Goal Recognition as Reinforcement Learning

Leonardo Amado[1], Reuth Mirsky, [2], Felipe Meneguzzi [3,1]

[1] Pontifícia Universidade Católica do Rio Grande do Sul, Brazil
[2] Bar Ilan University, Israel and The University of Texas at Austin, USA
[3] University of Aberdeen, Scotland
leorosaamado@gmail.com, mirskyr@cs.biu.ac.il, felipe.meneguzzi@abdn.ac.uk

Abstract

Most approaches for goal recognition rely on specifications of the possible dynamics of the actor in the environment when pursuing a goal. These specifications suffer from two key issues. First, encoding these dynamics requires careful design by a domain expert, which is often not robust to noise at recognition time. Second, existing approaches often need costly real-time computations to reason about the likelihood of each potential goal. In this paper, we develop a framework that combines model-free reinforcement learning and goal recognition to alleviate the need for careful, manual domain design, and the need for costly online executions. This framework consists of two main stages: offline learning of policies or utility functions for each potential goal, and online inference. We provide a first instance of this framework using tabular Q-learning for the learning stage, as well as three mechanisms for the inference stage. The resulting instantiation achieves state-of-the-art performance against goal recognizers on standard evaluation domains and superior performance in noisy environments.

Introduction

Goal recognition (GR) is a key task in artificial intelligence, where a *recognizer* infers the goal of an *actor* based on a seof observations. Consider a service robot that wishes to assist a person in the kitchen by fetching appropriate utensils without interrupting the task execution or demanding attention for specifying instructions (**????**). A common approach to enable the robot to perceive and infer the person's goal in this situation consists of a pipeline of activity recognition from raw images and translation into actions for a symbolic GR algorithm (Figure 1). After processing the raw images into observations, a goal recognizer further processes a seof these observations into a goal or a distribution of goals. Most GR approaches rely on arduously informing the recognizer about the feasibility and likelihood of the different actions that the actor can execute. This process might include crafting elaborate domain theories, multiple planner executions in real-time, intricate domain optimizations, or any combination of these tasks, leading to limitations:

Cost of Domain Description: Crafted domain theories require deliberate design and accurate specification of domain dynamics, which is usually a process done manually by an expert. In highly complex environments, manual elicitation of such a model might even be impossible.

Noise Susceptibility: As specifying accurate domain dynamics is costly, many specifications are incomplete and cannot inform the recognizer about unlikely observations or partial observation se. This property makes many goal recognizers susceptible to noise.

Online Costs: Some recognizers require costly online computations, such as multiple planner or parser executions. These computations can hinder the recognizer's real-time inference ability, especially when observations are processed incrementally and the goal of the actor needs to be reevaluated many times throughout the plan execution.

We address these limitations by replacing manually crafted representations and online executions with model-free Reinforcement Learning (RL) techniques. The resulting framework performs efficient and noise-resistant GR without the need to craft a domain model and without any planner or parser executions during recognition.

The three key contributions of this paper are: (1) Revisiting the GR problem definition to accommodate RL-based domains and developing a new framework that relies on policies or utility functions derived from any model-free RL technique. This framework consists of two main stages: Offline learning for each potential goal, and an online inference stage that compares an observed trajectory to those learned policies. (2) A first instance of the new framework for tabular RL using an off-the-shelf implementation of a Q-learning algorithm and three recognition measures: Accumulated utility, a modified KL-divergence, and Divergence Point. (3) Evaluation of the new framework on domains with partial and noisy observability. Experiments show that even with a very short learning process, we can still accurately and robustly perform GR on challenging problems. We show this framework's ability to perform comparably to a state-of-the-art goal recognizer on standard evaluation domains, and have superior performance in noisy environments.

Background

We begin by defining a GR problem in a way that is consistent with existing literature (**??**). Given a domain theory $\mathbb{T}$, a set of possible goals $\mathcal{G}$, and a seof observations $\boldsymbol{O}$, a goal recognition problem consists of a goal $g \in \mathcal{G}$

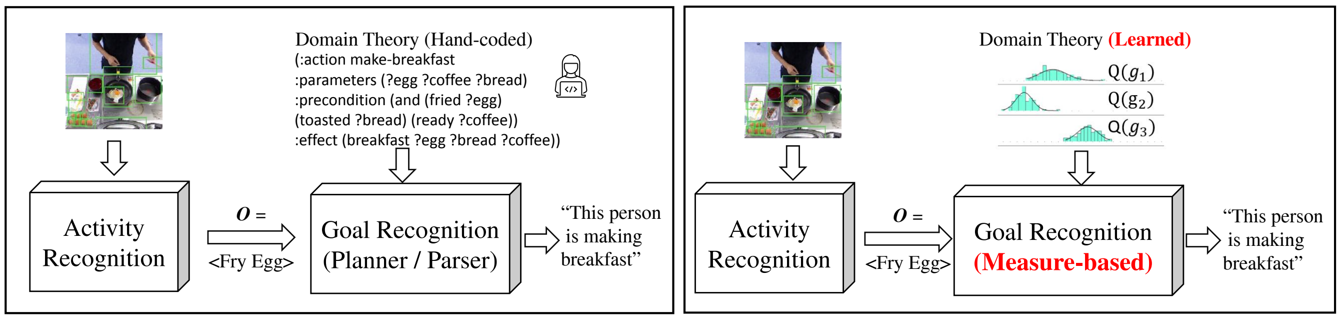
Domain Theory (Hand-coded)
(:action make-breakfast
:parameters (?egg ?coffee ?bread)
:precondition (and (fried ?egg)
(toasted ?bread) (ready ?coffee))
:effect (breakfast ?egg ?bread ?coffee))
Activity Recognition
O =
<Fry Egg>
Goal Recognition (Planner / Parser)
"This person is making breakfast"
Domain Theory (Learned)
Q(g1)
Q(g2)
Q(g3)
Activity Recognition
O =
<Fry Egg>
Goal Recognition (Measure-based)
"This person is making breakfast"

