Extended Goal Recognition Design with First-Order Computation Tree Logic

Tsz-Chiu Au

Department of Computer Science and Engineering
Ulsan National Institute of Science and Technology
Ulsan, Republic of Korea
chiu@unist.ac.kr

Abstract

Goal recognition design (GRD) is the task of modifying environments for aiding observers to recognize the objectives of agents during online observations. The worst case distinctiveness (WCD), a widely used performance measure in GRD research, can fail to provide useful guidance to the redesign process when some goals are too hard to be distinguished. Moreover, the existing WCD-based approaches do not work when an agent aims for a sequence of goals instead of just one goal. The paper presents a new GRD framework called *extended goal recognition design* (EGRD) for goal recognition that involves multiple goals. The objective of EGRD is to modify an environment to minimize the worst case distinctiveness of a *goal condition* that describes how an agent can reach a set of goals. A goal condition can be formally expressed in first-order computation tree logic (FO-CTL) that can be evaluated by model checking. We introduce a novel graphical representation of FO-CTL sentences that is suitable for extended goal recognition. Moreover, we present a search algorithm for EGRD with a novel caching mechanism. Our experimental results show that the caching mechanism can greatly speed up our EGRD search algorithm by reusing the previous evaluation of FO-CTL sentences.

Introduction

In goal recognition, an observer infers the goal of an agent, who is acting in an environment, from a sequence of observations (Kautz 1987; Carberry 2001; Ramirez and Geffner 2010; Sukthankar et al. 2014; Vered and Kaminka 2017; Pereira, Oren, and Meneguzzi 2017). Goal recognition design (GRD) is the task of designing an environment such that it becomes easier for an observer to recognize an agent's goal (Keren, Gal, and Karpas 2020). In recent years, there is a new line of research on GRD based on minimizing the *worst case distinctiveness* (WCD), put forward by Keren, Gal, and Karpas (2014). WCD is the highest number of observations that an observer needs to observe in order to be certain of the agent's goal in the worst case. In some application domains such as airport security, it is helpful to recognize inappropriate goals pursued by an agent as early as possible. This early detection can be achieved by redesigning an environment such that WCD is minimized.

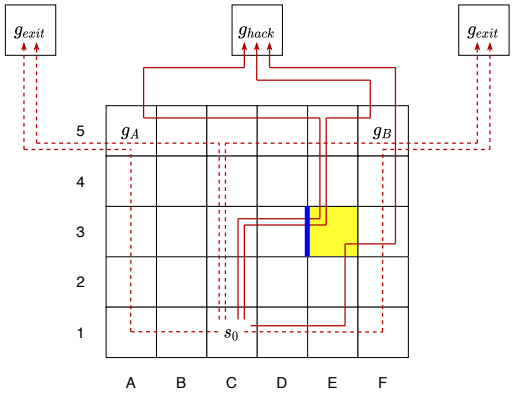

Figure 1: An example of EGRD that involves multiple goals.

However, there are situations in which WCD cannot provide helpful guidance for environmental design. Suppose there are two long paths that share a long prefix but lead to two different goals. It is difficult to redesign the environment to reduce WCD even if other goals can be recognized easily. Perhaps we should modify WCD to take the relative importance of goals into account. However, we pursue a different approach: instead of asking exactly which goal an agent aims for, an observer asks whether the agent aims for a *goal condition*, which describes conditions such as whether the agent aims for one of two goals but not any other goals. It is not always necessary for the observer to recognize a goal exactly. For example, if security guards have enough resources to protect two locations simultaneously, it is sufficient to know two possible locations an intruder plans to visit. More generally, when WCD is too restrictive for GRD, recognizing weaker goal conditions can provide more opportunities for the redesign process.

We propose using *first-order computation tree logic* (FO-CTL) to express goal conditions (Bohn et al. 1998). Computation tree logic is a widely used language for the formal verification of properties by model checking (Clarke and Emerson 1981). In this paper, we show that FO-CTL can be used to express a wide variety of goal conditions that are suitable for goal recognition. Specifically, FO-CTL is expressive enough for recognizing *extended goals* that potentially involve multiple goals (Pistore and Traverso 2001; Baier and

McIlraith 2006; Camacho et al. 2017). To illustrate when *extended goal recognition design* (EGRD) is necessary, consider the example in Figure 1, which extends the airport security example in (Keren, Gal, and Karpas 2020). In this example, an agent has to aim for two goals: the first goal is either g_A or g_B, which refer to the checkpoints at A5 or F5 respectively. The second goal is either g_{exit} (exiting the area normally) or g_{hack} (entering the computer room using the key the agent steals at E3). The agent must choose to follow one of the *legal paths*[1] (the red paths in Figure 1) starting at the initial state s_0. The observer does not know the chosen path ahead of time, let alone the goals on the chosen path. If the agent aims for g_{hack}, it must choose one of the solid red lines. The observer can recognize g_{hack} as one of the agent's goals only after the agent visits D3 or E2. After recognizing g_{hack}, the observer has to deploy one security guard to one of the checkpoints at A5 or F5 to intercept the agent. However, if the agent visits D3 but not E2, the observer cannot recognize the agent's first goal (g_A or g_B) and cannot decide which checkpoints it should deploy the security guard. If we put a barrier (the blue line in Figure 1) between D3 and E3, we can force the agent to reveal its goal g_{hack} as well as g_B at E2. If there is no barrier and the agent reaches D3, there is a chance the observer misplaces the security guard and the agent can reach the computer room. This example illustrates a goal condition that involves two goals, and the second goal (g_{hack}) should not be recognized before the first goal (g_A or g_B). In this paper, we will show how to use FO-CTL to express this type of goal conditions, such that we can redesign the environment (e.g., putting a barrier at the right place) to satisfy the goal condition while minimizing a cost function (e.g, time to reveal the goals).

This paper is organized as follows. We first define the syntax and the semantics of FO-CTL and explain what are goal conditions. Then we describe the structure of goal query graphs and how to compute WCD by model checking. After that, we define the task of EGRD and present a technique to speed up the search algorithm for EGRD. Finally, we present the experimental results and conclude this paper.

