Robust Reinforcement Learning for Linear Temporal Logic Specifications with Finite Trajectory Duration

Soroush Mortazavi Moghaddam[†, *], Yash Vardhan Pant[†, *], Sebastian Fischmeister[†, *]
[†] Electrical and Computer Engineering, University of Waterloo, Canada, ON

Abstract

Linear Temporal Logic (LTL), a formal behavioral specification language, offers a mathematically unambiguous and succinct way to represent operating requirements for a wide variety of Artificial Intelligence (AI) systems, including autonomous and robotic systems. Despite progress, learning policies that reliably satisfy complex LTL specifications in challenging environments remains an open problem. While LTL specifications are evaluated over infinite sequences, this work focuses on solving objectives within a given finite number of steps, as is to be expected in most real-world applications involving robotic or autonomous systems. We study the problem of generating trajectories of a system that satisfy a given LTL_f specification in an environment with a priori unknown transition probabilities. Our proposed approach builds upon the popular AlphaGo Zero Reinforcement Learning (RL) framework, which has found great success in the two-player game of Go, to learn policies that can satisfy an LTL_f specification given a limit on the trajectory duration. Extensive simulations on complex robot motion planning problems demonstrate that our approach achieves higher success rates in satisfying studied specifications with time constraints compared to state-of-the-art methods. Importantly, our approach succeeds in cases where the baseline method fails to find any satisfying policies.

Keywords: Reinforcement Learning (RL), Monte-Carlo Tree Search (MCTS), Linear Temporal Logic (LTL), Planning, Robotics

1. Introduction

LTL provides a concise and expressive language to formally express objectives, constraints, and temporal requirements in a broad range of applications, including autonomous systems. That is why it has become a popular choice for high-level task specification for agents in many AI domains for tasks such as planning, robot control, and navigation, especially in recent years [1–4]. However, synthesizing policies for these systems can be difficult for complex specifications or challenging environments. Recently, RL-based methods have demonstrated promise in solving such cases [1, 2, 4, 5] (more details in Sec. 2).

The semantics of LTL are designed with the assumption that the system in question will produce an infinite trace of states, yet that might not be realistic in practice. When dealing with physical systems, such as robots, it is reasonable to assume access to limited resources. Taking action in the real world takes energy and time and causes wear to the equipment. In the current work, time is considered a limited resource when synthesizing policies by limiting the trajectory's maximum duration. To evaluate a finite trajectory against an LTL specification, a variant language called LTL_f [6] can be used. LTL_f uses the same syntax as LTL, but its semantics assume interpretations are done over finite traces.

Among the frameworks in the domain of Machine Learning (ML), RL offers a potential framework for learning policies that can satisfy complex tasks specified as LTL_f objectives, acting as the *reward function* for the RL algorithm. Despite its advantages, constructing reward functions (in the context of RL) using LTL_f remains a difficult problem. The primary challenge in this setup stems from the lack of an immediate outcome in evaluating an LTL_f formula over states in a trajectory, resulting in *sparse rewards*. LTL_f evaluation

[*] smortaza@uwaterloo.ca (corresponding), yash.pant@uwaterloo.ca, sebastian.fischmeister@uwaterloo.ca

produces a boolean value indicating whether the given trajectory σ - a history of states - satisfies the LTL_f formula or not. With this interpretation, a positive reward can only be achieved by making no mistakes throughout the entire run. Conversely, not satisfying the specification provides no information about the extent of the violation or when it occurred. As RL approaches typically rely on a notion of reward propagation to produce their policies, dealing with sparse rewards often proves challenging [7]. A common approach [1, 3, 4] to deal with sparse rewards is to utilize a more informative construct indicating the degree to which the specification has been satisfied, e.g., via Limit Deterministic Büchi Automata (LDBA), Deterministic Rabin Automata (DRA), Reward Machines (RMs), etc.

Example 1.1. *Consider Fig. 1 inspired by the craft-worlds in [2]. The objective of the robot is to collect wood and iron (in order) and take them to the factory to build a bridge. This task can be formally written at a high level in* LTL_f*:* $\Diamond(wood \wedge \Diamond(iron \wedge \Diamond factory))$ *or in English: Eventually gather wood, and then eventually gather iron and then eventually visit the factory. (see Sec. 3.1 for details on the syntax of* LTL_f*)*

Another challenge in this domain is the substantial increase in the required number of samples for accurate reward propagation and estimation of states, particularly over complex specifications and extended horizons. Consequently, a sampling method such as Monte-Carlo Tree Search (MCTS) greatly influences both the efficiency of policy learning and the performance of the learned policy. MCTS has historically demonstrated its effectiveness in highly intricate problems with vast state spaces ([8] goes over a comprehensive list). When combined with a guiding heuristic like a neural network, MCTS has proven capable of remarkable performance in numerous tasks [9] [10]. Drawing inspiration from how AlphaGo Zero [11] uses the same technique, we have crafted our solution for satisfying LTL_f objectives (such as the one in Example 1.1) in potentially stochastic environments (like Fig. 1) described via an MDP.