Figure 1: A comparison of existing model-based approaches for goal recognition (left) and our proposed framework (right). The key changes in our approach are presented in red.

that **explains** $\boldsymbol{O}$. The semantics of $\mathbb{T}$ and **explains** can vary greatly between goal recognizers. For example, in Ramirez and Geffner (**?**), a domain theory is a planning domain instantiated in a specific initial state s_0 and goal g is explained by $\boldsymbol{O}$ if there is some optimal plan for g, generated by a planner, that begins in s_0 and is compatible with $\boldsymbol{O}$. They refine this interpretation to rank goals' likelihood when there is more than one goal with an optimal plan that is compatible with $\boldsymbol{O}$ (**?**). In this work we propose multiple semantics for **explains**, but we first focus on defining our RL-based domain theory. For that, we use the definition of a Markov Decision Process (MDP), a policy, and a Q-function (**?**).

Definition 1 (MDP) *A **Markov Decision Process** M, is a 4-tuple $(\mathcal{S}, \mathcal{A}, p, r)$ such that $\mathcal{S}$ are the possible states in the environment, $\mathcal{A}$ is the set of actions the actor can execute, $p(s' \mid s, a)$ is a transition function that gives the probability of transitioning from state s to state s' after taking action a and $r(s, a, s')$ defines a reward function.*

A **policy** $\pi(a \mid s)$ for an MDP is a function that defines the probability of the agent taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. Some RL algorithms, such as Q-learning, compute the policy of an agent using a **Q-function** $Q(s, a)$, which is an estimation of the expected return starting from s after taking action a. In our new framework, a domain theory $\mathbb{T}$ consists of the state and action spaces of an MDP and a set of policies or Q-functions. Unlike planning-based GR where the domain theory is decoupled from the problem instance (the set of possible goals $\mathcal{G}$), here $\mathbb{T}$ depends on the set of goals. We define two types of domain theories:

Definition 2 (Utility-based Domain Theory) *A utility-based domain theory $\mathbb{T}_Q(\mathcal{G})$ is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{Q})$ such that $\mathcal{Q}$ is a set of Q-functions $\{Q_g\}_{g \in \mathcal{G}}$.*

Definition 3 (Policy-based Domain Theory) *A policy-based domain theory $\mathbb{T}_\pi(\mathcal{G})$ is a tuple $(\mathcal{S}, \mathcal{A}, \Pi)$ such that Π is a set of policies $\{\pi_g\}_{g \in \mathcal{G}}$.*

Essentially, we can view both domain theories as a set of MDPs with the same transitions but different reward functions for different goals. Our aim is to learn either a good policy or a utility function that represents the expected behavior of actors under each of these MDPs. We use this formulation to provide a new definition for a goal recognition problem in which we replace the abstract notion of $\mathbb{T}$ and combine the goal set $\mathcal{G}$ into these domain theories.

Definition 4 (Goal Recognition Problem) *Given domain theory $\mathbb{T}_Q(\mathcal{G})$ or $\mathbb{T}_\pi(\mathcal{G})$ and a seof observations $\boldsymbol{O}$, output a goal $g \in \mathcal{G}$ that **explains** $\boldsymbol{O}$.*

Note that every GR approach using a policy-based domain theory $\mathbb{T}_\pi(\mathcal{G})$, can also be given by a utility-based domain theory $\mathbb{T}_Q(\mathcal{G})$. We can make this change by generating for each goal g a softmax policy π_g based on Q_g, as shown in Equation 1. Consider a case where all values in a Q-function are negative. This case would actually result in the policy π_g being more likely to take the worst action. Thus, if the utility function has some negative values, we rescale the function with the additive inverse of the largest negative value (some number $-C$): $Q' = Q(s, a) + C$. This modification ensures that the resulting policy π_g will prioritize high-value actions. Thus, for brevity, from now on we refer to our framework as one that relies on Q-functions, unless we explicitly wish to discuss specific properties of a policy-based domain theory.

$$\pi_g(a \mid s) = \frac{Q_g(s, a)}{\sum_{a' \in \mathcal{A}} Q_g(s, a')} \tag{1}$$

Using this new problem definition, we develop our framework to solve these goal recognition problems, discuss how to learn $\mathbb{T}_Q(\mathcal{G})$ or $\mathbb{T}_\pi(\mathcal{G})$, and how to decide which goal g best **explains** observations $\boldsymbol{O}$.

