Eventual Discounting Temporal Logic Counterfactual Experience Replay

Cameron Voloshin [1] **Abhinav Verma** [2] **Yisong Yue** [1 3]

Abstract

Linear temporal logic (LTL) offers a simplified way of specifying tasks for policy optimization that may otherwise be difficult to describe with scalar reward functions. However, the standard RL framework can be too myopic to find maximally LTL satisfying policies. This paper makes two contributions. First, we develop a new value-function based proxy, using a technique we call eventual discounting, under which one can find policies that satisfy the LTL specification with highest achievable probability. Second, we develop a new experience replay method for generating off-policy data from on-policy rollouts via counterfactual reasoning on different ways of satisfying the LTL specification. Our experiments, conducted in both discrete and continuous state-action spaces, confirm the effectiveness of our counterfactual experience replay approach.

1. Introduction

In the standard reinforcement learning (RL) framework, the goal is to develop a strategy that maximizes a reward function in an unknown environment. In many applications of RL, a practitioner is responsible for generating the reward function so that the agent will behave desirably after the learning process. However, it can be challenging to convey real-world task specifications through scalar rewards (Randløv & Alstrøm, 1998; Toromanoff et al., 2019; Ibarz et al., 2018; Zhang et al., 2021; Ng et al., 1999; Yang et al., 2022). Colloquially known as reward-shaping, practitioners often resort to using heuristic "breadcrumbs" (Sorg, 2011) to guide the agent towards intended behaviors. Despite the "reward function is enough" hypothesis (Sutton & Barto, 2018; Silver et al., 2021)), some tasks may not be reducible to scalar rewards (Abel et al., 2021).

In response to these challenges, alternative RL frameworks using Linear Temporal Logic (LTL) to specify agent behavior have been studied (see Section 7). LTL can express desired characteristics of future paths of a system (Baier & Katoen, 2008), allowing for precise and flexible task/behavior specification. For example, we may ask for a system to repeatedly accomplish a set of goals in specific succession (see Section 2 for more examples).

Without significant assumptions, there is no precise signal on the probability of a policy satisfying an LTL objective. Existing work overwhelmingly uses Q-learning with a sparse RL heuristic (Bozkurt et al., 2020; Cai et al., 2021a) meant to motivate an agent to generate trajectories that appear to satisfy the task. First, these heuristics involve complicated technical assumptions obscuring access to non-asymptotic guarantees, even in finite state-action spaces. Second, sparse reward functions pose substantial challenges to any gradient-based RL algorithm since they provide poor signal to adequately solve credit assignment. Resolving sparsity involves using a hierarchical approach (Bozkurt et al., 2020) or re-introducing reward-shaping (Hasanbeig et al., 2018). See Section 7 for elaboration on prior work.

Our contributions. In this paper, we focus on model-free policy learning of an LTL specified objective from online interaction with the environment. We make two technical contributions. First, we reformulate the RL problem with a modified value-function proxy using a technique we call *eventual discounting*. The key idea is to account for the fact that optimally satisfying the LTL specification may not depend on the length of time it takes to satisfy it (e.g., "eventually always reach the goal"). We prove in Section 4 that the optimal policy under eventual discounting maximizes the probability of satisfying the LTL specification.

Second, we develop an experience replay method[1] to address the reward sparsity issue. Namely, any LTL formula can be converted to a fully known specialized finite state automaton from which we can generate multiple counterfactual trajectories from a single on-policy trajectory. We call this method *LTL-guided counterfactual experience replay*. We empirically validate the performance gains of our counterfactual experience replay approach using both finite state/action spaces as well as continuous state/action spaces using both Q-learning and Policy Gradient approaches.

[1]Caltech [2]Penn State [3]Latitude AI. Correspondence to: Cameron Voloshin <cvoloshin@caltech.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1]Code here: https://github.com/clvoloshin/RL-LTL

2. Preliminaries

We give the necessary background and examples to understand our problem statement and solution approach. An *atomic proposition* is a variable that takes on a truth value. An *alphabet* over a set of atomic propositions AP is given by $\Sigma = 2^{\text{AP}}$. For example, if AP $= \{x, y\}$ then $\Sigma = \{\{\}, \{x\}, \{y\}, \{x, y\}\}$. $\Delta(A)$ represents the set of probability distributions over a set A.

2.1. Running Example

We will be using the environment illustrated in Figure 1 (First) as a running example. The agent is given by a green dot and there are 3 circular regions in the environment colored yellow, blue and red. The AP are given by $\{y, b, r\}$, referring to the respective colored zones. Elements of Σ indicate which zone(s) the agent is in.

2.2. MDPs with Labelled State Spaces

Following a similar notation as Voloshin et al. (2022), we assume that the environment follows the discounted, labelled Markov Decision Process (MDP) framework given by the tuple $\mathcal{M} = (\mathcal{S}^{\mathcal{M}}, \mathcal{A}^{\mathcal{M}}, P^{\mathcal{M}}, d_0^{\mathcal{M}}, \gamma, L^{\mathcal{M}})$ consisting of a state space $\mathcal{S}^{\mathcal{M}}$, an action space $\mathcal{A}^{\mathcal{M}}$, an ***unknown*** transition function $P^{\mathcal{M}} : \mathcal{S}^{\mathcal{M}} \times \mathcal{A}^{\mathcal{M}} \to \Delta(\mathcal{S}^{\mathcal{M}})$, an initial state distribution $d_0^{\mathcal{M}} \in \Delta(\mathcal{S}^{\mathcal{M}})$, and a labelling function $L^{\mathcal{M}} : \mathcal{S}^{\mathcal{M}} \to \Sigma$. Let $A^{\mathcal{M}}(s)$ to be the set of available actions in state s.

Unlike traditional MDPs, $\mathcal{M}$ has a labeling function $L^{\mathcal{M}}$ which returns the atomic propositions that are true in that state. For example, in Figure 1 (First), when the agent enters a state $s \in \mathcal{S}^{\mathcal{M}}$ such that it is both in the yellow and blue zone then $L^{\mathcal{M}}(s) = \{y, b\}$.

2.3. Linear Temporal Logic (LTL)

Here we give a basic introduction to LTL. For a more comprehensive overview, see Baier & Katoen (2008).

Definition 2.1 (LTL Specification, φ)**.** An LTL specification φ is the entire description of the task, constructed from a composition of atomic propositions with logical connectives: not ($\neg$), and (&), and implies ($\to$); and temporal operators: next (X), repeatedly/always/globally (G), eventually (F), and until (U).

Examples. For $AP = \{x, y\}$, some basic task specifications include safety ($G\neg x$), reachability (Fx), stability (FGx), response ($x \to Fy$), and progress (x & XFy).

Consider again the environment in Figure 1 (First) where $AP = \{y, r, b\}$. If the task is to eventually reach the yellow zone and stay there (known as stabilization) then we write $\varphi = FGy$. Or, if we would like the agent to infinitely loop between the yellow and red zone while avoiding the blue zone then $\varphi = GF(y \;\&\; XFr) \;\&\; G\neg b$, a combination of safety, reachability, and progress.

2.4. LTL Satisfaction

LTL has recursive semantics defining the meaning for logical connective satisfaction. Without loss of generality, we will be using a specialized automaton, an LDBA $\mathcal{B}_\varphi$ (Sickert et al., 2016), defined below to keep track of the progression of φ satisfaction. More details for constructing LDBAs are in Hahn et al. (2013); Baier & Katoen (2008); Křetínskỳ et al. (2018). We drop φ from $\mathcal{B}_\varphi$ for brevity.

Definition 2.2. (Limit Deterministic Büchi Automaton, LDBA (Sickert et al., 2016)) An ***LDBA*** is a tuple $\mathcal{B} = (\mathcal{S}^{\mathcal{B}}, \Sigma \cup \mathcal{A}_{\mathcal{B}}, P^{\mathcal{B}}, \mathcal{S}^{\mathcal{B}^*}, b_{-1}^{\mathcal{B}})$ consisting of (i) a finite set of states $\mathcal{S}^{\mathcal{B}}$, (ii) a finite alphabet $\Sigma = 2^{\text{AP}}$, $\mathcal{A}_{\mathcal{B}}$ is a set of indexed jump transitions (iii) a transition function $P^{\mathcal{B}} : \mathcal{S}^{\mathcal{B}} \times (\Sigma \cup \mathcal{A}_{\mathcal{B}}) \to \mathcal{S}^{\mathcal{B}}$, (iv) accepting states $\mathcal{S}^{\mathcal{B}^*} \subseteq \mathcal{S}^{\mathcal{B}}$, and (v) initial state $b_{-1}^{\mathcal{B}}$. There exists a mutually exclusive partitioning of $\mathcal{S}^{\mathcal{B}} = \mathcal{S}_D^{\mathcal{B}} \cup \mathcal{S}_N^{\mathcal{B}}$ such that $\mathcal{S}^{\mathcal{B}^*} \subseteq \mathcal{S}_D^{\mathcal{B}}$, and for $b \in \mathcal{S}_D^{\mathcal{B}}, a \in \Sigma$ then $P^{\mathcal{B}}(b, a) \subseteq \mathcal{S}_D^{\mathcal{B}}$, closed. $\mathcal{A}_{\mathcal{B}}(b)$ is only (possibly) non-empty for $b \in \mathcal{S}_N^{\mathcal{B}}$ and allows $\mathcal{B}$ to transition to $\mathcal{S}_D^{\mathcal{B}}$ without reading an AP. A *path* $\varrho = (b_0, b_1, \dots)$ is a sequence of states in $\mathcal{B}$ reached through successive transitions under $P^{\mathcal{B}}$.

