Partial Observability and Probabilistic Plan/Goal Recognition

Christopher W. Geib,
Honeywell Laboratories
3660 Technology Drive
Minneapolis, MN 55418
christopher.geib@honeywell.com

Robert P. Goldman,
SIFT, LLC
211 First Street, North
Minneapolis, MN 55401
rpgoldman@sift.info

Abstract

Partial observability of the domain prevents the application of many plan recognition algorithms to real world problems. This paper presents a treatment for partial observability for a specific probabilistic plan recognition algorithm and an initial analysis of the runtime impact.

1 Introduction

There has been significant recent work in plan recognition based on models of plan execution[Bui *et al.*, 2000; Goldman *et al.*, 1999; Pynadath and Wellman, 2000]. This research is motivated by an effort to make practical applications of this approach to a number of different domains, including computer network security, assistive systems for elders, and insider threat detection. All of these domains are only partially observable. Unfortunately, the large number of domain features, without significant modularity or probabilistic state transition models makes using an Abstract Markov Model(AMM)[Bui *et al.*, 2000] difficult. Methods based on probabilistic grammars[Pynadath and Wellman, 2000] do not handle concurrent and interleaved goals preventing their application. Other probabilistic plan recognition systems [Charniak and Goldman, 1993; Conati *et al.*, 1997] do not model plan execution, which is required for significant reasoning about missing observations.

In previous work [Geib and Goldman, 2001b],we have put forward a limited solution to the problem of partial observability that requires additional logical reasoning and places an *a priori* fixed bound on the number of unobserved actions that the system will consider. This earlier solution was somewhat *ad hoc*, and did not permit accurate modeling of situations where some actions are more easily observed than others. Further, our previous work spent significant effort to identify when other actions provided evidence of previously unobserved actions. In this paper, we present an alternative probabilistic formulation that avoids such reasoning and eliminates the limitations of our previous solution.

In the rest of this paper we will briefly review our approach to probabilistic plan recognition, and then describe how it can be extended to handle partially observable domains. Our solution adds significant computational costs for the algorithm, and we will provide some experimental data showing this in our implementation. After this we will discuss the theoretical implications for the runtime, and discuss how the effects of the computational cost can be ameliorated.

2 Background

The Probabilistic Hostile Agent Task Tracker(PHATT) [Goldman *et al.*, 1999; Geib and Goldman, 2001a; 2001b] is based on a model of the execution of simple hierarchical plans [Erol *et al.*, 1994] rather than plans as formal models. Figure 1 displays examples of a hierarchical plan of the kind that PHATT assumes. In this discussion, the plans in the library will be represented as partially ordered and/or trees: in Figure 1, "and nodes" are represented by an undirected arc across the lines connecting the parent node to its children; "or nodes" do not have this arc. Ordering constraints in the plans are represented by directed arcs between the actions that are ordered. For example in Figure 1 action *scan* must be executed before *get-ctrl* before *get-data*. Following standard use, we refer to the children of OR-nodes as "methods," and an agent's choice between these alternatives as "method choice." The leaf nodes in the tree are referred to as "primitive actions."

2.1 Algorithmic Intuition

The central realization of the PHATT approach is that plans are executed dynamically and that at any given moment the agent is able to execute any one of the actions in its plans that have been enabled by its previous actions. To formalize this, initially an agent chooses a set of goals. On the basis of these goals that agent chooses a set of plans to execute to achieve these goals. The set of plans chosen determines a set of pending primitive actions. That is, the set of actions within the plan that are enabled by some other action within the plans. Each initial pending set represents the possible first actions the agent could perform that would contribute to the plan(s) it has chosen to meet its goal(s).