The Goal Recognition as Reinforcement Learning Framework

Our new framework consists of two main stages: (1) learning a set of Q-functions; and (2) inferring the goal of an actor given a seof observations. Figure 2 illustrates this process. First, the initial inputs are state and action spaces, $\mathcal{S}$ and $\mathcal{A}$, and a set of goals $\mathcal{G}$. There is no restriction on the properties of $\mathcal{S}$ and $\mathcal{A}$; they can be either discrete or continuous, and no transition or reward function is required a-priori. Our framework can employ any off-the-shelf RL algorithm to learn a Q-function for each goal, $\{Q_g\}_{g \in \mathcal{G}}$ which together with the original $\mathcal{S}$ and $\mathcal{A}$ constructs the domain theory $\mathbb{T}_Q(\mathcal{G})$ for the recognition stage. Once the framework receives an observation se a measure between an observation seand a

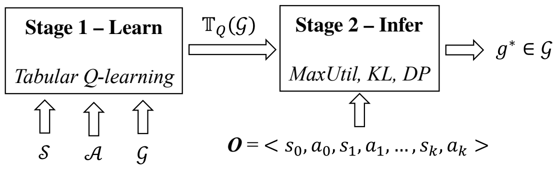

Figure 2: The GR as RL framework. The Goal Recognition as Q-Learning (GRAQL) instance appears in *italics*.

Q-function computes a distance between the observations $\boldsymbol{O}$ and each Q_g. Here we focus primarily on state-action observation se, where $\boldsymbol{O} = \langle s_0, a_0, s_1, a_1, \ldots \rangle$, but the next Section contains examples of how to handle state-only ($\boldsymbol{O}^s = \langle s_0, s_1, \ldots \rangle$) or action-only ($\boldsymbol{O}^a = \langle a_0, a_1, \ldots \rangle$) observations. The inferred goal g^* is the one that minimizes the measured distance between its respective Q-function and the observations, as defined in Equation 2.

$$g^* = \arg\min_{g \in \mathcal{G}} \textsc{Distance}(Q_g, \boldsymbol{O}) \qquad (2)$$

Stage 1 (Learning): The first part of the GR as RL framework is learning a set of Q-functions (or policies) for each goal following Algorithm 1. It defines a set of n RL problems where n is the number of goals in $\mathcal{G}$: for each goal g, we generate a reward function in which there is some positive gain when reaching the goal g (Line 3), and no reward otherwise (Line 2). This basic setting allows us to leverage additional reward shaping or other optimizations to improve the learning of the Q-functions, just as in any other RL problem. However, in the most naive form of this problem, we require no penalty for actions that fail to advance the agent towards reaching the goal, nor any specific discount factor. We explicitly chose this simple problem formulation to highlight the efficacy of our approach, as its aim is to generate informative-enough Q-functions, not to perfectly maximize the reward of the RL agent. As we show in our empirical evaluation, even though we do not train Q-functions until convergence and provide poor solvers for reaching their respective goals, they suffice to create an accurate and robust domain theory for our goal recognizers.

This formulation enables us to use well-established RL research to learn a set of Q-functions given the properties of our environment: discrete or continuous, deterministic or stochastic transitions, etc. Acquiring the domain theory then becomes an RL problem with its respective challenges: selecting the most appropriate algorithm for learning, tuning its hyperparameters, etc.

Stage 2 (Inference): Once we have a set of Q-functions $Q_{\mathcal{G}}$ and an observation se$\boldsymbol{O}$, the next stage in the framework is online GR: inferring the Q-function (i.e., goal) of the actor that explains $\boldsymbol{O}$. Traditional GR algorithms require complex computations, such as planner or parser executions to reason about the similarity of $\boldsymbol{O}$ to each goal. In this work, we take on a measure-based approach instead (and present potential measures). Algorithm 2 implements the inference process using DISTANCE as a measure function, which implements Equation 2: given $\boldsymbol{O}$, find for each goal g the distance between $\boldsymbol{O}$ and Q_g (Line 3). The algorithm then chooses the goal with the minimum distance value as the most likely goal of the actor (Line 6). This formulation of the GR task aligns well with Ramirez and Geffner (**?**), who introduce the notion of similarity between an observation seand optimal, goal- and observation-dependent plans. As the algorithm computes the similarity between the observation seand a Q-function that is defined over all state-action pairs rather than a single trajectory, it inherently reasons about noisy and missing observations, as our empirical evaluation shows.

Algorithm 1: Learn a Q-function for each goal

```
Require: $\mathcal{S}, \mathcal{A}$ : State and action spaces
Require: $\mathcal{G}$: a set of candidate goals
1: for all $g \in \mathcal{G}$ do
2:     $\forall a \forall s \neq g, r(s,a) \leftarrow 0$   ▷ Create a reward function
3:     $\forall a, r(g,a) \leftarrow C$   ▷ Reaching $g$ yields some positive value
4:     $Q_g \leftarrow \textsc{Learn}(\mathcal{S}, \mathcal{A}, r)$
5: return $\{Q_g\}_{g \in \mathcal{G}}$
```

Algorithm 2: Infer most likely goal for the observations

```
Require: $\mathbb{T}_Q(\mathcal{G})$: $\mathcal{S}, \mathcal{A}, \{Q_g\}_{g \in \mathcal{G}}$ : State and action spaces, and Q-functions per goal
Require: $\boldsymbol{O}$: an observation se$\langle s_0, a_0, s_1, a_1, \ldots \rangle$
1: $m_{g^*} \leftarrow \infty$   ▷ Init shortest distance
2: for all $g \in \mathcal{G}$ do   ▷ Compute distances from $\boldsymbol{O}$
3:     $m_g \leftarrow \textsc{Distance}(Q_g, \boldsymbol{O})$   ▷ Use distance measure
4:     if $m_g \leq m_{g^*}$ then
5:         $g^* \leftarrow g$ and $m_{g^*} \leftarrow m_g$
6: return $g^*$
```