First-Order Computation Tree Logic

The syntax of FO-CTL is just like the syntax of first order logic with the addition of *path quantifiers* (**A** and **E**), *temporal operators* (**F**, **G**, **X**, and **U**), and *path formulas*. We assume no function symbol. The syntax of FO-CTL formulas is specified by the following context-free grammar.

$$
\begin{array}{rcl}
\phi & ::= & \bot \mid \top \mid Atom \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \\
 & & \phi \Rightarrow \phi \mid \phi \Leftrightarrow \phi \mid \forall x\, \phi \mid \exists x\, \phi \mid \mathbf{A}\, \psi \mid \mathbf{E}\, \psi \\
\psi & ::= & \mathbf{F}\, \phi \mid \mathbf{G}\, \phi \mid \mathbf{X}\, \phi \mid [\phi\, \mathbf{U}\, \phi] \\
Atom & ::= & Predicate(Term_1, \ldots, Term_n) \mid \\
 & & Term_1 = Term_2 \\
Term & ::= & Constant \mid Variable
\end{array}
$$

[1] A typical assumption in GRD research is that an agent cannot move freely but follow one path in a given set of legal paths, which can be either the shortest paths, some feasible paths subject to physical constraints, or some paths in a path library.

where $Atom$ is an atom, $Predicate$ is a predicate symbol, and $Term$ is a term which can be either a constant or a variable. Moreover, ϕ is a *state formula*, and ψ is a *path formula*. A FO-CTL sentence is a state formula but not a path formula. A variable is *free* if it is not quantified by universal or existential quantifiers. A *substitution* is a binding list $\{x_1/C_1, \ldots, x_n/C_n\}$, where x_i and C_i are a variable and a constant, respectively, for $1 \leq i \leq n$. We denote the sentence after applying the substitution by $\textsc{Subst}(\theta, \phi)$. Let Θ be the set of all possible substitutions.

Semantics of FO-CTL FO-CTL sentences are interpreted over transition systems: $M = (S, E, L, s_0)$, where 1) S is a finite set of states; 2) $E \subseteq S \times S$ is a transition relation (i.e., $(s_1, s_2) \in E$ iff s_2 is a next state of s_1); 3) $L : S \to 2^{\mathcal{A}}$ maps a state s to the set of ground atoms that are true at state s, where $\mathcal{A}$ is the set of all ground atoms; and 4) $s_0 \in S$ is the initial state. A path p starting at state s_0 is a sequence of states $\langle s_0, s_1, \ldots \rangle$ such that $s_{i+1} \in T(s_i)$ for $i \geq 0$. Let $\mathbb{P}(s)$ be the set of all paths in M starting at s. Given a transition system $M = (S, E, L, s_0)$, a state $s \in S$, and a sentence ϕ, we say (M, s) *entails* ϕ (i.e., $(M, s) \models \phi$) iff ϕ is true at s in M. The entailment is defined recursively as follows:

1. $(M, s) \models \top$
2. $(M, s) \not\models \bot$
3. $(M, s) \models \rho$ iff $\rho \in L(s)$ where ρ is a predicate
4. $(M, s) \models \neg\phi$ iff $(M, s) \not\models \phi$
5. $(M, s) \models \phi_1\, op\, \phi_2$ iff $[(M, s) \models \phi_1]\, op\, [(M, s) \models \phi_2]$, for $op \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$
6. $(M, s) \models \forall x\, \phi$ iff $\forall(\theta \in \Theta)\, [(M, s) \models \textsc{Subst}(\theta, \phi)]$
7. $(M, s) \models \exists x\, \phi$ iff $\exists(\theta \in \Theta)\, [(M, s) \models \textsc{Subst}(\theta, \phi)]$
8. $(M, s) \models \mathbf{A}\, \psi$ iff $\forall(p \in \mathbb{P}(s))\, [(M, p) \models \psi]$
9. $(M, s) \models \mathbf{E}\, \psi$ iff $\exists(p \in \mathbb{P}(s))\, [(M, p) \models \psi]$

The entailment of a path formula ψ ($(M, p) \models \psi$ where $p = \langle s_0, s_1, \ldots \rangle$) in Statements 8 and 9 is defined as follows:

11. $(M, p) \models \mathbf{F}\, \phi$ iff $\exists(s' \in p)\, (M, s') \models \phi$
12. $(M, p) \models \mathbf{G}\, \phi$ iff $\forall(s' \in p)\, (M, s') \models \phi$
13. $(M, p) \models \mathbf{X}\, \phi$ iff $(|p| \geq 2) \wedge [(M, s_1) \models \phi]$
14. $(M, p) \models [\phi_1\, \mathbf{U}\, \phi_2]$ iff $\exists(s_i \in p)\, \{[(M, s_i) \models \phi_2] \wedge \forall(j \in [0, i))\, (M, s_j) \models \phi_1)\}$

Syntactic Sugar In goal recognition tasks, the set $\mathbb{C}$ of constants is the finite set of goals $\mathbb{G} = \{g_1, g_2, \ldots, g_m\}$ known to the observer, and there is exactly one predicate symbol Goal in $\mathbb{P}$ with one argument. The predicate $\mathsf{Goal}(g_i)$ is true if g_i is a goal at a given state. Hence, we omit the predicate symbol Goal when writing a FO-CTL sentence. When we see constant symbols or variable symbols in a sentence at which a predicate is expected, we can assume they are enclosed by the predicate symbol Goal. For example, the sentence "$\exists x\, \{x \wedge \forall x'\, [(x' \neq x) \Rightarrow \neg\mathbf{EF}\, x']\}$" should be translated into "$\exists x\, \{\mathsf{Goal}(x) \wedge \forall x'\, [\neg(x' = x) \Rightarrow \neg\mathbf{EF}\, \mathsf{Goal}(x')]\}$". Since there is only one type of ground atoms $\mathsf{Goal}(g)$ in L, we can replace L in $M = (S, E, L, s_0)$ with $\mathcal{G} : S \to 2^{\mathbb{G}}$, which maps a state s to a set $\mathcal{G}(s)$ of goals such that

$\mathsf{Goal}(g) \in L(s)$ for all $g \in \mathcal{G}(s)$. Hence, we shall define M as $(S, E, \mathcal{G}, s_0)$ for the rest of this paper. We shall use the symbol x to denote variables in a FO-CTL sentence.

Goal Conditions

Most existing works in goal recognition design assume an agent aims for one goal only. In this paper, we consider the scenarios in which an agent can aim for several goals. Let p^* be the legal path chosen by an agent. We consider all goals in all states in p^* are the goals aimed by the agent. We ignore the cases in which an agent visits a state with a goal g, but it does not intend to achieve g. We ignore unintentional goal achievements since an observer can only observe agents' behavior and cannot read agents' minds. From the observer's perspective, an agent does not accidentally achieve a goal.

