Learning from Demonstrations with High-Level Side Information*

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

Ivan Papusha
University of Texas at Austin
ipapusha@utexas.edu

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

Abstract

We consider the problem of learning from demonstration, where extra side information about the demonstration is encoded as a co-safe linear temporal logic formula. We address two known limitations of existing methods that do not account for such side information. First, the policies that result from existing methods, while matching the expected features or likelihood of the demonstrations, may still be in conflict with high-level objectives not explicit in the demonstration trajectories. Second, existing methods fail to provide a priori guarantees on the out-of-sample generalization performance with respect to such high-level goals. This lack of formal guarantees can prevent the application of learning from demonstration to safety-critical systems, especially when inference to state space regions with poor demonstration coverage is required. In this work, we show that side information, when explicitly taken into account, indeed improves the performance and safety of the learned policy with respect to task implementation. Moreover, we describe an automated procedure to systematically generate the features that encode side information expressed in temporal logic.

1 Introduction

Learning from demonstration [Argall *et al.*, 2009], also referred to as imitation learning or apprenticeship learning [Abbeel and Ng, 2004], aims at learning a *policy* to implement some *task*, using samples of an expert's behaviors as demonstrations. There is a wide range of applications of learning from demonstration in robotics, such as navigation and manipulation tasks.

One common approach to learning from demonstration is inverse reinforcement learning (IRL) [Ng *et al.*, 2000], in which the agent relies on rewards to interpret the expert's behaviors. The environment is modeled as a Markov decision process (MDP) with known transition dynamics. Given the environment MDP and expert's demonstrations as trajectories, IRL recovers a reward function and constructs policies based on the estimated reward function. Formulations of IRL in the literature differ primarily in their interpretation of expert demonstrations, or the "similarity" between the expert's policy and desired policies expressed in terms of rewards. Some common assumptions are, for example, that both the expert's policy and all desired policies are optimal [Abbeel and Ng, 2004; Ratliff *et al.*, 2006; Ramachandran and Amir, 2007; Dvijotham and Todorov, 2010]; or the expected total rewards of output policies should match the sample mean of total rewards of trajectories in demonstrations [Ziebart *et al.*, 2008; Boularias *et al.*, 2011; Bloem and Bambos, 2014].

Although human experts can directly provide low-level demonstrations to implicitly specify the learning task, it is usually beneficial to explicitly indicate high-level task requirements, which we naturally rely on to assess the performance of the learned policies. A high-level task can be "grasp an object without touching anything else," or "obey traffic lights and road signs while driving from A to B," which may not be sufficient to encode all desired properties of an ideal policy, but is crucial to the task performance. Existing IRL methods do not infer underlying high-level tasks and thus the agent's behavior at newly visited states may not satisfy the actual task requirements.

In this work, we join the strengths of high-level task requirements and demonstrations and formalize the problem of IRL with high-level side information. Given task specification as a co-safe Linear Temporal Logic (LTL) formula, and a collection of optimal expert demonstrations consistent with the task specification, we describe a learning framework that recovers both a reward function, as well as a deterministic finite automaton (DFA), which together guarantee a quantitative level of probability that the learned policy will satisfy the task. Crucially, the addition of an LTL side specification allows us to learn general policies that work even when the expert examples are scarce.

Following the many applications of formal methods to robotics and control [Fainekos *et al.*, 2005; Kress-Gazit *et al.*, 2011; Wolff *et al.*, 2012; Bobadilla *et al.*, 2012], we encode the task requirements in LTL [Pnueli, 1981], which is an expressive formal logic suitable for task requirements. These include reaching-a-goal, stability, obstacle avoidance, sequen-

*This work was supported in part by AFRL # FA8650-15-C-2546, DARPA # W911NF-16-1-0001, ARO # W911NF-15-1-0592, NSF # 1651089 and NSF # 1550212.

tially visiting regions of interest, and conditional reactive behaviors. Generally, LTL specifications can be evaluated on trajectories with infinite length; but since expert demonstrations are finite, we focus on a set of tasks to be implemented in finite time, which can be specified by a subset of LTL called co-safe LTL [Kupferman and Vardi, 2001].

We adopt the framework of maximum-likelihood inverse reinforcement learning (MLIRL) [Babes *et al.*, 2011] as a baseline approach, and learn policies in both the original environment MDP and in the product space of the MDP and the specification automaton. We further propose an algorithm that evaluates the learned policy using the co-safe LTL formula during learning. We report numerical results on a navigation example, in which policies are learned with MLIRL, MLIRL with a specification automaton, and with our own algorithm. We show that the learned policy benefits from both the construction of product automaton and the evaluation with co-safe LTL formula, because it attains higher probabilities of successfully implementing the task, and provides formal guarantees on task completion even in regions of state space not covered by the expert examples.

2 Notation and Preliminaries

For a finite set B and a nonnegative integer $l \in \mathbb{N}^+$, define B^l as the set of all sequences of length l composed by elements in B. In addition, define B^* (resp. B^ω) as the set of all finite (infinite) sequences composed by elements of B. Finally, define $\mathcal{D}(B)$ as the set of all probability distributions over B.