Definition 2.3. ($\mathcal{B}$ accepts) $\mathcal{B}$ **accepts** a path ϱ if there exists some state $b \in \mathcal{S}^{\mathcal{B}^*}$ in the path that is visited infinitely often.

Examples. Consider again the environment in Figure 1 (First) where $AP = \{y, r, b\}$. If we would like to make an LDBA for $\varphi = FGy$ (reach and stabilize at y) then we would get the state machine seen in Figure 1 (Second). In this state machine, the agent starts at state 0. The accepting set is given by $\mathcal{S}^{\mathcal{B}^*} = \{1\}$. The transition between state 0 and state 1 is what is formally referred to as a jump transition: $\mathcal{A}_{\mathcal{B}}(0) = \{\epsilon\}$ while $\mathcal{A}_{\mathcal{B}}(\cdot) = \varnothing$ otherwise. Whenever the agent is in state 0 of the LDBA, there is a choice of whether to stay at state 0 or transition immediately to state 1. This choice amounts to the agent believing that it has satisfied the "eventually" part of the LTL specification. When the agent takes this jump, then it must thereafter satisfy y to stay in state 1. The agent gets the decision of when it believes it is capable of satisfying y thereafter. When the agent takes the jump, if it fails to stay in y, it immediately transitions to the sink, denoted state 2. The LDBA accepts when the state 1 is reached infinitely often, meaning the agent satisfies "always y" eventually, as desired.

Another example, this time without jump transitions, would be for $\varphi = GF(y \;\&\; XFr) \;\&\; G\neg b$ (oscillate between y and r forever while avoiding b). The LDBA can be seen in Figure 1 (Third). In this state machine, the agent starts at state 1 and the accepting set is given by $\mathcal{S}^{\mathcal{B}^*} = \{1\}$. To make a loop back to state 1, the agent must visit both r and y.

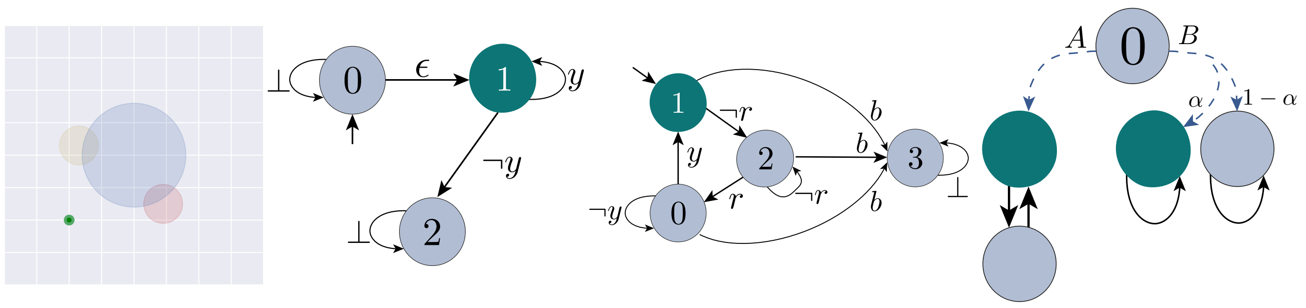

Figure 1. Examples. ***First:*** Illustration of the Flatworld environment. The agent is a green dot and there are 3 zones: yellow, blue and red. ***Second:*** LDBA $\mathcal{B}$ for "FGy". $\mathcal{S}^{\mathcal{B}^*} = \{1\}$, denoted by green circle. The initial state is $b_{-1} = 0$. ***Third:*** LDBA $\mathcal{B}$ for "$GF(y \,\&\, XFr) \,\&\, G\neg b$". $\mathcal{S}^{\mathcal{B}^*} = \{1\}$, denoted by green circle. The initial state is $b_{-1} = 1$. ***Fourth:*** For this example, an agent starting in state 0 and solving $\arg\max_{\pi\in\Pi} \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}[\sum_{i=0}^{\infty} \gamma^i \mathbf{1}_{\{b_i \in \mathcal{S}^{\mathcal{B}^*}\}}]$ where $\mathcal{S}^{\mathcal{B}^*}$ is illustrated as the green circles would choose to take action B with probability 1 if $\alpha \in (1/2, 1)$ for any $\gamma \in [0, 1]$. Such a policy has $\mathbb{P}[\pi \models \varphi] = \alpha < 1$. However the probability optimal policy deterministically takes action A, with $\mathbb{P}[\pi^* \models \varphi] = 1$. This illustrates catastrophic myopic behavior.

Doing so infinitely often satisfies the LDBA condition and therefore the specification. If at any point b is encountered then the agent transitions to the sink, denoted state 3.

3. Problem Formulation

We first introduce slightly more notation. Let $\mathcal{Z} = \mathcal{S}^{\mathcal{M}} \times \mathcal{S}^{\mathcal{B}}$. Let $\Pi : \mathcal{Z} \times \mathcal{A} \to \Delta([0, 1])$ be a (stochastic) policy class over the product space of the MDP and the LDBA (defined below), where $\mathcal{A}((s, b)) = \mathcal{A}^{\mathcal{M}}(s) \cup \mathcal{A}^{\mathcal{B}}(b)$, to account for jump transitions in $\mathcal{B}$.

Synchronizing the MDP with the LDBA. For any $(s, b) \in \mathcal{Z}$, a policy $\pi \in \Pi$ is able to select an action in $\mathcal{A}^{\mathcal{M}}(s)$ or an action in $\mathcal{A}^{\mathcal{B}}(b)$, if available. We can therefore generate a **trajectory** as the sequence $\tau = (s_0, b_0, a_0, s_1, b_0, a_1, \ldots)$ under a new probabilistic transition relation given by

$$P(s', b'|s, b, a) = \begin{cases} P^{\mathcal{M}}(s, a, s') & a \in A^{\mathcal{M}}(s), b' \in P^{\mathcal{B}}(b, L(s')) \\ 1, & a \in A^{\mathcal{B}}(b), b' \in P^{\mathcal{B}}(b, a), s = s' \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Let the LDBA projection of τ be the subsequence $\tau_{\mathcal{B}} = (b_0, b_1, \ldots)$. Elements of $\tau_{\mathcal{B}}$ can be thought of as tracking an agent's LTL specification satisfaction:

Definition 3.1 (Run Satisfaction, $\tau \models \varphi$)**.** We say a trajectory satisfies φ if $\mathcal{B}$ accepts $\tau_{\mathcal{B}}$, which happens if $\exists b \in \tau_{\mathcal{B}}$ infinitely often with $b \in \mathcal{S}^{\mathcal{B}^*}$.

Let $\mathrm{T}^P_\pi = \mathbb{E}_{z\sim d_0^{\mathcal{M}} \times \{b_{-1}\}}[\mathrm{T}^P_\pi(z)]$ be the distribution over all possible trajectories starting from any initial state $z \in d_0^{\mathcal{M}} \times \{b_{-1}\}$ where $\mathrm{T}^P_\pi(z)$ is the (conditional) distribution over all possible trajectories starting from $z \in \mathcal{Z}$ generated by π under relation P (given in (1)). The probability of LTL satisfaction results from counting how many of the trajectories satisfy the LTL specification:

Definition 3.2 (State Satisfaction, $z \models \varphi$)**.** $\mathbb{P}_\pi[z \models \varphi] = \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi(z)}[\mathbf{1}_{\{\tau\models\varphi\}}] = \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}[\mathbf{1}_{\{\tau\models\varphi\}}|z_0 = z]$

Definition 3.3 (Policy Satisfaction, $\pi \models \varphi$)**.** $\mathbb{P}[\pi \models \varphi] = \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}[\mathbf{1}_{\{\tau\models\varphi\}}]$ where $\mathbf{1}_X$ is the indicator for X.