Goal Recognition as Q-Learning

In this section, we detail the components of the first instance of our framework specifically for tabular Q-learning approaches — Goal Recognition as Q-Learning (GRAQL). We focus the first instance of this framework on tabular domains to enable an evaluation against existing GR baselines. Planning-based GR algorithms use PDDL as their domain descriptions, which can be easily translated into tabular representations (**??**). Figure 2 illustrates the specific components that require implementation in *italics*. We start by explaining the hyperparameters and discussing these choices in the learning stage. Then, we introduce three different measures for an observation seand a Q-function.

Learning $\{Q_g\}_{g \in \mathcal{G}}$ Using Q-Learning

For the learning stage, we use an off-the-shelf Q-learning algorithm. As learning the Q-functions for each goal are a means to an end rather than our ultimate aim, we do not focus on techniques to optimize this stage, but rather employ a single solution, showing that we can acquire an informative domain theory with minimal effort. We set the reward for reaching the goal to 100, and 0 otherwise, and the discount factor to 0.9. As exploration is more important in this case than maximizing the reward, the sampling strategy we use is

ϵ-greedy with linearly decaying values ($\epsilon = 1 \ldots 0.01$).

Instead of using random exploration to reach the goal, shaping the initial policy can speed up the learning process: for each goal g, an optimal planner generates a single trajectory to the goal $p_g = \{\langle s_0, a_0\rangle, \langle s_1, a_1\rangle, \ldots\}$. We can use this trajectory to initialize Q_g with positive values for state-action pairs that are part of its goal's optimal path p_g.

$$Q_g(s,a) = \begin{cases} 1, & \text{if } \langle s,a\rangle \in p_g \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

This shaping initializes the Q-function to give high utility to a single trajectory, which is similar to the original formulation of planning-based GR, where a single optimal plan constitutes the baseline for the actor's presumed path to the goal (**?**). In that sense, GRAQL bridges a gap between planning-based GR and RL: in that formulation, a planner outputs a single optimal plan for goal g, which might not be the plan the actor chooses to follow. Later work overcomes this problem by searching for a set of diverse plans for each goal (**?**). Instead of using search, GRAQL **refines** a single optimal plan into a policy that captures the cost of alternative plans, even if these plans are not necessarily close to optimal. We implemented our approach with and without this shaping process. To ensure that it does not overfit or create an unfair bias in the Q-functions towards the planning-based observation seused in our evaluation, we explicitly chose problems with multiple optimal plans per goal and ran different planners for shaping (LAMA (**?**)) and for testing (Fast Downward (**?**)), so that p_g is not the only possible optimal path. The resulting Q-functions with and without shaping were not significantly different, so our empirical results only show the performance of the Q-functions without the shaping process.

Measures for Tabular Q-functions

For the inference stage, we use three different distance measures for a distance between a Q-function Q_g and an action-state observation se$\boldsymbol{O}$, inspired by three common RL measures: MaxUtil, KL-divergence, and Divergence Point. We then extend MaxUtil's definition to handle state-only observation se and action-only observation se.

MaxUtil is an accumulation of the utilities collected from the observed trajectory.

$$MaxUtil(Q_g, \boldsymbol{O}) = \sum_{i \in |\boldsymbol{O}|} Q_g(s_i, a_i) \tag{4}$$

KL-Divergence is a measure for the distance between two distributions, so we construct two policies, π_g and $\pi_{\boldsymbol{O}}$ for Q_g and $\boldsymbol{O}$ respectively. The goal-dependent policy π_g is defined as a softmax stochastic policy (Equation 1). The observations policy $\pi_{\boldsymbol{O}}$ is a pseudo-policy where $\pi_{\boldsymbol{O}}(a_i \mid s_i) = 1$ for each $\langle s_i, a_i\rangle \in \boldsymbol{O}$ and provides a uniform distribution for all actions taken in unobserved states.

$$KL(\boldsymbol{O}, Q_g) = \mathrm{D}_{\mathrm{KL}}(\pi_{\boldsymbol{O}} \mid\mid \pi_g) = \sum_{i \in |\boldsymbol{O}|} \pi_{\boldsymbol{O}}(a_i \mid s_i) \log \frac{\pi_{\boldsymbol{O}}(a_i \mid s_i)}{\pi_g(a_i \mid s_i)} \tag{5}$$

Divergence Point (DP) is a measure adapted from Macke et al. (**?**), where given a trajectory $\boldsymbol{O}$ and a policy π, it is defined as the minimal point in time in which the action taken by $\boldsymbol{O}$ has zero probability to be chosen by π. We implement a softer version of the original measure, where the probability threshold is a parameter δ instead of exactly 0. The reason for this softened version of DP is that, for similar enough goals, the probability of an action to be chosen for both goals is unlikely to be exactly 0. Finally, as DP gets higher values when $\boldsymbol{O}$ and π share more resemblance, we take its additive inverse to get a distance compatible with the minimization formulation of Algorithm 2. Here too, the goal-dependent policy π_g is defined as the softmax stochastic policy from Equation 1:

$$DP(Q_g, \boldsymbol{O}) = -min\{t \mid \pi_g(a_{t-1} \mid s_{t-1}) \leq \delta\} \tag{6}$$