2.1 MDP and Policies

Let $\mathcal{M} = \langle S, S_I, A, T, R, \gamma \rangle$ be a *Markov decision process (MDP)*, where S is a finite set of states; $S_I \subseteq S$ is a set of initial states; A is a finite set of actions; $T : S \times A \times S \to [0, 1]$ is a transition function such that for each $(s, a) \in S \times A$, $T(s, a, \cdot) \in \mathcal{D}(S)$; $R : S \times A \to \mathbb{R}$ is a reward function, and $\gamma \in (0, 1)$ is a discounting factor.

A *path* or *trajectory* τ of $\mathcal{M}$ is an infinite alternating sequence of states and actions, $\tau = s_0, a_0, s_1, a_1, \ldots$, such that $s_0 \in S_I$, and for all $k \geq 0$, we have $a_k \in A$ and $T(s_k, a_k, s_{k+1}) > 0$. Given two states $s, s' \in S$, we say s' is *reachable* from s, denoted by $s \rightsquigarrow s'$, if and only if there exists a path $\tau = s_0, a_0, s_1, a_1, \ldots$ with $s = s_i$ and $s' = s_j$ for some integers $0 \leq i \leq j$. For any set of states $B \subseteq S$, define $Reach(B) = \{s' \in S : \exists s \in B, s' \rightsquigarrow s\}$ as the set of states from which B is reachable.

A (memoryless) *policy* π for $\mathcal{M}$ is a mapping from states to distributions over actions: $\pi : S \times A \to [0, 1]$ such that for any $s \in S$, $\pi(s, \cdot) \in \mathcal{D}(A)$. Given any policy π, we can define a state value function $V_\pi : S \to \mathbb{R}$ such that for each state $s \in S$, $V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \mid s_0 = s\right]$ is the expected future discounted reward that an agent can get by applying policy π from state s. Correspondingly, we can define an action value function $Q_\pi : S \times A \to \mathbb{R}$ such that for any state-action pair $(s, a) \in S \times A$, $Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \mid s_0 = s, a_0 = a\right]$ is the expected discounted reward if the agent takes policy π after taking action a from state s. The functions V_π, Q_π, R, and π satisfy the Bellman relations:

$$V_\pi(s) = \sum_a \pi(s, a) Q_\pi(s, a),$$

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') V_\pi(s').$$

These two equations can be combined into

$$Q_\pi(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \sum_{a'} \pi(s', a') Q_\pi(s', a'). \quad (1)$$

2.2 LTL Specifications

In order to evaluate policies with LTL specifications, we attach labels to states. The labels, consisting of *atomic propositions*, are boolean variables defined on S. Let AP be a set of atomic propositions. The *labeling function* $\mathcal{L} : S \to 2^{AP}$ maps each state $s \in S$ to its labels $\mathcal{L}(s) \subseteq AP$, which is the set of atomic propositions that are true at state s. With slight abuse of notation, we also use $\mathcal{L}(\tau)$ to denote the sequence of atomic propositions that hold at states in path $\tau = s_0, a_0, s_1, a_1, \cdots$ of $\mathcal{M}$, i.e., $\mathcal{L}(\tau) = \mathcal{L}(s_0), \mathcal{L}(s_1), \cdots$.

An LTL formula φ over AP is defined recursively by:

$$\varphi := \text{true} \mid p \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \mathtt{X}\,\varphi_1 \mid \varphi_1 \,\mathtt{U}\, \varphi_2,$$

where $p \in AP$, and φ_1, φ_2 are LTL formulas. The logical and temporal operators above can be combined to define other useful operators such as $\wedge$, $\rightarrow$, $\mathtt{G}$ and $\mathtt{F}$. See [Pnueli, 1981] for a detailed explanation of the semantics of LTL.

In general, an LTL formula φ is evaluated on $(2^{AP})^\omega$, i.e., infinite sequences of elements in 2^{AP}. To better suit the need to encode tasks that are implemented over finite horizons, we focus on a subset of LTL formulas, namely co-safe LTL formulas. These formulas are characterized by the key feature that every (infinite) sequence that satisfies the formula has a finite prefix [Kupferman and Vardi, 2001]. A wide range of learning from demonstration tasks that can be encoded as co-safe LTL formulas, for example: $\varphi_1 = (\neg\text{obstacle}\,\mathtt{U}\,\text{goal}) \wedge \mathtt{F}\,\textit{goal}$ means "reach the goal without running into obstacles," and $\varphi_2 = ((\neg\text{object}_2)\,\mathtt{U}\,\text{object}_1) \wedge (\mathtt{F}\,\text{object}_1) \wedge (\mathtt{F}\,\text{object}_2)$ means "grab object 1 first and then grab object 2."

Given a co-safe LTL formula φ_{cs}, we can construct a (non-unique) deterministic finite automaton (DFA) $\mathcal{A}_{\varphi_{\text{cs}}} = \langle Q, q_I, Q_F, 2^{AP}, \delta \rangle$ that accepts the finite prefixes all runs that satisfy φ_{cs}, where Q is a finite set of states, $q_I \in Q$ is the initial state, $Q_F \subseteq Q$ is a set of accepting (final) states, 2^{AP} is the alphabet, and $\delta : Q \times 2^{AP} \to Q$ is a deterministic transition function. All states in Q_F are absorbing states, i.e., for all $L \in 2^{AP}, q \in Q_F, \delta(q, L) = q$.

