Probabilistic Plan Recognition for Hostile Agents

Christopher W. Geib and **Robert P. Goldman**
Honeywell Technology Center
Minneapolis, MN 55418 USA
{geib,goldman}@htc.honeywell.com

Abstract

This paper presents a probabilistic and abductive theory of plan recognition that handles agents that are actively hostile to the inference of their plans. This focus violates a primary assumption of most previous work, namely complete observability of the executed actions.

Introduction

This paper extends a theory of plan recognition (Goldman, Geib, & Miller 1999) to handle agents that are actively hostile to the inference of their plans. This focus violates a primary assumption of most previous work, namely complete observability of the executed actions. Confronting hostile agents means, we can no longer assume that we see all of the agents actions.

The plans of hostile agents also require an account of the effects of: world state/context, multiple interleaved plans, partially ordered plans, and negative evidence (lack of a report of an action). In our previous work (Goldman, Geib, & Miller 1999) (GG&M) we presented a probabilistic, abductive theory of plan recognition centered around plan *execution*, which addressed these issues. This work extends our previous work to the case of hostile agents.

In this paper, we provide an understanding at the "knowledge level." We have implemented our plan recognition model, and the examples we discuss are taken from this implementation. However, we do not claim to provide an efficient solution to the plan recognition problem. There are many ways our model could be implemented; we discuss some of these in GG&M .

The rest of this paper is organized as follows. First we give a brief definition and review of plan recognition, then we present a brief review of our previous work in probabilistic plan recognition providing the intuition underlying the model. Following this, we discuss the problems raised by hostile agents and outline answers.

Plans

In this paper, we use simple hierarchical (task decomposition) plans(Erol, Hendler, & Nau 1994), as most plan recognition work does. We assume that agents have a *plan library* that provides recipes for achieving goals. Figure 1 shows a plan library for a "hacker" in a simplified computer network intrusion example.

If a hacker has a goal like stealing information from a computer (**theft**), the plan library breaks that goal into five *steps*: scan the system to determine vulnerabilities (**recon**), exploit the system's weaknesses to gain entry (**break-in**), escalate privileges (**gain-root**), export desired data (**steal**), and hide traces of presence on computer (**clean**). Ordering constraints within a method are represented by directed arcs. For example, the hacker must **break-in** before she can **gain-root**.

Finally, notice that there is a condition/event that is tied to the action **clean**. The dashed line represents the fact that this condition results from the execution of the action. Thus, if **clean** is executed it will result in deleted event logs (**deleted-logs**). This information about action effects will be critical to inferring the execution of unobserved actions.

Plan Recognition

Plan recognition is the process of inferring the goals of an agent from observations of an agent's actions. Previous work in this area has assumed that the agent being observed is not actively hostile to the process of plan recognition.

In 1986, Kautz and Allen (K&A) published "Generalized Plan Recognition," (Kautz & Allen 1986) that framed almost all work in plan recognition. K&A defined the problem of plan recognition as the problem of identifying a minimal set of *top-level actions* sufficient to explain the set of observed actions.

They treated the problem as one of computing minimal explanations, in the form of vertex covers, based on the plan graph. For example, if one observed **recon** (See Figure 1) the minimal explanations would be

$$(\textbf{theft}) \vee (\textbf{vandalism}) \vee (\textbf{info})$$

In fact, with only this observation we have no evidence to rule out the possibility that the agent has multiple goals. Therefore the system should also consider the possibility of *two* or even *three* top-level goals in order to explain this observation.

One problem with this work is that it does not take into account differences in the *a priori* likelihood of different plans. Charniak and Goldman (C&G) (1993) argued that, since plan recognition involves abduction,

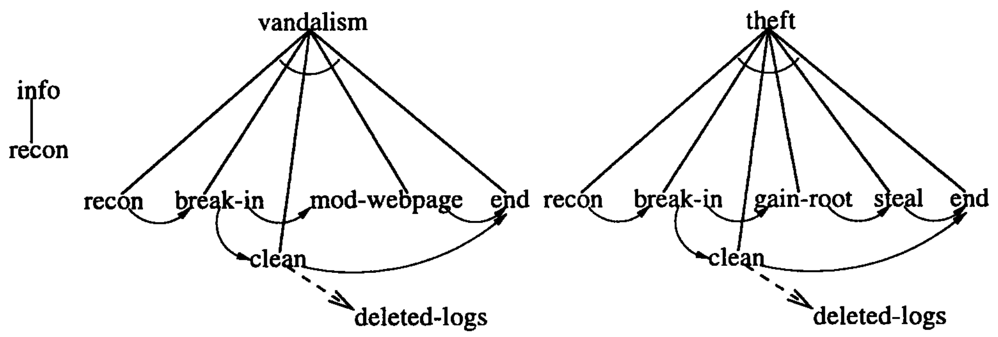

Figure 1: A hierarchical plan library in diagram form.