Ideally we would like to find a policy with highest probability of LTL specification satisfaction: one that generates the most number of LTL-satisfying runs. Formally,

$$\pi^* \in \arg\max_{\pi\in\Pi} \mathbb{P}[\pi \models \varphi]. \tag{2}$$

We note that Eq (2) is the standard starting point for formulating policy optimization for LTL satisfaction (Yang et al., 2022; Bozkurt et al., 2020; Cai et al., 2021a; Hasanbeig et al., 2018; 2020; Voloshin et al., 2022).

Remark 3.1. *The synchronized MDP/LDBA, referred to as a product MDP, is itself a larger MDP with a "mixed" state space given a continuous state from the MDP and a discrete state from the LDBA, e.g. states now take on the tuple-form (s, b). LTL satisfaction is not reachability within this product MDP. Satisfaction requires a policy that creates an infinite loop in the LDBA part of the product MDP (reaching any accepting buchi state infinitely often for arbitrary underlying MDP state).*

Remark 3.2. *Notice that the policy class Π is memoryless with respect to the product MDP as the LDBA takes care of adding in sufficent memory a policy would need to accomplish φ (Baier & Katoen, 2008). This comes at the cost of a significant increase in state-action space, but we never explicitly form the product MDP.*

4. RL-Friendly Form: Eventual Discounting

Unfortunately, the maximization problem in Eq (2) is not easily optimized since we dont have a direct signal on $\mathbb{P}[\pi \models \varphi]$. Without any additional assumptions (such as structured knowledge of the MDP), any finite subsequence can only give evidence on whether $\tau \models \varphi$ but not a concrete proof.

Eventual Discounting. To address the above issue, we develop a modified value-function based surrogate as follows. Given a trajectory $\tau = (s_0, b_0, a_0, \ldots)$, we keep track of how often $b_i \in \mathcal{S}^{\mathcal{B}^*}$ and incentivize an agent to visit $\mathcal{S}^{\mathcal{B}^*}$ as many times as possible. In particular, under eventual discounting, the value function will give the agent a reward of 1 when in a state $b_i \in \mathcal{S}^{\mathcal{B}^*}$ and not discount length of time between visits to $\mathcal{S}^{\mathcal{B}^*}$. Formally, we will be seeking

$$\pi^*_\gamma \in \arg\max_{\pi\in\Pi} \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}[\sum_{i=0}^{\infty} \Gamma_i \mathbf{1}_{\{b_i \in \mathcal{S}^{\mathcal{B}^*}\}}] \quad (\equiv V^\gamma_\pi), \tag{3}$$

where $\Gamma_0 = 1$ and

$$\Gamma_i = \prod_{t=0}^{i-1} \gamma(b_t), \quad \gamma(b_t) = \begin{cases} \gamma, & b_t \in \mathcal{S}^{\mathcal{B}^*} \\ 1, & \text{otherwise} \end{cases}. \tag{4}$$

Intuition for Γ_i. At first glance setting $\Gamma_i = \gamma^i$ to be the traditional RL exponential discount rate would seem reasonable. Unfortunately, $\nexists \gamma \in [0,1]$ with $\Gamma_i = \gamma^i$ that avoids catastrophic myopic behavior. In particular, take Figure 1 (Fourth). The agent starts in state 0 and only has two actions A and B. Taking action A transitions directly to an accepting state from which point the accepting state is visited every 2 steps. On the other hand, action B transitions to an accepting state with probability α and a sink state with probability $1-\alpha$. The accepting state reached by action B is revisited every step. Suppose $\beta = \pi(A) = 1 - \pi(B)$ then we can calculate:

$$\mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}[\sum_{i=0}^{\infty} \gamma^i \mathbf{1}_{\{b_i \in \mathcal{S}^{\mathcal{B}^*}\}}] = \frac{\beta}{1-\gamma^2} + \frac{(1-\beta)\alpha}{1-\gamma}. \tag{5}$$

For $\alpha > 1/2$, the optimal choice β is $\beta = 0$ implying that $P(\pi \models \varphi) = \alpha$. When $\alpha \in (1/2, 1)$ then this implies that π is not probability optimal. Indeed, $P(\pi \models \varphi) = \alpha < 1$ when $\beta = 0$ but $P(\pi^* \models \varphi) = 1$ by selecting $\beta = 1$. The intuition here, which can be formalized by taking $\gamma \to 1$, is that the average reward for taking action A is $\frac{1}{2}$ while the average reward for taking action B is 1 with probability α, which is worth the risk for large $\alpha > 1/2$.

To avoid this myopic behavior, we must avoid discriminating between return times between good states. The number steps (on average) it takes to return to $\mathcal{S}^{\mathcal{B}^*}$ is irrelevant: we only require that the system does return. For this reason we do not count time (hence $\gamma = 1$) in our definition of Γ_i when the system is not in $\mathcal{S}^{\mathcal{B}^*}$. We call this *eventual discounting*.

4.1. Analysis of π^*_γ

In this section we analyze how the probability of π^*_γ satisfying φ compares to that of the best possible one π^*.

Let the set $O(\tau) = \{i : b_i \in \mathcal{S}^{\mathcal{B}^*}\}$ denote the occurences (time steps) when a good state is reached. This quantity is natural since $|O(\tau)| = \infty$ if and only if $\tau \models \varphi$.

Lemma 4.1. *For any $\pi \in \Pi$ and $\gamma \in (0,1)$, we have*

$$|(1-\gamma)V^\gamma_\pi - \mathbb{P}[\pi \models \varphi]| \le \log(\frac{1}{\gamma})O_\pi$$

where $O_\pi = \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}\left[|O(\tau)| \middle| \tau \not\models \varphi\right]$ is the expected number of visits to an accepting state for the trajectories that do not satisfy φ.

Proof. Fix some state $z = (s,b) \in \mathcal{Z}$.

$$\begin{aligned} V^\gamma_\pi(z) &= \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}[\sum_{i=0}^{\infty} \Gamma_i \mathbf{1}_{\{b_i \in \mathcal{S}^{\mathcal{B}^*}\}} | z_0 = z] \\ &= \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}\left[\sum_{j=0}^{|O(\tau)|} \gamma^j | z_0 = z\right] \end{aligned}$$

Using the fact that $\sum_{j=0}^{k} \gamma^j = \frac{1-\gamma^k}{1-\gamma}$, we have

$$\begin{aligned} V^\gamma_\pi(z) = \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}&\left[\frac{1-\gamma^{|O(\tau)|}}{1-\gamma}\middle|_{z_0=z}^{\tau\models\varphi}\right] \mathbb{P}_\pi[z \models \varphi] \\ &+ \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}\left[\frac{1-\gamma^{|O(\tau)|}}{1-\gamma}\middle|_{z_0=z}^{\tau\not\models\varphi}\right] \mathbb{P}_\pi[z \not\models \varphi] \end{aligned} \tag{6}$$

Since $|O(\tau)| = \infty$ for any $\tau \models \varphi$,

$$\mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}\left[\frac{1-\gamma^{|O(\tau)|}}{1-\gamma}\middle|_{z_0=z}^{\tau\models\varphi}\right] = \frac{1}{1-\gamma} \tag{7}$$

together with $\mathbb{P}_\pi[z \not\models \varphi] \ge 0$ implies

$$V^\gamma_\pi(z) \ge \frac{1}{1-\gamma}\mathbb{P}_\pi[z \models \varphi]. \tag{8}$$

Taking the expectation over initial states we have

$$V^\gamma_\pi \ge \frac{1}{1-\gamma}\mathbb{P}[\pi \models \varphi]. \tag{9}$$

Now we find an upper bound. First, define

$$M_\pi(t) \equiv \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}\left[e^{t|O(\tau)|}\middle| \tau \not\models \varphi\right].$$

Starting again with Eq (6) and using Eq (7), we have

$$V^\gamma_\pi(z) \le \frac{\mathbb{P}_\pi[z \models \varphi]}{1-\gamma} + \frac{1 - \mathbb{E}_{\tau\sim \mathrm{T}^P_\pi}\left[e^{\log(\gamma)|O(\tau)|}\middle|_{z_0=z}^{\tau\not\models\varphi}\right]}{1-\gamma} \tag{10}$$

where we have used that $\mathbb{P}_\pi[z \not\models \varphi] \le 1$ for any $z \in \mathcal{Z}$. Taking the expectation with respect to the initial state distribution then we have

$$(1-\gamma)V^\gamma_\pi \le \mathbb{P}[\pi \models \varphi] + 1 - M_\pi(\log(\gamma)) \tag{11}$$