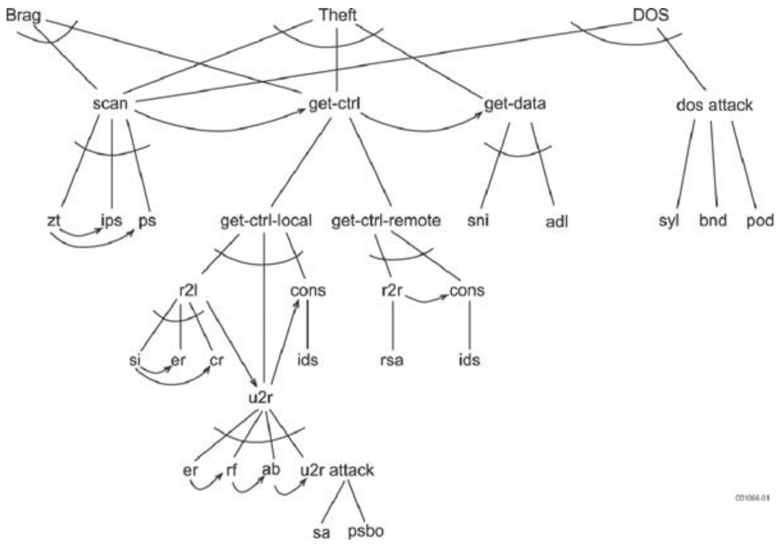


Figure 1: An example plan library.

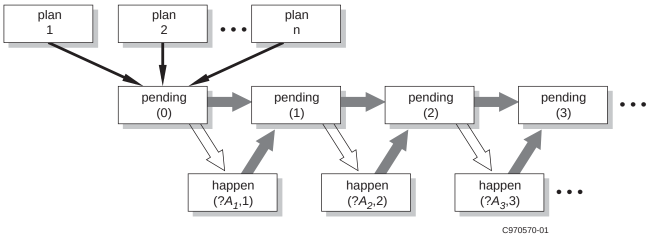


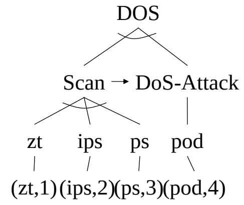
Figure 2: A simple model of plan execution.

The agent executes one of the pending actions, generating a new set of pending actions. In most cases the executed action itself is removed from the pending set [1] and actions that are enabled by the executed action are added to the pending set. The next action the agent executes will be chosen from this new pending set and the process repeats until the agent stops performing actions or finishes all of its plans. This process is illustrated in Figure 2. Each state is conditionally independent of the history, given its immediate predecessor, so we have a Markov chain. Since the pending sets are not observable, this Markov chain is a Hidden Markov Model (HMM).

To use this model to perform probabilistic plan recognition, PHATT treats the observed actions as an execution trace and hypothesizes goals and plan structures to explain the observed actions. Given a set of hypothesized goals the agent may have, PHATT can generate the agent's pending sets. When PHATT reaches the end of the set of observations it 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 such explanation. By building the complete set of explanations for the observations, PHATT can establish the probability of each explanation, and on the basis of this, the conditional probability of any particular goal.

More formally, we define an *explanation* as a minimal forest of plan trees with pending sets recorded for each time-step, and expansions chosen for each AND-node and OR-node node sufficient to allow the assignment of each observation to a specific primitive action in the plans. Note that for each possible explanation, there is a unique corresponding sequence of pending sets. An explanation is *not* simply a set of goals; it also includes method choices and ordering decisions and the pending sets. As a result there may be multiple possible explanations for any single set of goals.

As an example, given the plan library shown in Figure 1, one possible explanation for the sequence of observations **(zt,1), (ips,2), (ps,3), (pod,4)**, is shown in Figure 3. Note the use of an integer time notation in the



PS(0) = {(zt, DOS), (zt, Theft), (zt, Brag)}
PS(1) = PS(0) + {(ips, Scan), (ps, Scan)}
PS(2) = PS(0) + {(ps, Scan)}
PS(3) = PS(0) + {(syl, DoS-Attack),
(bnd, DoS-Attack), (pod, DoS-Attack)}

Figure 3: An explanation.

[1] Actions that can be repeatedly executed are left in the pending set.

observations.

