2020 59th IEEE Conference on Decision and Control (CDC)
Jeju Island, Republic of Korea, December 14-18, 2020

Active Task-Inference-Guided Deep Inverse Reinforcement Learning

Farzan Memarian, Zhe Xu, Bo Wu, Min Wen, Ufuk Topcu

***Abstract*— We consider the problem of reward learning for *temporally extended tasks*. For reward learning, inverse reinforcement learning (IRL) is a widely used paradigm. Given a Markov decision process (MDP) and a set of demonstrations for a task, IRL learns a reward function that assigns a real-valued reward to each state of the MDP. However, for temporally extended tasks, the underlying reward function may not be expressible as a function of individual states of the MDP. Instead, the history of visited states may need to be considered to determine the reward at the current state. To address this issue, we propose an iterative algorithm to learn a reward function for temporally extended tasks. At each iteration, the algorithm alternates between two modules, a *task inference module* that infers the underlying task structure and a *reward learning module* that uses the inferred task structure to learn a reward function. The task inference module produces a series of queries, where each query is a sequence of *subgoals*. The demonstrator provides a binary response to each query by attempting to execute it in the environment and observing the environment's feedback. After the queries are answered, the task inference module returns an *automaton* encoding its current hypothesis of the task structure. The reward learning module augments the state space of the MDP with the states of the automaton. The module then proceeds to learn a reward function over the augmented state space using a novel *deep maximum entropy IRL* algorithm. This iterative process continues until it learns a reward function with satisfactory performance. The experiments show that the proposed algorithm significantly outperforms several IRL baselines on temporally extended tasks.**

I. Introduction

In inverse reinforcement learning (IRL) [1]–[3], given a reward-free Markov decision process (MDP) and a set of demonstrations corresponding to the successful implementation of a task, the goal is to learn the underlying reward function. Existing IRL algorithms, such as [3], [4], learn a reward as a function of the states of the MDP, assuming the reward is independent of the history of states. However, this assumption may not be valid for *temporally extended tasks*, where we often need to consider the history of states to learn a reward function.

Another challenge for IRL methods is the dependence of the effectiveness of the learned reward function on the demonstrations provided beforehand. To address this issue, [5] proposed to actively ask for demonstrations at specific states, and [6] proposed to actively ask for *advice* in different subsets of the state space. However, for temporally extended tasks, these approaches may not reveal the underlying task structure. Without uncovering the underlying task structure, we may not be able to recover the underlying reward function.

To address the issues discussed above, we develop an iterative algorithm that learns a reward function for temporally extended tasks that are defined over a set of subgoals. Specifically, the algorithm alternates between two modules: A *task inference module* infers the underlying task structure and a *reward learning module* learns a reward function using the inferred task structure.

The task inference module produces a series of queries to be answered by the demonstrator. Each query is a sequence of subgoals, and the demonstrator must execute the sequence in the environment and answer the query based on the *binary* feedback from the environment. For queries for which the environment's feedback is positive, the demonstrator produces demonstrations according to the sequence of subgoals in the query and adds them to the set of demonstrations. The task inference module then returns a deterministic finite automaton (DFA) encoding the current hypothesis of the task structure.

The reward learning module augments the state space of the MDP with the states of the DFA. And then, the module proceeds to learn a reward function over the augmented state space using a novel deep MaxEnt IRL algorithm inspired by [4]. The IRL algorithm approximates the reward function by a convolutional neural network (CNN). The reward network takes as input the states of the MDP and the DFA and the current action and outputs a real-valued reward. By using a CNN, we avoid the need for manual feature engineering as CNN can learn to extract reward features. The experiments show that the learned reward network generalizes to test environments not seen during training.

In each iteration, after the two modules are called sequentially, the algorithm computes a maximum entropy policy from the learned reward function. This iterative process continues until the computed policy leads to a *satisfactory* performance in terms of the probability of completing the task.

To experimentally evaluate the proposed algorithm, we create a *task-oriented navigation* environment. We experiment with several navigation tasks and show that the proposed algorithm can successfully infer the underlying task structure and use it to guide IRL. We compare the results with two baselines, i.e., a memoryless IRL method, an IRL

This research was supported partly by the grants ARL W911NF2020132, ONR N00014-20-1-2115, ARL ACC-APG-RTP W911NF

Farzan Memarian, Zhe Xu and Bo Wu are with the Oden Institute for Computational Engineering and Sciences, University of Texas at Austin, Austin, TX 78712, Ufuk Topcu is with the Department of Aerospace Engineering and Engineering Mechanics, and the Oden Institute for Computational Engineering and Sciences, University of Texas at Austin, Austin, TX 78712, e-mails: farzan.memarian@utexas.edu, zhexu@utexas.edu, bwu3@utexas.edu, utopcu@utexas.edu.

Min Wen is with Google LLC. minwen@google.com,

978-1-7281-7447-1/20/$31.00 ©2020 IEEE

method augmented with memory bits. We show that the proposed method learns policies that outperform the baselines in terms of the probability of completing the navigation tasks. We observe the biggest difference in performance between the proposed method and the baselines for navigation tasks with complicated task structures (e.g., the task of reaching a long sequence of subgoals in a given order, where there are repeated subgoals in the sequence). For such tasks, the policy derived from the proposed method completes the task with a probability that is at least nine times higher than the baselines.