In particular, $M_\pi(t)$ is convex and therefore it lies above its tangents:

$$\begin{aligned} M_\pi(t) &\geq M_\pi(0) + tM'_\pi(0) \\ &= 1 + t\mathbb{E}_{\tau \sim \mathrm{T}^P_\pi}\left[|O(\tau)|\middle| \tau \not\models \varphi\right] \\ &= 1 + tO_\pi \end{aligned}$$

Plugging this inequality into Eq (11), together with Eq (9),

$$\mathbb{P}[\pi \models \varphi] \leq (1-\gamma)V^\gamma_\pi \leq \mathbb{P}[\pi \models \varphi] + \log(\frac{1}{\gamma})O_\pi \quad (12)$$

Subtracting $\mathbb{P}[\pi \models \varphi]$ from both sides and taking the absolute value completes the proof. □

Theorem 4.2. *(Non-asymptotic guarantee) For any $\gamma \in (0, 1)$,*

$$\sup_{\pi\in\Pi} \mathbb{P}[\pi \models \varphi] - \mathbb{P}[\pi^*_\gamma \models \varphi] \leq 2\log(\frac{1}{\gamma}) \sup_{\pi\in\Pi} O_\pi \quad (13)$$

where $O_\pi = \mathbb{E}_{\tau \sim \mathrm{T}^P_\pi}\left[|O(\tau)|\middle| \tau \not\models \varphi\right]$.

Proof. Consider any sequence $\{\pi_i\}_{i=1}^\infty$ such that $\mathbb{P}[\pi_i \models \varphi] \to \sup_\pi \mathbb{P}[\pi \models \varphi]$ as $i \to \infty$. Then we have for any π_i,

$$\begin{aligned} \mathbb{P}[\pi_i \models \varphi] - \mathbb{P}[\pi^*_\gamma \models \varphi] &= \mathbb{P}[\pi_i \models \varphi] - (1-\gamma)V^\gamma_{\pi_i} \\ &\quad + (1-\gamma)V^\gamma_{\pi_i} - (1-\gamma)V^\gamma_{\pi^*_\gamma} \\ &\quad + (1-\gamma)V^\gamma_{\pi^*_\gamma} - \mathbb{P}[\pi^*_\gamma \models \varphi] \\ &\overset{(a)}{\leq} |\mathbb{P}[\pi_i \models \varphi] - (1-\gamma)V^\gamma_{\pi_i}| \\ &\quad + |\mathbb{P}[\pi^*_\gamma \models \varphi] - (1-\gamma)V^\gamma_{\pi^*_\gamma}| \\ &\overset{(b)}{\leq} \log(\frac{1}{\gamma})(O_{\pi_i} + O_{\pi^*_\gamma}) \\ &\overset{(c)}{\leq} 2\log(\frac{1}{\gamma}) \sup_{\pi\in\Pi} O_\pi \end{aligned}$$

where (a) is triangle inequality together with removing the term $(1-\gamma)V^\gamma_{\pi_i} - (1-\gamma)V^\gamma_{\pi^*_\gamma}$ since it is nonpositive by definition of π^*_γ, (b) is an application of Lemma 4.1, and (c) is a supremum over all policies. Taking the limit on both sides as $i \to \infty$ completes the proof. □

Corollary 4.3. *If the number of policies in Π is finite then $\sup_{\pi\in\Pi} O_\pi = m < \infty$ is attained, is a finite constant and*

$$\sup_{\pi\in\Pi} \mathbb{P}[\pi \models \varphi] - \mathbb{P}[\pi^*_\gamma \models \varphi] \leq 2m\log(\frac{1}{\gamma})$$

Corollary 4.4. *In the case that $\mathcal{S}^\mathcal{M}$ and $\mathcal{A}^\mathcal{M}$ are finite, then $\mathcal{Z}$ and $\mathcal{A}$ are finite. It is known that optimal policies are deterministic (Puterman, 2014) and therefore there we need only consider deterministic policies, for which there are a finite number. Thus $\sup_{\pi\in\Pi} O_\pi = m < \infty$ is attained, is a finite constant and*

$$\sup_{\pi\in\Pi} \mathbb{P}[\pi \models \varphi] - \mathbb{P}[\pi^*_\gamma \models \varphi] \leq 2m\log(\frac{1}{\gamma})$$

Corollary 4.5. *For any $\epsilon > 0$, when γ is selected to be $\gamma > \frac{1}{e^{\epsilon/(2m)}}$ then*

$$\left|\sup_{\pi\in\Pi} \mathbb{P}[\pi \models \varphi] - \mathbb{P}[\pi^*_\gamma \models \varphi]\right| \leq \epsilon$$

4.2. Interpretation

Theorem 4.2 relies on the quantity $\sup_{\pi\in\Pi} O_\pi$ to be finite for the bound to have meaning. In fact, we need only make requirements on $M_\pi(\log(\gamma))$ but the requirements are more easily understood on O_π. As an aside, $M_\pi(\log(\gamma))$ can be interpreted as the moment generating function of the random variable which is the number of visits to $\mathcal{S}^{\mathcal{B}^*}$. Instead we consider the equally natural quantity O_π. O_π is the (average) number of times that a good state is visited by a trajectory that does not satisfy the specification. Ideally, this number would be small and it would be easy to discriminate against good and bad policies.

Unconstrained Policy Class. In the case that Π is an infinite class, conditions for ensuring $\sup_{\pi\in\Pi} O_\pi$ is finite is nontrivial and is dependent on the landscape of the transition function P of the MDP and Π.

Let us suppose $\sup_{\pi\in\Pi} O_\pi$ is infinite. This means there are policies that induce bad trajectories that eventually fail to reach $\mathcal{S}^{\mathcal{B}^*}$, but along the way visited $\mathcal{S}^{\mathcal{B}^*}$ an arbitrarily large (but finite) number of times. In other words, they are policies that are indistinguishable from actual probability-optimal policies until the heat death of the universe.

Consider the specification in Figure 1 (right), given by infinitely often cycle between red and yellow while avoiding blue. A good-looking bad policy is one that accomplishes the task frequently but, amongst the times that it fails, it would cycle between red and yellow many times before failing. $\sup_{\pi\in\Pi} O_\pi$ being infinite means that there are policies that will cycle arbitrarily many times before failing.

Constrained Policy Class. Corollary 4.3 suggests that having a finite number of policies suffices to generate probability optimal policies, with suboptimality shrinking at a rate of $\log(\frac{1}{\gamma})$. In practice, since all computers deal with finite precision, the number of policies is finite. Explicitly constraining the number of policies is another option.

On the other hand, Corollary 4.4 reveals that discretization is also a sufficient condition. This suggests that compactness of P and Π and continuity of P may be sufficient as well but we leave these conditions for future work.

Algorithm 1 `Learning with LCER`

```
Param: Maximum horizon $T$. Replay buffer $D = \{\}$.
1: for $k = 1, 2, \ldots$ do
2:     Run $\pi_{k-1}$ in the MDP for $T$ timesteps and collect
       $\tau = (s_0, b_0, a_0, \ldots, s_{T-1}, b_{T-1}, a_{T-1}, s_T, b_T)$
3:     $D_k \leftarrow$ LCER$(D_{k-1}, \tau)$
4:     $\pi_k \leftarrow$ Update$(\pi_{k-1}, D_k)$ // Q-learn/Policy grad.
5: end for
```

5. LTL Counterfactual Experience Replay

One can optimize the formulation in Eq (3) using any Q-learning or policy gradient approach, as seen in Algorithm 1 (Line 4). However, doing so is challenging since it suffers from reward sparsity: the agent only receives a signal if it reaches a good state.

We combat reward sparsity by exploiting the LDBA: $P^{\mathcal{B}}$ is completely known. By knowing $P^{\mathcal{B}}$, we can generate multiple off-policy trajectories from a single on-policy trajectory by modifying which stats in the LDBA we start in, which notably does not require any access to the MDP transition function $P^{\mathcal{M}}$. We call this approach *LTL-guided Counterfactual Experience Replay*, `LCER` (Algorithm 1, Line 3), as it is a modification of standard experience replay (Lin, 1992; Mnih et al., 2013; 2015) to include counterfactual experiences elsewhere in the LDBA. `LCER` is most simply understood through Q-learning, and needs careful modification for policy gradient methods.

