Domain-independent Plan Intervention

Paper #10

Abstract

Intervening is common in online assistive agents and safety-critical decision making, where an observer determines how to guide a user toward a desirable outcome. Intervention extends plan recognition to account for the goals of the user as well as undesirable outcomes. We formalize the plan intervention problem and characterize the observer's decision space as an Intervention Graph, from which we extract domain-independent features to assess critical actions leading to undesirable outcomes. We apply these features to learn whether to intervene in online scenarios while varying undesirable outcomes. In several benchmark problems, the learned models dominate three recent plan recognition approaches, although the performance varies by the application. To our knowledge, this is the first work to incorporate plan recognition for intervention, which lays a foundation for further applications of plan recognition for assistive and decision support agents.

1 Introduction

Even the best plan can go wrong. Dangers might arise from a user failing to follow the plan correctly or as a result of a nefarious agent. Consider route planning where a driver was unaware of upcoming road damage or a traffic jam. Or consider cybersecurity where a user was unaware of an unsafe hyperlink. In both, plans achieving the desirable goal might have similar prefixes to those that result in undesirable outcomes; only the plan suffix differs. State-of-the-art plan recognition techniques only provide a part of the solution for selecting interventions.

In this work, we create a domain-independent formulation of learning *whether* to intervene. At a minimum, intervention problems require a user and some intervening observer. In other settings, the observer also considers actions of a competing agent. The user and the competing agent have partial visibility and can not recognize the effects of certain actions that take place in the environment. The user and the competitor (if present) take turns in proposing actions to the observer. The observer evaluates how critical the presented action is for the user to reach his goal while avoiding the undesirable state using domain-independent features. If the observer finds that the presented action is critical, then it is flagged to warn the user. We discuss two types of actions that must be flagged: direct and indirect. Direct contributing actions have the undesirable state as their post-condition. Indirect contributing actions are sequences that add the necessary conditions leading to the undesirable state.

The contributions of this paper include:

- extending an existing plan recognition model to the intervention problem and demonstrating that three state-of-the-art approaches do not perform well for this problem
- formalizing the online intervention problem as determining when the criticality–i.e., the opportunity or urgency of possible damage–of a state warrants interruption;
- modeling the observer's decision space as an *Intervention Graph*, which can be constructed explicitly or sampled with the TopK planner (Riabov et al., 2014);
- defining domain-*independent* features to assess the criticality of a state using the Intervention Graph;
- extending existing goal/plan recognition benchmarks (Ramırez and Geffner 2009; 2010) to incorporate intervention and introducing a new plan intervention benchmark domain called Rush Hour[1],
- adapting four algorithms that learn whether to intervene,

Our results show that the proposed intervention solution outperforms existing plan recognition algorithms with fewer false positives and false negatives.

2 Background: Plan Recognition

We adapt the *offline* plan recognition formulation by Ramirez and Geffner (2010) to the problem of *online* plan intervention. In that work, a STRIPS (Fikes and Nilsson 1971) planning problem is a tuple $P = \langle F, A, I, G\rangle$ where F is the set of fluents, $I \subseteq F$ is the initial state, $G \subseteq F$ represents the set of goal states and A is the set of actions. Each action $a \in A$ is a triple $a = \langle Pre(a), Add(a), Del(a)\rangle$ that consists of preconditions, add and delete effects respectively, where $Pre(a), Add(a), Del(a)$ are all subsets of F. An action a is applicable in a state s if preconditions of a are true in s; $pre(a) \in s$. If an action a is executed in state s, it results in a new state $s' = (s \setminus del(a) \cup add(a))$. A solution for P is a plan $\pi = \{a_1, \ldots, a_k\}$ of length k that modifies I into G by execution of actions $a_1, \ldots, a_k$.

[1]We will release these benchmarks and code upon publication

Ramirez and Geffner (2010) define the plan recognition problem as $T = \langle D, \mathcal{G}, O, Pr \rangle$ where $D = \langle F, A, I \rangle$ is a planning domain, $\mathcal{G} \subseteq F$ is the possible set of goals, the observation sequence $O = o_1, \dots, o_m$ are actions $o_i \in A, i \in [1, m]$, and Pr is a prior probability distribution over $\mathcal{G}$. A solution for T is a probability distribution over $G \in \mathcal{G}$ indicating their likelihood.

3 Intervention Example

Consider an example where a user present actions while an observer decides whether to intervene. The observer is interested in two kinds of states: a desirable state $\mathrm{d} \subseteq F$ is the user's goal and the undesirable state $\mathrm{u} \subseteq F$ is an outcome the user wants to avoid, but may not be able to detect. (For convenience, we may discuss d or u as single states even though they are sets.) The observer's objective is to help the user achieve d while avoiding u. Thus the observer must identify actions that lead to u and intervene when appropriate.

Figure 1 shows a grid navigation task where the user navigates from `w1` by moving vertically and horizontally and `y3` contains a pit the user can not see. Thus, $\mathrm{d} =$ `at(z3)` and $\mathrm{u} =$ `at(y3)`. The user computes a satisficing plan for d and reveals by presenting an action at a time to the observer. A, B, C are all feasible from the user's perspective. However, the observer knows plans B and C are unsafe. If the user presents actions `move(y2,y3)` or `move(x3,y3)` observer must flag them because they are direct contributors to u. If the user is executing an optimal plan, the observer can flag indirect contributors to u. In this case, if the action `move(w2,w3)` is revealed, it must be flagged because the user can only move up to reach d (without violating optimality). Thus u becomes inevitable. Also, `move(w3,x3)`, `move(x3,y3)` must be flagged. We formally define the terms *directly contributing action* and *indirectly contributing sequence* later.

