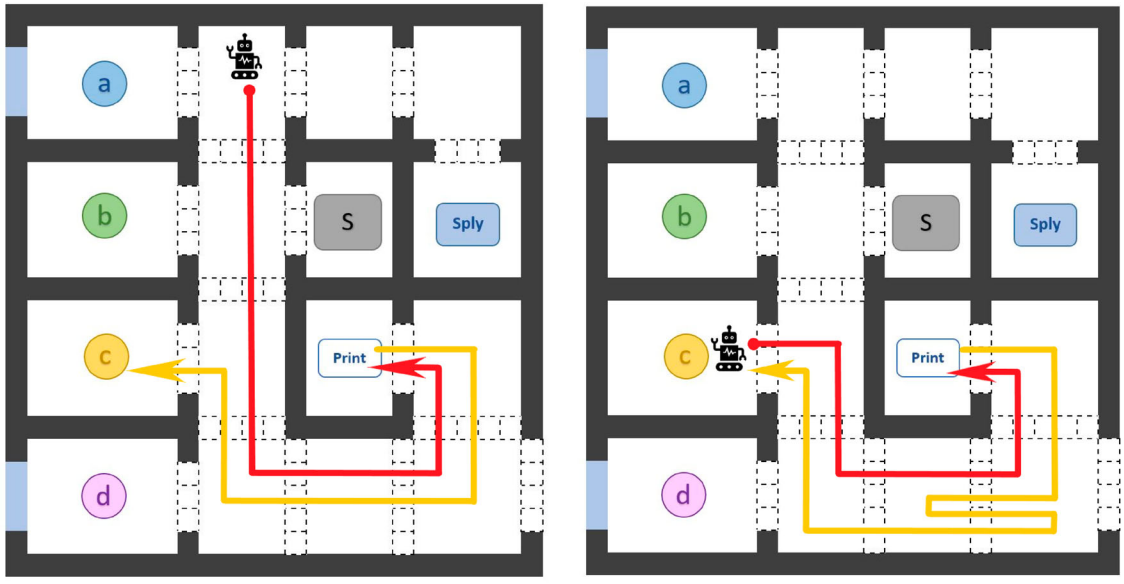

Figure 8. A path of two complete runs for Case 1 with the observations in four directions.

example, office 'b' has the observation, O('b') = 'door' 'wall' 'wall' 'wall'. Since it is the only observation at this state, its observation probability is 1.0. However, it shall be noted that office 'c' and the printer's room have the same observation as Office 'b'.

In Case 1, starting from an initial position, the TurtleBot visits the printer's room to collect the documents and then carries the documents to offices 'a' or 'c' recurrently. Meanwhile, TurtleBot shall avoid the storage room. This task can be formulated via LTL as below, and the induced LDGBA with only single-label transitions has four automaton states.

$$\begin{aligned}\varphi_{case1} &= \square(\mathit{Print} \rightarrow \bigcirc(\neg \mathit{Print}\ \mathrm{U}\ (a \vee c))) \wedge \square(\neg S) \\ &= \square\varphi_1 \wedge \square(\neg S) \end{aligned} \tag{21}$$

Figure 8(a) shows that, following the optimal policy, the TurtleBot starts from the end of the hallway (i.e. its initial position) and arrives at the printer's room first, as the red route indicates. The agent then leaves immediately after picking up the documents and heads to office room 'c', as the yellow route shows. Figure 8(b) shows that the TurtleBot leaves office 'c' and continuously completes the second round. Since TurtleBot can observe all four directions, it made the belief convergence fast, and the agent can plan the path accordingly. The path appears that TurtleBot efficiently completed the task with small action uncertainty indicated as the to-and-fro part of the yellow path.