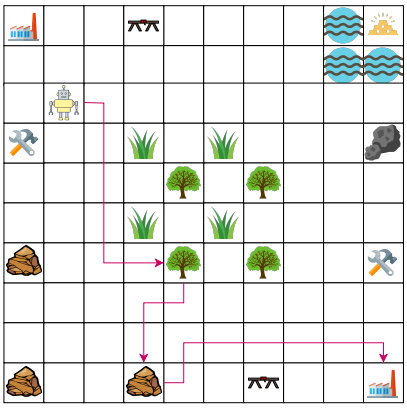


Figure 1. An example of a task in the craft-world (see Sec. 5.1.2 for more details) and a possible trajectory that satisfies it. More details of the the task in Example 1.1.

Contribution: The main contribution of this work is a proposed model-free Monte-Carlo-based RL approach for learning policies to satisfy LTL_f specifications within a given number of time-steps. Through extensive simulations, we show it can learn robust policies for problems characterized by complex formulas, even in stochastic environments with multiple unsafe regions and a high risk of violating the specification, all while requiring no knowledge of the MDP that defines the environment dynamics. In these challenging cases, our approach outperforms state-of-the-art methods that fail even in the deterministic setting.

Outline of the paper: Sec. 2 discusses the related work in this domain. After that, the preliminaries and the problem formulation are presented in Sec. 3. The details of the proposed method are then discussed in Sec. 4. Subsequently, Sec. 5 shows the results obtained from three scenarios, including comparisons with another RL-based state-of-the-art approach. Finally, Sec. 6 concludes and summarizes this work in a brief discussion.

2. Related Work

Policy synthesis for systems with LTL specifications has been well studied [12–16] however, there remains the challenge of generating policies that satisfy complex specifications in large or stochastic environments such as the ones considered in this work (see Sec. 5.1). Here, the discussion is restricted to RL-based methods dealing with LTL specifications.

In [1], the authors frame the task of synthesizing control policies for LTL specified problems as a model-free, Q-learning problem. it involves integrating the environment's MDP and an LDBA corresponding to the given LTL formula to construct a product MDP. Similarly, [5] tackles the problem of satisfying the LTL formula as a two-player zero-sum game between the agent and the environment. However, the environment can dynamically react to the agent's actions in that setting which is not the case in the current work where only a non-reactive (but possibly stochastic) environment is considered. Both [1] and [5] rely on the propagation of rewards in the product system based on the Bellman equations, which suffers from slow reward propagation if the corresponding product system lacks a sufficient number of states with positive rewards or if those states are unlikely to be reached (like the sequential delivery case study in Sec. 5.1). Moreover, similar to most of the work in this domain, [1] and [5] assume an infinite time horizon for solving their tasks. This leads to lower success rates for finite runs compared to this work, despite the optimality of the solution for trajectories of unbounded duration.

Authors of [17] introduce a new method for creating off-policy data from on-policy runs to be used via Q-learning to learn policies for a broad set of applications. As demonstrated in Sec. 5.2, Q-learning-based approaches for LTL specification policy synthesis can struggle to scale as environments become challenging and stochastic. Furthermore, [17] is prone to learning instability, especially with complex specifications, as it relies on counterfactual experience replay, meaning it generates data that is possibly not valid for its training.

Decomposing the LTL task into sub-tasks has been explored in [18] where multiple co-safe LTL tasks are taught to an RL agent. Co-safe LTL is less expressive than LTL_f. Moreover, [18] suffers from an exponential growth of the number of decomposed sub-tasks and Q functions it needs to learn, which can quickly become a problem if the given specifications consist of multiple conjunct parts. The problem becomes more profound if the Q functions are implemented as neural networks as their training could get computationally infeasible.

To tackle the sparse rewards problem, [3] and later [19] proposes Reward Machines (RMs), that form the basis of their proposed Q-learning for Reward Machines (QRM) framework. Aside from not taking time limits into account, the construction of reward machines is a manual task and requires expert knowledge to build suitable machines for any given specification. The current work exploits product systems that can be built automatically with no expert knowledge. Furthermore, QRM does not generalize by default to stochastic environments for the same problem it was meant to solve in the deterministic setting.

Unlike the other works discussed, the work in [20] proposes a way to take into account the MDP cost associated with the movement of the robot while trying to achieve optimal solutions for a LTL task. This is similar to our assumption of limited resources when devising a solution. Their approach assumes access to a lower bound on the transition probabilities in the underlying MDP, but the current work does not require such information.

Authors of [21] use the compositional syntax and the semantics of LTL to learn policies that generalize to previously unseen specifications during training. They encode LTL formulas into Graph Neural Networks and offer a general framework to deal with new randomly generated specifications. In their case studies however, they do not use the full grammar of LTL, and use a context-free fragment of it which is less expressive than LTL_f. For example, the office-world task studied here (Sec. 5.1) cannot be expressed in their setup.

Using a notion of time windows to address loop requiring specifications in finite time steps, [22] proposes a new task-specification language: Geometric LTL. Their case studies

are much simpler than the ones studied here, both in the complexity of the environment (size, number of items in the world, and the amount of stochasticity) and, the LTL specification.