Q-learning with `LCER`. See Algorithm 2 for a synopsis of `LCER` for Q-learning. Regardless of whatever state $s \in \mathcal{S}^{\mathcal{M}}$ the agent is in, we can pretend that the agent is in any $b \in \mathcal{S}^{\mathcal{B}}$. Then for any action the agent takes we can store experience tuples:

$$\{(s, b, a, r, s', \tilde{b}') \mid \forall b \in \mathcal{S}^{\mathcal{B}}\} \tag{14}$$

where $\tilde{b}' = P^{\mathcal{B}}(b, L^{\mathcal{M}}(s'))$ is the transition that would have occurred from observing labelled state $L(s')$ in state (s, b) and $r = \mathbf{1}_{\tilde{b}' \in B^*}$. Furthermore we can add all jump transitions:

$$\{(s, b, \epsilon, r, s, \tilde{b}') \mid \forall b \in \mathcal{S}^{\mathcal{B}}, \forall \epsilon \in \mathcal{A}^{\mathcal{B}}(b)\} \tag{15}$$

since jumps also do not affect the MDP. Notice when we add the jumps that $s' = s$, since only the LDBA state shifts in a jump.

Policy Gradient with `LCER`. See Algorithm 3 for a summary of `LCER` for policy gradient. For policy gradient, unlike Q-learning, it is necessary to calculate future reward-to-go: $R_k(\tau) = \sum_{i=k}^{T} \Gamma_i \mathbf{1}_{\{b_i \in \mathcal{S}^{\mathcal{B}^*}\}}$. Thus, we have to generate entire trajectories that are consistent with $P^{\mathcal{B}}$ rather than independent transition tuples as in Eq (14). We will show how to generate all feasible trajectories.

Algorithm 2 `LCER` for Q-learning

```
Param: Dataset $D$. Trajectory $\tau$ of length $T$.
1: for $(s_t, a_t, s_{t+1}) \in \tau$ do
2:     for $b \in \mathcal{S}^{\mathcal{B}}$ do
3:         Set $\tilde{b} \leftarrow P^{\mathcal{B}}(b, L^{\mathcal{M}}(s_{t+1}))$
4:         $D \leftarrow D \cup (s_t, b, a_t, \mathbf{1}_{\tilde{b} \in B^*}, s_{t+1}, \tilde{b})$
5:         for $\epsilon \in \mathcal{A}^{\mathcal{B}}(s)$ do
6:             Set $\tilde{b} \leftarrow P^{\mathcal{B}}(b, \epsilon)$
7:             $D \leftarrow D \cup (s_t, b, \epsilon, \mathbf{1}_{\tilde{b} \in B^*}, s_t, \tilde{b})$
8:         end for
9:     end for
10: end for
11: return $D$
```

Consider a trajectory $\tau = (s_0, b_0, a_0, \ldots, s_T, b_T)$ was collected. Let us remove jump transitions (s_i, b_i, a_i) where $a_i \in \mathcal{A}^{\mathcal{B}}(b_i)$ and consider the projection of the trajectory to the MDP $\tau_{\mathcal{M}} = (s_0, s_1, \ldots, s_T)$. We should only have control over the initial LDBA state b_0 as all other automaton states $(b_1, \ldots, b_T)$ in a trajectory sequence are determined by $\tau_{\mathcal{M}}$ and $b_{i+1} = P^{\mathcal{B}}(b_i, L^{\mathcal{M}}(s_i))$.

Therefore we add

$$\tilde{\mathcal{T}}(\tau) = \{(s_0, \tilde{b}_0, a_0, \ldots, s_T, \tilde{b}_T) \mid \\ \forall \tilde{b}_0 \in \mathcal{S}^{\mathcal{B}}, \tilde{b}_i = P^{\mathcal{B}}(\tilde{b}_{i-1}, L^{\mathcal{M}}(s_i))\} \tag{16}$$

where only the LDBA states are different between the trajectories.

Now we handle jump transitions. Consider some $\tilde{\tau} \in \tilde{\mathcal{T}}(\tau)$. Recall, a jump transition can occur whenever $\mathcal{A}^{\mathcal{B}}(\tilde{b}_i)$ is non-empty. This involves adding a trajectory that is identical to $\tilde{\tau}$ all the way until the jump occurs. The jump occurs and then the same action sequence and MDP state sequence follows but with different LDBA states. Specifically, suppose $\tilde{b}_i$ had an available jump transitions, $\epsilon \in \mathcal{A}^{\mathcal{B}}(\tilde{b}_i)$. Then:

$$\tilde{\tau}_{i,\epsilon} = (s_0, \tilde{b}'_i, a_0, \ldots, s_i, \tilde{b}'_i, \epsilon, s_i, \tilde{b}'_{i+1}, a_i, \ldots, s_T, \tilde{b}'_T) \tag{17}$$

where $\tilde{b}'_k = \tilde{b}_i$ for $k \leq i$ and $\tilde{b}'_k = P^{\mathcal{B}}(\tilde{b}'_{k-1}, L^{\mathcal{M}}(s_k))$ otherwise.

We have to add all possible $\tilde{\tau}'_{i,\epsilon}$ that exist. Let $\mathcal{E}$ be the operator that adds jumps to existing sequences:

$$\mathcal{E}(\tilde{\mathcal{T}}(\tau)) = \tilde{\mathcal{T}}(\tau) \cup \{\tilde{\tau}_{i,\epsilon} \text{ from Eq (17)} | \\ \forall \tilde{\tau} \in \tilde{\mathcal{T}}(\tau), \exists b_i \in \tilde{\tau} \text{ s.t. } \exists \epsilon \in \mathcal{A}^{\mathcal{B}}(b_i)\}. \tag{18}$$

We can only apply $\mathcal{E}(\mathcal{E}(\ldots(\mathcal{E}(\tilde{\mathcal{T}}(\tau)))))$ at most T times since the original length of τ is T.

Remark 5.1. *The length of τ has to be sufficiently large to make sure the LDBA has opportunity to reach $\mathcal{S}^{\mathcal{B}^*}$. A sufficient condition is $T \geq |\{b | \mathcal{A}^{\mathcal{B}}(b) \neq \varnothing\}|$, the number of LDBA states with jump transitions.*

Algorithm 3 LCER for Policy Gradient

```
Param: Dataset $D$. Trajectory $\tau$ of length $T$.
 1: Set $\tilde{\mathcal{T}}_0 \leftarrow \tilde{\mathcal{T}}(\tau)$
 2: for $k = 1, \ldots, T-1$ do
 3:     $\tilde{\mathcal{T}}_k \leftarrow \mathcal{E}(\tilde{\mathcal{T}}_{k-1})$
 4:     if $\tilde{\mathcal{T}}_k == \tilde{\mathcal{T}}_{k-1}$ then
 5:         Set $\tilde{\mathcal{T}}_{T-1} \leftarrow \tilde{\mathcal{T}}_k$
 6:         break
 7:     end if
 8: end for
 9: Set $D \leftarrow D \cup \tilde{\mathcal{T}}_{T-1}$
10: return $D$
```

It is possible to constructively generate feasible trajectories during the rollout of a policy rather than after-the-fact, see Appendix B.

6. Experiments

We perform experiments in four domains with varying LTL formulas, state spaces, action spaces, and environment stochasticity summarized in the following section. Our aim is to answer the following two questions: (1) Can we achieve policies that behave the way we expect an LTL-satisfying policy to behave? (2) How does LCER impact the performance of learning. [2]

6.1. Environment Details

Minecraft The Minecraft environment is a 10×10 deterministic gridworld with 5 available actions: left, right, up, down, nothing. The agent, given by a red triangle starts in the cell $(9, 2)$. The environment, as well as the final behavior of the agent (given by blue dots) can be seen in Figure 2 (First).

Pacman The Pacman environment is a 5×8 deterministic gridworld with 5 available actions: left, right, up, down, nothing. The agent, given by a red triangle starts in the cell $(0, 3)$. The ghost chases the agent with probability 0.8 and takes a random action with probability 0.2, for this reason the environment is stochastic. The starting position of the environment can be seen in Figure 2 (Second).

Flatworld The Flatworld environment (seen in Figure 2 Third and Fourth) is a two dimensional continuous world. The agent (given by a green dot) starts at $(-1, -1)$. The dynamics of the world are given by $x' = x + a/10$ where both $x \in \mathbb{R}^2$ and $a \in [0, 1]^2$. We also allow the action space to be discrete by letting there be 5 actions (right, up, left, down, nothing) where the agent takes a full-throttle action in each respective direction.

Carlo The Carlo environment (seen in Figure 2 Fifth) is a simplified self-driving simulator that uses a bicycle model for the dynamics. The agent observes its position, velocity, and heading in radians for a total of 5 dimensions. The agent has control over its heading and throttle, for an action space of $[-1, 1]^2$. For this domain, we have chosen to use a circular track where the agent starts in the center of the road at an angle of $\{\pi(1+2i)/4\}_{i=0}^3$ and drive counterclockwise around in a circle without crashing.