Interrupting the presented actions directly nor indirectly contributing to u results in false alarms. We tune the intervention sensitivity; i.e., direct vs. indirect using a set of intervention features defined in Section 7. But next we consider how to formulate intervention as an extension of plan recognition.

4 Intervention as Planning

We develop an approach whereby plans leading to u or d are identified using the recognition approach advanced by Ramirez and Geffner (2009, 2010). The observer should allow the user to pursue plans leading to d and intervene when the user takes actions from plans that get "too close" to u. Our key insight is considering u as a "goal", which is justified for planning where we want to identify plan suffixes that lead to undesirable outcomes. Undesirable plans Π_{u} are plans that achieve u and desirable plans Π_{d} achieve d. Thus, the observer must consider the two states together $\mathcal{G} = \{\mathrm{u} \cup \mathrm{d}\}$ The observer's recognition problem is: in domain D, given $O = \{o_1, \dots o_{i-1}\}$, what is the most likely "goal" from $\mathcal{G}$ if o_i will be observed? Plan recognition algorithms should help the observer identify the most likely goal and intervene if u is most likely. When u and d are far apart, recognition is straightforward. When they are proximate, it is difficult for the observer to recognize u on time.

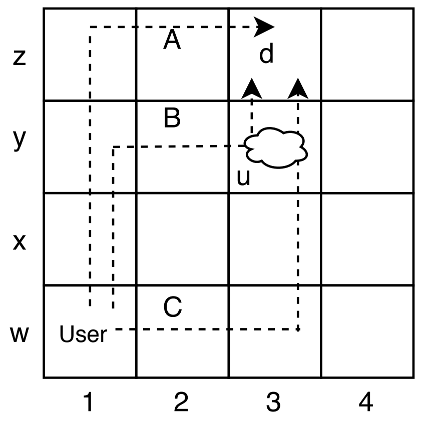

Figure 1: An example of helpful intervention.

In intervention, the observer asks the question: in domain D, given $O = \{o_1, \dots o_{i-1}\}$ must o_i be flagged to prevent reaching u first? Note that $O \not\models \mathrm{u}$. If we are to solve intervention as a plan recognition problem, we need to assume a prior probability (e.g., uniform) over $\mathcal{G}$. Sometimes, it may be hard for the observer to correctly determine the priors. From the user's point of view, the prior of $\mathrm{u} \approx 0$ because he does not know about it, whereas the observer may assign a different prior. This leads to false alarms/misses if intervention happens based on the most likely goal given O.

Definition 1. *The* online plan intervention problem *is a tuple $\mathcal{I} = \langle D, O, \mathcal{G} \rangle$ where $D = \langle F, A, I \rangle$ is a planning domain, O is sequence of observed actions $(o_1, o_2, \dots o_{i-1})$, o_1 is the presented, new action,* $\mathrm{d}, \mathrm{u} \in \mathcal{G}$ *and* $\mathcal{G} \subseteq F$

The user solves the planning problem $P_0 = \langle F_0, A_0, I_0, \mathrm{d} \rangle$, where $F_0 \subset F$, $A_0 \subseteq A$, $I_0 \subseteq I$ and $\mathrm{d} \subset F$. Because $\mathrm{u} \subset F$, $\mathrm{u} \neq \mathrm{d}$ and u is hidden to the user, some solutions to P_0 become undesirable. Therefore, the observer can partition the solution space for P_0 to two disjoint plan sets: desirable (Π_{d}), which do not contain actions that lead to u and undesirable (Π_{u}). The user following his own plan may present actions to the observer from both Π_{u} and Π_{d}.

A solution to $\mathcal{I}$ is a binary decision that maps each action the user presents o_i to $\{Yes, No\}$ indicating whether it requires intervention. For example, suppose in Figure 1 the user is executing the plan C = { `move(w1,w2)`, `move(w2,w3)`, `move(w3,x3)`, `move(x3,y3)`,...}. When he presents `move(w1,w2)` it is flagged as No, when `move(w2,w3)` is presented, it flagged as No and so on.

We use machine learning to derive functions that perform the mapping from observations to flagging decisions $\{Yes, No\}$, for an application specific constant $\Theta \mapsto$ {'direct','indirect'}. The first method uses features derived from a data structure called an *Intervention Graph*. The second method uses plan distance metrics that compare distances between the user's projected plan to Π_{u} and Π_{d} as features. When $\Theta = direct$, we define the directly contributing action.

Definition 2. *A* directly contributing action a_{crit} *occurs in an undesirable plan* $\pi_{\mathrm{u}} \in \Pi_{\mathrm{u}}$ *and execution of* a_{crit} *in state* s *results in a state* s' *such that* $s' \models \mathrm{u}$.

When $\Theta = indirect$, we define an indirectly contributing sequence q_{crit}.