MaxUtil for State-only $\boldsymbol{O}$ applies to states-only: $\boldsymbol{O}^s = \langle s_0, s_1, \ldots\rangle$. In this case, similar to offline policy learning, we optimistically take the action with the highest Q-value:

$$MaxUtil(Q_g, \boldsymbol{O}^s) = \sum_{i \in |\boldsymbol{O}|} \max_a Q_g(s_i, a) \tag{7}$$

MaxUtil for Action-only $\boldsymbol{O}$ applies to actions-only: $\boldsymbol{O}^a = \langle a_0, a_1, \ldots\rangle$. We optimistically take the state with the highest Q-value in this case as well. To do that, we first need to find the set of all states for which the observation a_i is an optimal action according to Q_g. This set can be formally defined as $Opt(a_i \mid Q_g) = \{s \in \mathcal{S} \mid Q_g(s, a_i) \geq Q_g(s,a) \forall a \in \mathcal{A}\}$. From this set, we choose the state with the maximal utility (presumed to be the state in the optimal path) and associate this utility with the observation.

$$MaxUtil(Q_g, \boldsymbol{O}^a) = \sum_{i \in |\boldsymbol{O}|} \max_{s \in Opt(a_i \mid Q_g)} Q_g(s, a_i) \tag{8}$$

Experimental Evaluation

To be able to compare GRAQL and planning-based GR, we use PDDLGym (**?**) as our evaluation environment. PDDLGym is a python framework that automatically constructs OpenAI Gym environments from PDDL domains and problems. Thus, for each PDDL domain used by state-of-the-art GR algorithms, we generate the parallel representation in Gym for GRAQL. We use three domains from the PDDLGym library for their similarity with commonly used GR evaluation domains: Blocks, Hanoi, and SkGrid (The latter highly resembles common GR navigation domains such as those used by **?** (**?**)). For each domain, we generate 10 GR problems with 4 candidate goals in $\mathcal{G}$. We manually choose ambiguous goals, i.e., goals that are close to one another rather than in different corners of a grid. Each problem has 7 variants, including partial and noise observations. We have 5 variants with varying degrees of observability (10%, 30%, 50%, 70%, and full observability), and 2 variants that include noise observations with varying degrees of observability (50% and full observability). Thus, our test set includes 210 GR problems. These GR problems can pose a real challenge to existing recognizers, especially when they are partially observable or noisy. Note that, while PDDLGym uses a PDDL description to drive the simulations, our approach only has access to the same data available through a regular Gym simulator, i.e., discrete observations, action names, and reward function.

Next, we discuss the hyperparameters used in this evaluation and explain our three sets of performance tests, using different types of $\boldsymbol{O}$ as input: state-action pairs where $\boldsymbol{O}$ is fully observable; state-action pairs with missing observations, noisy observations, or both; and state-only or action-only trajectories with missing observations. We use standard machine learning metrics in our evaluation: accuracy, precision, recall, and F-score. We note that the accuracy metric reported here is different from the accuracy metric in **?** (**?**), which refers to the ratio of problems where the recognizer ranks the true goal (i.e., True Positives) with the highest likelihood, and the number of problems tested. This measure differs from standard ML accuracy, which also takes into consideration whether the algorithm successfully ranks incorrect goals *lower* than the correct one (i.e., True Positives + True Negatives). In our experiments, we report accuracy using the latter definition.

Hyperparameters and Algorithmic Choices

For the learning stage of our experiments, we use an off-the-shelf model-free Q-learning algorithm (**?**). For each goal, we run the learner for a fixed number of episodes, whether it reaches convergence or not. We evaluate greedy policy executions after training the Q-functions for 500, $10k$ and $30k$ episodes. The difference in the resulting plan lengths was around 5%, and went both ways—sometimes the policy learned after 500 episodes was not informed enough and drifted away from the goal during execution, and sometimes policies learned after $30k$ episodes overfitted a specific solution. The key differentiator is often whether the learner reaches the goal once during training, as after that the learned Q-function can generate an optimal plan for that goal. Since the performance of these training regimes does not vary much, we report our empirical results with a consistent value of 500 training episodes.

For the inference stage, we compare our framework with each of our three proposed measures, denoted MaxUtil, KL, and DP, as well as the probabilistic planning-based recognizer by Ramirez and Geffner (**?**), denoted R&G. For the DP measure, we use a threshold probability for divergence of $\delta = 0.1$. As we wish to be able to compare GRAQL against a planning-based algorithm, we focus our evaluation on discrete state and action spaces, but we conjecture that the framework can directly handle continuous spaces by using the same measures, only with a different learning algorithm.