3 Maximum-Likelihood Inverse Reinforcement Learning (MLIRL)

We adopt the framework of maximum-likelihood inverse reinforcement learning (MLIRL) [Babes *et al.*, 2011] as the baseline algorithm that does not use any high-level side information. In this section we introduce the key components of MLIRL: policy structure, reward parameterization, and optimization objective.

Softmax policy We restrict the policy search to the subclass of policies for which the probability to take an action a at state s is a softmax function of the action-value function. For any function $Q : S \times A \to \mathbb{R}$, define the *softmax* policy as

$$\pi_Q(s,a) := \frac{\exp(Q(s,a))}{\sum_{\tilde{a}} \exp(Q(s,\tilde{a}))}, \quad \forall (s,a) \in S \times A. \quad (2)$$

The softmax policy, as a special case of the Boltzmann exploration policy [John, 1994], has been used in several instances of IRL [Neu and Szepesvári, 2007; Babes *et al.*, 2011; Macglashan and Littman, 2015]. It defines a valid distribution π_Q for all Q, i.e., $\pi_Q(s,a) \geq 0$ for all $s \in S$ and $a \in A$, and $\sum_{a \in A} \pi_Q(s,a) = 1$ for all $s \in S$. It is also smooth in the components of Q, allowing easy computation of a policy gradients. With the softmax policy, the agent prefers to select actions with higher action-values, but still has the freedom to explore suboptimal actions. Such freedom is particularly important in accommodating any inconsistency in the expert's demonstrations.

Reward parameterization The reward function R of $\mathcal{M}$ is approximated by a linear combination of k pre-designed features, with parameter $\theta \in \mathbb{R}^{k \times 1}$. We denote the feature matrix as $F = [\mathbf{f}_1, \cdots \mathbf{f}_k] \in \mathbb{R}^{|S| \cdot |A| \times k}$ with $\mathbf{f}_i$ representing the i^{th} reward feature vector. For convenience, we denote the row of F corresponding to the state-action pair (s,a) as $F(s,a) \in \mathbb{R}^{1 \times k}$. The overall reward matrix is $R = F\theta$ for some feature weight $\theta \in \mathbb{R}^k$. Substituting the softmax policy and the reward matrix into (1) yields

$$\begin{aligned} Q(s,a) =& F(s,a)\theta + \\ & \gamma \sum_{s'} T(s,a,s') \sum_{a'} \pi_Q(s',a') Q(s',a'). \end{aligned} \quad (3)$$

In the following, we treat θ as the free variable, and denote the action value function Q and policy π_Q satisfying (2) and (3) as Q_θ and π_θ.

Expert demonstrations The demonstrations consist of a set $D = \{\tau_1, \cdots \tau_m\}$ of m finite prefixes of trajectories in $\mathcal{M}$. For each $l \in \{1, \cdots m\}$, $\tau_l = s_{l,0}, a_{l,0}, \cdots s_{l,t_l}, a_{l,t_l}$ is the l^{th} demonstration trajectory, which is an ordered sequence of $t_l + 1$ state-action pairs. We refer to such demonstrated trajectories as expert trajectories.

Maximum-likelihood objective The goal is to find θ and an induced policy π_θ that maximize the likelihood of observing the expert demonstrations. An equivalent optimization objective is to minimize the negative log-likelihood

$$\begin{aligned} J^{\text{mle}}(\theta \mid \mathcal{M}, D) &:= -\sum_{l=1}^{m} \sum_{t=1}^{t_l} \log \pi_\theta(s_{l,t}, a_{l,t}) \\ &= -\sum_{l=1}^{m} \sum_{t=1}^{t_l} \Big(Q_\theta(s_{l,t}, a_{l,t}) - \log \big(\sum_{\tilde{a}} \exp(Q_\theta(s_{l,t}, \tilde{a})) \big) \Big), \end{aligned} \quad (4)$$

with equality constraints given by Eq. (2) and (3). The objective function is smooth and convex in θ, but regularization on θ may be needed to avoid separation problems, and to get a finite solution [Albert and Anderson, 1984].

4 MLIRL with High-Level Side Information

Assume that in addition to the standard inputs to MLIRL problems, i.e., a reward-free MDP $\mathcal{M}$, expert demonstrations D, and feature matrix F, we also know some high-level task requirements encoded as a co-safe LTL formula φ_{cs}. This side information is utilized in two steps: we first extend the original MDP $\mathcal{M}$ into a product automaton incorporating the task structure, and then augment the optimization objective by explicitly evaluating the policy.

4.1 Extending the State Space

An implicit assumption in all MDP-based IRL methods is that the expert's policy is memoryless, i.e., the distribution of the next action is decided by the current state and independent on trajectory history. The assumption breaks if the task has some hierarchical structure and can be easily decomposed into several sub-tasks, which is a common case in practice. Side information as high-level task requirements can be used to generate memory states automatically, which enables us to construct a product automaton $M_{\varphi_{\text{cs}}}$ with the original environment $\mathcal{M}$ and a DFA $\mathcal{A}_{\varphi_{\text{cs}}}$. Then we learn a memoryless policy over the extended state space of $M_{\varphi_{\text{cs}}}$.