Definition 3. *An* indirectly contributing sequence q_{crit} *is a totally ordered action sequence that starts with the presented action* o_i *from in an undesirable plan* $\pi_{\mathrm{u}} \in \Pi_{\mathrm{u}}$. *Executing actions in* q_{crit} *from state* s *results in a state* s' *such that* $s' \models \mathrm{u}$.

5 Intervention With a "Competitor"

Above, we introduced the simple version of the intervention problem. But sometimes there is another agent interacting with the environment at the same time as the user. We introduce an additional actor, the *competitor*. The competitor is not a standard adversary like you would find in a two-player game. The competitor has a limited set of actions, which he uses to create states that will lead to u before d is reached.

Figure 2 shows examples with the competitor in a modified version of the block-words domain used in (Ramırez and Geffner 2009). Figure 2 (top) shows a problem with 4 blocks (T, B, A, D) initially on the table and time flows from left to right. $\mathrm{d} = \{$(CLEAR T)(ON T A)(ON A D)(ONTABLE D)$\}$(i.e., TAD) and $\mathrm{u} = \{$(CLEAR B)(ON B A)(ON A D)(ONTABLE D)$\}$ (i.e., BAD). The user does not know about the competitor's modifying the state of block B (dotted line block) nor about u. The competitor can only modify the state of block B, and execute actions with the hidden block (stack B, pickup B etc.). The user may enable d with the action `stack(user,A,D)` and at the same time creating an opportunity for the competitor to reach u first. $a_{crit} = \{$`stack(competitor,B,A)`$\}$. q_{crit}=$\{$`pickup(competitor,B)`, `stack(competitor,B,A)`$\}$.

In Figure 2 (bottom) d = (CUP) and u = (CUT) are defined the same way as the TAD/BAD example. Block T is hidden from the user. Intervention must happen when `STACK(competitor, T, P)` is observed. If not, the user could unwittingly reach u by placing U on T (thinking P is free) and stacking C on U. Therefore, $q_{crit} = \{$`stack(competitor,T,P)`, `pickup(user,U)`,`stack(user,U,T)`, `pickup(user,C)`,`stack(user,C,U)`$\}$.

Definition 4. *The* online plan intervention problem *with a competitor is a tuple* $\mathcal{I} = \langle D, O, \mathcal{G}\rangle$ *where* $D = \langle F_0 \cup F_1, A_0 \cup A_1, I\rangle$ *is a planning domain, O is sequence of observed actions,* $\mathrm{d}, \mathrm{u} \in \mathcal{G}$ *and* $\mathrm{d}, \mathrm{u} \subseteq F_0 \cup F_1$

The user solves the planning problem $P_0 = \langle F_0, A_0, I, \mathrm{d}\rangle$. The competitor wants to solve the planning problem $P_1 = \langle F_1, A_1, I, \mathrm{u}\rangle$. However, the competitor does not have access to all the actions needed to achieve u and requires the user to enable some conditions for u. Also, $A_1 \cap A_0 = \emptyset$ and $F_1 \neq F_0$. Because u is hidden to the user, some solutions to P_0 become undesirable. Observations for $\mathcal{I}$, contain actions from A_0 and A_1.

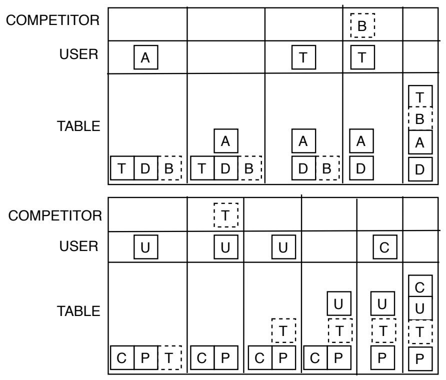

Figure 2: Reaching u = (BAD) (top) and u = (CUT) (bottom) with an competitor and a user. The table initially contains all four blocks in each example. A block in the user or competitor row indicates a pickup action that is removed using a stack action.

Given an observation trace $O = \{o_1, o_2, \ldots o_{i-1}\}$, the observer sees the effects of the actions upto o_{i-1}. An intervention episode is defined from state I until one of u or d is satisfied. If the competitor is present, which agent presents first action in the episode is decided randomly. The observer makes the intervention decision for the presented action and adds it as o_i. The agents take turns in presenting the next action a_{i+1} to execute in the episode from A_0 and A_1 until the episode terminates. We make several assumptions. **(Observability)** The observer has full observability, knows about d and u and helps the user avoid u. u is unknown to the user. d is unknown to the competitor (if present). The user can not recognize effects of the competitor's (if present) actions. **(Plans)** The user follows a satisficing plan to reach d, but may reach u unwittingly. There is a satisficing plan to reach u and we assume that it has a common prefix with a plan to reach d. In this work, we are only interested in correctly identifying actions that are in a_{crit} and q_{crit} for a given intervention problem. Therefore, we assume that the user continues to present the observer with actions from his original plan even after the first positive flag and does not replan. We leave the problem of handling positive flags to future work. **(Competitor)** When present, the competitor only perform actions using objects hidden to the user; this restriction follows from many security domains where an attacker is a remote entity that plants traps and expects the user to become an unwitting accomplice (e.g., attacker sends clickbait phishing email to the user). The user and the competitor (if present) are (bounded) rational agents.