Case 2 requires the TurtleBot to visit the supply station right after it accomplishes the delivery task in Case 1, and the storage room must also be avoided during the entire task. The LTL formula of Case 2 in (22) is extended from part of the formula (21) used in Case 1, denoted as φ_1. Different from the task in Case 1, we only require the agent to complete one run in Case 2.

$$\varphi_{case2} = \varphi_1 \wedge ((a \vee c) \rightarrow \bigcirc(\neg(a \vee c)\ \mathrm{U}\ \mathit{Sply})) \wedge \square(\neg S) \tag{22}$$

As shown in Figure 9, the TurtleBot starts from office 'b' (as the initial state), moves to the print room, and delivers the documents to office 'a'. Then, it arrives at the 'Sply' station to complete this task. Two states on the red route are visited more than once because of the action uncertainty.

5.3.2. Observation in a single direction

In another observation setting, the agent can only observe on one direction at each state it visits. The observation set is 'wall', 'hallway', 'door', 'window'. Assuming each observation has the same chance of being detected by the agent's sensors, the observation probabilities can be calculated. For example, the agent can observe 'door', 'wall', and 'window' with probabilities of 0.25, 0.5, and 0.25, respectively, in office 'a'. It can be seen that the observation uncertainty is much higher than that in the previous observation setting. Consequently, the computation time is longer for the agent to learn the optimal policies, as shown in Table 2.

Compared with the path with all four-directional observations, a single observation element provides the agent less sense of what the current state is, the increased observation uncertainty brought the difficulty of the optimal path planning. As Figures 10 and 11 show, the agent

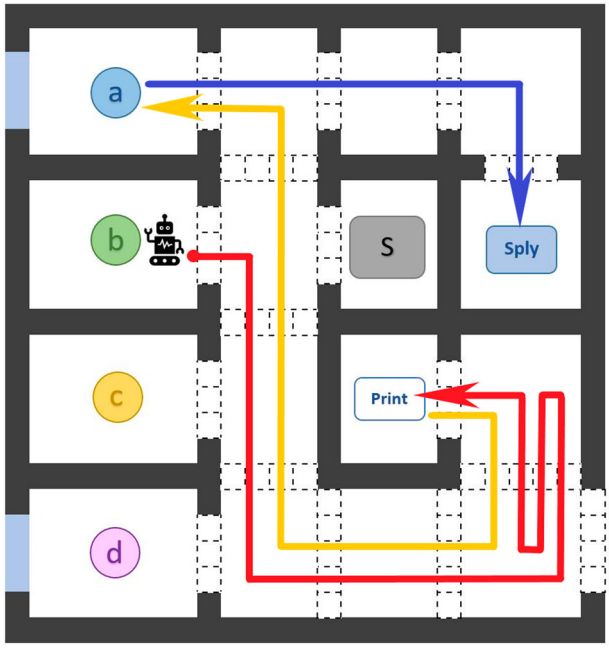

Figure 9. A path for Case 2 with the observation in four directions.

encountered difficulty in selecting the suitable actions especially around the 'hallway' states between the office rooms 'c', 'd', and printer's room, which appears as the bends and edges in the planned paths.

6. Conclusions

This paper tends to solve POMDP problems with high-level and complex tasks that can be formulated via LTL and then converted to LDGBAs. Such a motion planning problem became equivalent to finding an optimal policy on the product of POMDP and the induced LDGBA. Two model-based RL approaches are adopted to derive the optimal policy as a function of the belief state. One approach is based on the PBVI methods, and we utilize the SARSOP solver on the product POMDP to approximate the optimal value functions. In another approach, we employ a CNN architecture for Q-networks in DQN to map the relationship between a belief state and its optimal state-action values (i.e. Q values). To address the issue that LDGBA may fail to find the deterministic policy for some task specifications. We redesign the reward function and introduce a frontier-tracking function for the above-mentioned approaches, respectively. The simulations demonstrate that the agent could accomplish the specified tasks by following the derived optimal policies.

In addition, we perform the scalability study on the value iteration approach. In contrast, DQN is more computationally intensive because many episodes are needed for convergence while updating the belief state at each step. However, with the implementation of the frontier-tracking function, DQN is adaptive to various

Table 2. Time consumption of generating optimal policies in office simulations.