The contributions of the paper are as follows: **1)** We develop an algorithm for actively inferring the underlying task structure in the form of a DFA. **2)** We develop a method for inducing the task information encoded as a DFA in deep IRL. The DFA tracks the progress in the execution of the task. **3)** We propose a novel deep IRL algorithm that learns a reward function over the extended state space. The trained reward network extracts specialized features for each stage of task completion.

II. Related Work

As part of the pipeline of the proposed algorithm, we learn the task structure and then incorporate it in reward learning. In prior work such as [7] and [8] they use assumptions about the task structure to guide policy learning. In particular they use Bayesian inference to segment unstructured demonstration trajectories. [9] assumed that the expert performs a given sequence of (symbolic) subtasks in each demonstration. They solve the problem of temporal alignment for the demonstrations and policy learning for each subtask simultaneously. All of these works learn a policy directly, whereas we learn a reward function. Moreover, unlike these works, we do not make any assumptions about the task structure prior to training. There is prior work attempting to learn the task structure or specialized subtask policies. In [10], the authors extend the generative adversarial imitation learning (GAIL) algorithm proposed in [11] to learn policy options as well as reward options for complex multi-stage tasks. They claim success in simple and complex continuous control tasks. In [12], they propose a method for learning segments of hierarchically-structured behavior from demonstration data and learn specialized policies or each segment. Both previous works assume access to a set of demonstrations a priori, whereas we use an active learning approach.

Our work is closely related to the use of formal methods in RL, such as RL for reward machines [13]–[15] and RL with temporal logic specifications [16]–[20], [20]–[24]. For example, in [25], the authors propose an RL algorithm that learns a policy that maximizes the probability of satisfying a specification expressed as a linear temporal logic formula. They construct an automaton based on the linear temporal logic formula and use it to guide the RL agent. The idea of using high-level knowledge such as automaton or temporal logic to guide reinforcement learning [26], [27] or inverse reinforcement learning has been explored in some previous work. For example, in [26], the authors propose an iterative algorithm that performs joint inference of reward machines and policies for RL. Our work is different from this line of work as we focus on inverse reinforcement learning rather than reinforcement learning. In [28], they propose to incorporate side information into the IRL algorithm. Our work is different from this work in that we do not assume any prior knowledge over the task. In addition, they model the reward function as a linear function of known features, whereas we model the reward function by a deep neural network.

Our work is also related to hierarchical IRL (HIRL) [29], where they assume each demonstration trajectory corresponds to a set of subtasks. The subtasks are separated by critical *transition states*, i.e., states where transitions between subtasks occur. Once the transition states are recognized, the state space is augmented with features that capture the visitation history of the transition states. HIRL suffers from two limitations compared to the proposed method. First, it assumes that the task can always be decomposed into a sequence of subtasks. This assumption may not hold in general, for example, for tasks that are expressed with more than one sequence of subgoals. Second, their method does not benefit from the use of deep neural networks and automatic feature learning.

III. Background and Preliminaries

The problem of inverse reinforcement learning (IRL) can be described as follows: Given a reward-free MDP $\mathcal{M}$, and a set of demonstration trajectories D, learn a reward function R that can optimally interpret the demonstrations in some pre-specified way.

We adopt the following definitions and notations. The environment is modeled as a reward-free MDP $\mathcal{M} = \langle S, A, T, \rho, \eta \rangle$ where S is the state space; A is the action space; $T : S \times A \to \mathcal{D}(S)$ (where $\mathcal{D}(S)$ is the set of all probability distributions over S) is the transition function; $\rho \in \mathcal{D}(S)$ is an initial distribution over S; and $\eta : S \to E$ is a labeling function with E as a finite set of subgoals. Let $D = \{\tau_1, \ldots, \tau_N\}$ be a set of demonstration trajectories, where $\tau_i = \{(s_{i,0}, a_{i,0}), \ldots, (s_{i,n_i}, a_{i,n_i})\}$ for all $i = 0, \ldots, n_i$.

We define the task to be learned by a mapping $\mathcal{T} : E^* \to \{0, 1\}$, where E^* is the Kleene star of E, and $\mathcal{T}(\omega) = 1$ denotes that a sequence of subgoals $\omega \in E^*$ can complete the task. The task structure can be encoded by a deterministic finite automaton (DFA). A DFA $\mathcal{A}$ is a tuple $\langle Q_\mathcal{A}, \Sigma, \delta, q_0, F \rangle$ where $Q_\mathcal{A}$ is a set of states; Σ is a set of input symbols (also called the alphabet); $\delta : Q_\mathcal{A} \times \Sigma \to Q_\mathcal{A}$ is a deterministic transition function; $q_0 \in Q_\mathcal{A}$ is the initial state; $F \subseteq Q_\mathcal{A}$ is a set of final states (also called *accepting* states). Given a finite sequence of input symbols $w = \sigma_0, \sigma_1, \ldots, \sigma_{k-1}$ in Σ^k for some $k \in \mathbb{N}^+$, the DFA $\mathcal{A}$ generates a unique sequence of $k+1$ states $\tau_\mathcal{A} = q_0, q_1, \ldots, q_k$ in $Q_\mathcal{A}^{k+1}$ such that for each $t = 1, \ldots, k$, $q_t = \delta(q_{t-1}, \sigma_{t-1})$. We denote the last state q_k by taking the sequence w of inputs from q_0 as $\underline{\delta}(q_0, w)$. $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if and only if $\underline{\delta}(q_0, w) \in F$. Let $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^*$ be all the finite sequences of input symbols that

are accepted by $\mathcal{A}$, which is also referred to as the *accepted language* of $\mathcal{A}$.