6 The Intervention Graph

We define the intervention graph, which models the decision space of the observer. We then extract several features from the intervention graph, which we use to derive functions that map the observations to flagging decisions. Unfortunately, extracting these features can be intractable when the graph is large, as can happen with larger problems, multiple paths reaching d, or multiple undesirable goals. Therefore, we define an additional set of features by sampling the plan space, where the samples are obtained with the Top-K planner (Ri-

Algorithm 1 Build Intervention Graph

```
Require: $D$, $s$, $\mathcal{G}$
 1: $i = 0$; $s_i \leftarrow I$
 2: procedure EXPANDGRAPH($D, s, \mathcal{G}$)
 3:     if $s_i \models \mathrm{u}, \mathrm{d}$ then return $\langle V, E \rangle$
 4:     else
 5:         for $a \in A$ where $Pre(a) \in s_i$ do
 6:             $s_{i+1} \leftarrow ((s_i \setminus Del(a)) \cup Add(a))$
 7:             if $s_{i+1} \equiv s_i$ then continue
 8:             $v \leftarrow$ AddVertex ($s_{i+1}$)
 9:             $e \leftarrow$ AddEdge ($s, s_{i+1}, a$)
10:             $V \cup \{v\}$ ; $E \cup \{e\}$
11:             ExpandGraph ($D, s_{i+1}, \mathcal{G}$)
```

abov, Sohrabi, and Udrea 2014).

The Intervention Graph allows the observer to evaluate how close the current projection of the observed partial plan is to u. The Intervention Graph consists of alternating state and action layers where each state layer consists of predicates that have been made true by the actions in the previous layer. An action layer consists of actions $a \in A$ whose preconditions are satisfied in the state. Algorithm 1 describes the process of building the Intervention Graph. The algorithm takes as input a domain theory D, state s and $\mathcal{G} = \{\mathrm{u}, \mathrm{d}\}$ (lines 1-2). Before any observations have been made, the current state (i.e., root of the tree) is set to initial state I. Next, using the domain theory D, actions $a \in A$ whose preconditions are satisfied at current state are added to the graph (lines 5-6). Each action in level i spawns possible states for level $i + 1$. Line 7 ensures that the actions that immediately inverts the previous action are not added to the graph. For each resulting state a search node is created, with an edge representing the action responsible for the state transition (lines 8-10). Calling the method recursively for each open search node until d and u are added to the graph generates the plan hypotheses for the observer (line 11). To ensure that only realistic plans are explored, we do not add no-op actions to the action layers in the graph. As a new observation arrives, the root of the graph is changed to reflect the new state after the observation and subsequent layers are modified to that effect.

The Intervention Graph is a weighted, single-root, directed acyclic connected graph $IG = \langle V, E \rangle$, where V is the set of vertices denoting possible states the user could be in until d is reached, and E is the set of edges representing actions from $A_0 \cup A_1$. A path from the root of the tree to d that goes through u represents an undesirable plan, while a path from root to a node containing d that bypasses u represents a desirable plan.

7 Extracting Intervention Graph Features

We extract a set of features from the intervention graph that help determine when to intervene. These features include: Risk, Desirability, Distance to d, Distance to u and Percentage of active undesirable landmarks. We use the definition for landmarks by Hoffman et al. (2004), which are propositions that have to be true at some time in every solution plan in a planning problem. Formally: a fact $L_P \subset F$ of a planning task $P = \langle F, A, I, G \rangle$ is a **fact landmark** for P iff L_P is true in some state in all valid plans that modifies I to G.

We use these features to train a classifier that learns to identify actions in a_{crit} and q_{crit}. Figure 3 illustrates a fragment of the intervention graph from Figure 2 (top) after the user presents the action `PICK-UP A`, which we will use as a running example to discuss feature computation.

Risk (**R**) quantifies the probability that the presented action will lead to u. R is also coupled with the uncertainty the observer has about what the next action the user or the competitor (if present) will present. We model the uncertainty as a uniform probability distribution across the set of actions whose preconditions are satisfied in current state. We define R as the posterior probability of reaching u while the user is trying to achieve d. We extract plans from the intervention graph from the root to any leaf containing the d, including the plans in which the user has been subverted to reach u instead. By construction, d will always be a leaf. Let $\Pi_{\mathcal{C}}$ be the candidate plans reaching d and let $|\Pi_{\mathcal{C}}| = n$. The plan set Π_u contains action sequences that reach state u such that, $\Pi_u \subseteq \Pi_{\mathcal{C}}$, $|\Pi_u| = m$ and $(m <= n)$. We compute posterior probability of reaching u for a path $\pi \in \Pi_u$, using chain rule in probability as, $P_\pi = \prod_{j=1}^{k} P(\alpha_j | \alpha_1, \alpha_2, ..., \alpha_{k-1})$, and $\alpha_j \in A$ and k is the length of path until u is reached. Then:

$$R = \begin{cases} \frac{\sum_{i=1}^{m} P_{\pi_i}}{m} & m > 0 \\ 0 & m = 0 \end{cases}$$

In Figure 3, $(n = 6)$ and $(m = 1)$. Since we assumed full observability for the observer, the root of the tree (current state) is assigned the probability of 1.0. Actions that are immediately possible after the current state are each assigned probabilities following a uniform distribution across the branching factor (0.33). Then for each applicable action in the current state, the resulting state gets the probability of $(1.0 \times 0.33 = 0.33)$. Similarly, we apply the chain rule of probability for each following state and action level in the graph until u first appears in the path. $R = \frac{0.08}{1} = 0.08$.