Recently, [4] proposed the use of trajectory transformers to satisfy LTL_f specified tasks simulated in mini-grid environments. It uses deterministic finite automata (DFA) to deal with sparse rewards. However, the LTL_f specifications and the environments tested in their work are significantly simpler than the ones considered here.

While avoiding the pitfalls of the approaches discussed above, we propose a model-free RL method that can deal with any LTL_f specification. Through extensive simulations, we show our approach is scalable to a wide variety of complex problems and can learn robust policies to solve these tasks in potentially stochastic environments with a priori unknown transition probabilities.

3. Prerequisites and Problem Formulation

This section briefly discusses the building elements of our problem setup and then formally defines the problem formulation.

3.1. Prerequisites

Markov decision process (MDP): The environments discussed in this work are modeled as (potentially stochastic) MDPs, with their transition probabilities not directly available to the agent. A Markov decision process (MDP) is defined as a tuple (S, s_0, A, P, R) where S is a finite set of discrete states, $s_0 \in S$ is the initial state, A is a finite set of actions, P is the transition probability function, $P(S, a, S') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ mapping each state-action pair to a next state with certain probabilities, and $R : S \times A \times S \to \mathbb{R}$ is a reward function. Given a set of *atomic propositions* AP, where each member represents a statement about the state of a system (like visiting a factory in Example 1.1), an MDP could be augmented by a *labeling function* $L : L_{ap}(s) \to \{True, False\}$, mapping each atomic proposition ap to a boolean value given a state s of the MDP. By adding L and AP to the MDP it becomes a *labeled MDP*. The labeling function is needed for evaluating LTL_f formulas over sequences of MDP states as explained in the LTL_f description below.

Linear Temporal Logic over Finite Traces (LTL_f): LTL_f [6] is a variant of the formal language LTL [23] used to express temporal constraints for a system by providing a set of logical operators. From a syntactic perspective, LTL_f formulas are written as a combination of atomic propositions with Boolean and temporal operations. The Boolean operations include: True ($\top$), False ($\bot$), negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), implication ($\Rightarrow$). The temporal operations include: Always ($\Box$), Eventually ($\Diamond$), Next ($\bigcirc$), and Until ($\cup$). The grammar of this language can be represented in Backus-Naur form as shown below. A formula φ can be created as:

$$\varphi ::= ap \mid True \mid \varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \cup \varphi_2$$

The rest of the operators ($False, \vee, \Diamond, \Box, \Rightarrow$) can be derived via this grammar (see [23]).

A trajectory of visited labeled MDP states can be evaluated against an LTL_f formula by checking the sequence of observed labels (true or false) for each ap against the LTL_f formula (see [6] for details). This evaluation will generate a boolean reward. As an example, the red trajectory in Fig. 1 satisfies the LTL_f formula in Example 1.1 and would produce a reward of one. The current work uses these boolean rewards instead of an immediate reward (from a reward function R in the MDP definition) when learning its internal estimates of how good a state is when training the network (Sec. 4.1) and during MCTS (Sec. 4.4).

Limit Deterministic Buchi Automata (LDBA): LDBAs provide a way to represent an LTL formula via a transition system where states encode information about the truth of sub-formulas of the LTL formula and transitions capture how these truths evolve given

a sequence of inputs. In our case, these inputs are the true observed MDP labels during a trajectory. An LDBA can be defined by a tuple $(Q, q_0, \Sigma, \delta, F)$ where Q is a finite set of the LDBA states and $q_0 \in Q$ the initial state, Σ is a finite set of the LDBA alphabet, δ is a transition function $Q \times \Sigma \to 2^Q$, and F is a set of acceptance states. Further details of the LDBA structure are available in [24].

Product Markov decision process (MDP): The product MDP can be seen as the Cartesian product of an LDBA and an MDP. The product MDP states include information about both the MDP state and the LDBA state of the LTL$_f$ specification. Given a labeled MDP and an LTL$_f$ formula along with its corresponding LDBA, the product MDP can be constructed as $(S^*, s_0^*, A^*, P^*, F^*)$ where $S^* = S \times Q$ is the set of all possible pairs of MDP and LDBA states. s_0^* is the initial state of the product MDP, A^* is the MDP action set augmented with the LDBA's *epsilon* actions (see [24]), F^* is a set of accepting states $\in S^*$, and P^* is a transition function mapping a state pair $s_i^* = (s_i, q_i)$ to another $s_j^* = s(_j, q_j)$ with a certain probability for an action a^* and LDBA label input $(a^*, L(s_i))$. Details of the construction of the product MDP are available in [25].

3.2. Problem formulation

Problem 1. *For an environment represented via a labeled MDP M with a priori unknown transition probabilities and its initial state s_0, an LTL$_f$ formula φ describing the task specification, and a time horizon T: find a sequence of labeled MDP states $\sigma = \{s_0, s_1, \ldots, s_{T-1}\}$ such that their corresponding observed labels satisfies φ.*

The following section explains how the proposed approach solves Problem 1.