Maximum Entropy IRL. In the original MaxEnt IRL algorithm [3], the reward function is parameterized as a linear combination of a given set of feature functions. In other words, given a set of features $\{f_1, \ldots, f_K\}$ where $f_k : S \times A \to \mathbb{R}$ for $k = 1, \ldots, K$, the reward function R_θ is parameterized by $\theta = [\theta_1, \ldots, \theta_K]^\intercal$ and $R_\theta(s, a) = \sum_{k=1}^{K} \theta_k f_k(s, a)$. In MaxEnt IRL, we want to learn a reward function such that the corresponding maximum entropy policy produces the same expected total reward over trajectories as the average reward over trajectories obtained from demonstrations. With the principle of maximum entropy, it can be derived that the resulting probability distribution over any dynamically feasible (finite-length) trajectory $\tau = s_0, a_0, \ldots, s_{|\tau|}, a_{|\tau|}$ is proportional to the exponential of the total reward of τ:

$$Pr(\tau|R_\theta) \propto \exp \sum_{(s_i,a_i)\in\tau} R_\theta(s_i, a_i). \tag{1}$$

While linear parameterization is commonly used in IRL literature [2], [30]–[32], it suffers from several drawbacks. On the one hand, it requires human knowledge to provide properly designed reward features, which can be labor-intensive; on the other hand, if the given features fail to encode all the essential requirements to generate the demonstrations, there is no way to recover this flaw by learning from demonstrations. To address these drawbacks, we model the reward function by a deep neural network to automatically construct reward features from expert demonstrations.

IV. Active Task-Inference-Guided Deep Inverse Reinforcement Learning

In this section, we introduce the active task-inference-guided deep IRL (ATIG-DIRL), which iteratively infers the task structure and incorporates the task structure into reward learning.

A. Overview

ATIG-DIRL, as illustrated in Algorithm 1, consists of a task inference module and a reward learning module. The task inference module utilizes L* learning [33], an active automaton inference algorithm, as the template to iteratively infer a DFA from queries and counterexamples. The inferred DFA encodes the high-level task structure to help IRL recover reward functions. Following L* learning, the task inference module generates two kinds of queries with Boolean answers, namely the *membership* query and the *conjecture* query. A membership query asks whether a sequence of subgoals can lead to task completion. We defer the details of answering membership queries to Sec. IV-B. After a number of membership queries are answered, the inference engine outputs a hypothesis DFA and a set of corresponding demonstrations for membership queries that lead to task completion (line 5 in Alg. 1). The task inference module then asks a conjecture query about whether the hypothesis DFA can help recover a satisfactory reward function (line 6 in Alg. 2). The conjecture query is to be answered by the reward learning module, and the details are in Sec. IV-C. If the answer to the conjecture query is $True$, both the automaton inference process and the reward learning are finished. Otherwise, the answer is $False$, and there exists a counterexample in the form of a sequence of subgoals to illustrate the difference between the conjectured DFA and the task structure. Such a counterexample will trigger a new iteration with the next round of membership queries.

Algorithm 1 Active task-inference-guided deep IRL (ATIG-DIRL)

1: **Input:** A reward-free MDP $\mathcal{M} = \langle S, A, T, \rho, \eta \rangle$, stopping threshold κ
2: **Output:** A hypothesis DFA $\mathcal{A} = \langle Q_\mathcal{A}, \Sigma, \delta, q_0, F \rangle$, a reward network $R_\theta : S \times Q_\mathcal{A} \to \mathbb{R}$ and a policy $\pi_\theta : S \times Q_\mathcal{A} \to \mathcal{D}(A)$
3: Initialize success ratio (β) as 0, counterexample (CE) as null, set of demonstrations (D) as empty set
4: **while** $\beta \leq \kappa$ **do**
5: $\mathcal{A}$, $D \leftarrow$ TaskInferenceModule($\mathcal{M}$, CE)
6: R_θ, π_θ, β, $CE \leftarrow$ RewardLearningModule($\mathcal{A}$, $\mathcal{M}$, D, κ)
7: **end while**

B. Task Inference Module

To generate a hypothesis DFA $\mathcal{A} = \langle Q_\mathcal{A}, \Sigma, \delta, q_0, F \rangle$ where $\Sigma = E$, ATIG-DIRL produces a number of membership queries. Each membership query asks whether following a subgoal sequence $\omega \in E^*$ leads to task completion, i.e. $\mathcal{T}(\omega) = 1$. The task $\mathcal{T}$ is unknown, but given a sequence $\omega \in E^*$, one can observe $\mathcal{T}(\omega)$ from the environment. To answer a membership query, we rely on a demonstrator to make a demonstration in the MDP environment and generate a state sequence $\lambda \in S^*$ where $\eta(\lambda) = \omega$. Then if the subgoal sequence ω completes the task, i.e. $\mathcal{T}(\omega) = 1$, the answer to this membership is $True$, otherwise the answer is $False$. If it is not possible to execute the query, the answer is $False$. After answering the membership queries, ATIG-DIRL will generate a hypothesis DFA $\mathcal{A}$ following procedures in [33].