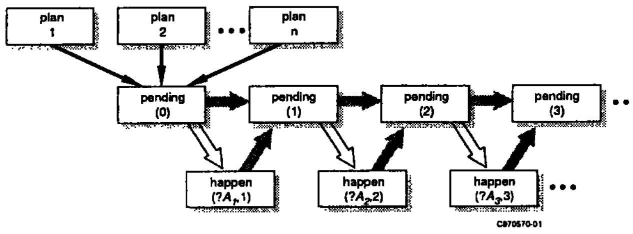

Figure 2: Generation of pending sets.

it could best be done as probabilistic (Bayesian) inference. Bayesian inference supports the preference for minimal explanations, in the case of hypotheses that are equally likely, but also correctly handles explanations of the same complexity but different likelihoods.

Recognition based on execution

The plan recognition framework developed in GG&M is based on the realization that plans are executed dynamically and that at any given moment the agent is able to choose to execute any of the actions that have been enabled by its previous actions. Thus, at any time an agent will have a *pending set* of actions that are enabled by its previous actions. The agent is free to choose to execute any of the actions in the current pending set.

To formalize this slightly, initially the executing agent has a set of goals and chooses a set of plans to execute to achieve these goals. The set of plans chosen determines the set of *pending* primitive actions. As the episode proceeds, the agent will repeatedly execute one of the pending actions, and generate a new set of pending actions from which further actions will be chosen.

The new pending set is generated from the previous set by removing the action just executed and adding newly enabled actions. Actions become enabled when their required predecessors are completed. This process is illustrated in Figure 2. To provide some intuition, the sequence of pending sets can be seen as a Markov chain, and the addition of the action executions with unobserved actions makes it a hidden Markov model.

To use this model to perform probabilistic plan *recognition*, we use the observations of the agent's actions as an execution trace. By stepping forward through the trace, and hypothesizing goals the agent may have, we can generate the agent's resulting pending sets. Once we have reached the end of the execution trace we will have the complete set of pending sets that are consistent with the observed actions and the sets of hypothesized goals that go with each of these sets. Once we have this set we establish a probability distribution over it. We can then determine which of the possible goals the agent is most likely pursuing.

Notice that the observations of the agent's actions are used to construct the execution traces. In the case of hostile agents, the observations will not, in general, be a complete record of the execution trace. Instead it will be necessary to consider execution traces containing unobserved actions.

This theory was designed to handle: partially ordered actions, overloaded actions, the effects of context, and negative evidence from not observing actions (i.e. the dog didn't bark). While some of these problems are partially handled by other systems, no other system handles all of them. We refer the reader to GG&M for a complete discussion of this formalism. We will now consider extending this formalism to the problems presented by hostile agents.

Infering unobserved actions

Existing work on plan recognition has assumed complete observability of the agents actions. The central idea behind this research is to remove this assumption. That is, we want to infer the goals of an agent given that the behavior of the agent is only partially observable. The rest of this paper will be organized as follows, first we will discuss two kinds of reasoning that we can do to infer the execution of unobserved actions: inferring unobserved actions from observed actions, and inferring unobserved actions from state changes. We then discuss our general algorithm for plan inference and we will conclude with a discussion of the assumptions and limitations of the algorithm.

Inferring unobserved actions from observed actions

Consider the following observations:

(gain-root,mod-webpage)

These two observations indicate with very high probability the hacker is engaged in both stealing information from a computer and defacing a webpage. We can conclude this because these actions are members of disjoint plans, that is, no single root goal will explain both of these actions.

However these actions are even more informative since they are both *unenabled* by the observed actions. We define an *unenabled action* is one that is observed without having first observed the actions the plan library specifies must come before it. In this case, the plan library specifies that **recon** and **break-in** must occur before **gain-root** or **mod-webpage**. Therefore, in order to explain these two observations we must assume the execution of at least one instance of **recon** and **break-in** each. Thus, these two actions provide evidence of two distinct plans:

(recon, break-in, mod-webpage)
and
(recon, break-in, gain-root)

Consider our model of plan recognition. Unenabled actions provide more information for us to use to reconstruct the agent's actual actions than other observations. They require that the action itself be in the sequence, but they also provide evidence of unobserved actions. Consider generating the execution traces needed to produce the pending sets for the last example. Not only does this set of observations allow us to prune out any execution sequence that doesn't contain a **gain-root**, followed sometime later by a **mod-webpage**, but it also allows us to ignore any trace that doesn't have a **recon** followed by a **break-in** preceding the **gain-root**. These unenabled actions are very important pieces of information when attempting to infer the plans of hostile agents.