A *goal condition* is a boolean query about temporal and logical relationships among goals that an agent tries to achieve. Given a transition system that models an environment, a goal condition describes a certain property of a *substructure* in the transition system. In FO-CTL, a substructure of a given state s is a subgraph formed by the set of paths starting from s. In extended goal recognition, we are interested in checking whether a substructure that satisfies a goal condition exists in a given transition system. In extended goal recognition design, we are interested in modifying a transition system such that the location of the substructure that satisfies a goal condition can be optimized (e.g., make it closer the the initial state). We choose FO-CTL to encode goal conditions due to the expressiveness of FO-CTL and the availability of efficient model checking algorithms. For example, consider the following goal condition:

$$\phi_{\mathsf{unique}} = \exists x \, \{\mathbf{AF}\,(x \wedge \forall x' \; [(x' \neq x) \Rightarrow \mathbf{AG}\,\neg x'])\} \quad (1)$$

ϕ_{unique} is a FO-CTL sentence that checks whether a goal g exists such that the agent must eventually achieve g while the agent will not achieve any other goal. Hence, if ϕ_{unique} is true at state s, the agent will certainly achieve one goal only, and the observer can successfully recognize the goal at s even if the agent has not achieved the goal at s yet.

To compute the WCD of a goal condition ϕ, we find a set $S_\phi(p)$ of states on a legal path p such that ϕ is true in these states. Then the WCD can be computed by

$$WCD = \left\{ \max_{p \in P^{leg}} \min_{s_i \in S_\phi(p)} [dist(s_0, s_i)] \right\} - 1, \quad (2)$$

where P^{leg} is the set of all legal paths and $dist(s_0, s_i)$ is the distance between s_i and the initial state s_0. Note that the -1 term is needed because the WCD does not include the state at which a goal has already been recognized.

Goal Query Graph

FO-CTL sentences can become quite long and complicated when multiple goals are involved. In this section, we propose to express a goal condition in a graphical form that is suitable for goal recognition. We also provide an algorithm to translate a graph into a FO-CTL sentence.

A *goal query graph* is a directed acyclic graph that describes a substructure in a transition system that models an environment. There are three types of vertices in a goal query graph: state vertices, nil vertices, and choice vertices. A *state vertex* v matches a state s in an environment according to a *state condition* $\mathsf{cond}(v)$ associated with v. A state condition is a first-order logic statement without the modal operators in FO-CTL and can have free variables. If $\mathsf{cond}(v)$ is true at state s, we say v matches s (or s matches v). A *nil vertex* is a placeholder for connecting different edges and is not used to match any state. A nil vertex cannot be a terminal vertex and must be followed by an edge. A *choice vertex* corresponds to the beginning of alternative paths in a goal query graph. It must be followed by two or more *choice edges*.

There are five types of edges: **AP** edges, **EP** edges, **AX** edges, **EX** edges, and choice edges. An **AP** edge (v_1, v_2) describes the following substructure if both v_1 and v_2 are state vertices: after an agent reaches a state s_1 that satisfies the state condition of v_1, it will eventually reach another state that matches v_2 *in all future paths* after s_1. In other words, after matching v_1 at s_1 the agent will certainly reach another state that matches v_2. Each **AP** edge (v_1, v_2) has an *edge condition* $\mathsf{cond}(v_1, v_2)$, a first-order logic statement without the modal operators in FO-CTL and can have free variables. All intermediate states between s_1 and the states that matches v_2 must satisfy $\mathsf{cond}(v_1, v_2)$. If we omit the edge condition when we draw an **AP** edge in a goal query graph, the edge condition is assumed to be True. We say an **AP** edge *matches* a state s_1 if there are states that matches v_2 on all paths after matching s and $\mathsf{cond}(v_1, v_2)$ is satisfied on all states between s and the states that matches v_2. Likewise, an **EP** edge (v_1, v_2) matches a state s_1 if an agent will *possibly* (but not necessarily) reach another state that matches v_2 after matching v_1 at s_1. More precisely, there is at least one future path after s_1 that matches v_1 such that the agent will reach another state s_2 that matches v_2, and all intermediate states between s_1 and s_2 satisfy $\mathsf{cond}(\epsilon)$. In other words, there is a chance of reaching s_2 after s_1. Note that the length of the path between s_1 and s_2 is at least 1.

AX edges and **EX** edges are similar to **AP** edges and **EP** edges except that the length of the path between the states that match v_1 and v_2 is exactly 1. **AX** edges and **EX** edges do not have edge conditions because there is no intermediate state between the states that match v_1 and v_2. A choice edge denotes an alternative path after a choice vertex. It does not match any substructure and has no edge condition.

The state conditions and edge conditions can contain free variables that match goals in states. Moreover, a free variable can appear multiple times in a goal query graph, matching the same goal. There are two ways to match free variables with goals in a substructure that matches a goal query graph. A free variable x *strongly* matches a goal g when x always matches g in *all* paths in the substructure. On the other hand, x *weakly* matches some goals when x matches a goal, but the goal can be different on different paths that lead to the vertices in which x is matched. For example, the goal query graph in Figure 2 has two free variables x_1 and x_2. If we want both x_1 and x_2 to strongly match two goals, the graph should be translated into this FO-CTL sentence: $\exists x_1 \exists x_2 [\mathbf{AF}\, x_1 \wedge \mathbf{AX}\,\mathbf{AF}\, x_2]$, which means that there exist two goals that an agent will certainly reach one after another.

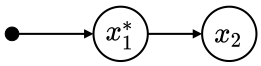

Figure 2: A goal query graph that recognizes two goals sequentially. The black dot is a nil vertex, the circles are state vertices, and the solid arrows are **AP** edges. The asterisk on x_1 denotes that x_1 will weakly match a goal.

Figure 3: A goal query graph for recognizing a goal but excluding other goals.