Given $\mathcal{A}_{\varphi_{\text{cs}}}$ and $\mathcal{M}$, define the *product automaton* $M_{\varphi_{\text{cs}}} = \langle \bar{S}, \bar{S}_I, \bar{S}_F, A, \bar{T}, \gamma \rangle$, where $\bar{S} = S \times Q$ is a finite state space; $\bar{S}_I = S_I \times q_I$ is the set of initial states; $\bar{S}_F = S \times Q_F$ is the set of final states; $\bar{T} : \bar{S} \times A \times \bar{S} \to [0,1]$ is a transition function such that for any $(s,q), (s',q') \in \bar{S}, a \in A$, $\bar{T}((s,q), a, (s',q')) = T(s,a,s')$ if $\delta(q, \mathcal{L}(s')) = q'$ and 0 otherwise. Policies in $M_{\varphi_{\text{cs}}}$ can be defined analogously to those in $\mathcal{M}$. Similar to the evaluation of $\mathcal{A}_{\varphi_{\text{cs}}}$, a finite path $\tau_M = (s_0,q_0), a_0, (s_1,q_1), a_1 \cdots (s_l,q_l), a_l \in (\bar{S} \times A)^{l+1}$ of $M_{\varphi_{\text{cs}}}$ satisfies φ_{cs} if and only if $(s_l,q_l) \in \bar{S}_F$, or equivalently $q_l \in Q_F$.

Any finite (resp. infinite) trajectory in $\mathcal{M}$ can be uniquely mapped to a trajectory of equal length in the product automaton $M_{\varphi_{\text{cs}}}$. We define an operator $h(\cdot \mid \mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}}) : (S \times A)^* \to (\bar{S} \times A)^*$ to translate finite trajectories in $\mathcal{M}$ into the corresponding trajectories in the product automaton $M_{\varphi_{\text{cs}}}$. The operator h enables us to interpret the demonstrations D in $M_{\varphi_{\text{cs}}}$. For any $\tau_l \in D$, define

$$h(\tau_l \mid \mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}}) = \bar{s}_{l,0}, a_{l,0}, \bar{s}_{l,1}, a_{l,1}, \cdots, \bar{s}_{l,t_l}, a_{l,t_l}$$

such that $\bar{s}_{l,0} := (s_{l,0}, q_I)$ and for $j = 1, \cdots, t_l$, $\bar{s}_{l,j} := (s_{l,j}, \delta(q_{l,j-1}, \mathcal{L}(s_{l,j})))$. Any trajectory in $M_{\varphi_{\text{cs}}}$ can be uniquely projected to a trajectory in $\mathcal{M}$, simply by dropping the second component of each state in $\bar{S}$. In the following, we assume that the learning procedure occurs in the product automaton in order to take advantage of the side information. For simplicity we use $h(D \mid \mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}}) := \{h(\tau_l \mid \mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}}) : \tau_l \in D\}$ to represent the set of projected trajectories of D in $M_{\varphi_{\text{cs}}}$.

The construction of $\mathcal{A}_{\varphi_{\text{cs}}}$ and $M_{\varphi_{\text{cs}}}$ is internal to the learning algorithm and may not be accessible by the expert. Correspondingly the agent has no access to the expert policy. The

only shared inputs between the agent and expert are the high-level task specification φ_{cs}, the environment dynamics $\mathcal{M}$, and the set D of demonstrated trajectories in $\mathcal{M}$. Any equivalent DFA for φ_{cs} works in principle, except with varying computation time due to possibly different sizes of $M_{\varphi_{\text{cs}}}$.

4.2 Augmenting Objective with Side Information

In order to guarantee the performance of the learned policy, we explicitly compute the probability of satisfying φ_{cs} from all valid initial states. This is done by computing a function $\bar{y}(\cdot \mid \bar{\pi}) : \bar{S} \to [0,1]$ such that $\bar{y}(\bar{s} \mid \bar{\pi})$ is the probability of satisfying φ_{cs} by taking policy $\bar{\pi}$ from initial state $\bar{s}$. By a result in model checking [Baier *et al.*, 2008],

$$\bar{y}(\bar{s} \mid \bar{\pi}) = \begin{cases} 1, & \text{if } \bar{s} \in \bar{S}_F, \\ 0, & \text{if } \bar{s} \notin \mathit{Reach}(\bar{S}_F), \\ \sum_{a \in A} \bar{\pi}(\bar{s}, a) \sum_{\bar{s}' \in \bar{S}} T(\bar{s}, a, \bar{s}') \bar{y}(\bar{s}' \mid \bar{\pi}), & \text{otherwise.} \end{cases} \tag{5}$$

There is a unique $\bar{y}(\cdot \mid \bar{\pi})$ for any given $\bar{\pi}$; it can be obtained either by linear programming, or by computing the least fixed point of the operator

$$\Gamma_{\bar{\pi}}(\bar{y})(\bar{s}) = \begin{cases} 1, & \text{if } \bar{s} \in \bar{S}_F, \\ 0, & \text{if } \bar{s} \notin \mathit{Reach}(\bar{S}_F), \\ \sum_{a \in A} \bar{\pi}(\bar{s}, a) \sum_{\bar{s}' \in \bar{S}} T(\bar{s}, a, \bar{s}') \bar{y}(\bar{s}' \mid \bar{\pi}), & \text{otherwise.} \end{cases}$$