Note that in this discussion, we have implicitly assumed the agent can perform any action without detection, however in practice this is not true. Some actions are simply harder to hide the execution of than others. For example, the probability that a person could conduct a port scan of my machine without my knowledge is much higher than the probability that they could successfully carry out a denial of service attack against it without my noticing. In this framework it is trivial to add probabilities about the likelihood of an agent performing a specific action undetected.

Inferring unobserved actions from state changes

Most existing work on plan recognition has ignored reports of the effects of actions. Simply put, reports of state changes are unnecessary and redundant when one has a complete log of the agent's actions. However, there are a large number of domains where the actions of the agent cannot be observed directly. Instead indirect observation is the only possible course. In these cases, reports of state changes, can provide evidence of the agent's unobserved actions.

Often, when it is possible to prevent an observer from seeing the performance of an action, it is **not** possible to prevent the observation of the action's effects. Consider weapons testing, while a nation might be able to covertly build a nuclear device, testing one is hard to hide. In our network security domain consider the **clean** action; the execution of the action might be hidden, but the deleting the log files is very visible.

Reports of state changes can provide evidence of unobserved actions that have the desired effect. From them we can infer that the action must occur before the report of the state change. Reports of state change can also simply provide confirming information about a previously observed action.

Consider the following sequence of observations:

(recon,break-in,deleted-logs)

The report of the deleted event logs implies an unobserved **clean** action. Further the ordering constraints in the plan library imply that it must fall between the execution of **break-in** and the report of **deleted-logs**.

However, if the sequence of observations were:

(recon, break-in, clean, deleted-logs)

The report would provide no extra information since it is consistent with the observed actions. Like acquiring evidence from unenabled actions these reports give more information about the execution traces that are consistent with the observation.

Gathering this information from state change reports requires a subtle assumption. We must assume there is only a single report for any given state change. Since this approach assumes that reports indicate changes in the state, multiple reports of the same state change will cause the system to believe that the given proposition has changed state more than once. In our network intrusion detection domain this assumption is not a problem. "Chattering" sensors that would produce multiple reports of the same change can be filtered.

The use of reports of state changes along with action observations is a significant contribution to plan recognition in its own right. We know of no other plan recognition systems that makes use of reports of state changes. However, it is particularly powerful when observing non-cooperative agents if an action's effects are harder to hide than the action's execution.

The solution

The central idea behind our original plan recognition algorithm is the production of a probability distribution over the set of all pending sets. This is generated using the observations as an execution trace of the agent's actions. Since each pending set represents the results of at least one execution trace, we generated the pending sets by stepping through observations. In the case of cooperative agents with complete and correct observations, this is sufficient.

However, as we have pointed out, in the case of hostile agents we face a problem with the execution traces. We can no longer assume that the observation stream is complete; it no longer represents the complete execution trace. Instead, for each set of observations we must construct the *set* of possible execution traces, inserting hypothesized unobserved actions to complete them.

For easy implementation we have assumed a bound on the number of unobserved actions. The next section discusses removing this assumption. Given a finite set of primitive actions, bounding the number of unobserved actions provides a limit on the length and number of execution traces that must be considered. In the worst case we only need to consider all execution traces whose length is equal to the maximum number of unobserved actions plus the number of observed actions.

This sounds like a very large search space, however we can prune this set of execution traces with the ordering constraints provided by the observations. We are only interested in execution traces consistent with the observations, therefore if a sequence does not contain all the observed actions or doesn't obey the ordering constraints imposed by the sequence or plan library it cannot generate one of the pending sets we are interested in and therefore can be filtered from consideration. The execution traces can also be filtered to be consistent with the unobserved actions that are implied by unenabled actions and observed state changes.

To summarize then, we handle hostile agents by extending the observed sequence of actions with hypothesized unobserved actions consistent with both the observed actions, observed state changes, and the plan graph to create a set of possible execution traces. Then we follow the plan recognition algorithm as before. We use the set of execution traces to construct the pending sets and then the probability distribution over the sets of hypotheses of goals and plans implicated by each of the traces and pending sets.

Example

The following example will illustrate this algorithm at a high level. Consider the following set of action and state change observations with a bound of three unobserved actions.