After matching the graph to a substructure, we can tell exactly which two goals are matched. If x_1 weakly matches a goal but x_2 strongly matches a goal, the FO-CTL statement should be $\exists x_2[\mathbf{AF}\,\exists x_1[x_1] \wedge \mathbf{AX}\,\mathbf{AF}\,x_2]$, which means that an agent will first reach a goal before reaching a second goal. However, we cannot tell exactly which goal is the first goal because different paths to the states that match x_1 can cause x_1 to match different goals. At the time a goal query graph is matched, all we know is that the agent will eventually reach two goals but we cannot determine exactly which goal is the first goal. In some applications, weakly-matched variables are exactly what we want. For example, the observer in Figure 1 does not need to know whether the first goal is g_A or g_B as long as it recognizes g_{hack} as the second goal. Knowing the existence of the first goal can play an important role in protecting the airport in an extended scenario in which the first goal is optional (e.g., there is a new path from s_0 to g_{hack} without going through A5 or F5) and the deployment of security guards depends on the knowledge of the first goal's existence (sending security guards to locations other than A5 and F5).

The goal query graph in Figure 2 ignores other goals in the matched substructure. If we prefer to recognize a goal g such that the agent will reach a state with g only and will not reach other goals thereafter, we can add additional requirements to the state conditions and the edge conditions. Let $\mathsf{XA} = \forall x[\neg x]$ be a predicate called "Exclude All". Let $\mathsf{XA}_{y_1,\ldots,y_m} = \forall x[(x \neq y_1) \wedge \ldots \wedge (x \neq y_m) \Rightarrow \neg x]$ be a predicate called "Exclude All But $y_1, y_2, \ldots, y_m$", where y_i is either a variable or a goal, for $1 \leq i \leq m$. Let End be a special predicate that matches the end of a legal path (i.e., the "state" after the last state of a legal path) only. The goal query graph in Figure 3 can be translated into $\exists x\{\mathbf{AF}\,(x \wedge (\forall x_1[(x_1 \neq x) \Rightarrow \neg x_1]) \wedge (\mathbf{AX}\,\mathbf{A}\,[\forall x_2\,[\neg x_2]\,\mathbf{U}\,\mathsf{End})]\}$, which is mathematically equivalent to ϕ_{unique} in Sentence 1.

The goal query graphs in Figures 2 and 3 match states in a sequential manner. Choice vertices and choice edges offer a way to match states in a nonlinear fashion. Figure 4 shows a goal query graph that offers a choice to match either g_1 or g_2 after matching x. An agent must eventually reach either g_1 or g_2 after matching a goal. However, the observer cannot know ahead of time which one it is—it only knows either g_1 or g_2 (or both since there is no XA predicate) will be reached. If the agent's second goal is g_1, the agent will immediately

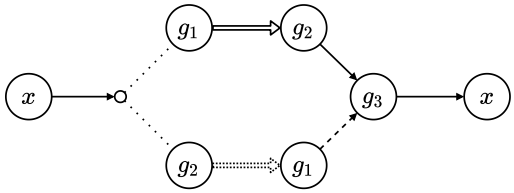

Figure 4: A goal query graph with all five types of edges. The empty circle is a choice vertex and the dotted lines are choice edges. The solid unfilled arrow from g_1 to g_2, the dashed unfilled arrow from g_2 to g_1, and the dashed arrow from g_1 to g_3 are **AX**, **EX**, and **EP** edges, respectively.

reach g_2 after g_1, and then reach g_3 and x; otherwise, the agent will possibly reach g_1 immediately after g_2 and then may or may not reach g_3 and x due to the **EP** edges. Note that the first x and the last x match the same goal. This goal query graph can be translated into this FO-CTL sentence: $\exists x[x \wedge \mathbf{AX}\,\mathbf{AF}\,[(g_1 \wedge \mathbf{AX}\,(g_2 \wedge \mathbf{AX}\,\mathbf{AF}\,(g_3 \wedge \mathbf{AX}\,\mathbf{AF}\,x))) \vee (g_2 \wedge \mathbf{EX}\,(g_1 \wedge \mathbf{EX}\,\mathbf{EF}\,(g_3 \wedge \mathbf{AX}\,\mathbf{AF}\,x)))]]$. The supplementary material contains additional examples showing the usage of choice vertices and choice edges.

Translation to FO-CTL sentences

We devised an algorithm to translate goal query graphs into FO-CTL sentences that can be used to identify substructures in a transition system by model checking. The algorithm proceeds by a depth-first search starting from the start state vertex v_0 of a graph. Let $\mathsf{tr}(v_1)$ be the FO-CTL sentence of a subgraph starting at the vertex v_1. Let $\mathsf{tr}(v_1, v_2)$ be the FO-CTL sentence of a subgraph that includes the edge (v_1, v_2) and the subgraph starting at v_2. If v_1 is a terminal vertex, $\mathsf{tr}(v_1) = \mathsf{cond}(v_1)$. If v_1 is not a terminal vertex and v_1 is a state vertex, $\mathsf{tr}(v_1) = \mathsf{cond}(v_1) \wedge \mathsf{tr}(v_1, v_2)$ where (v_1, v_2) is the edge that follows v_1. If v_1 is not a terminal vertex and v_1 is a nil vertex, $\mathsf{tr}(v_1) = \mathsf{tr}(v_1, v_2)$. If v_1 is a choice vertex and the choice edges incident on v_1 are (v_1, v_2), (v_1, v_3), $\ldots (v_1, v_m)$, then $\mathsf{tr}(v_1) = \mathsf{tr}(v_2) \vee \mathsf{tr}(v_3) \vee \ldots \vee \mathsf{tr}(v_m)$.

Given an **AP** edge (v_1, v_2) where v_1 is a state vertex, if $\mathsf{cond}(v_1, v_2)$ is True, $\mathsf{tr}(v_1, v_2) = \mathbf{AX}\,\mathbf{AF}\,\mathsf{tr}(v_2)$; otherwise, $\mathsf{tr}(v_1, v_2) = \mathbf{AX}\,\mathbf{A}\,[\mathsf{cond}(v_1, v_2)\,\mathbf{U}\,\mathsf{tr}(v_2)]$. The modal operator **AX** makes sure that both $\mathsf{cond}(v_1, v_2)$ and $\mathsf{cond}(v_2)$ are not used to match the state that matches v_1. If v_1 is a nil state, **AX** is dropped: $\mathsf{tr}(v_1, v_2) = \mathbf{AF}\,\mathsf{tr}(v_2)$ if $\mathsf{cond}(v_1, v_2)$ is True and $\mathsf{tr}(v_1, v_2) = \mathbf{A}\,[\mathsf{cond}(v_1, v_2)\,\mathbf{U}\,\mathsf{tr}(v_2)]$ otherwise. The translation of **EP** edges is the same except the modal operators **AF**, **AX**, and **AU** are replaced with **EF**, **EX**, and **EU**, respectively. The translation of **AX** edges and **EX** edges are $\mathbf{AX}\,\mathsf{tr}(v_2)$ and $\mathbf{EX}\,\mathsf{tr}(v_2)$, respectively.