Cases(Observation)	Computing time[seconds]
Case 1 (four directions)	1.21
Case 1 (one direction)	18.64
Case 2 (four directions)	3.74
Case 2 (one direction)	29.83

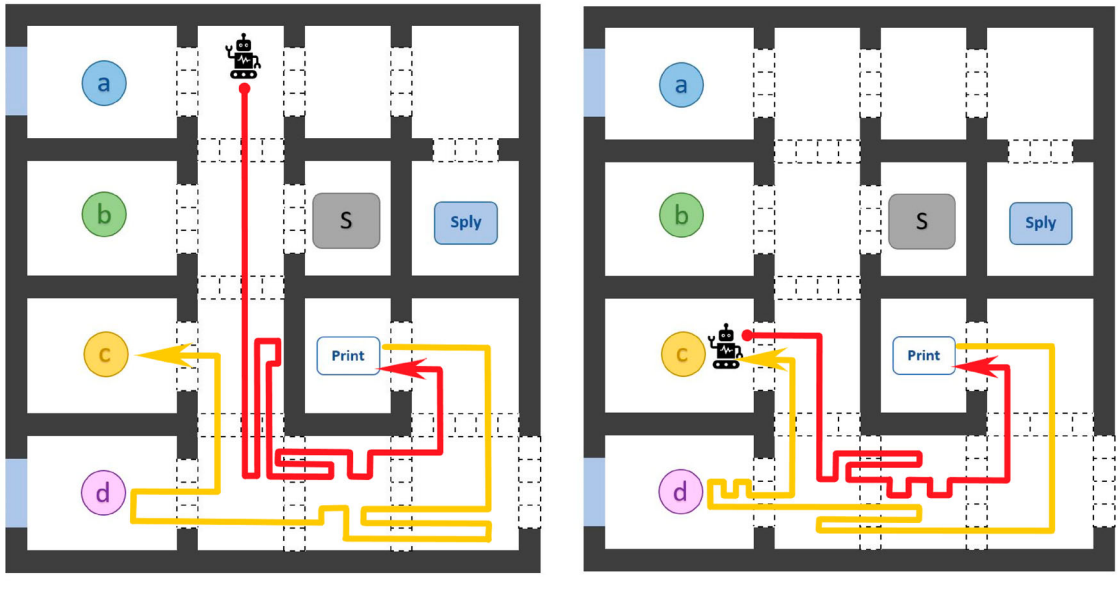

Figure 10. The path of office simulation Case 1 with the observation in a single direction.

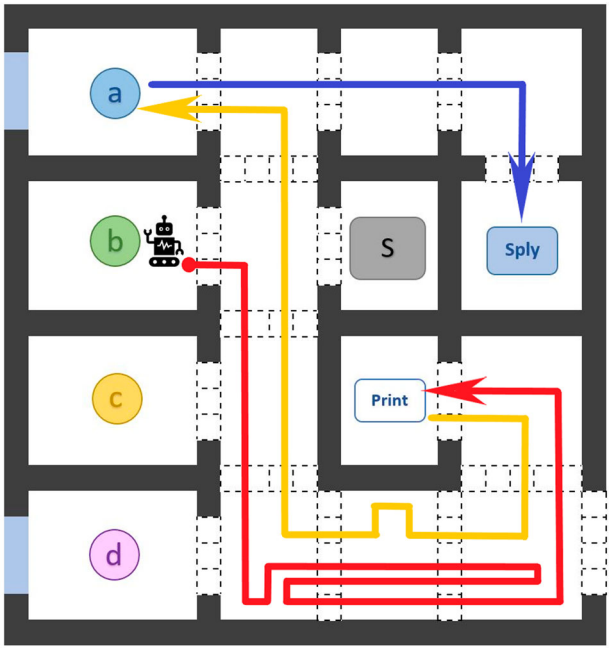

Figure 11. The path of office simulation Case 2 with the observation in a single direction.

task specifications. In future works, our proposed DQN approach can be potentially extended to a hybrid framework accommodating both model-free and model-based approaches, in which the belief state can be updated based on the approximation of transition and observation likelihoods. Furthermore, this DQN approach can also be combined with Inverse reinforcement learning (IRL) in POMDPs to use the learned Q-values as a feature to extract the rewards in problems with large state spaces. This approach has been explored in other works [39, 40] and can be further researched.