(break-in,deleted-logs)

Given these observations and the bound on unobservable actions, the algorithm (implemented in Poole's PHA (Poole 1993)) walks forward through the list of observations, adding unobserved actions as required to build a set of consistent execution traces. To explain the given observations requires the introduction of two unobserved actions, one to enable the action **break-in** and one to cause the state change reported in **deleted-logs**. The complete process results in 9 possible execution traces. The first:

(recon,break-in,clean,deleted-logs)

is consistent with the agent not having executed any further unobserved actions beyond those required to justify the observations. This trace is only consistent with the high level goals of **theft** or **vandalism**.

There are four traces that are consistent with the execution of a second unobserved **recon** action performed at various points in the sequence. These traces are not shown here, however they would be consistent with the goal of pursuing any two of the top level goals concurrently.

Of the four remaining traces:

(recon,break-in,gain-root, clean,deleted-logs)
and
(recon,break-in,clean,deleted-logs,gain-root)

are consistent only with the goal of **theft**. Note the ordering differences due to the partial ordering in the plan library. The final two execution traces:

(recon,break-in,mod-webpage,clean,deleted-logs)
and
(recon,break-in,clean,deleted-logs,mod-webpage)

are consistent only with the goal of **vandalism**. Again, note the ordering differences due to the partial ordering in the plan library.

In constructing this set of possible execution traces PHA has already established a probability distribution over the explanations and establishes the most likely goal. In this case, since the number of explanations for **theft** and **vandalism** are equal and there are no environmental factors that would weigh in favor of one over the other, these goals are equally likely. The conjunctive plans of **theft** or **vandalism** with **info** is a much less likely third alternative.

Assumptions

In our implementation of this algorithm we have made two assumptions about the observation stream: 1) There is a fixed and known upper bound on the number of unobserved actions, and 2) the given observations are true and correctly ordered. Neither of these assumptions is strictly necessary. We will consider each of them in turn.

Bounding the number of possible unobserved actions enables reasoning about where the agent could be in the execution of its plans. Suppose we bound the number of unobserved actions at two, and we observe a **break-in** action. This observation is not consistent with the agent having already executed **steal**. We have seen one action and the agent may have executed two more unobserved. The agent can have executed a total of three actions. Since, **steal** is the fourth step in its plan, the agent could not yet have executed it.

This bound can be removed from the algorithm in a number of ways including: running the algorithm multiple times with increasing bounds or replacing the bound with a probability distribution over the number of unobserved actions and weighing the execution traces accordingly. We see determining the best way to remove this limitation as an area for future work.

Second, we assumed that the observed actions happen and in the order indicated by the sequence. Thus if we have a sequence of three observations: **recon**, **break-in**, and **gain-root**, we know **recon** happened before **break-in** which happened before **gain-root**. The observation sequences are **not** assumed to be complete, therefore we can't conclude **clean** *didn't* happen between **break-in** and **gain-root** or even after **gain-root**. However, ordering constraints provided by the plan library allow us to rule out some possibilities. For example, the ordering constraints allow us conclude that if **clean** did occur unobserved it couldn't have occurred before the **break-in** unless there were an earlier *unobserved* **break-in**.

This assumption means we need not question the validity of observations. However, in environments with hostile agents, this assumption must be questioned. Consider a military example, if we receive a report of troops massing at a particular location, we must first determine the validity of the report before considering the effect this would have on our assessment of the enemy's goals. It is however straightforward to complicate the model by including a traditional model of noisy observations.

Conclusions

In this paper we have extended a probabilistic model of plan recognition to handle hostile agents. We have extended a model of plan recognition based on plan execution to enable the inference of unobserved actions on the basis of observations of changes to the world state and the execution of unenabled actions. These extensions remove a major assumption of previous research in plan recognition and significantly broadens the areas where plan recognition can be applied.

Acknowledgments

This material is based upon work supported by the DARPA Cyber Panel program and the Air Force Research Laboratory under Contract No. F30602-99-C-0177. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, the U.S. Government, or the Air Force Research Laboratory. Special thanks to David Poole for providing his Probabilistic Horn Abduction PHA interpreter and help and guidance on working with it.

References

Charniak, E., and Goldman, R. P. 1993. A Bayesian model of plan recognition. *Artificial Intelligence* 64(1):53–79.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task network planning. In Hammond, K. J., ed., *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference*, 249–254. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Goldman, R. P.; Geib, C. W.; and Miller, C. A. 1999. A new model of plan recognition. In *Proceedings of the 1999 Conference on Uncertainty in Artificial Intelligence.*

Kautz, H., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 32–38.

Poole, D. 1993. Logic programming, abduction and probability: a top-down anytime algorithm for stimating prior and posterior probabilities. *New Generation Computing* 11(3–4):377–400.