[2]Our choice of baseline is meant to reveal the impacts of LCER, while controlling for underlying method such as PPO or Q-learning. Creating a benchmark for RL+LTL with a standard set of experiments and baselines remains interesting and relevant future work.

6.2. Methods and Baseline

When the action space is discrete, we use Q-learning with LCER otherwise we use PPO with LCER. The baseline we compare against is the same method without LCER. This allows us to verify the extent to which LCER impacts performance. We also plot a trajectory from the final policy for each environment in the middle of each column of Figure 2, except for Pacman as it is difficult to visualize the interaction between the ghost and pacman outside of video.

Dealing with $\mathcal{A}$. For PPO, the agent's policy is a Gaussian (as in standard implementations) over the continuous action space. In order to deal with jump transitions (in the LDBA) when in a continuous action space (in the MDP), we first let the agent decide whether to execute a jump transition or not (ie. a probabilistic coin flip). If the agent chooses to not, then we take the action according to the Gaussian. The coin flip probability is learned, as well as the Gaussian. For the importance sampling term of PPO, the density of π is modified to account for the coin flip. For more details see Appendix A.

6.3. Results

Can we achieve desired behavior? The answer here is a resounding yes. For each environment (except Pacman) we illustrate the trajectory of the final policy above each learning curve in Figure 2. Determining the probability of satisfaction of the final policy is currently a challenging open problem (except in finite-state action spaces). Nevertheless, in each environment the agent qualitatively accomplishes the task. Even for challenging tasks with continuous action spaces, the agent is able to learn to accomplish the LTL specification.

Does LCER help in the learning process? According to the learning curves in the last row of Figure 2, LCER demonstrably expedites learning. In every environment with the exception of Carlo, LCER generates significant lift over lack of experience replay.

Intuition for why LCER helps? One way of viewing an LDBA is as a curriculum for what steps need to be taken in

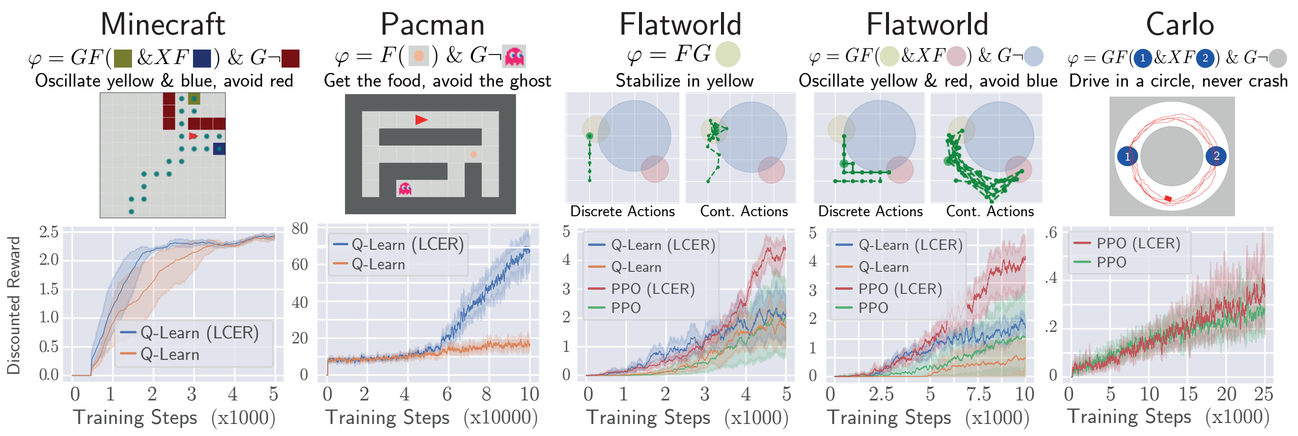

Figure 2. Results. Each column is an environment and a LTL formula we'd like an agent to satisfy. The environment and a trajectory from the final policy is illustrated in the center of the column (except for Pacman, which is the initial state). The learning curves at the bottom of each column show that adding off-policy data using `LCER` has strong benefits for empirical performance. ***First Column:*** Minecraft, where an agent should visit the yellow and blue areas while avoiding the red. The final policy is illustrated via blue dots. ***Second Column:*** Pacman, where an agent should collect the food while avoiding a ghost. ***Third Column:*** Flatword, where an agent should eventually stabilize in the yellow region. When the actions are discrete we use Q-learning, when the actions are continuous we use PPO. ***Fourth Column:*** Same as the third column except an agent should oscillate between the yellow and red regions while avoiding the blue. ***Fifth Column:*** Carlo, where an agent should drive in a circle without crashing by visiting the blue regions labelled 1 and 2 infinitely often.

order to accomplish a task. By replacing the LDBA state of the agent with some other dream LDBA state, we are allowing the agent to "pretend" that it has already accomplished some portion of the task.

As an example, consider the Flatworld example in Figure 2 with $\varphi = GF(y \ \& \ XF(r)) \ \& \ (G\neg b)$. A baseline agent (without `LCER`) would need to accomplish the entirety of the task in order to see any reward. However, an agent with counterfacual data, need only visit y from state 0 of the LDBA (see figure 1 for the LDBA). Then once the agent is really good at getting to y, it needs to learn how to reach r from state 2. After both of these tasks are accomplished, independently, the agent has solved the whole task. By placing the agent in state 0 of the LDBA, we are effectively letting the agent pretend that it has already visited r. In this sense, part of the task has been accomplished.

Limitations of LCER LCER can generate data outside of the "reachable" set of state-action pairs (when starting from the initial distribution). If all of the data generated by LCER is outside the reachable set, then LCER does not contribute meaningful off-policy experience. For a concrete example, consider the task "A then B then C" (e.g. "$A\&XF(B\&XFC))$"). Suppose the MDP is such that C can only be reached once B was reached which can only be reached once A was reached. Then the task and MDP are aligned in a worst-case way where any experience generated by swapping buchi states in the trajectories is technically outside of the "reachable" space.

We want to generate many trajectories containing some accepting state b^*, since these trajectories provide reward signal. However, it is likely that most of the trajectories generated by LCER contain no reward signal, imbalancing the dataset for training. Practitioners should pay attention to the data imbalance to avoid learning instability.

Finally, as the size of the product MDP increases, generating all possible trajectories using LCER may become computationally challenging. One way of generating only positive-signal trajectories would be to generate trajectories "backwards": set the buchi state at time t to $b_t = b^*$ and see which buchi state(s) b_{t-1} (if any) would have reached b_t under the taken action a_{t-1}. Then see which b_{t-2} would have reached the states b_{t-1} under the taken action a_{t-2}, etc. This procedure would generate all valid trajectories that reached $b_t = b^*$. Doing so for every $t <= T$ would give all "positive" signal trajectories. Further computational savings due to knowledge of the connectedness of the LDBA can be exploited. We leave these engineering optimizations to future work.

7. Related Work

Finding LTL-satisfying policies. Among the attempts at finding LTL-satisfying policies, Q-learning approaches have been the primary method of choice when the dynamics are unknown and Linear Programming methods when the dynamics are known (Sadigh et al., 2014; Hasanbeig et al., 2018; Bozkurt et al., 2020; Cai et al., 2021b; Ding et al., 2014). The Q-learning approaches are predominantly constrained to finite state-action spaces. Among the works that extend to continuous action spaces (Hasanbeig et al., 2020), DDPG is used and takes the form of hierarchical RL which is known to potentially find myopic policies (Toro Icarte et al., 2022).

Handling a subset of LTL specifications involving those expressible as finite expressions can also be addressed with Reward machines (Toro Icarte et al., 2022; Camacho et al., 2019; Vaezipoor et al., 2021). Our work handles ω-regular expressions, subsuming regular expressions. Many problems are ω-regular problems, but not regular, such as liveness (something good will happen eventually) and safety (nothing bad will happen forever).

On the formulation in Eq (3). Notable prior work on defining the value function as a function of the number of visits to $\mathcal{S}^{\mathcal{B}}$ and a state-dependent Γ_i function include Bozkurt et al. (2020); Cai et al. (2021a). Most notably, these authors use multiple different state-dependent discount rates that have a complicated relationships between them that needs to be satisfied in the limit as $\gamma \to 1^-$. Our work drastically simplifies this, getting rid of the technical assumptions, while strengthening the guarantees. This allows us to find a non-asymptotic dependence on the suboptimality of a policies' probability of LTL satisfaction as a function of γ.

Off-policy data. One may view the counterfactual samples in Toro Icarte et al. (2022) as an instantiation of LCER, limited to finite LTL expressions and discrete action spaces. Extension to continuous action space and full LTL requires a careful treatment. In the continuous action and full LTL setting, (Wang et al., 2020) incorporate starting the agent from a different initial LDBA state (than b_{-1}) which is still on-policy but from a different starting state and doesn't take advantage of the entire LDBA structure. This work can be seen as complimentary to our own.