This study focuses on high-level motion planning to find the optimal policy for the agent to accomplish user-specified complex tasks. Although we use a virtual TurtleBot2 on PyBullet3.0 to validate the derived policy, it would be interesting to incorporate the internal controllers of a robot in real-world applications. Fainekos *et al.* [41] designed a closed–loop hybrid controller to integrate the continuous trajectory control to discrete path planning, which satisfied high-level task specifications via LTL. Specifically, the transitions of the agent between states in POMDP can be considered as many subtasks. For example, suppose the high-level policy outputs a discrete action, such as 'go forward' or 'turn left.' In that case, the discrete action needs to be mapped to the desired linear and angular velocities, by considering the robot's dynamics and kinematics. The low-level controllers, such as a PID controller, can send the control signals to the actuators (e.g. the motors) to drive the wheels. At the same time, the observations captured by the agent's embedded sensors serve dual purposes in the POMDP framework. Firstly, they provide feedback to the trajectory control between the states. Secondly, the observations can also be used to update the agent's belief state for the high-level decision-making process.

The proposed model-based approaches have limitations because the transition and observation probabilities must be provided to update the belief state. Model-free RL methods will be considered for POMDPs with LTL specifications in future works. In that case, the decision-making will depend on the history of observations or actions.

Note

1. https://github.com/JunchaoLi001/LDGBA-Model_Checking.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

Li, Wang, and Xiao would like to thank US Department of Education (ED#P116S210005) and US National Science Foundation NSF (#2226936) for supporting this research.

Notes on contributors

Junchao Li received his B.S.E. degree from the Department of Mechanical Engineering at the University of Iowa in 2017. He is currently a Ph.D. student in the same department and a research assistant in The Multi-scale Computational Science and Engineering Laboratory. His research interests include robotics, automation, machine learning, path planning, and decision-making.

Mingyu Cai received the B.Eng. in Aerospace Engineering from the Beijing Institute of Technology, Beijing, China in 2015, and the M.S.E. degree in Mechanical and Aerospace Engineering from the University of Florida, Gainesville, USA in 2017. He obtained his Ph.D. degree in Mechanical Engineering with the University of Iowa in 2021. He is currently a postdoctoral associate in the Explainable Robotics Lab (ERL) with the Department of Mechanical Engineering, Lehigh University, Bethlehem, USA. His research interests include motion planning and decision making employing formal methods, machine learning, and optimization.

Zhaoan Wang received a B.S.E. degree in Automotive System Engineering from Arizona State University in 2019 and an M.S.E. degree in Engineering Technology from Wayne State University in 2021. He is currently a Ph.D. Student in Mechanical Engineering at the University of Iowa. He is a graduate research assistant in the Multi-scale Computational Science and Engineering Laboratory. His research interests include machine learning, robotics, and big data analytics.

Shaoping Xiao is a professor in the Department of Mechanical Engineering at the University of Iowa. He graduated from Northwestern University with a Ph.D. degree in mechanical engineering before joining the University of Iowa in 2003. His original expertize lies in computational mechanics and materials science, and one of his papers has been cited over 1000 times. In the past several years, he has extended his efforts to artificial intelligence (AI) and its applications in engineering problem-solving. His group's current research interests include machine-learning enhanced numerical modeling of composite materials, reinforcement learning and formal methods for robotics control, AI-powered design of distributed reservoir systems to mitigate the flood, intelligent traffic lights, and quantum computing.