Desirability (**D**) measures the effect of the observed action to help the user pursue the desirable goal safely. Given $\Pi_{\mathcal{C}}$ as the set of plans extracted from the intervention graph that reach d and $|\Pi_{\mathcal{C}}| = n$. The plan set Π_d contains action sequences that reach state d without reaching u, $\Pi_d = \Pi_{\mathcal{C}} \setminus \Pi_u$, we compute posterior probability of reaching d without reaching u for a path $\pi \in \Pi_d$, using chain rule in probability as, $P_\pi = \prod_{j=1}^{k} P(\alpha_j | \alpha_1, \alpha_2, ..., \alpha_{k-1})$, and $\alpha_j \in A$ and k is the length of path. Then:

$$D = \begin{cases} \frac{\sum_{i=1}^{n-m} P_{\pi_i}}{n-m} & n - m > 0 \\ 0 & n - m = 0 \end{cases}$$

In Figure 3, there are five instances where user achieved d without reaching u (two in sub tree T_1, three in the expanded branch). Following the same approach to assign probabilities for states and actions, $D = \frac{(0.08+0.08+0.08+0.04+0.04)}{5} = 0.07$. R and D are based on probabilities indicating the confidence the observer has about the next observation. We also use simple distance measures: (1) distance to u (δ_u) and (2)

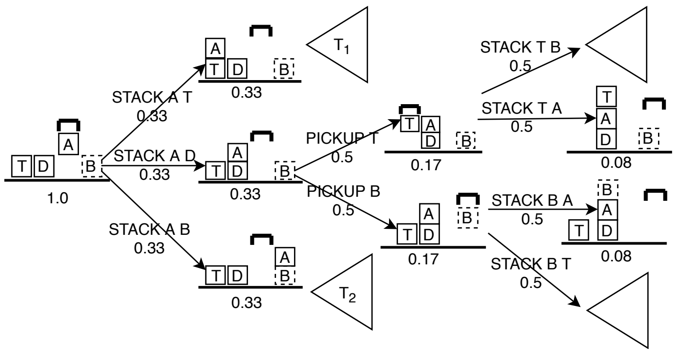

Figure 3: Fragment of the decision space after PICKUP A has been observed for block-words example in Figure 2 (top). Numbers under each state and action indicate the probability. Sub trees T_1 and T_2 are not expanded for simplicity.

distance to d (δ_d). Both distances are measured in the number of actions required to reach a state containing d or u from root in the intervention graph.

Distance to u (δ_u) measures the distance to state u from the current state in terms of the number of actions. $\Pi_\mathcal{C}$ is the set of paths extracted from the intervention graph that reach d and $|\Pi_\mathcal{C}| = n$. The path set Π_u contains action sequences that reach state u such that, $\Pi_u \subseteq \Pi_\mathcal{C}$, $|\Pi_u| = m$ and ($m <= n$). We count s, the number of the edges (actions) before u is reached for each path $\pi \in \Pi_u$ and δ_u is defined as the average of the distance values:

$$\delta_u = \begin{cases} \frac{\sum_{i=1}^{m} s_i}{m} & m > 0 \\ -1 & m = 0 \end{cases}$$

In this formula, -1 indicates that the undesirable state is not reachable from the current state. For the example problem illustrated in Figure 3, $\delta_u = \frac{3}{1} = 3$.

Distance to d (δ_d) measures the distance to d from current state. The path set Π_d contains action sequences that reach d without reaching u, $\Pi_d = \Pi_\mathcal{C} \setminus \Pi_u$, we count t, the number of the edges where d is achieved without reaching u for each path $\pi \in \Pi_d$. Then, δ_d is defined as the average of the distances given by the formula:

$$\delta_d = \begin{cases} \frac{\sum_{i=1}^{n-m} t_i}{n-m} & n-m > 0 \\ -1 & n-m = 0 \end{cases}$$

In this formula, -1 indicates that d can not be reached safely from the current state. For the example problem illustrated in Figure 3, $\delta_d = \lceil \frac{3+3+7+7+3}{5} \rceil = 5$.

Percentage of active attack landmarks ($\mathcal{L}_{ac}$) captures the criticality of the current state toward contributing to u. Landmarks (Hoffmann, Porteous, and Sebastia 2004) are predicates (or actions) that must be true in every valid plan for a planning problem. We used the algorithm in Hoffmann et al. (2004) to extract fact landmarks for the planning problem $P = \langle D, \mathrm{u} \rangle$. These landmarks are referred to as attack landmarks ($\mathcal{L}_u$) because they establish predicates that must be true to reach u. Landmarks have been successfully used in deriving heuristics in plan recognition (Vered et al. 2018) and generating alternative plans (Bryce 2014). We compute the percentage of active attack landmarks in the current state ($\mathcal{L}_{ac}$). To compute $\mathcal{L}_{ac}$ in Figure 3, we count the number of landmark predicates that have become active (l) in the root of the intervention graph. Then, ($\mathcal{L}_{ac}$) is given by the formula: $\mathcal{L}_{ac} = \frac{l}{|\mathcal{L}_u|}$. In Figure 3, $l = 4$ and $\mathcal{L}_{ac} = 4/10 = 0.4$.