Testing under Full Observability

First, we show the performance of MaxUtil, KL, DP, and R&G on fully observable, optimal state-action trajectories. The average length of these trajectories is 8 $(\min = 5, \max = 15)$ steps for Blocks, 10.4 $(\min = 5, \max = 15)$ for Hanoi, and 12.4 $(\min = 6, \max = 24)$ for SkGrid. Figure 3 summarizes the performance of each approach over ten instances per setup. With full observability, almost all results

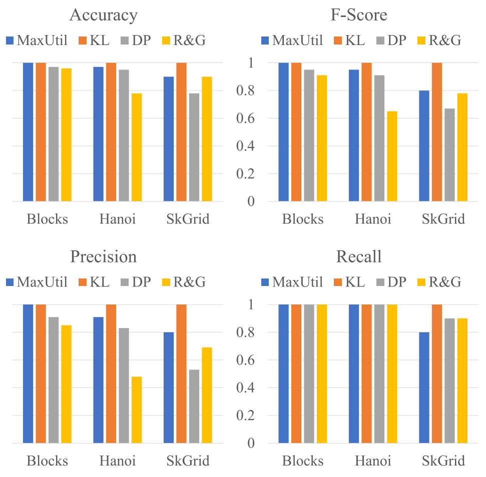

Figure 3: Comparison of R&G, MaxUtil, KL, DP by their accuracy, precision, recall, and F-score for full observability.

are above 80%, and KL-divergence achieves perfect performance, consistently predicting the actor's goal correctly. Notably, DP performs worse than the other GRAQL measures, especially in the SkGrid environment. As DP requires an action to be highly unlikely for some goal to rank it as incorrect, the high variability of SkGrid presents a real challenge to it. In this domain, the same goal might have a number of equally optimal trajectories requiring diagonal movement due to the nature of square-grid navigation environments. All GRAQL approaches perform similarly or better than R&G, except for DP on SkGrid, where the largest difference was in precision (DP: 0.53, R&G: 0.69). On the other hand, the performance of R&G in the Hanoi environment is inferior to all GRAQL methods in terms of accuracy, (DP: 0.95, R&G: 0.78), precision (DP:0.83, R&G: 0.48), and F-score (DP:0.91, R&G:0.65). Hanoi has many actions that appear in plans for different goals, causing high ambiguity in recognition time, which makes it especially challenging to R&G to distinguish between those goals. These results show that GRAQL is able to achieve comparable results to the state-of-the-art with fully observable trajectories.

Testing under Partial Observability and Noise

We evaluate our approaches under partial observability with varying degrees of observability (10%, 30%, 50%, 70%, and full observability). We use the same trajectories in all experiments, removing steps randomly to achieve a specific observability ratio. Table 1 shows the average performance of each approach over ten instances per setup. It is clear that as observability decreases, so does the performance of all approaches. However, partial observability seems to highly affect the performance of R&G, with values that decrease to about a half in the 10% observability level (e.g., accuracy of 0.96 drops to 0.44 in Blocks). In general, KL and MaxUtil perform better than DP and R&G, except for the recall metric, where these latter ones are the superior approaches. The reason for this loss of recall is that DP and R&G are more likely to have ties between potential goals, so when these

		Accuracy				Precision				Recall				F-Score			
O	Domain	MxU	KL	DP	RG	MxU	KL	DP	RG	MxU	KL	DP	RG	MxU	KL	DP	RG
10%	Blocks	**0.93**	0.90	**0.93**	0.44	**0.82**	0.80	0.77	0.25	0.90	0.80	**1.00**	0.90	0.86	0.80	**0.87**	0.39
	Hanoi	0.93	**0.95**	0.90	0.50	0.77	**0.90**	0.71	0.29	**1.00**	0.90	**1.00**	**1.00**	0.87	**0.90**	0.83	0.44
	SkGrid	0.80	**0.90**	0.55	0.72	0.60	**0.80**	0.36	0.42	0.60	0.80	**1.00**	**1.00**	0.60	**0.80**	0.53	0.59
30%	Blocks	**1.00**	0.95	0.97	0.74	**1.00**	0.90	0.91	0.42	**1.00**	0.90	**1.00**	0.80	**1.00**	0.90	0.95	0.55
	Hanoi	**0.95**	**0.95**	0.93	0.68	0.83	**0.90**	0.77	0.38	**1.00**	0.90	**1.00**	**1.00**	**0.91**	0.90	0.87	0.55
	SkGrid	0.90	**0.95**	0.70	0.88	0.80	**0.90**	0.45	0.63	0.80	0.90	**1.00**	**1.00**	0.80	**0.90**	0.62	0.77
50%	Blocks	**1.00**	**1.00**	0.97	0.80	**1.00**	**1.00**	0.91	0.50	**1.00**	**1.00**	**1.00**	0.90	**1.00**	**1.00**	0.95	0.64
	Hanoi	**0.95**	**0.95**	0.93	0.72	0.83	**0.90**	0.77	0.42	**1.00**	0.90	**1.00**	**1.00**	**0.91**	0.90	0.87	0.59
	SkGrid	0.80	**0.90**	0.72	0.88	0.60	**0.80**	0.48	0.63	0.60	0.80	**1.00**	**1.00**	0.60	**0.80**	0.65	0.77
70%	Blocks	**1.00**	**1.00**	0.97	0.94	**1.00**	**1.00**	0.91	0.77	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.95	0.87
	Hanoi	**0.95**	0.90	0.93	0.72	**0.83**	0.80	0.77	0.42	**1.00**	0.80	**1.00**	**1.00**	**0.91**	0.80	0.87	0.59
	SkGrid	0.85	**0.95**	0.72	0.92	0.70	**0.90**	0.47	0.71	0.70	0.90	0.90	**1.00**	0.70	**0.90**	0.62	0.83
100%	Blocks	**1.00**	**1.00**	0.97	0.96	**1.00**	**1.00**	0.91	0.85	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.95	0.92
	Hanoi	0.97	**1.00**	0.95	0.78	0.91	**1.00**	0.83	0.48	**1.00**	**1.00**	**1.00**	**1.00**	0.95	**1.00**	0.91	0.65
	SkGrid	0.90	**1.00**	0.78	0.90	0.80	**1.00**	0.53	0.69	0.80	**1.00**	0.90	0.90	0.80	**1.00**	0.67	0.78
Avg	Blocks	**0.98**	0.97	0.96	0.78	**0.96**	0.94	0.88	0.56	0.98	0.94	**1.00**	0.92	**0.97**	0.94	0.93	0.67
	Hanoi	**0.95**	**0.95**	0.93	0.68	0.83	**0.90**	0.77	0.40	**1.00**	0.90	**1.00**	**1.00**	**0.91**	0.90	0.87	0.56
	SkGrid	0.85	**0.94**	0.69	0.86	0.70	**0.88**	0.45	0.62	0.70	0.88	0.96	**0.98**	0.70	**0.88**	0.61	0.75