Assume $\bar{y}^{(0)}(s) = 0$ for all $\bar{s} \in \bar{S} \backslash \bar{S}_F$ and $\bar{y}^{(0)}(s) = 1$ for all $s \in \bar{S}_F$, and $\bar{y}^{(k)}$ is updated as $\bar{y}^{(k+1)} = \Gamma_{\bar{\pi}}(\bar{y}^{(k)})$ for all $k \in \mathbb{N}$, then it can be shown that $\lim_{k \to +\infty} \bar{y}^{(k)}$ exists and is the unique solution to (5) [Baier *et al.*, 2008]. Note that since $\pi_\theta(\bar{s}, a) > 0$ for all θ and $(\bar{s}, a)$ such that there exists $\bar{s}' \in \bar{S}$ with $\bar{T}(\bar{s}, a, \bar{s}') > 0$ by definition of softmax policy, $\mathit{Reach}(\bar{S}_F)$ is independent on θ.

We can augment the MLIRL objective (4) by adding a non-decreasing differentiable function $g : \mathbb{R}^{|\bar{S}|} \to 1$ of $\bar{y}(\cdot \mid \pi_\theta)$ to explicitly consider the performance of π_θ with respect to the task specification. The new objective is to minimize

$$\begin{aligned} &J^{\text{side}}(\theta \mid M_{\varphi_{\text{cs}}}, h(D|\mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}})) \\ =&J^{\text{mle}}(\theta \mid M_{\varphi_{\text{cs}}}, h(D|\mathcal{M}, \mathcal{A}_{\varphi_{\text{cs}}})) - \mu \cdot g(\bar{y}), \end{aligned} \tag{6}$$

where $\mu > 0$ is a trade-off parameter adjusting the weight between the objective and the task performance objective. The optimization is subject to constraints (2), (3) and (5).

We solve the optimization problem by gradient descent, in which the key is to compute the derivative of Q_θ, π_θ and $\bar{y}$ with respect to θ. For any matrix B, we denote its component at row i and column j as $B(i,j)$. If we assume that π_θ does not change much in the neighborhood of θ, we can estimate $\frac{\partial Q_\theta}{\partial \theta}$ while considering π_θ as constant. Then for any $i = 1, \cdots k$ and $(\bar{s}, a) \in \bar{S} \times A$,

$$\begin{aligned} &\frac{\partial Q_\theta(\bar{s}, a)}{\partial \theta_i} \\ =&F_i(\bar{s}, a) + \gamma \sum_{\bar{s}' \in \bar{S}} \sum_{a' \in A} T(\bar{s}, a, \bar{s}') \pi_\theta(\bar{s}', a') \frac{\partial Q_\theta(\bar{s}', a')}{\partial \theta_i}. \end{aligned} \tag{7}$$

$$\frac{\partial \pi_\theta(\bar{s}, a)}{\partial \theta_i} = \pi_\theta(\bar{s}, a) \Big(\frac{\partial Q_\theta(\bar{s}, a)}{\partial \theta_i} - \sum_{\tilde{a}} \pi_\theta(\bar{s}, \tilde{a}) \frac{\partial Q_\theta(\bar{s}, \tilde{a})}{\partial \theta_i} \Big). \tag{8}$$

$$\begin{aligned} \frac{\partial}{\partial \theta_i} \bar{y}(\bar{s}) =& \sum_{a \in A} \pi_\theta(\bar{s}, a) \sum_{\bar{s}' \in \bar{S}} T(\bar{s}, a, \bar{s}') \Bigg(\frac{\partial \bar{y}(\bar{s}')}{\partial \theta_i} + \\ & \Big(\frac{\partial Q_\theta(\bar{s}, a)}{\partial \theta_i} - \sum_{\tilde{a}} \frac{\partial Q_\theta(\bar{s}, \tilde{a})}{\partial \theta_i} \Big) \bar{y}(\bar{s}') \Bigg), \end{aligned} \tag{9}$$

$$\frac{\partial}{\partial \theta_i} \bar{y}(\bar{s}) = 0, \text{ if } \bar{s} \in \bar{S}_F \bigcup (\bar{S} \backslash \mathit{Reach}(\bar{S}_F)).$$

The derivatives of Q_θ, π_θ and $\bar{y}$ with respect to θ are unique solutions of (7)–(9), given π_θ and $\bar{y}$. The uniqueness of the solution $\frac{\partial Q_\theta}{\partial \theta_i}$ in (7) holds for any stationary (i.e., time-invariant) policy π_θ given that F is bounded [Bertsekas *et al.*, 1995], which trivially holds as F is fixed. The uniqueness of the solution $\frac{\partial \bar{y}}{\partial \theta_i}$ in (9) holds for any stationary policy π_θ, $\bar{y}$ and $\frac{\partial Q_\theta}{\partial \theta}$, which can be proved by contradiction: assume that there exist two different functions $y_1, y_2 : \bar{S} \to \mathbb{R}$ that are both solutions to (9). Then for any $\bar{s} \in \bar{S}$, $y_1(\bar{s}) - y_2(\bar{s}) = \sum_{a \in A} \pi_\theta(\bar{s}, a) \sum_{\bar{s}' \in \bar{S}} T(\bar{s}, a, \bar{s}') \big(y_1(\bar{s}') - y_2(\bar{s}') \big)$. As (5) is known to have a unique solution, $y_1(\bar{s}) - y_2(\bar{s})$ has to be zero for all $\bar{s} \in \bar{S}$, which leads to a contradiction to the assumption. Therefore (9) has a unique solution $\frac{\partial \bar{y}}{\partial \theta_i}$.