After obtaining a hypothesis DFA $\mathcal{A}$, ATIG-DIRL asks whether $\mathcal{A}$ is sufficient to recover the reward function (conjecture query). To answer this query, we follow the procedure introduced in Sec. IV-C, which is summarized in Alg. 2.

Demonstration trajectories. When the task inference module asks a query ω, if $\mathcal{T}(\omega) = 1$, it means the query encodes a subgoal sequence that leads to task completion. The demonstrator will then produce several demonstrations following the same subgoal sequence and adds them to the set of demonstrations.

C. Reward Learning Module

This module is concerned with learning a reward function, which is modeled as a deep neural network, using the hypothesis DFA. Although previous deep MaxEnt IRL methods [4], [34]–[36] can learn deep reward networks

Algorithm 2 RewardLearningModule

Input: A reward-free MDP $\mathcal{M}$, a hypothesis DFA $\mathcal{A}$, a set of demonstrations $D = \{\tau_1, \ldots, \tau_N\}$, threshold for producing counterexamples κ
Output: Reward network $R_\theta : S \times Q_\mathcal{A} \to \mathbb{R}$, policy $\pi_\theta : S \times Q_\mathcal{A} \to \mathcal{D}(A)$, success ratio ($\beta$), counterexample ($CE$)
Hyper-parameters: number of iterations between updating the success ratio N, stopping threshold ϵ
Initialize the reward network parameter θ_0; Stop $\leftarrow$ False; $t \leftarrow 0$
while *Stop* = *False* **do**
  Use the parameter vector θ_t at the current time step t to compute Q_{θ_t} and the policy π_{θ_t} via Eq. 4 and Eq. 5
  Compute $\frac{\partial Q_\theta}{\partial \theta} \mid_{\theta=\theta_t}$ and $\frac{\partial \pi_\theta}{\partial \theta} \mid_{\theta=\theta_t}$ via Eq. 7 and Eq. 8
  Compute $\frac{\partial L_D(\theta_t)}{\partial \theta_t}$ via Eq. 6
  Update θ: $\theta_{t+1} \leftarrow \theta_t + \alpha_t \frac{\partial L_D(\theta)}{\partial \theta} \mid_{\theta=\theta_t}$
  if $(t \bmod N = 0)$ **then**
    Compute the current success ratio β_t at time step t using Monte Carlo evaluation
    if $\beta_t - \beta_{t-N} \leq \epsilon$ **then** *Stop* $\leftarrow$ *True*
  end if
  $t \leftarrow t + 1$
end while
if $(\beta_t \leq \kappa)$ **then**
  Produce a CE using Monte Carlo simulation
end if
$\beta \leftarrow \beta_t$

from demonstrations, they assume the reward function is a function of the current MDP state. As a result, the learned policy has to be independent of the history of states, which does not suffice for temporally extended tasks. To address this issue, we propose a new maximum entropy deep IRL algorithm (which is inspired by the work in [4]), as described in Alg. 2.

The key idea is to extend the state space of the MDP using the states of the hypothesis DFA $\mathcal{A}$ and to create a product MDP and then learn a reward function over the extended state space. We define the product MDP as follows:

Product MDP. Let $\mathcal{M} = \langle S, A, T, \rho, \eta \rangle$ be a reward-free MDP and $\mathcal{A} = \langle Q_\mathcal{A}, \Sigma, \delta, q_0, F \rangle$ be a DFA. The product MDP $\mathcal{M}_\mathrm{p} := \mathcal{M} \otimes \mathcal{A} = (Z, Z_0, \delta_\mathrm{p}, \eta_\mathrm{p}, F_\mathrm{p})$ is a tuple such that, $Z = S \times Q_\mathcal{A}$ is a finite set of states. Z_0 is the initial set of states where for each $z_0 = (s, q) \in Z_0$, $s \in S_0$, $q = \delta(q_0, \eta(s))$. δ_p is the transition function of the product MDP defined as

$$\delta_\mathrm{p}((s,q),a,(s',q')) = \begin{cases} T(s,a,s') & \text{if } q' = \delta(q, \eta(s')); \\ 0 & \text{otherwise;} \end{cases}$$

$\eta_\mathrm{p}((s,q)) = \{q\}$ is a labeling function; and $F_\mathrm{p} = S \times F$ is a finite set of accepting states. The state space of the product MDP is essentially an extension of the state space of the original MDP.

We apply the proposed IRL algorithm on the product MDP. As a result, the reward depends on both the current state s in $\mathcal{M}$ and the current state q in $\mathcal{A}$, which together form the current state of the product MDP $z \in Z$. DFA states can be considered as different stages in task implementation. Since the DFA state is an input to the reward function, the learned reward, and the corresponding induced policy will be a function of the stage of the task.