Table 1: Impact of partial observability: comparing MxU, KL, DP, RG with varying observability levels of $\boldsymbol{O}$.

approaches cannot distinguish between them using $\boldsymbol{O}$, they return multiple goals, trading off recall with precision.

Finally, we evaluate our approaches with the addition of noise in the observations. We add noise to the observations by first generating an optimal plan using the Fast Downward planner. We then randomly choose a step-index and inject two consecutive non-optimal actions, which induces states along this sub-optimal path as a state-action pair, from that step, thus forcibly deviating the observed plan from the optimal. Finally, to get back to the goal with no additional noise, we rerun the planner to get an optimal plan to the goal from the state reached after executing these two noisy actions. Using this process to add noise, we have four additional noisy actions per trajectory for all of our environments: two that make the agent drift away from its goal, and two additional actions to backtrack. We chose this form of noise as these actions are still valid, even if not optimal. An alternative noise could have been an injection of invalid actions, or any random state-action pair that is not part of the generated trajectory. However, R&G and most other planning-based approaches cannot trivially reason about this type of noise, as they will simply label that noisy plan as an impossible plan for the goal. We tested the noisy trajectories with full observability and with partial observability set to 0.5. Table 2 shows these results. Unlike the noise-free case, in these results MaxUtil outperforms KL in most setups in terms of accuracy, precision, and F-score. When comparing the overall performance of each approach with and without noise, KL and R&G are more noise-sensitive than MaxUtil and DP.

State-Only and Action-Only Observations

R&G, for example, uses only actions in its inference process, but no states (or vice versa), even if they are available. Our next set of experiments shows the performance of GRAQL when given such observations, and compares it with R&G. We use the same observation se as in our partial observability experiments, but we provide our MaxUtil-based approaches with either the full se, the states $\boldsymbol{O}^s = \langle s_0, s_1, \ldots \rangle$ (MaxUtil for state-only, in Equation 7) or the actions $\boldsymbol{O}^a = \langle a_0, a_1, \ldots \rangle$ (MaxUtil for action-only, in Equation 8). Figure 4 summarizes these results, where each bar represents the average performance for all observation levels (from 10% to full). All versions of MaxUtil perform well in Blocks and Hanoi and outperform R&G in all metrics but recall. The main issue is the Action-only version in the SkGrid domain (e.g. accuracy of 0.54 and precision of 0.23), which underperforms significantly against other versions. Equation 8 estimates states using the most optimistic of all of the states for which the observed action is an optimal action. Given SkGrid's structure, every action is an optimal action for about half of the states, thus taking this optimistic approach is unlikely to be accurate.

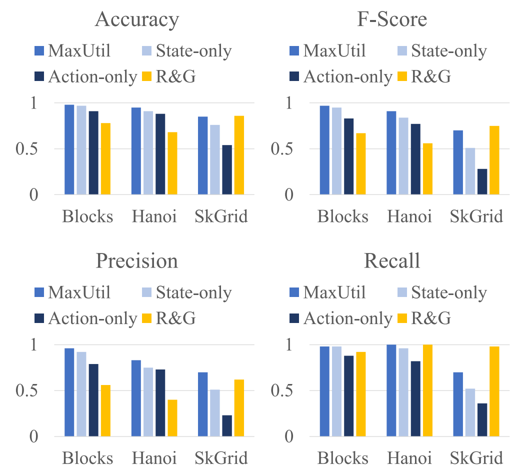

Figure 4: Performance comparison of R&G and MaxUtil with $\boldsymbol{O}^s$ (state-only), $\boldsymbol{O}^a$ (action-only), and $\boldsymbol{O}$ (state-action).