References

[1] Howard RA. Dynamic programming and Markov processes. Cambridge (MA, USA): MIT Press; 1960.

[2] Monahan G. State of the art-A survey of partially observable Markov decision processes: theory, models, and algorithms. Manag Sci. 1982;28(1):1–16. doi: 10.1287/mnsc.28.1.1

[3] Sutton RS, Barto AG. Reinforcement learning: an introduction. Cambridge (MA, USA): MIT Press; 2018.

[4] Kaelbling L, Littman M, Cassandra A. Planning, uncertainty, partially observable Markov decision processes. Artif Intell. 1998;101(1):99–134. doi: 10.1016/S0004-3702(98)00023-X

[5] Sondik EJ. The optimal control of partially observable Markov decision processes [PhD thesis]. Stanford, California; 1971.

[6] Cheng HT. Algorithms for partially observable Markov decision process [PhD thesis]. University of British Columbia; 1988.

[7] Littman ML. Markov games as a framework for multi-agent reinforcement learning.New Jersey.International conference on machine learning.1994.p. 157–163.

[8] Zhang NL, Liu W. Planning in stochastic domains: problem characteristics and approximation. THKUST-CS96-31. Hong Kong University; 1996.

[9] Pineau J, Gordon G, Thrun S. Point-based value iteration: an anytime algorithm for POMDPs. Proceedings of the 18th international joint conference on artificial intelligence (IJCAI'03). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2008. p. 1025–1030.

[10] Shani G, Pineau J, Kaplow R. A survey of point-based POMDP solvers. Auton Agent Multi Agent Syst. 2013;27:1–51. doi: 10.1007/s10458-012-9200-2

[11] Kurniawati H, Hsu D, Lee WS. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces.ETH Zurich, SwitzerlandRobotics: Science and systems.2008.

[12] Boettiger C, Ooms J, Memarzadeh M. Sarsop: approximate POMDP planning software. (July 6, 2022) CRAN.R-project.org/package = sarsop.

[13] Watkins CJCH, Dayan P. Q-learning. Machine learning. 1992;8(3–4):279–292. doi:10.1007/BF00992698.

[14] Egorov M. Deep reinforcement learning with POMDPs [Tech. Rep.]. Stanford, CA, USA: Stanford University; 2015.

[15] Mnih V, Silver D, Riedmiller M. Playing Atari with deep Q learning. Nips. 2013.

[16] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature. 2015;518(7540):529–533. doi: 10.1038/nature14236

[17] Baier C, Katoen JP. Principles of model checking. Cambridge (MA, USA): MIT Press; 2008.

[18] Křetínský J, Meggendorfer T, Sickert S. Owl: a library for ω-words, automata, and LTL.Los Angeles, CA, USA, Automated Technology for Verification and Analysis: 16th International Symposium.2018.

[19] Hahn EM, Perez M, Schewe S. Omega-regular objectives in model-free reinforcement learning.Prague, Czech Republic,Tools and Algorithms for the Construction and Analysis of Systems: 25th International Conference, TACAS.2019.p. 395–412.

[20] Hasanbeig M, Kantaros Y, Abate A, et al. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. Proceedings of the IEEE 58th conference on decision and control (CDC). Nlce, France: IEEE; 2019. p. 5338–5343.

[21] Oura R, Sakakibara A, Ushio T. Reinforcement learning of control policy for linear temporal logic specifications using limit-deterministic generalized Büchi automata. IEEE Control Syst Lett. 2020;4(3):761–766. doi: 10.1109/LCSYS.7782633

[22] Chatterjee K, Chmelík M, Gupta R, et al. Qualitative analysis of POMDPs with temporal logic specifications for robotics applications. Proceedings of the IEEE international conference on robotics and automation. Seattle, Washington, USA: IEEE; 2015. p. 325–330.

[23] Sharan R, Burdick J. Finite state control of POMDPs with LTL specifications. 2014 American control conference. Portland, Oregon, USA: IEEE. 2014. p. 501–508.