Lastly, the algorithm identifies all free variables in the FO-CTL sentence ϕ and inserts existential quantifiers in ϕ. If x is a strongly-matched variable, the algorithm put $\exists x$ in the front of ϕ. Otherwise, x is a weakly-matched variable, and the algorithm first computes the common prefix of all paths to all occurrences of x in ϕ, and then insert $\exists x$ before the last node in the common prefix. The pseudo-code of the translation algorithm can be found in the supplementary material. The time complexity of the algorithm is $O(|V|+|E|)$, where

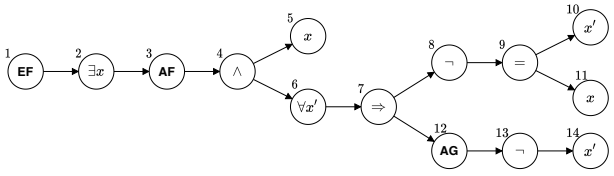

Figure 5: The tree structure of $\mathbf{EF}\,\phi_{\mathsf{unique}}$.

V and E are the set of nodes and the set of edges in the goal query graph, respectively.

Finding WCD by Model Checking

Suppose we want to calculate the WCD of a goal condition ϕ according to Equation 2. Instead of using a specialized algorithm (e.g., the latest-split method in Keren, Gal, and Karpas (2014)), we can integrate the calculation into model checking by evaluating an extended version of the FO-CTL sentence at the initial state:

$$\mathbf{EF}\,\phi \tag{3}$$

Typically a model checking algorithm evaluates a FO-CTL sentence recursively according to the tree-structure of the sentence as shown in Figure 5. The truth values of subsentences are propagated to the root node by recursive function calls. We can modify the model checking algorithm by attaching a *cost* to the truth value returned by each recursive function call. Moreover, we attach an *evaluation function* to each node in a sentence. An evaluation function takes the truth values and the costs returned from the children nodes and computes a new cost. In Sentence 3, the cost returned by the root node of ϕ is always the depth of the state s_i that matches ϕ (i.e., $dist(s_0, s_i)$ in Equation 2). The cost is minimum on any legal path to s_i because the **EF** node always matches the first state that matches ϕ on a legal path. Then the evaluation function of the **EF** node collects the minimum costs from all legal paths that contain a state that matches ϕ, and returns the maximum cost among them. The maximum cost minus 1 is the WCD as defined in Equation 2.

This method can be generalized to compute performance measures other than the WCD. Indeed, when there are multiple goals, we may be interested in optimizing other performance measures such as the number of goals that can be recognized. However, we do not get into this discussion here.

Extended Goal Recognition Design

In this section, we define the design models for EGRD and planning with goal sequences.

Modifications and Design Models

In this paper, we only allow modifications to the transition relation E in a transition system $M = (S, E, \mathcal{G}, s_0)$. We define a *modification* m as a pair $(\mathsf{Add}(m), \mathsf{Del}(m))$, where both $\mathsf{Add}(m)$ and $\mathsf{Del}(m)$ are sets of edges such that $\mathsf{Add}(m) \cap \mathsf{Del}(m) = \emptyset$. m is *applicable* in $M = (S, E, \mathcal{G}, s_0)$ if and only if $E \cap \mathsf{Add}(m) = \emptyset$ and $(\mathsf{Del}(m) \setminus E) = \emptyset$. After m is applied to M, the new set of edges is $E' = (E \setminus \mathsf{Del}(m)) \cup \mathsf{Add}(m)$, and the new transition system is $M' = (S, E', \mathcal{G}, s_0)$.

According to Keren, Gal, and Karpas (2019), a *design model* is a triple $\langle \mathcal{W}, \delta, \eta \rangle$, where $\mathcal{W}$ is a set of atomic modifications, δ is transition function between goal recognition models (which are transition systems in our context), and η specifies whether a modification sequence can be applied to a goal recognition model. However, according to our definition of modifications, δ can be deduced from $\mathcal{W}$ (i.e., $\delta(M, m) = M'$ where M' is the new transition system after applying M' at M). We can also define η as follows: if a modification sequence $\vec{m} = \{m_0, m_1, \ldots, m_k\}$ is applicable in M (i.e., m_{i+1} is applicable in $M_{i+1} = \delta(M_i, m_i)$ for $0 \leq i \leq k-1$), $\eta(\vec{m}, M) = 1$; otherwise, $\eta(\vec{m}, M) = 0$. For the rest of the paper, we shall denote a design model as $\mathcal{W}$ without δ and η.

Legal Paths and Legal Transition Systems

Given an initial transition system $M_0 = (S, E_0, \mathcal{G}, s_0)$, let $\mathbb{M}$ be the set of all possible systems that are reachable from M_0 by the modifications in $\mathcal{W}$. In other words, $\mathbb{M}$ is the closure of $\{M_0\}$ under $\mathcal{W}$. Let $E_{\mathsf{max}} = \bigcup_{(S,E,\mathcal{G},s_0)\in\mathbb{M}} E$ be the set of all edges in all transition models in $\mathbb{M}$. Basically, the edges that are not in E_{max} will never be visited by an agent and they can be ignored. Let M_{max} be $(S, E_{\mathsf{max}}, \mathcal{G}, s_0)$.

We denote the set of all paths starting at s_0 in M by $P(M)$. We assume the agent will eventually choose to follow one path $p^* \in P^{leg}$, where $P^{leg} \subseteq P(M_{\mathsf{max}})$ is a finite set of *legal paths*, each of which is a finite path. Basically, P^{leg} is determined by the agent's controller. For example, P^{leg} can be the set of the shortest paths to the goals in $\mathbb{G}$. However, other controllers can use a different set of paths. While the observer does not know p^*, he does know P^{leg} by either being given a database of legal paths or getting hold of a controller that can generate legal paths on demand.