4. Proposed methodology: solving LTL$_f$ tasks via RL

The RL framework in this work uses a combination of Monte-Carlo rollouts (i.e., trajectory samples from a given state) and a neural network to select actions that lead to a higher probability of satisfying the task. Overall, the network acts as a guide for MCTS, helping it focus on promising actions so it requires fewer samples. MCTS on the other hand, helps train the network by providing improved policies for each state and provides the agent with policies at each step to choose its actions. These two components and their adaptation to solve Problem 1 are explained in this section along with Fig. 2 illustrating an overview of the proposed framework. Alg. 1 provides these steps in more detail.

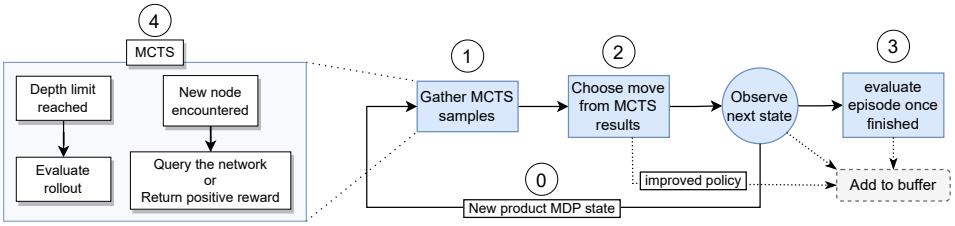


Figure 2. The overall procedure of how the proposed method works (see Sec. 4). At each state, actions are sampled from the improved policies gathered via MCTS, then used to train the network which in turn helps the MCTS with better rollouts later on.

4.1. Neural network for policy and value function estimation

The actions MCTS chooses during its rollouts are guided by a neural network. The network takes as input an encoded product MDP state, $s^* \in S^*$ as a multi-channeled matrix,

passes it through multiple Conv2D and Dense layers and outputs: i) a probability distribution $\mathbf{p}$ over the set of actions, as the *prior* policy for unexplored nodes in MCTS and ii) a real value v indicating the probability of satisfying the task from s^*. Thus, unlike AlphaGo Zero our network does not require a history of visited states as input. The corresponding loss function of the network (from [11]) for a given input s^* is:

$$L(\text{encoded}(s^*)) = (r_{\text{LTL}_f} - v)^2 - \pi^T \log \mathbf{p} + c||\theta||^2 \tag{4.1}$$

where $v \in (0, +1)$ is the network's predicted value and, $\mathbf{p} \in \mathbb{R}^{dim(A^*)}$ is the network's suggested policy. $\pi : S^* \to A^*$ is the improved MCTS policy (see next subsection for details). r_{LTL_f} is the Boolean reward of the entire episode (stage ③ in Fig. 2). The last term is the L2 regularization for the network's parameters θ and its coefficient c, which determines the amount of regularization when training the network weights[1].

4.2. MCTS for policy improvement

In this work, the agent and its internal MCTS explore the same product MDP state space. MCTS maintains a tree where nodes represent the product MDP states, and edges are possible transitions to the next states. The results of MCTS rollouts from each state s^* are kept in its corresponding node in a set of parameters. These parameters include $N(s^*)$: visit counts of the node's children, $W(s^*)$: accumulated backed-up rewards through the node's children, $Q(s^*) = W(s^*)/N(s^*)$: average backed-up rewards through the node's children and, $P(s^*)$ the *prior* policy for the node. Whenever MCTS encounters a new node, it queries the network to get this prior, possibly along with a value (see Sec. 4.4 for details) and updates the rest of the parameters as it gathers more samples (more details in [11]).

As described in Alg. 1, at each step of the episode with the agent in state s_t^* (stage ⓪ in Fig. 2), S number of MCTS rollouts are performed (stage ①), which result in a probability distribution over the possible actions (i.e., a policy) π_t for that state. This policy is usually better than the prior policy suggested by the network for s_t^* and so, the agent samples from this policy at the t^{th} step to choose an action (stage ②), and keeps it in a buffer as the target to train the network's policy output for s_t^* based on Eq. 4.1.

4.3. From the game of Go to LTL$_f$ objectives

Our proposed method draws inspiration from AlphaGo Zero [11], but differs from it since satisfying an LTL$_f$ task fundamentally differs from the game of Go. The main differences and our adaptation of the MCTS (component ④ in Fig. 2) are described below.

Revisiting states within a run: A fundamental difference between Go and instances of Problem 1 is that in each episode of a game of Go, no state is visited more than once. This is not generally true, e.g., in a robot navigation task the agent could revisit the same states. The following issue explains why this makes our agent prone to exploring repetitive states that do not advance the product MDP state in any meaningful direction. Originally, AlphaGo Zero relies on a variant of the PUCT algorithm [26] to balance exploration-exploitation in its MCTS search. But, due to binary evaluations over finite traces, if MCTS fails to find a satisfactory trajectory after gathering some samples, revisiting the same states over and over will have the same result as exploring the environment with no success since both are returning the same rewards. Early experiments confirm this issue as MCTS eventually ends up oscillating between a few repetitive states, not learning anything new. This fact suggests MCTS as implemented in AlphaGo Zero must be modified to solve instances of Problem 1, as it makes exploring the product MDP state space a challenge.