Unlike MaxEnt IRL where the reward function is modeled as a linear combination of pre-specified features, we model the reward function as a neural network. The set of reward parameters are represented by θ, which is the weight vector of the reward network. The objective is to maximize the posterior probability of observing the demonstration trajectories and reward parameters θ given a reward structure:

$$\begin{aligned} L(\theta) :&= \log Pr(D, \theta | R_\theta)) \\ &= \underbrace{\log Pr(D|R_\theta)}_{L_D} + \underbrace{\log P(\theta)}_{L_\theta} \end{aligned} \tag{2}$$

L_D is the log-likelihood of the demonstration trajectories in D given the reward function R_θ. L_θ can be interpreted as either the logarithm of the prior distribution $P(\cdot)$ at θ or as a differentiable regularization term on θ. In this work, we assume a uniform prior distribution over θ, so what remains is the maximization of L_D.

Since we apply IRL on the state space of the product MDP, the demonstration trajectories are projected onto the state space of the product MDP, i.e. $\tau_i = \{(z_{i,0}, a_{i,0}), \ldots, (z_{i,n_i}, a_{i,n_i})\}$, where $z_{i,j} = (s_{i,j}, q_{i,j})$. Let π_θ be the policy corresponding to R_θ, then L_D can be expressed as

$$L_D = \sum_{\tau_i \in D} \sum_{j=0}^{n_i - 1} \log \pi_\theta(a_{i,j} | z_{i,j}) + C, \tag{3}$$

where C is a constant that is dependent on D and the transition dynamics δ_p of the product MDP.

The computation of π_θ given R_θ is essentially a maximum entropy reinforcement learning problem [37]. It can be proved [37] that for any $R_\theta(z,a)$, there exists a unique function $Q_\theta(z,a)$ which is the fixed point of

$$\begin{aligned} Q_\theta(z,a) &= R_\theta(z,a) \\ &\quad + \gamma \sum_{z' \in Z} \delta_p(z'|z,a) \log \sum_{a'} \exp(Q_\theta(z',a')), \end{aligned} \tag{4}$$

where γ is the discount factor which is a hyper-parameter. Note that $Q_\theta(z,a)$ is an implicit function of θ, as $R_\theta(z,a)$ is parametrized by θ and $Q_\theta(z,a)$ is derived from $R_\theta(z,a)$ by Eq. 4. The maximum entropy policy π_θ, which is alternatively called the *soft Bellman policy* in [37], can be derived from Q_θ as shown in the equation below

$$\pi_\theta(z|a) = \frac{\exp Q_\theta(z,a)}{\sum_{a'} \exp(Q_\theta(z,a'))}. \tag{5}$$

To find the optimal θ, we perform gradient ascent using the gradient of L_D with respect to θ expressed as

$$\begin{aligned}\frac{\partial L_D}{\partial \theta} =& \frac{\partial}{\partial \theta}\sum_{i=1}^{N}\sum_{l=0}^{n_i}\log \pi_\theta(z_{i,l}, a_{i,l}) \\ =& \sum_{i=1}^{N}\sum_{l=0}^{n_i}\Big(\frac{\partial Q_\theta(z_{i,l}, a_{i,l})}{\partial \theta} - \sum_{a'} w_\theta(z_{i,l}, a')\Big),\end{aligned} \tag{6}$$

where $w_\theta(z,a) := \pi_\theta(a|z)\frac{\partial Q_\theta(z,a)}{\partial \theta}$ for any $z \in Z, a \in A$. To compute the right hand side of Eq.6, we compute the gradient of π_θ and Q_θ as

$$\begin{aligned}\frac{\partial Q_\theta(z,a)}{\partial \theta} &= \frac{\partial R_\theta(z,a)}{\partial \theta} \\ &+ \gamma \sum_{z'} T(z'|z,a)\sum_{a'}\pi_\theta(a'|z')\frac{\partial Q_\theta(z',a')}{\partial \theta},\end{aligned} \tag{7}$$

$$\frac{\partial \pi_\theta(a|z)}{\partial \theta} = z_\theta(z,a) - \pi_\theta(a|z)\sum_{a'} w_\theta(z,a'). \tag{8}$$

Since $\gamma \in (0,1)$, it can be shown that for any θ, there exists a unique solution $\frac{\partial Q_\theta(z,a)}{\partial \theta}$ which is the fixed point to Eq. 7. Therefore, there is also a unique solution $\frac{\partial \pi_\theta(a|z)}{\partial \theta}$ to Eq. 8.

Once we have performed a gradient ascent step using Eq.6 to update the weight vector θ of the neural network, we have automatically updated the reward network R_θ.

Monte Carlo evaluation. Evaluating the task performance of a reward network R_θ, amounts to computing the *success ratio* of the maximum entropy policy π_θ for R_θ. We use the Monte Carlo approach to empirically compute the success ratio of the policy π_θ. Concretely, we use π_θ to produce a set of state sequences λ, convert the state sequences to subgual sequences, i.e, $\eta(\lambda) = \omega$, and then observe $\mathcal{T}(\omega)$. The ratio of sequences with $\mathcal{T}(\omega) = 1$ to the total number of sequences yields the success ratio (line 11 in Alg. 2).