Mathematically, we should not directly evaluate ϕ in $M = (S, E, \mathcal{G}, s_0)$ since the agent cannot traverse any path not in P^{leg}. Instead, we should evaluate ϕ in a different transition system whose paths are in $P(M)$ as well as P^{leg}. We define the *legal transition system of M w.r.t. P^{leg}* as $\mathcal{T}(M) = (S', E', \mathcal{G}', s_0)$, which is formed by merging the longest common prefixes of the paths in $P^{leg}(M) = P^{leg} \cap P(M)$. $\mathcal{T}(M)$ is a tree rooted at s_0. Note that a state in $s \in S$ can appear in several paths in $P^{leg}(M)$, and the different instances of s in $\mathcal{T}(M)$ should be named differently in S'. Although ϕ should be evaluated in $\mathcal{T}(M)$ instead of M, an evaluation algorithm such as Algorithm 2 does not have to construct $\mathcal{T}(M)$ explicitly, as long as it keeps checking whether the current path is not in P^{leg} while evaluating ϕ in M.

Extended Goal Recognition Design

An *extended goal recognition model* R is a tuple $\langle M, P^{leg}, \phi, \mathsf{cost} \rangle$, where $M = (S, E, \mathcal{G}, s_0)$ is a transition system, P^{leg} is a set of legal paths, ϕ is a FO-CTL sentence, and cost is a cost function. An *extended goal recognition design model* is a pair $\langle R_0, \mathcal{W} \rangle$, where $R_0 = \langle M_0, P^{leg}, \phi, \mathsf{cost} \rangle$ is the initial goal recognition model and $\mathcal{W}$ is a design model. Given $\langle R_0, \mathcal{W} \rangle$, the task of *extended*

goal recognition design (EGRD) is to search for a transition system M^* in the closure $\mathbb{M}$ of $\{M_0\}$ under $\mathcal{W}$ such that 1) the legal transition system $T(M^*)$ of M^* w.r.t. P^{leg} satisfies $(\mathcal{T}(M^*), s_0) \models \phi$, and 2) $\mathcal{T}(M^*)$ minimizes the cost function cost w.r.t. ϕ. More precisely,

$$M^* = \underset{\forall M \in \mathbb{M} \text{ s.t. } (\mathcal{T}(M), s_0) \models \phi}{\arg\min} \mathsf{cost}(\phi, \mathcal{T}(\mathsf{M})), \qquad (4)$$

where $\mathsf{cost}(\phi, \mathcal{T}(\mathsf{M}))$ is the cost of ϕ in $\mathcal{T}(M)$.

Planning with Goal Sequences

A transition system $M = (S, E, \mathcal{G}, s_0)$ can be defined in terms of classical planning with goal sequences using the STRIPS formalism (Fikes and Nilsson 1971). In classical planning, a state $s \in S$ is defined as a set of *fluents*, each of them is a ground, functionless atom that is true in s. Note that a fluent is not an atom in FO-CTL sentences. A *planning domain* is a pair $\langle \mathcal{F}, A \rangle$, where $\mathcal{F}$ is the finite set of all fluents and A is a set of actions. A plan π is a sequence of actions $\langle a_0, a_1, \dots a_{k-1} \rangle$, where $a_i \in A$ for $0 \leq i < k$. Given an initial state $I \subseteq \mathcal{F}$, a plan π is *valid*[2] if a_{i+1} is applicable in $s_{i+1} = \mathsf{apply}(\mathsf{s_i}, \mathsf{a_i})$ for $0 \leq i < k$ and $s_0 = I$. The path of a valid π is the sequence of states $p(\pi) = \langle s_0, s_1, \dots, s_k \rangle$ visited by the agent if it executes π. Let Π^{leg} be a finite set of *legal plans* from which the agent can choose a plan and execute the plan. Note that unlike the definition of the legal plans in (Keren, Gal, and Karpas 2019), a legal plan in Π^{leg} does not have to associate with a unique goal; there could be legal plans that lead to no goal or multiple goals. Given Π^{leg}, we can construct the corresponding set of legal paths: $P^{leg} = \{p(\pi) : \forall \pi \in \Pi^{leg}\}$.

Many existing works consider action removal modifications only, which remove actions from a planning domain (Keren, Gal, and Karpas 2014; Son et al. 2016; Ang et al. 2017; Wayllace, Hou, and Yeoh 2017; Mirsky et al. 2019). Action conditioning modification adds preconditions to actions (Keren, Gal, and Karpas 2018). We adopt a similar definition of modifications, whereby atoms or literals can be added to or removed from the preconditions, the add lists, or the delete lists of actions. Such modifications will only update E in the corresponding transition system $M = (S, E, \mathcal{G}, s_0)$—some new edges can be added to E while some existing edges can be removed from E. Adding new edges to E will enlarge $P(M)$ such that there can be more legal paths in $P^{leg}(M) = P^{leg} \cap P(M)$ for the agent to choose, whereas removing edges from E can reduce the set of $P^{leg}(M)$. These effects of a modification m can be summarized by $\mathsf{Add}(m)$ and $\mathsf{Del}(m)$.

In partial observable environments, sensor refinements modify sensory input models (Keren, Gal, and Karpas 2016a,b; Wayllace et al. 2020; Shvo and McIlraith 2020). Since we assume the outcomes of actions are deterministic and the model is fully observable to both the agent and the observer, our design model do not allow sensor refinement.

[2]Unlike classical planning, a valid plan does not have to achieve any goal in $\mathbb{G}$ in our framework. A plan that does not reach any goal can still reach some hidden goals not in $\mathbb{G}$. However, the hidden goals are irrelevant to the goal recognition task and can be ignored.