For each presented action, the intervention graph is generated and features are computed, producing the corresponding feature vector for the presented action. Landmarks for the problem $P = \langle D, \mathrm{u} \rangle$ are computed apriori.

8 Sampling the Intervention Graph

To overcome the state space explosion in large domains, we propose a second method. Instead of finding plans from the full search space, the observer samples only a subset of plans to compute a new feature vector called Sampled Features. Here, we estimate the Risk and Desirability using plan distance measures. If the user is executing an unsafe plan, then that plan should be more similar to a sample of unsafe plans, compared to a sample of safe plans.

The observer computes plan distances between a reference plan (π') and sampled plans (Π'') for both u and d for each presented action. We follow the method proposed by Vered et al. (2016) to generate the observation compatible plan by concatenating the observation history with the optimal plan that reaches u (and d) to produce π'. We use the Top-K planner with K=50 (Riabov, Sohrabi, and Udrea 2014), to sample the plan space. We use action set distance (ACD), state sequence distance (SSD), causal link distance (CLD) (Nguyen et al. 2012), Generalized Edit Similarity (GED) for sequences of states and actions (Sohrabi et al. 2016) as to measure the distance. When an action is presented to the observer, π' is computed. Then, observation compatible Top-K plans are produced for u and d separately. Then we compute the medians of ACD, CLD and SSD, minimum remaining actions to u and d, minimum action GED and state GED are computed for u and d for all ⟨reference, sample⟩ pairs. This produces the Sampled Feature vector for the presented action.

9 Learning When to Intervene

We train a classifier to categorize the presented action into two classes: (Y) indicating intervention is required and (N), indicating otherwise. Given a observations labeled as Y/N and corresponding feature vectors, we train the classifiers with 10-fold cross validation. The trained model is used to predict intervention for previously unseen intervention problems. We chose Naive Bayes, K-nearest neighbors, decision tree and logistic regression classifiers from Weka [2]. Attribute selected classifiers filter the feature vector to only select critical features. This step reduces complexity of the model, makes the outcome of the model easier to interpret, and reduces over-fitting.

We generated training data from twenty intervention problems using the benchmark domains. Additionally, we created a new planning domain for the Rush Hour puzzle. In this puzzle, user moves vehicle objects on a grid to clear a path for a target vehicle to move to the exit located on the perimeter of the grid. We created training data from twenty Rush Hour puzzles. We restricted the number of observation traces per intervention problem to 100.

A full parameter search revealed the best parameters for the classifiers. The decision tree classifier is tuned to pruning confidence=0.25 and minimum number of instance per leaf=2. K-nearest neighbor classifier is tuned to use k=1 and distance measure=Euclidean. The logistic regression classifier is tuned for ridge parameter = 1.0E-8. Naive Bayes classifier is tuned with the supervised discretization=True. For the intervention graph approach, the learned models chose distance to u and Risk as the dominant features for the benchmark domains.

10 Results and Discussion

We focus on two questions: (1) Using domain-independent features indicative of the likelihood to reach u from current state, can the observer correctly interrupt to prevent the user from reaching u? and (2) How does the learning approach perform against state-of the-art goal recognition? To address the first question, we evaluated the performance of the learned model on unseen problems.

The benchmark suite consists of Blocks-words, IPC Grid, Navigator and Ferry domains and the Generalized Rush Hour domain. For the **Blocks-words** domain, we chose word building problems. The words user and the competitor want to build are different but they have some common letters. In the **IPC grid** domain, the user agent moves through a grid to get from point A to B. Certain locked positions on the grid can be opened by picking up keys. In the **Navigator** domain, the user agent moves from one point in grid to another. In IPC Grid and Navigator domains, we designated certain locations on the grid as traps. The goal of the user is to navigate to a specific point on the grid without passing through the trap. In the **Ferry** domain, a single ferry moves cars between different locations. We assigned a port as *compromised*. The ferry's objective is to transport cars to specified locations without passing the compromised port. In the **Rush Hour** domain, we created puzzles on arbitrary grid sizes and exits. The player wins by moving a target car to the exit. We also designated some vehicles on the grid as forbidden. The forbidden vehicles need not be moved to solve the puzzle. Any user action that moves any of the forbidden vehicles raises the intervention flag.

We generate 3 separate test instances of 20 problems each (total of 60) for the benchmark domains with problems different from training instances. For example, number of blocks in the blocks words domain, size of grid (navigator, IPC-Grid), number of cars and exit locations in Rush Hour domain, accessible and inaccessible paths on the grid (navigator, IPC-Grid), properties of artifacts in the grid (IPC-Grid). For each problem in test instance we generated 10 observation traces (total of 600 test observation traces). For the Rush Hour domain we created 2200 observation traces and split to training and testing 60/40. We define true-positive as the classifier correctly identifying the presented action as action that must be in a_{crit} or q_{crit}. True-negative is an instance where the classifier correctly identifying an action as not belonging to a_{crit} or q_{crit}. False-positives are instances where classifier incorrectly identifies an action as belonging to a_{crit} or q_{crit}. False-negatives are instances where the classifier incorrectly identifies the presented action as not belonging to a_{crit} or q_{crit}. Naturally, our test observation traces contain a large number of negatives. To offset the bias introduced to the classifier by the class imbalance, we report Matthews correlation coefficient (MCC) because it gives an accurate measure the quality of a binary classification while taking into account the different class sizes. We also report the F-score $= \frac{tp}{tp+1/2(fp+fn)}$ for the classifiers. tp, fp, fn are the number of true positives, false positives and false negatives respectively.