Stopping criterion. After every N iterations of gradient ascent (N is hyper-parameter), the module evaluates the success ratio of the computed maximum entropy policy π_θ for R_θ, and stops the iterations once the success ratio stops changing significantly according to a pre-specified threshold ϵ (line 12 in Alg.2).

Counterexamples. After we finish the iterations of gradient ascent, if the success ratio of the maximum entropy policy π_θ for the obtained reward network R_θ is smaller than a threshold κ, then the module produces a counterexample to be used by the task inference module in the next iteration of Alg. 1. To produce the counterexample, we apply Monte Carlo simulations and find an ω such that $\mathcal{T}(\omega) \neq \xi(\omega, \mathcal{A})$ where $\xi(\omega, \mathcal{A})$ is 1 when ω is accepted by the DFA $\mathcal{A}$ and is 0 otherwise (line 17 in Alg. 1).

V. Experiments

We create the *task-oriented navigation* environment, which is a simulated environment that can model various navigation tasks. Different objects are present in the environment, such as {building, grass, tree, rock, barrel, and tile}. A region is defined as any 3×3 square neighborhood in the environment.

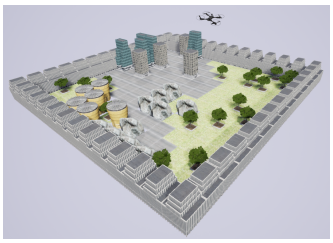


Fig. 1: Training environment for task-oriented navigation.

TABLE I: Definition of region types.

Type-0 region (R_0)	a region with more than 6 buildings
Type-1 region (R_1)	a region with more than 6 trees
Type-2 region (R_2)	a region with more than 6 barrels
Type-3 region (R_3)	a region with more than 6 stones
Irrelevant region	none of the above

Each region belongs to one of the types defined in Table. I. Region types are used to define navigation tasks, as we will see later. Visiting a region means visiting the center block of that region. Fig. 1 depicts one instantiation of the environment used as the training environment for our experiments.

Agent. The agent is an aerial vehicle that can navigate in the environment. At each time step, it is located on top of an environment block, and it can choose either of the following actions {"up", "down", "left", "right"}, and move one block in that direction.

The navigation task. We experiment with three different navigation tasks. The underlying DFA corresponding to all the three tasks can be found in the extended ArXiv version[1]. Among these tasks, task #3 is the most challenging one, and we observe the biggest performance difference between the proposed method and the baselines for this task. As such, we explain this task in more detail, and the rest of the tasks follow the same notation and conventions. Task #3 is defined as follows: {visit R_0, R_1, R_2, R_3, R_0 in this order. Any other order leads to failure}. Fig. 2 shows the DFA encoding this task.

With a given task, the goal of ATIG-DIRL is to infer a DFA that is equivalent to the underlying DFA for that task and use it for reward learning. We have used the libalf library [38] to infer the DFA.

Reward network. The input to the convolutional layers of the reward network is the 7×7 neighborhood of the agent in the environment. Each block in the neighborhood is represented by a one-hot-encoded vector, specifying the object that occupies that block. There are 6 objects, so the input to convolutional layers is a $7\times7\times6$ tensor. The network has two convolutional layers, the first layer has 6 kernels and the second layer has 8 kernels, all kernels are of size 2×2 with a stride of 1. After the convolutions, there is a flattening layer. And this is where the DFA's state and

[1]https://arxiv.org/abs/2001.09227

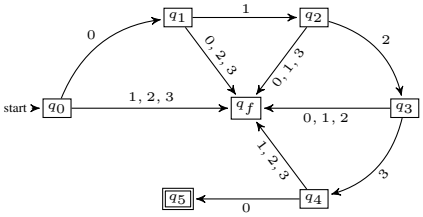
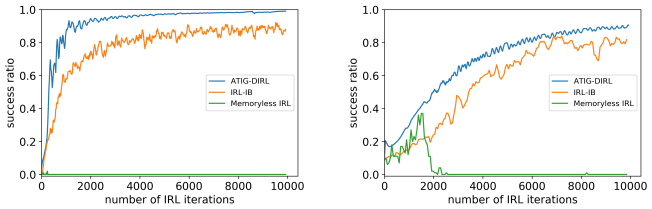
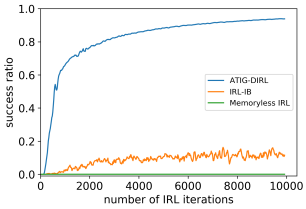


Fig. 2: Underlying DFA corresponding to task #3.



(a) Task #1. (b) Task #2.



(c) Task #3.

Fig. 3: Performances of ATIG-DIRL versus the IRL baselines at training time.

the action are provided as input, by appending them to the output of the flattening layer. The DFA's state and the action are represented by one-hot-encoded vectors of length 10 and 4, respectively. Note that the DFAs we learn have at most 7 states, but to make the architecture applicable to bigger DFAs, we use zero padding to make the one-hot-encoded vector be of size 10. After appending the state of the DFA and action, the resulting vector passes through two fully connected layers of size 214 and 50, and the output layer has one neuron. Each hidden layer is followed by a ReLU layer.