5 Examples

We illustrate our approach on a path planning task in a 10-by-10 grid world map, as shown in Figure 1. Each cell represents a state in $\mathcal{M}$, from which the agent has 4 available actions: up, down, left, and right. States are labeled by their colors: *r* (red), *w* (white), *y* (yellow) and *b* (blue). The two green states are labeled as g_1 (green$_1$) and g_2 (green$_2$). The yellow state is an absorbing state, i.e., it has no outgoing transitions.

The specification task is to visit both green cells (intermediate goals) in any order, and end at the yellow cell (final

Figure 1: Grid world example and demonstration trajectories.

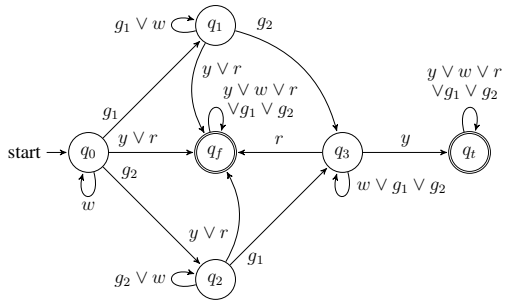


Figure 2: An equivalent DFA for φ_{cs}.

goal), while avoiding red cells (obstacles). These requirements are encoded as the co-safe LTL formula $\varphi_{\text{cs}} = \varphi_{\text{init}} \rightarrow (\varphi_{\text{safe}} \wedge \varphi_{\text{goal}})$, where

$$\varphi_{\text{init}} = \neg r \wedge \neg y \qquad \text{(Initial state)},$$
$$\varphi_{\text{safe}} = \neg r \,\mathtt{U}\, y \qquad \text{(Obstacle avoidance)},$$
$$\varphi_{\text{goal}} = \big((\neg y)\,\mathtt{U}\,(\mathtt{F}\, g_1 \wedge \mathtt{F}\, g_2)\big) \wedge (\mathtt{F}\, y) \qquad \text{(Goal reaching)}.$$

An equivalent DFA is shown in Figure 2. Each state in the DFA corresponds to some task status (see Table 2), and each transition represents some progress toward task completion. These transitions can be automatically encoded into features to facilitate learning. In fact, three features were constructed this way (see $\mathbf{f}_2, \mathbf{f}_3, \mathbf{f}_4$ in Table 1). The other two features come from transitions observed in demonstrations ($\mathbf{f}_1$), and a penalty for each transition ($\mathbf{f}_5$). Note that the final state has no outgoing transitions, and outgoing loops have zero reward.

The agent is given a set of demonstrated trajectories that successfully implemented the task, as shown in Figure 1. In this example all demonstrated trajectories start from the blue cell, pass both green cells (in arbitrary order), avoid all red cells, and eventually end at the yellow cell. States in the upper right corner are never observed in demonstration.

We now discuss the learned policy in three different cases, where the agent is provided with a different amount of side information, and the policies are learned with or without high-level task information.

Case 1 (MLIRL in $\mathcal{M}$) The agent only knows about the MDP $\mathcal{M}$, the labeling function $\mathcal{L}$, and the demonstrations $\mathcal{D}$, and learns a policy with MLIRL, i.e., by minimizing $J^{\text{mle}}(\theta \mid \mathcal{M}, D)$ in Eq. (4) while satisfying the constraints (2) and (3).

Table 1: Design of features in Case 2 and 3.

Feature	Explanation
$\mathbf{f}_1$	$f_1(s,a) = 1$ if (s,a) appeared in demonstration; otherwise $f_1(s,a) = 0$.
$\mathbf{f}_2$	$f_2(s,a)$ is the probability to reach q_t for the first time by taking a at state s.
$\mathbf{f}_3$	$f_3(s,a)$ is the probability to reach q_2, q_3 for the first time by taking a at state s.
$\mathbf{f}_4$	$f_4(s,a)$ is the negative probability to reach red states.
$\mathbf{f}_5$	$f_5(s,a) = -1$ if $s \notin S_F$.

Table 2: Interpretation of DFA states.

DFA State	Interpretation
q_0	None of g_1, g_2, y or r visited.
q_1	Visited g_1, never visited g_2, y, r.
q_2	Visited g_2, never visited g_1, y, r.
q_3	Visited g_1 and g_2, never visited y, r.
q_t	Visited g_1, g_2, y without visiting r (success).
q_f	Visited r, or visited y before visiting both g_1 and g_2 (failure).