We implemented three state-of-the art plan recognition algorithms to compare intervention accuracy to the proposed learning based solution. We selected Ramirez and Geffener's probabilistic plan recognition algorithm (Ramırez and Geffner 2010) (both the satisficing, and optimal implementations) and the Goal Recognition with Goal Mirroring algorithm by Vered et al. (2018). For each presented action, the observer solves a plan recognition problem using each approach. We assumed uniform priors for over u and d. If u is the top ranked goal for the presented action, then it is flagged as requiring intervention. The assumption is that these algorithms must also be able to correctly identify u as most likely goal for the actions in a_{crit} and q_{crit}. We used the same test data suite to evaluate accuracy.

Table 1 shows that the classifiers trained with features from the intervention graph achieve high accuracy for all the domains when predicting intervention. The MCC value shows that the class size does not bias the classifier. Low false positives and false negatives suggests that the user will not be unnecessarily interrupted. As expected performance degrades when we use a sampled plan space to derive features. For the benchmark domains we were able to discover at least one classifier responded with very high F-score the plan similarity features, the exception being the Ferry domain. The Rush Hour domain produced comparatively better results, with the decision tree and K-Nearest neighbor classifiers being the best.

Features derived from the intervention graph accurately separate actions where the user has limited options available to reach the desirable goal while avoiding the undesirable

[2] `http://www.cs.waikato.ac.nz/ml/weka/`

Domain	Naive Bayes						Decision Tree						Logistic Regression						K-Nearest					
	Inst 1		Inst 2		Inst 3		Inst 1		Inst 2		Inst 3		Inst 1		Inst 2		Inst 3		Inst 1		Inst 2		Inst 3	
	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC
Intervention Graph Method																								
Blocks	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EasyIPC	1	1	1	1	1	1	1	1	1	1	1	1	.88	.87	.88	.87	.86	.86	1	1	1	1	1	1
Ferry	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Navigator	1	1	1	1	.99	.99	.87	.87	.72	.74	.90	.90	1	1	1	1	.99	.99	1	1	.96	.96	.99	.99
Plan Space Sampling Method																								
Blocks	.25	.33	.25	.33	.25	.33	.25	.33	.25	.33	.25	.33	.25	.33	.25	.33	.25	.33	1	1	1	1	1	1
EasyIPC	1	1	1	1	1	1	1	1	1	1	1	1	.64	.63	.46	.44	.67	.66	.05	-.04	.04	-.03	.05	-.02
Ferry	.34	.33	.32	.31	0.02	-.004	.25	.28	.24	.23	.86	.86	.31	.32	.23	.22	1	1	.33	.40	.13	.15	.81	.82
Navigator	1	1	1	1	1	1	.62	.65	1	1	1	1	.60	.59	.98	.94	.97	.97	.61	.65	1	1	1	1
Rush Hour	.44	.40					.89	.89					.56	.54					.89	.89				

Table 1: F-score and MCC for predicting intervention using Intervention Graph and plan space sampling methods

Domain	Inst 1						Inst 2						Inst 3					
	RG (LAMA)		RG (HSP)		GM		RG (LAMA)		RG (HSP)		GM		RG (LAMA)		RG (HSP)		GM	
	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC	F-score	MCC
Blocks	.38	.45	.38	.45	.36	.43	.43	.49	.43	.49	.39	.45	.40	.47	.40	.47	.38	.45
EasyIPC	.13	.05	.13	.05	.10	.01	.21	.17	.18	.13	.12	.06	.23	.19	.22	.19	.14	.09
Ferry	.17	.18	.22	.20	.10	.08	.22	.23	.11	.06	.15	.09	.15	.17	.47	.52	.21	.34
Navigator	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 2: F-score and Maththews Correlation Coefficient (MCC) for recognizing intervention with probabilistic goal recognition (RG) (Ramırez and Geffner 2010) using a satisficing (LAMA) and optimal (HSP) planners goal mirroring (GM) (Vered et al. 2018). For the Navigator domain intervention problems, the true-negative, false negative rates were 100% each. Therefore, F-score and MCC are not reported for that problem set.

state. Thus the classifiers perform well in recognizing critical actions in new problems. The sampled features rely on the learning algorithm to produce better output when predicting intervention.

Comparing results in Table 2, learning methods outperform existing plan recognition algorithms when predicting intervention. The algorithms we selected clearly struggled to predict intervention in the Navigator domain. These results suggest that although we can adopt existing plan recognition algorithms to identify when the user needs intervention, it also produces a lot of false negatives and false positives in correctly identifying which state (desirable/undesirable) is most likely given the observations. The learning approach is better suited for intervention because the observer can target specific critical actions and allow the user some freedom.