Baselines. We implemented two baseline methods to compare with ATIG-DIRL: *memoryless IRL* and *IRL augmented with information bits (IRL-IB)*. The memoryless IRL is a basic deep MaxEnt IRL method [4] that learns a reward function as a function of the states of the MDP. The IRL-IB method is a deep MaxEnt IRL method that uses a fixed size memory to augment the state space of the MDP. In our experiment, since there are 4 region types, the memory is a vector of size 4, where each element corresponds to one of the region types. If a region type has been visited, the value of the corresponding element is 1, and 0 otherwise.

Evaluation criterion. The primary criterion we use in evaluating the performance of a trained model is the task success ratio. All methods are trained using the same training environment (Fig. 1).

Training performance. We compares the training performance of ATIG-DIRL algorithm to the IRL baselines for the tasks described by the DFAs depicted in the extended ArXiv paper.

The ATIG-DIRL was able to infer a DFA equivalent to the underlying DFA for all three tasks in at most three iterations of the main algorithm (Alg. 1). We have shown an intermediate hypothesis DFA inferred by the algorithm for task #3 in the extended ArXiv version. This DFA recognizes the correct sequence for completing the task, i.e., R_0, R_1, R_2, R_3, R_0, but it fails to capture the fact that any deviation from this path results in failure and the task cannot be completed anymore. For example, this DFA hypothesizes that the sequence $R_0, R_2, R_0, R_1, R_2, R_3, R_0$ completes the task, whereas by looking at the underlying DFA, we realize that this sequence leads to failure. The reported results for ATIG-DIRL correspond to the last call to Alg. 2 where the final hypothesized DFA is used in the IRL loop.

Fig. 3 shows that as tasks become more challenging from task #1 to task #3, the difference between the performance of ATIG-DIRL and the baselines becomes more pronounced. Tasks become more challenging if there are more ways to fail, fewer ways to complete the task, and if the sequences that lead to task completion are longer. Another factor that makes tasks more complicated is the existence of repeated subgoals in the sequences that complete the task. Task #3 is particularly challenging due to all the factors discussed above, particularly the existence of a repeated subgoal (R_0) in the sequence that completes the task.

IRL-IB is able to learn a reward function with comparable performance to the proposed method for tasks #1 and #2, but it fails to do so for task #3. The reason is that the memory structure that IRL-IB uses is unable to capture sequences with repeated subgoals. The memoryless IRL method performs poorly as expected because, without a memory to keep track of task progress, it is not possible to uncover the underlying reward function for the temporally extended tasks considered here.

Test performance. To compare the generalization capability of the three methods, we have tested them on 10 different randomly generated environments. The ATIG-DIRL method outperforms all the baselines on test cases (Tables IIa, IIb, IIc). The memoryless IRL method has a poor performance across all tasks as expected because it did not perform well even in the training environment. The IRL-IB method, however, performs well for tasks #1 and #2, but fails at generalization for task #3 which is the most challenging task. The main reason for its failure is the existence of a repeated subgoal in the subgoal sequence that completes the task.

VI. Conclusion and Future Work

We have proposed a new IRL algorithm, active task-inference-guided deep IRL (ATIG-DIRL). The algorithm learns the task structure in the form of a deterministic finite automaton (DFA) and uses the DFA to extend the state space of the original MDP and learns a reward function over the extended state space. By representing the reward

TABLE II: Mean and standard deviation of success ratio on 10 different test environments for each of the three tasks.

(a) Task #1.

Model	Mean	STD
ATIG-DIRL	0.95	0.02
IRL-IB	0.93	0.02
Memoryless IRL	0.002	0.004

(b) Task #2.

ATIG-DIRL	0.95	0.03
IRL-IB	0.89	0.04
Memoryless IRL	0.02	0.03

(c) Task #3.

ATIG-DIRL	0.96	0.02
IRL-IB	0.11	0.06
Memoryless IRL	0.003	0.006

functions as a convolutional neural network, the algorithm can automatically extract reward features that are essential for task implementation. We experiment with various navigation tasks and show that the learned reward function can be used to generate policies that outperform the baselines in both training and test environments. For future work, we will apply our algorithm to more complex environments with continuous state spaces.

REFERENCES

[1] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning." in *Proceedings of ICML*, 2000, pp. 663–670.

[2] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of ICML*, 2004.

[3] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *Proceedings of AAAI*, 2008, pp. 1433–1438.

[4] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

[5] M. Lopes, F. Melo, and L. Montesano, "Active learning for reward estimation in inverse reinforcement learning," in *ECML PKDD*, 2009.

[6] P. Odom and S. Natarajan, "Active advice seeking for inverse reinforcement learning," in *Proceedings of AAMAS*, 2016, p. 512–520.

[7] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto, "Learning and generalization of complex tasks from unstructured demonstrations," in *IROS*. IEEE, 2012, pp. 5239–5246.

[8] B. Michini, T. J. Walsh, A.-A. Agha-Mohammadi, and J. P. How, "Bayesian nonparametric reward learning from demonstration," *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 369–386, 2015.