[1]More details about the network architecture can be found at rrl-ltl.github.io/RRL-LTL

Lack of a second player: Another difference between Go and the problems studied here is that Go can be played in a self-play framework. In this framework, when their agent loses a game, the opponent (which is the agent itself) has won. So their agent can *always* learn what to do to win a game. In our setup, however, only receiving zero rewards (i.e., not solving the task) is insufficient to guide the agent towards satisfying the LTL_f specification. The self-play framework also provides the AlphaGo Zero agent with an opponent that gradually improves along with it. This does not apply to our problem setup, making the learning problem more challenging.

4.4. **Adapting MCTS for LTL_f objectives**

The following describes some of the main modification to MCTS for solving Problem 1.

(1) When encountering a new state (leaf node): If the agent has satisfied the specification so far during training, query the network for a predicted value. If not, return a positive reward.
(2) If the depth limit is reached: evaluate the trajectory, if it satisfies the specification, return a positive reward, else return zero.

Regarding the first point, if the agent has not seen any satisfying trajectories during training, the network has not seen any meaningful positive rewards yet. So the network will output arbitrary values which do not provide helpful guidance for MCTS. Therefore, until at least one satisfactory trajectory has been generated, MCTS will return a constant positive value (less than one) upon visiting a new node, which encourages visiting new states. Upon satisfying the specification once, the algorithm switches to consult the network for its suggested values. This strategy mitigates the challenge of exploration mentioned above. The second point states how MCTS evaluates a full rollout. The positive reward that MCTS backs up is more than or equal to one. There are two reasons for this. First, this reward must be greater than the positive reward of finding a new node. Also, as mentioned when discussing the lack of a second player, our approach critically relies on MCTS finding a satisfactory rollout at some point. Therefore, once such a rollout is found, its reward must be significant enough to ensure the same trajectory will be sampled again.

Algorithm 1 generating a trajectory to satisfy a LTL_f formula

Input: Trajectory duration T, The product MDP, LTL_f formula φ, labeling function L, the network net(θ), MCTS #samples S, and search_depth
Initialize Q, N, P, W, with zero, and with appropriate dimensions
Initialize σ, Π as empty lists ▷ Observed MDP states and their MCTS policies
Sample s_0 and append it to σ ▷ s_0^* is initialized based on s_0
for each t $\in \{0, ..., T\}$ **do**
  $\pi_t \leftarrow$ MCTS(net(θ), s_t^*, φ, Q, N, P, W, S, search_depth)
  $a_t^* \leftarrow$ sample from π_t
  $s_{t+1}^* \leftarrow$ observe from $P(a_t^*, L(s_t))$ ▷ Next product MDP state
  append s_{t+1} to σ, and π_t to Π
end for
Calculate r_{LTL_f} by evaluating σ against φ

- s_t^* is the product MDP state of the t^{th} step, while s_t is the corresponding MDP state

4.5. Training and policy extraction

During training, generated trajectories (sequences of $\{s_0^*, s_1^*, ...\}$ in Alg. 1) with their rewards (r_{LTL_f} in Alg. 1) and sequences of MCTS policies (Π in Alg. 1) are kept in a buffer and the network's weights are updated on a number of random samples after each run.

Alg. 1 is computationally heavy and can be slow at run-time. Thus we propose a simple method to extract a policy via Eq. 4.2 from the MCTS parameters after training.

$$\pi(s^*) = \text{argmax}_a \, N(s^*) \tag{4.2}$$

Training stops once the agent can fully solve the task from all valid initial states or after a maximum number of iterations. Deploying the agent with this policy is computationally much more lightweight and allows testing our method across a wide range of simulation-based experiments, as presented in the next section.

5. Simulation Studies

In this section, the proposed approach, i.e., the extracted policy from Eq. 4.2 after training the agent via Alg. 1 is compared to a state-of-the-art Q-learning based method, CSRL [1], via three case studies. We also include the results of another approach, i.e., Q-learning with counterfactual experiences for RMs (CRM) [19] in the case that was brought from their work. Initial experiments showed that the extracted policy (Eq. 4.2) has similar performance to that of Alg. 1 after training while being computationally much more tractable. All experiments were conducted on a desktop computer with an AMD Ryzen 7 6800H (3,2 Ghz, 8 core, 16 thread) CPU, RTX 3070ti GPU, and 16 GB of RAM. The implementations were carried out using Python, and the code is available at https://github.com/CL2-UWaterloo/RL-LTL.