Theory. Works with strong theoretical guarantees on policy satisfaction include Fu & Topcu (2014); Wolff et al. (2012); Voloshin et al. (2022) but are once again limited to discrete state/action spaces. Extensions of these work to continuous state space is not trivial as they make heavy use of the discrete Markov chain structure afforded to them.

8. Discussion

Our work, to the best of our knowledge, is the first to make full use of the LDBA as a form of experience replay and first to use policy gradient to learn LTL-satisfying policies. Our eventual discounting formulation is unrestricted to Finitary fragments of LTL like most prior work.

Despite the guarantees afforded to us by eventual discounting, in general the problem given in Eq (2) is not PAC learnable (Yang et al., 2022). Though, like SAT solvers, it is still useful to find reasonable heuristics to problems that are difficult. We show that under particular circumstances, eventual discounting gives a signal on the quantity of interest in (2) and even when it fails, it selects a policy that is difficult to differentiate from a successful one. Further, the bad news discussed in Section 4.2 we speculate is unavoidable in general LTL specifications, without significant assumptions on the MDP. For example, for stability problems in LTL and assuming control-affine dynamics then Lyapunov functions can serve as certificates for a policies' LTL satisfaction. A reasonable relaxation to this would be require a system to behave a certain way for a long, but finite amount of time.

Acknowledgements. Cameron Voloshin is funded partly by an NSF Graduate Research Fellowship. This work is also supported in part by NSF #1918865, ONR #N00014-20-1-2115, and NSF #2033851.

References

Abel, D., Dabney, W., Harutyunyan, A., Ho, M. K., Littman, M., Precup, D., and Singh, S. On the expressivity of markov reward. In *Advances in Neural Information Processing Systems*, 2021. URL https://proceedings.neurips.cc/paper/2021/file/4079016d940210b4ae9ae7d41c4a2065-Paper.pdf.

Baier, C. and Katoen, J.-P. *Principles of model checking*. The MIT Press, Cambridge, Mass, 2008. ISBN 978-0-262-02649-9.

Bozkurt, A. K., Wang, Y., Zavlanos, M. M., and Pajic, M. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10349–10355, 2020. doi: 10.1109/ICRA40945.2020.9196796.

Cai, M., Hasanbeig, M., Xiao, S., Abate, A., and Kan, Z. Modular deep reinforcement learning for continuous motion planning with temporal logic. *IEEE Robotics and Automation Letters*, 6(4):7973–7980, 2021a.

Cai, M., Xiao, S., Li, Z., and Kan, Z. Optimal probabilistic motion planning with potential infeasible ltl constraints. *IEEE Transactions on Automatic Control*, pp. 1–1, 2021b. doi: 10.1109/TAC.2021.3138704.

Camacho, A., Toro Icarte, R., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 6065–6073. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/840. URL https://doi.org/10.24963/ijcai.2019/840.

Ding, X., Smith, S. L., Belta, C., and Rus, D. Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control*, 59(5):1244–1257, May 2014. ISSN 0018-9286, 1558-2523. doi: 10.1109/TAC.2014.2298143.

Fu, J. and Topcu, U. Probably approximately correct MDP learning and control with temporal logic constraints. In Fox, D., Kavraki, L. E., and Kurniawati, H. (eds.), *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*, 2014. doi: 10.15607/RSS.2014.X.039. URL http://www.roboticsproceedings.org/rss10/p39.html.

Hahn, E. M., Li, G., Schewe, S., Turrini, A., and Zhang, L. Lazy probabilistic model checking without determinisation. *arXiv preprint arXiv:1311.2928*, 2013.

Hasanbeig, M., Abate, A., and Kroening, D. Logically-constrained reinforcement learning, 2018. URL https://arxiv.org/abs/1801.08099.

Hasanbeig, M., Kroening, D., and Abate, A. Deep reinforcement learning with temporal logics. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 1–22. Springer, 2020.

Hasselt, H. v., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2094–2100. AAAI Press, 2016.

Ibarz, B., Leike, J., Pohlen, T., Irving, G., Legg, S., and Amodei, D. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems*, 31, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Křetínskỳ, J., Meggendorfer, T., and Sickert, S. Owl: a library for ω-words, automata, and ltl. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 543–550. Springer, 2018.

Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321, 1992.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287, 1999.

Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Randløv, J. and Alstrøm, P. Learning to drive a bicycle using reinforcement learning and shaping. In *ICML*, 1998.

Sadigh, D., Kim, E. S., Coogan, S., Sastry, S. S., and Seshia, S. A. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pp. 1091–1096, 2014. doi: 10.1109/CDC.2014.7039527.

Sickert, S., Esparza, J., Jaax, S., and Křetínský, J. Limit-deterministic büchi automata for linear temporal logic. In Chaudhuri, S. and Farzan, A. (eds.), *Computer Aided Verification*, pp. 312–332, Cham, 2016. Springer International Publishing. ISBN 978-3-319-41540-6.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

Sorg, J. *The Optimal Reward Problem: Designing Effective Reward for Bounded Agents*. PhD thesis, University of Michigan, USA, 2011. URL https://hdl.handle.net/2027.42/89705.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Toro Icarte, R., Klassen, T. Q., Valenzano, R., and McIlraith, S. A. Reward machines: Exploiting reward function structure in reinforcement learning. *J. Artif. Int. Res.*, 73, may 2022. ISSN 1076-9757. doi: 10.1613/jair.1.12440. URL https://doi.org/10.1613/jair.1.12440.

Toromanoff, M., Wirbel, E., and Moutarde, F. Is deep reinforcement learning really superhuman on atari? leveling the playing field. *arXiv preprint arXiv:1908.04683*, 2019.

Vaezipoor, P., Li, A. C., Icarte, R. T., and McIlraith, S. A. Ltl2action: Generalizing LTL instructions for multi-task RL. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10497–10508, 2021. URL http://proceedings.mlr.press/v139/vaezipoor21a.html.

Voloshin, C., Le, H. M., Chaudhuri, S., and Yue, Y. Policy optimization with linear temporal logic constraints. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=yZcPRIZEwOG.

Wang, C., Li, Y., Smith, S. L., and Liu, J. Continuous motion planning with temporal logic specifications using deep neural networks, 2020. URL https://arxiv.org/abs/2004.02610.

Wolff, E. M., Topcu, U., and Murray, R. M. Robust control of uncertain markov decision processes with temporal logic specifications. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 3372–3379, 2012. doi: 10.1109/CDC.2012.6426174.

Yang, C., Littman, M. L., and Carbin, M. On the (in)tractability of reinforcement learning for ltl objectives. In Raedt, L. D. (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 3650–3658. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/507. URL https://doi.org/10.24963/ijcai.2022/507. Main Track.

Zhang, B., Rajan, R., Pineda, L., Lambert, N., Biedenkapp, A., Chua, K., Hutter, F., and Calandra, R. On the importance of hyperparameter optimization for model-based reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, 2021.

A. Experiments

A.1. Environment Details

The environment and experiment details are summarized in Table 1.

Table 1. Environment Details

Environment	Experiment	$\mathcal{S}^{\mathcal{M}}$	$\mathcal{A}^{\mathcal{M}}$	Dynamics	LTL Formula
Minecraft	Q-learning	Discrete	Discrete	Deterministic	$GF(y \ \& \ XF(b)) \ \& \ (G\neg r)$
Pacman	Q-learning	Discrete	Discrete	Stochastic	$F(\text{food}) \ \& \ (G\neg\text{ghost})$
Flatworld 1	Q-learning	$\mathbb{R}^2$	Discrete	Deterministic	FGy
Flatworld 2	Q-learning	$\mathbb{R}^2$	Discrete	Deterministic	$GF(y \ \& \ XF(r)) \ \& \ (G\neg b)$
Flatworld 3	PPO	$\mathbb{R}^2$	$[0,1]^2$	Deterministic	FGy
Flatworld 4	PPO	$\mathbb{R}^2$	$[0,1]^2$	Deterministic	$GF(y \ \& \ XF(r)) \ \& \ (G\neg b)$
Carlo	PPO	$\mathbb{R}^5$	$[-1,1]^2$	Deterministic	$GF(\text{zone1} \ \& \ XF(\text{zone2})) \ \& \ (G\neg\text{crash})$

A.2. Experiment Setup

Each experiment is run with 10 random seeds. Results from Figure 2 are from an average over the seeds.