[9] K. Shiarlis, M. Wulfmeier, S. Salter, S. Whiteson, and I. Posner, "Taco: Learning task decomposition via temporal alignment for control," in *Proceedings of ICML*, 2018, pp. 4661–4670.

[10] P. Henderson, W.-D. Chang, P.-L. Bacon, D. Meger, J. Pineau, and D. Precup, "Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[11] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in neural information processing systems*, 2016, pp. 4565–4573.

[12] T. Kipf, Y. Li, H. Dai, V. Zambaldi, A. Sanchez-Gonzalez, E. Grefenstette, P. Kohli, and P. Battaglia, "Compile: Compositional imitation learning and execution," *arXiv preprint arXiv:1812.01483*, 2018.

[13] Z. Xu, B. Wu, D. Neider, and U. Topcu, "Active finite reward automaton inference and reinforcement learning using queries and counterexamples," 2020.

[14] C. Neary, Z. Xu, B. Wu, and U. Topcu, "Reward machines for cooperative multi-agent reinforcement learning," 2020.

[15] R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *ICML'2018*, 2018.

[16] M. Hasanbeig, A. Abate, and D. Kroening, "Logically-constrained neural fitted q-iteration," *arXiv preprint arXiv:1809.07823*, 2018.

[17] L. Z. Yuan, M. Hasanbeig, A. Abate, and D. Kroening, "Modular deep reinforcement learning with temporal logic specifications," *arXiv preprint arXiv:1909.11591*, 2019.

[18] M. Hasanbeig, A. Abate, and D. Kroening, "Certified reinforcement learning with logic guidance," *arXiv preprint arXiv:1902.00778*, 2019.

[19] J. Fu and U. Topcu, "Probably approximately correct MDP learning and control with temporal logic constraints," *Robotics: Science and Systems*, vol. abs/1404.7073, 2014.

[20] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Teaching multiple tasks to an RL agent using LTL," in *Proc. AAMAS'2018*, 2018, pp. 452–461.

[21] M. Wen, I. Papusha, and U. Topcu, "Learning from demonstrations with high-level side information," in *Proc. IJCAI'17*, 2017, pp. 3055–3061. [Online]. Available: https://doi.org/10.24963/ijcai.2017/426

[22] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *AAAI'18*, 2018.

[23] A. Camacho, R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "LTL and beyond: Formal languages for reward function specification in reinforcement learning," in *IJCAI'2019*, 7 2019, pp. 6065–6073. [Online]. Available: https://doi.org/10.24963/ijcai.2019/840

[24] Z. Xu and U. Topcu, "Transfer of temporal logic formulas in reinforcement learning," in *Proc. IJCAI'2019*, 7 2019, pp. 4010–4018. [Online]. Available: https://doi.org/10.24963/ijcai.2019/557

[25] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee, "Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees," *arXiv preprint arXiv:1909.05304*, 2019.

[26] Z. Xu, I. Gavran, Y. Ahmad, R. Majumdar, D. Neider, U. Topcu, and B. Wu, "Joint inference of reward machines and policies for reinforcement learning," in *To appear in Proc. International Conference on Automated Planning and Scheduling (ICAPS), Special Track on Planning and Learning*, 2020.

[27] R. A. T. Icarte, E. Waldie, T. Klassen, R. Valenzano, M. P. Castro, and S. A. McIlraith, "Learning reward machines for partially observable reinforcement learning," in *NeurIPS*, 2019.

[28] M. Wen, I. Papusha, and U. Topcu, "Learning from demonstrations with high-level side information," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[29] S. Krishnan, A. Garg, R. Liaw, L. Miller, F. T. Pokorny, and K. Goldberg, "Hirl: Hierarchical inverse reinforcement learning for long-horizon tasks with delayed rewards," *arXiv preprint arXiv:1604.06508*, 2016.

[30] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of ICML*, 2006, pp. 729–736.

[31] G. Neu and C. Szepesvári, "Apprenticeship learning using inverse reinforcement learning and gradient methods," in *Proceedings of UAI*, 2007, pp. 295–302.

[32] E. Klein, M. Geist, B. Piot, and O. Pietquin, "Inverse reinforcement learning through structured classification," in *Proceedings of NeurIPS*, 2012, pp. 1007–1015.

[33] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and computation*, vol. 75, no. 2, pp. 87–106, 1987.

[34] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *Proceedings of ICML*, 2016, pp. 49–58.

[35] M. Wulfmeier, D. Z. Wang, and I. Posner, "Watch this: Scalable cost-function learning for path planning in urban environments," in *IROS*. IEEE, 2016, pp. 2089–2095.

[36] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1073–1087, 2017.

[37] Z. Zhou, M. Bloem, and N. Bambos, "Infinite time horizon maximum causal entropy inverse reinforcement learning," *IEEE Transactions on Automatic Control*, vol. 63, no. 9, pp. 2787–2802, 2018.

[38] B. Bollig, J.-P. Katoen, C. Kern, M. Leucker, D. Neider, and D. R. Piegdon, "libalf: The automata learning framework," in *CAV*, 2010, pp. 360–364.