5.1. Simulation setup

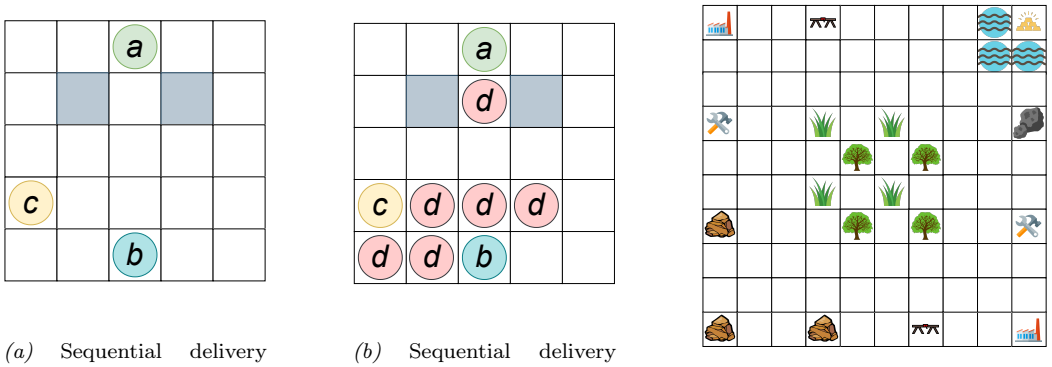
The dynamics of the environments are the same in all cases. The robot has four (MDP) actions to choose from: going up, right, down, or left. The robot may or may not have access to (LDBA) *epsilon* actions, depending on the LDBA's structure. Taking an ϵ-action only changes the LDBA state of the product MDP, and not the location of the robot in the environment (i.e., the MDP state). The environments can be deterministic or stochastic. In the deterministic case, the robot always moves in the desired direction. In the stochastic environment, the robot has a 0.8 probability of moving in the intended direction. There is a 0.1 probability it will move to the left and a 0.1 probability it will move to the right of the intended direction instead. Upon collision with a wall or an obstacle (grey areas), the robot will stay at its location. We report *success rates*, i.e., the probability of satisfying the task starting from a random initial position. All reported success rates are measured by running 1000 test runs. The robot did not start any runs from cells where it was impossible to satisfy the formula. Next, the case studies are discussed.

5.1.1. Case study 1 - sequential delivery with varying number of unsafe zones:

In this case, the robot must navigate in the environment shown in Fig. 3a which can have up to 6 unsafe zones d (Fig. 3b), such that it satisfies the following LTL_f formula:

$$\varphi_{\text{\,sequential delivery}} = \Box\neg d \wedge (\neg c \cup b) \wedge (\neg b \cup a) \wedge \Diamond\Box c \tag{5.1}$$

In other words, avoid unsafe zones d at all times, first visit a and then visit b and then visit c only in that order. The agent must not visit these labels in any other order, otherwise, it has failed. As the number of unsafe zones increases, this particular case presents a high chance of violating the specifications. As shown in Sec. 5.2, once the number of unsafe zones crosses a certain threshold, Q-learning (CSRL) fails to find a solution to the problem.



(a) Sequential delivery case: zero unsafe zones

(b) Sequential delivery case: six unsafe zones

(c) The craft-world

Figure 3. Environmets for the sequential delivery case (Fig. 3a), with up to six unsafe zones (Fig. 3b). The craft-world (Fig. 3c) inspired from [2].

5.1.2. **Case study 2 - Craft-world:**

In [2] ten tasks are defined for a craft-world and they are brought here as LTL$_f$ objectives[2]. The tasks include gathering resources to build tools and gaining access to the gold or extracting the gem from the ore[3]. In this case study, CSRL cannot be tested on the ninth task due to the limitations of its implementation. The ninth task, which is the hardest task of all, involves gathering iron and wood (in any order) and taking it to a factory to build a bridge, and then crossing the river to reach the gold (see Eq. 5.2).

$$\begin{aligned}\varphi^{(9)}_{\text{get gold}} = \Diamond((iron \wedge \Diamond(wood \wedge \Diamond(factory \wedge \Diamond gold)))\vee \\ (wood \wedge \Diamond(iron \wedge \Diamond(factory \wedge \Diamond gold)))) \wedge \Box(\neg river \vee bridge)\end{aligned} \tag{5.2}$$

5.1.3. **Case study 3 - Office world from [19]:**

In this case, the robot must navigate in the office world environment shown in Fig.4 such that it satisfies the following LTL$_f$ formula:

$$\varphi_{\text{office-world}} = \Box\neg d \wedge \Diamond((m \wedge \bigcirc\Diamond(c \wedge \bigcirc\Diamond o)) \vee (c \wedge \bigcirc\Diamond(m \wedge \bigcirc\Diamond o))) \tag{5.3}$$

or in English, avoid decoration locations d at all times, and end up at o after visiting c and m in any order. The office-world is brought from [19][4] along with the hardest task they defined on it (Eq. 5.3).

5.2. **Simulation results and discussion:**

Sequential delivery case with differing number of unsafe zones (Eq. 5.1): In the deterministic case, our approach fully solves the task in all seven environments within 30 steps. As a reference, the minimum number of steps required to solve the task from any position is 25. Q-learning baseline (CSRL) only manages to solve the first four (up to three unsafe zones) and fails to find a solution once four or more unsafe zones are added, for any trajectory length. In the stochastic case, average success rates are shown in Fig. 5a. In this plot, the agent's trajectory length is fixed at 50. In this comparison, CSRL was training

[2]Our agent does not have the special *use* action that was available to the agent in [2].
[3]See rrl-ltl.github.io/RRL-LTL for the complete list of craft-world tasks, and sample playbacks of trajectories generated for different cases.
[4]The office world from [19] was implemented with extra cells on each axis to recreate walls.

for three million iterations for each environment, allowing for two orders of magnitude more training time than our approach needs to solve the problems. The reason why CSRL cannot solve three of the cases is possibly due to the LDBA accepting states being too hard to reach. Q-learning relies on the propagation of the positive reward of these accepting states. As the number of unsafe zones increases, it becomes more unlikely that these states are found. As a result, its associated reward never propagates for the Q function to learn. However, MCTS coupled with a neural network can still find satisfying trajectories in such scenarios.