Algorithm 1: The EGRD search algorithm.

procedure EGRD-Search($M_0 = (S, E_0, \mathcal{G}, s_0)$, ϕ, $\mathcal{W}$)
  Let F be the frontier and H be a set of visited sets of edges.
  Let $\mathbf{C}$ be a cache of the evaluation results of subsentences.
  node_1 := the first node of ϕ; $p^* := \infty$; $F := \{(E_0, \{\})\}$
  While F is not empty and the current time $<$ the time limit
    Use a heuristic function to select $(E, \vec{m})$ from F
    Remove $(E, \vec{m})$ from F; $M := (S, E, \mathcal{G}, s_0)$
    $t :=$ **EVAL**(node_1, s_0, $\{\}$) with ϕ, M, $\mathbf{C}$, P^{leg}
    Add E to H; Let p be the cost attached to t.
    If $t =$ True and $p < p^*$, **then** $E^* := E$; $\vec{m}_* := \vec{m}$
    For each $m \in \mathcal{W}$ that is applicable to E,
      Get E' by applying m to E
      If $E' \notin H$, **then** insert $(E', \vec{m} \oplus \{m\})$ into F.
  return $M^* = (S, E^*, \mathcal{G}, s_0)$ and $\vec{m}_*$

Algorithm 2: Evaluate a FO-CTL sentence ϕ in $M = (S, E, \mathcal{G}, s_0)$ with a cache $\mathbf{C}$.

procedure EVAL(node_i, s, θ)
  /* ϕ, M, $\mathbf{C}$, and P^{leg} are given by Algorithm 1 */
  $t := \mathbf{C}((\mathsf{node}_i, s, \theta), P^{leg}(M, s))$; **If** t exists, **Return** t
  Use a model checking algorithm $\mathbf{MC}$ to evaluate ϕ.
  /* $\mathbf{MC}$ will call $\mathbf{EVAL}()$ recursively */
  Let t be the truth value of the subsentence rooted at node_i
  Call the evaluation function of node_i to compute $\mathsf{cost(t)}$
  $\mathbf{C}((\mathsf{node}_i, s, \theta), P^{leg}(M, s)) := t$ with $\mathsf{cost(t)}$
  return t with $\mathsf{cost(t)}$.

The EGRD Search Algorithm with Caches

Given an EGRD model $\langle R_0, \mathcal{W} \rangle$ where $R_0 = \langle M_0, P^{leg}, \phi, \mathsf{cost} \rangle$ and $M_0 = (S, E_0, \mathcal{G}, s_0)$, the objective is to search for a modification sequence $\vec{m}_*$ that minimizes the cost function $\mathsf{cost}(\phi, \mathcal{T}(\mathsf{M}))$ and leads to M^*. Algorithm 1 is a best-first search for finding $\vec{m}_*$. A node in Algorithm 1 is a pair $(E, \vec{m})$, where E is a set of edges and $\vec{m}$ is a modification sequence that leads to E from E_0. The algorithm maintains a set of nodes in the frontier F, which initially contains $(E_0, \{\})$. The algorithm chooses a node in F according to a heuristic function and then expands the node. To prevent a cycle in the search process, the algorithm avoids inserting a node into F if the set of edges has appeared previously. The algorithm keeps track of the solution with the minimum cost and returns it at the end. Since all transition systems are finite, the number of possible sets of edges is also finite. Hence, Algorithm 1 will eventually terminate.

In Algorithm 1, the evaluation of FO-CTL sentences in a given transition system is conducted by $\mathbf{EVAL}()$, which returns the truth value of the sentence as well as the cost. Algorithm 2 is the pseudocode of $\mathbf{EVAL}()$. Algorithm 2 relies on an external model checking algorithm, which can be a naive algorithm that traverses the transition system by depth-first search while checking the sentence recursively at each state, using the definition of entailment directly. The input of the algorithm is a tuple $(\mathsf{node}_i, s, \theta)$ called a *search node*, where node_i is the current sentence node in ϕ, s is the current state in M, and θ is a substitute. At the beginning, the input of Algorithm 2 is the initial search node $(\mathsf{node}_1, s_0, \{\})$,

as shown in Line 8 in Algorithm 1. Other inputs are ϕ, M, $\mathbf{C}$, and P^{leg}; these inputs remain unchanged in subsequent recursive function calls of $\mathbf{EVAL}()$. Whenever the model checking algorithm needs to call itself recursively, the recursive function call should invoke $\mathbf{EVAL}()$ instead such that the cache and the evaluation functions can be used recursively as well. In Line 11–13, the frontier is expanded even if t is false in Line 10 because an environment may need two or more modifications to form a new environment that satisfies ϕ, but the intermediate environments do not satisfy ϕ. In Line 7, the evaluation function of the current search node is called to compute the cost $\mathsf{cost}(\mathtt{t})$ that will be returned along with the truth value t in Line 9.

The time complexity of Algorithm 1 is exponential to the length of the modification sequences. It is difficult to come up with a good evaluation function for node selection since 1) we allow modifications to add and delete edges simultaneously, and 2) the FO-CTL sentence and the cost function can be arbitrary. One way to speed up Algorithm 1 is to avoid running Algorithm 2 from scratch all the time. We propose a *caching mechanism* that stores the results of the evaluation of *subsentences* for reuse. This mechanism works because 1) the cost function is defined according to the tree structure of a FO-CTL sentence ϕ such that the cost of a subsentence of ϕ can be computed separately, and 2) the evaluation of a subsentence starting at node_i is always the same given the set same search node $(\mathsf{node}_i, s, \theta)$ in Algorithm 2 and the same set of legal paths $P^{leg}(M, s) = \{p \in P^{leg}(M) : p \text{ has the same prefix from } s_0 \text{ to } s\}$. Therefore, we can associate $(\mathsf{node}_i, s, \theta)$ and $P^{leg}(M, s)$ with the truth value t and $\mathsf{cost}(\mathtt{t})$. This association can be stored in a cache and can be reused for different transition systems in Algorithm 2. See Lines 3 and 8 in Algorithm 1 and Lines 3 and 8 in Algorithm 2 for the implementation of this caching mechanism. The cache is created in Lines 3 of Algorithm 1 and is utilized by Algorithm 2: if the truth value can be found in the cache, the model checking algorithm will not be invoked; otherwise, the truth value and the cost will be stored in the cache after invoking the model checking algorithm.

Empirical Evaluation

We conducted two experiments to evaluate the EGRD search algorithm. In Experiment 1, we checked how well the EGRD search algorithm scales with the number of goals being recognized. We extended the goal query graph in Figure 2 with an increasing number of goals in sequential order, and half of the goals are weakly-matched goals. In Experiment 2, we evaluated the improvement of the execution time due to the caching mechanism, using the goal query graph in Figure 3.