Since the agent does not know φ_{cs} or the DFA, we cannot use all features from Table 1. Instead, we replace $\mathbf{f}_2(\bar{s}, a)$ and $\mathbf{f}_3(\bar{s}, a)$ by the probability of reaching the yellow state or a green state from $\bar{s}$ by taking a. All other features have the same interpretation as in Table 1. The learned feature weight vector is

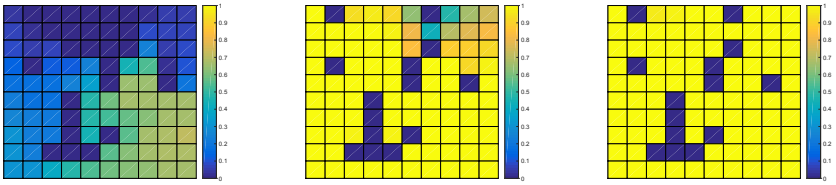
$$\hat{\theta}^{(1)} = [8.5176, 4.2678, -0.0442, -0.8208, 3.7336]^\intercal.$$

The sign of the learned weights is instructive: they define a policy that seeks to follow demonstrations ($\mathbf{f}_1$) and tries to reach the yellow state ($\mathbf{f}_2$) in a timely manner ($\mathbf{f}_5$). As the weights of $\mathbf{f}_3$ and $\mathbf{f}_4$ are negative, the agent fails to realize the importance of visiting green states and avoiding red states. There are at least two reasons for such behavior. First, as there is a feature ($\mathbf{f}_1$) marking the state-action pairs observed in demonstrations, the agent may simply try to follow the demonstrations whenever possible to minimize $J^{\text{mle}}(\theta \mid \mathcal{M}, D)$, without further reasoning about the demonstrations, which results in overfitting. Second, there is no side information for the agent to evaluate its policy or identify important features. As shown in Figure 3a, the agent behaves best in the lower right region, where it can follow some expert expert demonstration easily; it behaves the worst in the upper middle region, where the expert demonstrations are lacking.

Case 2 (MLIRL in $M_{\varphi_{\text{cs}}}$) The agent has all inputs in Case 1 and the DFA, and learns a policy with MLIRL within the product automaton. Compared with Case 1, the agent can now construct a product automaton. With the extended state space, the agent may behave differently based on the current status with respect to intermediate goals and potentially learn the importance of avoiding red cells. A simple check of the structure of the product automaton reveals that any visit to a red cell will lead to a transition to q_f in the DFA, which makes it impossible to reach $\bar{S}_F$ later. Therefore in order to reach a final state, it is necessary to add some penalty on visiting red states. The learned feature weight vector is

$$\hat{\theta}^{(2)} = [9.6090, 2.3128, 2.7393, -0.0121, 2.3221]^\intercal.$$

As shown in Figure 3b, the probability of satisfying φ_{cs} has been greatly improved. The weight for $\mathbf{f}_3$ is away from zero as expected, but the weight for $\mathbf{f}_4$ is still small, which suggests that the agent still has not learned to always avoid red cells. As a result, the probability of task success is the lowest in the upper right region, which is not covered by demonstrations. The two sources of problems explained in Case 1 still exist



(a) MLIRL policy in $\mathcal{M}$ (Case 1). Min prob: 6.90×10^{-9}; average prob: 0.304.

(b) MLIRL policy in $M_{\varphi_{\text{cs}}}$ (Case 2). Min prob: 0.430; average prob: 0.954.

(c) Policy with augmented objective (Case 3). Min prob: 0.962; average prob: 0.999.

Figure 3: Probability of satisfying φ_{cs} when following the corresponding policy from each initial state.

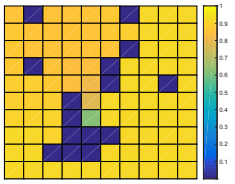


Figure 4: Probability of satisfying φ_{cs} for policies learned without the obstacle avoidance requirement.

here, which calls for the augmentation of objective function using LTL side information.

Case 3 (Policy with augmented objective) The agent has the same input as in Case 2, but now the policy is learned with the augmented objective function J^{side} in (6), where we set $g(\bar{y}) = \sum_{\bar{s}\in\bar{S}} \bar{y}(\bar{s} \mid \pi_\theta)$, i.e., the sum of probabilities of satisfying φ_{cs} from all initial states. With $\mu = 0.01$, the learned feature weight vector is

$$\hat{\theta}^{(3)} = [10.2010, 1.8908, 3.9550, 8.1854, 1.8855]^\intercal.$$

Compared with $\hat{\theta}^{(2)}$, the most significant change in $\hat{\theta}^{(3)}$ is that the weight on $\mathbf{f}_4$ is almost as large as $\mathbf{f}_1$, and much larger than the weights on other features. The agent now learns the importance of avoiding red states, and the performance with respect to task implementation has been significantly improved, especially from states that demonstrations fail to cover, as shown in Figure 3c. It shows that, by evaluating policies with side information φ_{cs}, the agent manages to get rid of the overfitting problem and the induced policy can now be generalized well into regions not previously seen in demonstrations.

To check the effect of the weight μ, we solved Case 3 with a series of μ and plotted the corresponding minimum and average probabilities of satisfying specification from all possible initial states, and corresponding negative log-likelihood of demonstrated trajectories (see Figure 5). Each experiment is repeated three times. Note that the value of the original objective $J^{\text{mle}}(\theta \mid M_{\varphi_{\text{cs}}}, D)$ is only slightly affected by μ, while the average probability of satisfying φ_{cs} is very sensitive to μ. This confirms that the augmentation of the objective function with LTL side information is necessary.