The craft-world tasks (Sec. 5.1.2): In the deterministic case, our approach fully solves all ten tasks within 40 steps. As a reference, the minimum number of steps required to solve the hardest task from any position is 29. CSRL cannot be tested on the ninth task (Eq. 5.2) due to its implementation but solves the rest within 40 steps. In the stochastic environments, our approach eventually achieves perfect success rates given enough time steps and so does CSRL on nine of the tasks. As an example, Fig. 5c shows our average success rate for the ninth task (Eq. 5.2) with varying trajectory durations[5].

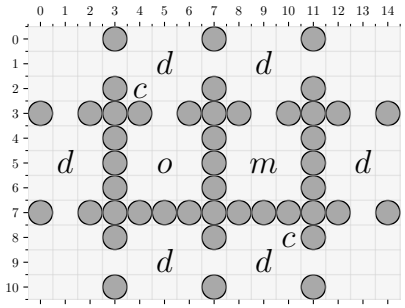
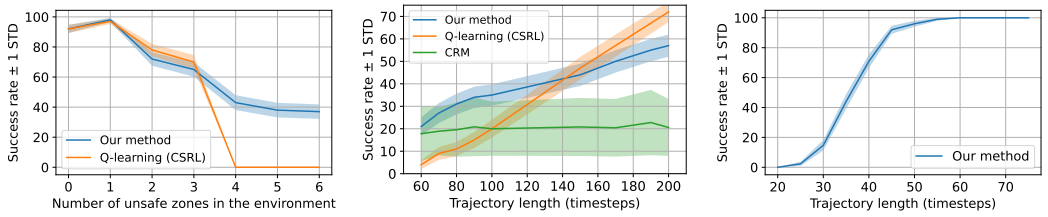


Figure 4. Environment for the office-world (see Sec. 5.1.3)

The office-world task (Eq. 5.3): In the deterministic case, both our approach and CSRL fully solve the task within 70 steps. In the stochastic case, our approach solves the task with a higher success rate when the agent is given a short trajectory length to solve the task and, CSRL outperforms our method in long trajectories (see Fig. 5b). As mentioned in Sec. 2, CSRL assumes an infinite horizon when learning its policies, and therefore its performance degrades when given short trajectory lengths to solve the task. But when allowed to run a long trajectory, its policy can eventually solve the case. CRM [19] (source of this case study) shows they solve this task in the deterministic case, starting from one initial state. Here, we tested their method in the stochastic environment from random initial states and reported the average results across 60 independent runs (as they do in their work). As seen in Fig. 5b, CRM, with the reward machine they designed for the task, does not offer the same performance as it did in the deterministic environment. We posit CRM's performance is hindered because the reward machine, designed under the assumption of a deterministic environment, may not provide appropriate signals for effective learning in a stochastic setting.



(a) sequential delivery (Sec. 5.1.1) *(b)* office-world (Sec. 5.1.3) *(c)* Task 5.2 in the craft-world

Figure 5. Results for the three case studies in stochastic environments (see Sec. 5.2).

Summary: Our proposed approach successfully solves complex tasks in environments presenting a high chance of violating a safety specification (e.g., Fig 3b) where the baseline method fails to solve the same tasks even in the deterministic setting. By treating time (i.e., allowed trajectory length) as a limited resource, our approach achieves higher success

[5] See rrl-ltl.github.io/RRL-LTL for similar plots of the other nine tasks.

rates compared to existing solutions when given a short duration to solve tasks in stochastic environments such as the one in Sec. 5.1.3. Finally, we tested our agent in a set of tasks proposed by [2], and show our approach can tackle complex specifications that require multiple levels of planning in stochastic environments.

6. Conclusion

This work presents a model-free RL approach that satisfies LTL_f specifications in MDPs with a priori unknown transition probabilities. Our approach first employs a neural network combined with MCTS to generate runs that satisfy the task. We further extract a policy from the MCTS parameters to make the approach amenable to online deployment and testing. Extensive simulations demonstrate that this approach can successfully learn policies to satisfy complex LTL_f specifications within a given maximum number of steps (e.g., corresponding to robot battery life or time limit). This approach learns robust policies for studied stochastic cases where either a state-of-the-art approach could not satisfy the specification of interest or, performs better when given a limited trajectory duration.

Limitations and future work: While the simulation studies show the applicability and performance of this approach across a range of problems, the proposed method does have limitations. Primarily, unlike [1], the current approach does not have theoretical guarantees of maximizing the probability of satisfying the LTL specification. Additionally, this approach lacks completeness guarantees, i.e., it may fail to find a trajectory that satisfies a specification, even when one exists. While our approach may lack theoretical soundness guarantees, rigorous simulation evaluations demonstrate practical success in cases where even state-of-the-art methods fail.