Experimental Setup We adopted four domains in the International Planning Competition: BLOCK-WORLD LOGISTICS, GRID, and DEPOTS. To generate EGRD problem instances that involve n goals, we implemented a problem generator for each domain. First, the problem generator randomly generates a set of goals that are ordered in a binary tree. For each path from the root to a terminal node in the binary tree, we used a planner to find plans for the goals on the path one by one without resetting the initial state after finding a plan for one goal. The planner we used was Fast Downward (Helmert 2006). Then we collected the first set of plans that can reach different goals in the binary tree. We randomly selected some actions that were crucial to the plans to reach different branches in the tree, and assigned them to the set $\mathcal{W}$ of action removal modifications. For each combination $\mathcal{W}'$ of the modifications in $\mathcal{W}$, we reran the planner for goals on the binary tree but forbade the planner to use the actions in $\mathcal{W}'$. This will force the planner to generate slightly different plans. Starting with the first set of plans, we ran the EGRD algorithm to optimize the WCD of the goal query graph using the modifications in $\mathcal{W}$. In both experiments, the number of EGRD problem instances in each domain was 50, and Algorithm 1 did not use any heuristic function.

Table 1: Execution times (in sec.) vs. the number of goals.

	1 Goal	2 Goals	3 Goals	4 Goals
LOGISTICS	1.67	7.85	10.86	14.99
DEPOTS	0.54	2.08	2.41	3.02
GRID	0.44	4.96	53.55	102.83
BLOCK-WORLD	0.79	4.43	9.63	17.54

Table 2: Execution times (in sec.) with and without cache.

	No Cache	With Cache	Improvement
LOGISTICS	6.43	0.90	86.0%
DEPOTS	5.87	0.90	84.5%
GRID	1.55	0.53	65.8%
BLOCK-WORLD	2.94	0.68	76.9%

Results Table 1 shows the results of Experiment 1. As the number of goals in a goal query graph increases, the execution times of the algorithm increase rapidly, especially in GRID. We anticipate that the execution time will be even larger for larger goal query graphs. Note that caching was enabled in Experiment 1. In Table 2, we can see that the caching mechanism can greatly enhance the performance of the algorithm. In fact, the caching mechanism is crucial to the performance of the evaluation of FO-CTL sentences and the EGRD search algorithm.

We have implemented the *pruned-reduce* algorithm in (Keren, Gal, and Karpas 2014) and conducted a preliminary experiment to evaluate its effectivness in EGRD. However, *pruned-reduce* do not always give a correct answer since the removal of action can falsify some FO-CTL statements.

Conclusions and Future Work

Previous works on GRD aim to recognize an agent's goal exactly. This paper extended the existing GRD frameworks by allowing the recognition of goal conditions that involve multiple goals. We described what goal conditions are and presented a novel graphical representation of goal conditions called goal query graph. We devised an algorithm to translate a goal query graph into a FO-CTL sentence for model checking. The EGRD search algorithm can modify an environment to optimize certain goal recognition behavior according to a given goal query graph. The caching mechanism is highly effective in speeding up the redesign process. In the future, we intend to extend our EGRD framework with partial observability of the agent and the observer.

Acknowledgment

This work has been taken place at UNIST and was supported by NRF (2016R1D1A1B0101359816) and UNIST (1.210080.01).

References

Ang, S.; Chan, H.; Jiang, A. X.; and Yeoh, W. 2017. Game-Theoretic Goal Recognition Models with Applications to Security Domains. In *GameSec 2017*, 256–272.

Baier, J. A.; and McIlraith, S. A. 2006. Planning with First-Order Temporally Extended Goals Using Heuristic Search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 788–795.

Bohn, J.; Damm, W.; Grumberg, O.; Hungar, H.; and Laster, K. 1998. First-Order-CTL Model Checking. In *Foundations of Software Technology and Theoretical Computer Science*, 283–294.

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J. A.; and McIlraith, S. A. 2017. Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 3716–3724.

Carberry, S. 2001. Techniques for Plan Recognition. *User Modeling and User-Adapted Interaction*, 11(1–2): 31–48.

Clarke, E. M.; and Emerson, E. A. 1981. Design and synthesis of synchronisation skeletons using branching time temporal logic. In *Logic of Programs, Proceedings of Workshop, Lecture Notes in Computer Science*, 52–71.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2: 189–208.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research (JAIR)*, 26: 191–246.

Kautz, H. 1987. *A Formal Theory of Plan Recognition*. Ph.D. thesis, Department of Computer Science, University of Rochester.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal Recognition Design. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 154–162.

Keren, S.; Gal, A.; and Karpas, E. 2016a. Goal Recognition Design with Non-Observable Actions. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 3152–3158.

Keren, S.; Gal, A.; and Karpas, E. 2016b. Privacy Preserving Plans in Partially Observable Environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 3170–3176.

Keren, S.; Gal, A.; and Karpas, E. 2018. Strong Stubborn Sets for Efficient Goal Recognition Design. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 141–149.

Keren, S.; Gal, A.; and Karpas, E. 2019. Goal Recognition Design in Deterministic Environments. *Journal of Artificial Intelligence Research (JAIR)*, 65: 209–269.

Keren, S.; Gal, A.; and Karpas, E. 2020. Goal Recognition Design - Survey. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 4847–4853.

Mirsky, R.; Gal, K.; Stern, R.; and Kalech, M. 2019. Goal and Plan Recognition Design for Plan Libraries. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2).

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-Based Heuristics for Goal Recognition. In *AAAI*, 3622–3628.

Pistore, M.; and Traverso, P. 2001. Planning as Model Checking for Extended Goals in Non-deterministic Domains. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Ramirez, M.; and Geffner, H. 2010. Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners. In *AAAI*, 1121–1126.

Shvo, M.; and McIlraith, S. A. 2020. Active Goal Recognition. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 9957–9966.

Son, T. C.; Sabuncu, O.; Schulz-Hanke, C.; Schaub, T.; and Yeoh, W. 2016. Solving Goal Recognition Design Using ASP. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 3181–3187.

Sukthankar, G.; Geib, C.; Bui, H. H.; Pynadath, D.; and Goldman, R. P. 2014. *Plan, Activity, and Intent Recognition: Theory and Practice*. Newnes.

Vered, M.; and Kaminka, G. A. 2017. Heuristic Online Goal Recognition in Continuous Domains. *arXiv:1709.09839*.

Wayllace, C.; Hou, P.; and Yeoh, W. 2017. New Metrics and Algorithms for Stochastic Goal Recognition Design Problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 4455–4462.

Wayllace, C.; Keren, S.; Gal, A.; Karpas, E.; Yeoh, W.; and Zilberstein, S. 2020. Accounting for Observer's Partial Observability in Stochastic Goal Recognition Design: Messing with the Marauder's Map. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*.