[24] Bouton M, Tumova J, Kochenderfer MJ. Point-based methods for model checking in partially observable Markov decision processes. In: Proceedings of the AAAI conference on artificial intelligence. New York, USA: AAAI; 2020. p. 34(6):2159–5399.

[25] Wang Y, Bozkurt AK, Pajic M. Reinforcement learning with temporal logic constraints for partially-observable Markov decision processes. arXiv preprint arXiv:2104.01612. 2021.

[26] Perez A, Platt R, Konidaris G, et al. LQR-RRT∗: optimal sampling-based motion planning with automatically derived extension heuristics. St Paul, MN, USA, 2012 IEEE international conference on robotics and automation. IEEE 2012 p. 2537–2542.

[27] Icarte RT, Waldie E, Klassen T, et al. Learning reward machines for partially observable reinforcement learningVancouver, Canada,.Advances in Neural Information Processing Systems 32 (NeurIPS 2019).2019.ISBN: 9781713807933.

[28] Lin LJ. Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach Learn. Aug. 1992;8(3/4):293–321. doi: 10.1023/A:1022628806385 Special issue on reinforcement learning.

[29] Bozkurt AK, Wang Y, Zavlanos MM, et al. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. Paris, France,.2020 IEEE international conference on robotics and automation (ICRA).2020.p. 10349–10355.

[30] Sickert S, Esparza J, Jaax S, et al. Limit-deterministic Büchi automata for linear temporal logic. Toronto, ON, Canada, Computer Aided Verification: 28th International Conference, CAV 2016. 2016. p. 312–332.
[31] Cai M, Xiao S, Li Z, et al. Optimal probabilistic motion planning with potential infeasible LTL constraints. IEEE Trans Autom Control. 2021;68:1–1. doi:10.1109/TAC.2021.3138704
[32] Cai M, Xiao S, Li B, et al. Reinforcement learning based temporal logic control with maximum probabilistic satisfaction. Xi'an, China, 2021 IEEE international conference on robotics and automation (ICRA) 2021. doi:10.1109/ICRA48506.2021.9561903
[33] Cai M, Hasanbeig M, Xiao S, et al. Modular deep reinforcement learning for continuous motion planning with temporal logic. IEEE Robot Autom Lett. 2021;6(4):7973–7980. doi: 10.1109/LRA.2021.3101544
[34] Hauskrecht M. Value-function approximations for partially observable Markov decision processes. J Artif Intell Res. 2000;13:33–94. doi: 10.1613/jair.678
[35] Křetínský J, Meggendorfer T, Sickert S, et al. Rabinizer 4: From LTL to your favourite deterministic automaton. Available from: https://www7.in.tum.de/ ~ kretinsk/rabinizer4.html.
[36] Coumans E, Bai Y. PyBullet, a Python module for physics simulation for games, robotics and machine learning. Available from: https://pybullet.org/wordpress/.
[37] Grady DK, Moll M, Kavraki LE. Combining a POMDP abstraction with replanning to solve complex, position-dependent sensing tasks. 2013 AAAI fall symposium series. 2013.
[38] Grady DK, Moll M, Kavraki LE. Extending the applicability of POMDP solutions to robotic tasks. IEEE Trans Robot. 2015;31(4):948–961. doi: 10.1109/TRO.2015.2441511
[39] You C, Lu J, Filev D, et al. Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. Robot Auton Syst. 2019;114:1–18. doi: 10.1016/j.robot.2019.01.003
[40] Lv H, Qi C, Song C, et al. Energy management of hybrid electric vehicles based on inverse reinforcement learning. Energy Rep. 2022;8:5215–5224. doi: 10.1016/j.egyr.2022.03.176
[41] Fainekos GE, Kress-Gazit H, Pappas GJ. Hybrid controllers for path planning: a temporal logic approach. Seville, Spain.Proceedings of the 44th IEEE Conference on Decision and Control.2005. doi:10.1109/CDC.2005.1582935