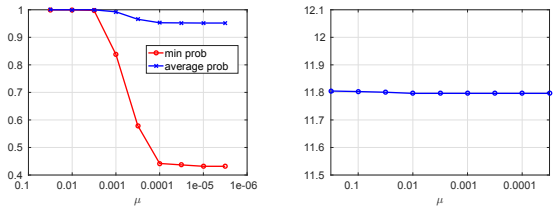


Figure 5: Probability of satisfying φ_{cs} (left) and negative log-likelihood of the state-action pairs in demonstration (right), as function of μ.

If the given high-level task description is incomplete or inaccurate, as is often the case in practice, demonstrations can compensate for an imperfect specification as long as the features are expressive enough, and no extra memory is needed to implement the missing part of the task.

To illustrate this point, we learned a policy where the side requirement to avoid red cells φ_{safe} is missing, and evaluated it with respect to the *true* task encoded by φ_{cs}. The result is shown in Figure 4. We observe the overall performance is only slightly worse than the case with accurate high-level task information, which suggests that the agent manages to learn from expert demonstration to avoid visiting red states at most initial states. Still, performance can be significantly worse at states that are close to obstacles, such as the cell in row 7, column 5. This example illustrates the agent's ability to learn actions preferences from demonstrations. However if the inaccurate high-level task information leads to insufficient memory states, the performance of the learned policies can be poor, as it is impossible to recover enough missing memory states from demonstration purely by learning the rewards.

6 Conclusions and Future Work

We formulated the problem of IRL with high-level side information on task requirements encoded as co-safe LTL formulas. We proposed two steps to improve the performance of the learned policy with respect to the probability of successfully implementing the task: the first is to construct a product automaton with the original MDP and an equivalent DFA of the co-safe LTL formula, and the second is to augment the objective function by evaluating the probability of satisfying the task requirements. We showed with a path planning example that the induced policy benefits from high-level side information with significantly better performance with respect to the task, especially in regions with few demonstrations.

This work can be extended in several directions. First, algorithms that can account for large state spaces are of interest. Another extension is to consider cases where high-level task requirements are imperfect, for example, if part of the requirements is missing in the high-level specification.

References

[Abbeel and Ng, 2004] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learn-

ing. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[Albert and Anderson, 1984] A. Albert and J. A. Anderson. On the existence of maximum likelihood estimates in logistic regression models. *Biometrika*, 71(1):1, 1984.

[Argall *et al.*, 2009] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[Babes *et al.*, 2011] Monica Babes, Vukosi Marivate, Kaushik Subramanian, and Michael L Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 897–904, 2011.

[Baier *et al.*, 2008] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008.

[Bertsekas *et al.*, 1995] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[Bloem and Bambos, 2014] Michael Bloem and Nicholas Bambos. Infinite time horizon maximum causal entropy inverse reinforcement learning. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 4911–4916. IEEE, 2014.

[Bobadilla *et al.*, 2012] Leonardo Bobadilla, Oscar Sanchez, Justin Czarnowski, Katrina Gossman, and Steven M LaValle. Controlling wild bodies using linear temporal logic. In *Robotics: Science and systems*, volume 7, page 17, 2012.

[Boularias *et al.*, 2011] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. pages 182–189, 2011.

[Dvijotham and Todorov, 2010] Krishnamurthy Dvijotham and Emanuel Todorov. Inverse optimal control with linearly-solvable MDPs. In *International Conference on Machine Learning*, pages 335–342, 2010.

[Fainekos *et al.*, 2005] Georgios E Fainekos, Hadas Kress-Gazit, and George J Pappas. Temporal logic motion planning for mobile robots. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2020–2025. IEEE, 2005.

[John, 1994] George H John. When the best move isn't optimal: Q-learning with exploration. In *AAAI*, page 1464. Citeseer, 1994.

[Kress-Gazit *et al.*, 2011] Hadas Kress-Gazit, Tichakorn Wongpiromsarn, and Ufuk Topcu. Correct, reactive, high-level robot control. *IEEE Robotics & Automation Magazine*, 18(3):65–74, 2011.

[Kupferman and Vardi, 2001] Orna Kupferman and Moshe Y Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[Macglashan and Littman, 2015] James Macglashan and Michael L Littman. Between imitation and intention learning. In *International Conference on Artificial Intelligence*, pages 3692–3698, 2015.

[Neu and Szepesvári, 2007] Gergely Neu and Csaba Szepesvári. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, UAI'07, pages 295–302, Arlington, Virginia, United States, 2007. AUAI Press.

[Ng *et al.*, 2000] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[Pnueli, 1981] Amir Pnueli. The temporal semantics of concurrent programs. *Theoretical computer science*, 13(1):45–60, 1981.

[Ramachandran and Amir, 2007] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI 2007, Proceedings of the International Joint Conference on Artificial Intelligence, Hyderabad, India, January*, pages 2586–2591, 2007.

[Ratliff *et al.*, 2006] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006.

[Wolff *et al.*, 2012] Eric Wolff, Ufuk Topcu, and Richard Murray. Optimal control with weighted average costs and temporal logic specifications. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[Ziebart *et al.*, 2008] Brian D Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *National Conference on Artificial Intelligence*, pages 1433–1438, 2008.