		Accuracy				Precision				Recall				F-Score			
O	Domain	MU	KL	DP	RG	MU	KL	DP	RG	MU	KL	DP	RG	MU	KL	DP	RG
0.5	Blocks	**0.95**	0.62	0.93	0.84	**0.95**	0.33	0.77	0.56	0.90	0.50	**1.00**	**1.00**	**0.90**	0.40	0.87	0.71
	Hanoi	**0.97**	0.90	0.93	0.68	**0.91**	0.80	0.77	0.38	**1.00**	0.80	**1.00**	**1.00**	**0.95**	0.80	0.87	0.56
	SkGrid	0.75	0.75	0.57	**0.88**	0.50	0.50	0.35	**0.64**	0.50	0.50	0.80	**0.90**	0.50	0.50	0.48	**0.75**
1.0	Blocks	**1.00**	**1.00**	0.95	0.96	**1.00**	**1.00**	0.83	0.83	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.91	0.91
	Hanoi	**1.00**	**0.95**	0.90	0.78	**1.00**	0.90	0.71	0.48	**1.00**	0.90	**1.00**	**1.00**	**1.00**	0.90	0.83	0.65
	SkGrid	0.85	**0.95**	0.65	0.90	0.70	**0.90**	0.40	0.69	0.70	**0.90**	0.80	**0.90**	0.70	**0.90**	0.53	0.78
Avg	Blocks	**0.97**	0.81	0.94	0.90	**0.97**	0.60	0.80	0.70	0.95	0.75	**1.00**	**1.00**	**0.95**	0.67	0.89	0.81
	Hanoi	**0.99**	0.93	0.91	0.73	**0.95**	0.85	0.74	0.43	**1.00**	0.85	**1.00**	**1.00**	**0.98**	0.85	0.85	0.61
	SkGrid	0.80	0.85	0.61	**0.89**	0.60	**0.70**	0.37	0.67	0.60	0.70	0.80	**0.90**	0.60	0.70	0.51	**0.77**

Table 2: Impact of noise: comparing MU, KL, DP, RG with varying observability and with 4 noisy observations in $\boldsymbol{O}$.

Related Work

A large body of work involves learning for *planning* domains (**??**). While some approaches learn action models from data, they do not link these action models to policies for reaching specific goals (**????**).

For example, Zeng et al. (**?**) use inverse reinforcement learning (IRL) to learn the rewards of the actor and then use an MDP-based GR. However, for GR, the motivation (rewards) that lead the actor to choose one action over another is redundant. By directly using RL, we skip this stage and learn utility functions or policies based on past actor experiences towards achieving specific goals. Amado et al. (**?**) learn domain theories for GR using autoencoders. However, they require observation of all possible transitions of a domain in order to infer its encoding, whereas we need only a small sample of transitions to learn a utility function informative enough to carry out GR effectively.

Unlike approaches that learn models for planning, we do not reason about the plan of the acting agent, but rather about the plan of another agent. In this case, we cannot control the actor's choices, and we might not know or care how the actor represents the environment and the task. Nevertheless, we need to be able to find a good-enough explanation for its actions to be able to assist it (as in the kitchen example from Figure 1). This setup is not the one used in existing work on learning other agent's behavior, e.g., the LOPE system (**?**), (**?**), and IRALe (**?**), as these systems choose the execution se it learns from. We can, however, use observed actions of other agents to improve our learning process. Gil (**?**) does so by investigating cases where executing new experiments can refine operators.

Other metric-based GR use distance metrics between an optimal plan and the observation se which can somewhat alleviate the need in online planner executions (**??**). This work differs from this problem statement, as it relies on the distance between a Q-function and an observation sequence rather than an optimal plan and an observation sequence.

Discussion and Conclusion

Recognition (GR) as model-free reinforcement learning, which obviates the need for an explicit model of the environment and candidate goals in the GR process. Our framework uses learned Q-values implicitly representing the agents under observation in lieu of explicit goals from traditional GR. This approach allows us to solve GR problems by minimizing the distance between an observation sequence and Q-values representing goal hypotheses or policies extracted from them. The GRAQL instantiation includes several possible distance measures we can derive from the Q-tables, based on KL-divergence, MaxUtil, and Convergence point. Our distance measures are competitive with the reference approach from the literature (**?**) in all experimental environments, and some distance measures outperform the reference approach in most domains, especially when the observation sequence is noisy or partial.

Besides recognition performance, GR needs to be computationally efficient so that an observer can quickly make decisions in response to the recognized goal in real-time. In this respect, our approaches differ substantially from recent planning-based GR, as it shifts almost all computation load to a pre-processing stage, instead of costly online planner runs. While computing the policies for each candidate goal is even more costly than running a planner for each goal, this computation can be done once prior to the recognition stage, and then the computation of processing observations is trivial and proportional to the number of observations. Finally, learning the policies saves the time of a domain expert who needs to carefully design the planning dynamics—a cost which is even harder to quantify.

In closing, our work paves the way for a new class of GR approaches based on model-free reinforcement learning. Future work will focus on new, more robust distance measures and mechanisms to handle noise explicitly, as well as experimenting with models learned using function approximation (e.g., neural networks). While our work is theoretically compatible with non-tabular representations of the value functions, we chose to focus our experiments on domains that are translatable to PDDL so our approach can be compared to planning-based GR. GRAQL does not depend on PDDL, as we can apply the learning stage on any domain where we can compute a policy, and then infer the correct goal using the set of observations and learned policies.

Ea deserunt perferendis modi, mollitia numquam deserunt incidunt ut doloremque voluptates ab amet aliquam atque natus, magni eum expedita eius cupiditate eaque?Hic nostrum illum quidem laboriosam dolores voluptatum odit molestiae ullam, inventore labore ducimus odio, officiis eligendi omnis odio, reiciendis cumque eaque atque itaque reprehenderit cupiditate autem, similique nesciunt suscipit vitae?Quisquam nobis tempora hic in quam nemo error, quibusdam nemo consequuntur placeat, quam ullam officiis obcaecati sit hic fuga magni quo sunt, nihil facere nemo perferendis inventore odio neque eum vitae adipisci incidunt.