Future works aim to mitigate these limitations by embedding structure into the learned policies to enable proof of completeness. We also aim to modify the proposed approach to make the learned policies generalize to environments other than the ones they were trained in, e.g., if the office world has additional decorations when the robot is deployed.[6]

References

[1] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic. "Control synthesis from linear temporal logic specifications using model-free reinforcement learning". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 10349–10355.

[2] J. Andreas, D. Klein, and S. Levine. "Modular multitask reinforcement learning with policy sketches". In: *International conference on machine learning*. PMLR. 2017, pp. 166–175.

[3] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith. "Using reward machines for high-level task specification and decomposition in reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2107–2116.

[4] D. Tian, H. Fang, Q. Yang, H. Yu, W. Liang, and Y. Wu. "Reinforcement learning under temporal logic constraints as a sequence modeling problem". In: *Robotics and Autonomous Systems* 161 (2023), p. 104351.

[5] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic. "Model-free reinforcement learning for stochastic games with linear temporal logic objectives". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 10649–10655.

[6] G. De Giacomo and M. Y. Vardi. "Linear temporal logic and linear dynamic logic on finite traces". In: *IJCAI'13 Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. Association for Computing Machinery. 2013, pp. 854–860.

[7] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg. "Learning by playing solving sparse reward tasks from scratch". In: *International conference on machine learning*. PMLR. 2018, pp. 4344–4353.

[6]This work was supported in part by Magna International and the Natural Sciences and Engineering Research Council (NSERC).

[8] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. "A survey of Monte Carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[9] T. Anthony, Z. Tian, and D. Barber. "Thinking fast and slow with deep learning and tree search". In: *Advances in neural information processing systems* 30 (2017).

[10] A. Laterre, Y. Fu, M. K. Jabri, A.-S. Cohen, D. Kas, K. Hajjar, T. S. Dahl, A. Kerkeni, and K. Beguir. "Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization". In: *arXiv preprint arXiv:1807.01672* (2018).

[11] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. "Mastering the game of Go without human knowledge". In: *nature* 550.7676 (2017), pp. 354–359.

[12] I. Papusha, J. Fu, U. Topcu, and R. M. Murray. "Automata theory meets approximate dynamic programming: Optimal control with temporal logic constraints". In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE. 2016, pp. 434–440.

[13] G. De Giacomo, M. Y. Vardi, et al. "Synthesis for LTL and LDL on finite traces". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*. AAAI Press. 2015, pp. 1558–1564.

[14] X. Ding, S. L. Smith, C. Belta, and D. Rus. "Optimal control of Markov decision processes with linear temporal logic constraints". In: *IEEE Transactions on Automatic Control* 59.5 (2014), pp. 1244–1257.

[15] X. C. Ding, C. Belta, and C. G. Cassandras. "Receding horizon surveillance with temporal logic specifications". In: *49th IEEE Conference on Decision and Control (CDC)*. IEEE. 2010, pp. 256–261.

[16] H. Kress-Gazit, M. Lahijanian, and V. Raman. "Synthesis for robots: Guarantees and feedback for robot behavior". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 211–236.

[17] C. Voloshin, A. Verma, and Y. Yue. "Eventual Discounting Temporal Logic Counterfactual Experience Replay". In: *Proceedings of the 40th International Conference on Machine Learning*. Vol. 202. 2023, pp. 35137–35150.

[18] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. "Teaching multiple tasks to an RL agent using LTL". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. 2018, pp. 452–461.

[19] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. "Reward machines: Exploiting reward function structure in reinforcement learning". In: *Journal of Artificial Intelligence Research* 73 (2022), pp. 173–208.

[20] C. Voloshin, H. Le, S. Chaudhuri, and Y. Yue. "Policy optimization with linear temporal logic constraints". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 17690–17702.

[21] P. Vaezipoor, A. C. Li, R. A. T. Icarte, and S. A. Mcilraith. "Ltl2action: Generalizing ltl instructions for multi-task rl". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10497–10508.

[22] M. L. Littman, U. Topcu, J. Fu, C. Isbell, M. Wen, and J. MacGlashan. "Environment-independent task specifications via GLTL". In: *arXiv preprint arXiv:1704.04341* (2017).

[23] A. Pnueli. "The temporal logic of programs". In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. ieee. 1977, pp. 46–57.

[24] S. Sickert, J. Esparza, S. Jaax, and J. Křetínskỳ. "Limit-deterministic Büchi automata for linear temporal logic". In: *Computer Aided Verification: 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*. Springer. 2016, pp. 312–332.

[25] M. Hasanbeig, A. Abate, and D. Kroening. "Logically-constrained reinforcement learning". In: *arXiv preprint arXiv:1801.08099* (2018).

[26] C. D. Rosin. "Multi-armed bandits with episode context". In: *Annals of Mathematics and Artificial Intelligence* 61.3 (2011), pp. 203–230.