Q-learning experiments. Let k be the greatest number of jump transitions available in some LDBA state $k = \max_{b \in \mathcal{S}^{\mathcal{B}}} |\mathcal{A}^{\mathcal{B}}(b)|$. Let $m = \max_{s \in \mathcal{S}^{\mathcal{M}}} |\mathcal{A}^{\mathcal{M}}(s)|$. The neural network $Q_\theta(s)$ takes as input $s \in \mathcal{S}^{\mathcal{M}}$ and outputs $\mathbb{R}^{(m+k)\times|\mathcal{S}^{\mathcal{B}}|}$ a $(m+k)$-dim vector for each $b \in \mathcal{S}^{\mathcal{B}}$. For our purposes, we consider $Q_\theta(s,b)$ to be the single $(m+k)$-dim vector cooresponding to the particular current state of the LDBA b.

When $\mathcal{S}^{\mathcal{M}}$ is discrete then we parametrize $Q_\theta(s,b)$ as a table. Otherwise, $Q_\theta(s,b)$ is parameterized by 3 linear layers with hidden dimension 128 with intermediary ReLU activations and no final activation. After masking for how many jump transitions exist in b, we can select $\arg\max_{i \in [0,\ldots,|\mathcal{A}^{\mathcal{B}}(b)|]} Q_\theta(s,b)_i$ the highest Q-value with probability $1-\eta$ and uniform with η probability. Here, η is initialized to η_0 and decays linearly (or exponentially) at some specified frequency (see Table 2).

At each episode (after a rollout of length T), we perform K gradient steps with different batches of size given in Table 3. We use Adam optimizer (Kingma & Ba, 2015) with a learning rate also specified by the table.

When in a continuous state space, we implement DDQN (Hasselt et al., 2016) (rather than DQN) with a target network that gets updated at some frequency specified by Table 3.

Table 2. Hyperparameters for Q-learning experiments (Discrete Action Space)

	η		η Decay								
Experiment	η_0	Min η	Type	Rate	Freq	Batch size	K (# batches)	LR	Target update	T	γ
Minecraft	.3	0	Exponential	.9	100	128	20	-	-	100	.99
Pacman	.4	0	Linear	.05	400	512	200	-	-	100	.999
Flatworld 1	.8	.15	Exponential	.9	100	128	5	.001	15	20	.95
Flatworld 2	.8	.15	Exponential	.9	100	128	5	.001	15	50	.95

PPO experiments. Let k be the greatest number of jump transitions available in some LDBA state $k = \max_{b \in \mathcal{S}^{\mathcal{B}}} |\mathcal{A}^{\mathcal{B}}(b)|$. The neural network $f_\theta(s)$ takes as input $s \in \mathcal{S}^{\mathcal{M}}$ and outputs $\mathbb{R}^{(k+2)\times|\mathcal{S}^{\mathcal{B}}|}$ is a $(k+2)$-dim vector for each $b \in \mathcal{S}^{\mathcal{B}}$. For our purposes, we consider $f_\theta(s,b)$ to be the single $(k+2)$-dim vector cooresponding to the particular current state of the LDBA b.

$f_\theta(s,b)$ is parameterized by 3 linear layers with hidden dimension 64 with intermediary ReLU activations. The first dimension corresponds to sampling a Gaussian action $a \sim \mathcal{N}(f_\theta(s,b)[0], \text{diag}(\sigma^2))$ where σ is initialized to σ_0 (see Table 3) and decays exponentially (at a rate given in the table) every 10 episodes. The remaining $k+1$ dimensions (after proper masking to account for the size of $|\mathcal{A}^{\mathcal{B}}(b)|$ and softmax) represent the probability $p = [p_a, p_{\epsilon_0}, \ldots, p_{\epsilon_k}]$ of taking either the MDP action a or a some jump transition ϵ_i. We sample from a Categorical(p) variable to select whether to return

$a \sim \mathcal{N}(\text{Tanh}(f_\theta(s,b)[0]), \text{diag}(\sigma^2))$ or $a = \epsilon_i$ for some i. The density can be calculated by multiplying p_a by the Gaussian density when a is selected, and p_{ϵ_i} otherwise.

For the critic, we have a parametrized network $f_\phi(s,b) \to \mathbb{R}$ of 3 linear layers with hidden dimension 64 with intermediary Tanh activations and no final activation.

At each episode (after a rollout of length T), we perform 5 gradient steps with different batches of size given in Table 3. The importance sampling term in PPO is clipped to $1 \pm .4$. The critic learning rate is $.01$. We use Adam optimizer (Kingma & Ba, 2015) for both the actor and critic.

Table 3. Hyperparameters for PPO experiments (Continuous Action Space)

Experiment	σ_0	σ Decay Rate	Min σ	Batch size	LR Actor	T
Flatworld 3	1.8	.98	.3	128	.001	20
Flatworld 4	1.8	.99	.1	128	.001	50
Carlo	.5	.999	.3	16	.0001	500

B. Constructing feasible trajectories for policy gradient during rollout

Suppose we wanted to generate feasible trajectories in realtime while the policy is being rolled out. That is, we have a partial trajectory of the form $\tau_t = (s_0, b_0, a_0, \ldots, s_t, b_t)$ generated by running π in P. Let $a_t = a \in \mathcal{A}$ be the t-th action taken by π and $s_{t+1} = s' \in \mathcal{M}$ be the next observed state observed in the MDP.

Let $\mathcal{T}_t$ be the current set of feasible (partial) trajectories at timestep t. Elements $\tau_k = (s_0, b_0, a_0, \ldots, s_k, b_k) \in \mathcal{T}_t$ denote k-step (partial) trajectory, not necessarily part of the trajectory observed during the course of a rollout of π. Here, $k \geq t$. Then, for each $\tau_k \in \mathcal{T}_t$, one of 4 cases holds:

Case 1. Action a is not a jump transition (ie. $a \in \mathcal{A}^{\mathcal{M}}(s_k)$) and there are no jump transitions available in b_k ($\mathcal{A}^{\mathcal{B}}(b_k) = \varnothing$). Then we can form the concatenation: $\tau_{k+1} = \tau_k \cup (a, s', b_{k+1})$ where $b_{k+1} = P^{\mathcal{B}}(b_k, L^{\mathcal{M}}(s'))$. We set $\mathcal{T}_\epsilon = \varnothing$.

Case 2. Action a is a jump transition and is currently feasible in b_k (ie. $a \in \mathcal{A}^{\mathcal{M}}(b_k)$). Then we can form the concatenation $\tau_{k+1} = \tau_k \cup (a, s', b_{k+1})$ where $b_{k+1} = P^{\mathcal{B}}(b_k, a)$. We set $\mathcal{T}_\epsilon = \varnothing$.

Case 3. Action a is a not a jump transition (ie. $a \in \mathcal{A}^{\mathcal{M}}(s_k)$), but there is at least one feasible jump transition in b_k (ie. $\mathcal{A}^{\mathcal{B}}(b_k) \neq \varnothing$). Then, in addition to forming τ_{k+1} from Case 1, we have all the possible jumps:

$$\mathcal{T}_\epsilon = \{\tau_k \cup (\epsilon, s_k, b_{k+1}, a, s', b_{k+2}) | \forall \epsilon \in \mathcal{A}^{\mathcal{B}}(b_k), b_{k+1} = P^{\mathcal{B}}(b_k, \epsilon), b_{k+2} = P^{\mathcal{B}}(b_{k+1}, a_t)\}$$

Case 4. Action a is a jump transition is infeasible in b_k (ie. $a \notin \mathcal{A}^{\mathcal{B}}(b_k)$). In this case, we just pass this trajectory. Setting $\tau_{k+1} = \tau_k$ and $\mathcal{T}_\epsilon = \varnothing$.

At the end of iterating over each element of $\tau_k \in \mathcal{T}_t$ and forming τ_{k+1} and $\mathcal{T}_\epsilon$, we can update our current set of feasible trajectories:

$$\mathcal{T}_{t+1} = \cup_{\tau_k \in \mathcal{T}_t} \Big((\mathcal{T}_t \setminus \{\tau_k\}) \cup \{\tau_{k+1}\} \cup \mathcal{T}_\epsilon \Big) \quad (19)$$

To put this process simply, we are swapping out τ_k for τ_{k+1} and also adding in any jump transitions if they are available. The algorithm can be seen in Algo 4.

Algorithm 4 LCER for Policy Gradient (Option 2)

```
Param: Dataset $D$. Trajectory $\tau$ of length $T$.
1: Set $\mathcal{T}_0 \leftarrow \{(s_0, b) | b \in \mathcal{B}\}$
2: for $(s_t, a_t, s_{t+1}) \in \tau$ do
3:     Form $\mathcal{T}_t$ according to Eq (19)
4: end for
5: Set $D \leftarrow D \cup \mathcal{T}_T$
6: return $D$
```