11 Related Work

Plan recognition is the problem of inferring the course of action (i.e., plan) an actor may take towards achieving a goal from a sequence of observations (Schmidt, Sridharan, and Goodson 1978; Kautz and Allen 1986). The constructed plan is expected to result in states that correspond to goals of the actor, which in turn presupposes that the actor intends to achieve those goals. In domain-dependent methods, agents rely heavily on domain knowledge for inference. Kabanza et al. (2010) presents a solution that recognizes an agents adversarial intent by mapping observations made to date to a plan library. Boddy et al. (2005) discuss construction and manipulation of domain models that describe behaviors of adversaries in computer security domain and use these models to generate plans. Goal Driven Autonomy (GDA) that allows agents to continuously monitor the current plans execution and assess if the current state matches with expectation (Klenk et al.2013). Geib et al. (2001) extended plan recognition to handle hostile agents.

Domain-independent goal recognition uses planning to infer agents goals. Ramirez and Geffner (2009; 2010) used an existing planner to generate hypotheses from observations to infer a single agent's plan. Their approaches offer advantages of being more adaptive to input as well as exploiting existing planning systems and plan representations. Their first approach computed the set of goals that can be achieved by optimal plans that match the observations. The second approach removed the optimality constraint and computed a probability distribution across likely goals. (Ramırez and Geffner 2010). Keren et al. (2014) introduced the worst-case distinctiveness (WCD) metric as a measurement of the ease goal recognition in a domain. By limiting the set of available actions in the model, WCD can be minimized, allowing the agent to reveal it's goal early.

Landmarks are used in recognition to minimize the number of hypotheses the agent has to evaluate. Pozanco et al. (2018) combines probabilistic plan recognition and landmarks to counter plan and block an opponent's goal achievement. This is approach uses offline recognition. In plan recognition, identifying the plan at the right time is not a priority. In intervention this is critical because we want to reduce false alarms and misses. Our approach complements (and improves) existing recognition methods by using classifiers to recognize intervention.

12 Conclusion

When agents work in environments with hidden information, their plans can go wrong. We formalized the online plan intervention problem to recognize when plans will not achieve the intended goal. We proposed two learned models that use features extracted from the intervention graph and the sampled plan space to identify critical actions and critical sequences, which if not intervened will cause the user agent's plan to fail. We evaluated how existing classifiers perform in predicting intervention. The learning approach outperforms existing online plan recognition approaches in predicting intervention with lower false positives and false negatives.

References

Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 12–21.

Bryce, D. 2014. Landmark-based plan distance measures for diverse planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 56–64.

Fernau, H.; Hagerup, T.; Nishimura, N.; Ragde, P.; and Reinhardt, K. 2003. On the parameterized complexity of the generalized rush hour puzzle. In *Proceedings of the 15th Canadian Conference on Computational Geometry*, 6–9.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3):189–208.

Geib, C. W., and Goldman, R. P. 2001. Probabilistic plan recognition for hostile agents. In *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, 580–584. AAAI Press.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278.

Jaidee, U.; Muñoz-Avila, H.; and W. Aha, D. 2011. Integrated learning for goal-driven autonomy. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2450–2455.

Kabanza, F.; Bellefeuille, P.; Bisson, F.; Benaskeur, A. R.; and Irandoust, H. 2010. Opponent behaviour recognition for real-time strategy games. In *Proceedings of the 5th AAAI Conference on Plan, Activity, and Intent Recognition (PAIR)*, AAAIWS'10-05, 29–36. AAAI Press.

Kautz, H. A., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of 5th National Conference on Artificial Intelligence (AAAI)*, 32–37.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 154–162.

Klenk, M.; Molineaux, M.; and Aha, D. W. 2013. Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence* 29:187–206.

Nguyen, T. A.; Do, M.; Gerevini, A. E.; Serina, I.; Srivastava, B.; and Kambhampati, S. 2012. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence* 190:1–31.

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Proceedings of the Thirty-First Association for the Advancement of Artificial Intelligence (AAAI)*, 3622–3628.

Pozanco, A.; Yolanda, E.; Fernández, S.; and Borrajo, D. 2018. Counterplanning using goal recognition and landmarks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 4808–4814.

Ramırez, M., and Geffner, H. 2009. Plan recognition as planning. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI)*, 1778–1783.

Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, 1121–1126.

Riabov, A.; Sohrabi, S.; and Udrea, O. 2014. New algorithms for the top-k planning problem. In *Proceedings of the theScheduling and Planning Applications woRKshop (SPARK) at the 24th International Conference on Automated Planning and Scheduling*, 10–16.

Schmidt, C. F.; Sridharan, N.; and Goodson, J. L. 1978. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence* 11(1-2):45–83.

Sohrabi, S.; Riabov, A. V.; Udrea, O.; and Hassanzadeh, O. 2016. Finding diverse high-quality plans for hypothesis generation. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, 1581–1582.

Vered, M.; Pereira, R. F.; Magnaguagno, M.; Meneguzzi, F.; and Kaminka, G. A. 2018. Online goal recognition as reasoning over landmarks. In *The AAAI 2018 Workshop on Plan, Activity, and Intent Recognition (PAIR)*.

Vered, M.; Kaminka, G. A.; and Biham, S. 2016. Online goal recognition through mirroring: Humans and agents. In *Proceedings of the 4th Annual Conference on Advances in Cognitive Systems*.