In the pending sets, each pair refers to the use of a particular primitive action to contribute to the achievement of a specific method that will be added to or extends the existing plan. For example the pair **(zt, DOS)** in the initial pending set (PS(0)) represents the possibility of an observed zone transfer action, **zt**, contributing to a plan for a denial of services attack, **DOS**, while **(ips, Scan)** in PS(1) represents the possibility of an ip-sweep, **ips**, contributing to the **Scan** method in the previously hypothesized **DOS** plan explaining **(zt, 1)**.

Thus, PHATT first builds a complete and covering set of explanations for the set of observations, then it establishes the probability of each explanation. On the basis of the probability of each explanation it then computes the conditional probability that the agent is pursuing any particular goal. While the worst case cost of computing the covering set of explanations can be prohibitive, our experience suggests this is not a problem in practice. We refer the reader to Geib [2004] for a discussion of the complexity of this process.

2.2 Computing Probabilities

PHATT takes a Bayesian approach to plan recognition. That is, it computes $Pr(exp|obs)$, the probability of an explanation *exp* given a set of observations, *obs*, using Bayes' rule:

$$Pr(exp \wedge obs)/Pr(obs)$$

The implementation, like most Bayesian systems, exploits the equivalent formulation:

$$Pr(exp \wedge obs)/\sum_i Pr(exp_i \wedge obs)$$

PHATT computes the joint probability of a particular explanation and observation set, $Pr(obs|exp)Pr(exp)$, by factoring it into three kinds of probabilistic features. First, the prior probability of each of the root goals, G_i, being adopted by the actor, $Pr(G_i)$. In PHATT, the set of root goals and prior probabilities is given with the plan library.

Second, for each choice action (OR-node), PHATT must have a probability of method choice given parent node. For example if a cyber attacker could use a land, synflood, or POD for a denial of service (DoS) attack, PHATT must have a distribution over how likely each of these possible attacks are given that the agent is going to commit a DoS attack. Typical PHATT models assume that each method is equally likely given its parent, but this simplifying assumption is not required by the framework. Given this assumption, if the number of children for OR-node j is $|Ch_j|$, we have $1/|Ch_j|$.

Taken together, the above two factors are sufficient to describe a probability distribution over subtrees of the plan library. However, they are not enough to describe an *ordered* trace of primitive action executions. For this, we need to order the action executions, subject to the constraints imposed by the plan library. The third factor takes care of this: for each pending set, PHATT computes the probability that the observed action is the one chosen next for execution. Again, we usually assume that all members of a pending set, $PS(k)$ are equally likely to be chosen, giving us $1/|PS(k)|$.

On the basis of these three classes of parameters, PHATT computes the joint probability of a given explanation and the observation trace. It does so by multiplying together the priors for each goal, the probability of the expansion choices, and the probability of the observed actions being chosen. More formally:

Definition 1 $Pr(exp, obs) =$

$$\prod_{i=0}^{I} Pr(G_i) \prod_{j=0}^{J}(1/|Ch_j|) \prod_{k=0}^{K}(1/|PS(k)|)$$

where the first term is the probability of the agent's goal set, the second term the probability that the agent chose the particular plans to achieve the goals, and the final term captures the probability that the given observations occurred in the specified order.

Since, the set of explanations is, by construction, an exclusive and exhaustive covering of the set of observations, these computed probabilities for each explanation can be normalized to yield conditional probabilities for each explanation given the observations. To find the probability that an agent has a particular goal, G_i, given a set of observations, we sum the probabilities of all explanations that contain G_i, which we notate as the set Exp_{G_i} That is:

Definition 2 $Pr(G_i|Obs) =$

$$\sum_{e}^{Exp_{G_i}} Pr(e, Obs)/\sum_{e}^{Exp} Pr(e, Obs)$$

where the denominator sums the probability of all explanations for the observations, and the numerator sums the probability of the explanations in which the goal G_i occurs. Further conditional queries can be computed by summing the probability mass for those explanations that meet the terms of the query and dividing by the total probability mass associated with all the explanations for the observations.

A common misconception is that the denominator in Definition 2 will always be one, making this division unnecessary. Recall, however, that this is an application of Bayes' law, so that the denominator is also $Pr(obs)$: the probability of the observation set. It is a rare (and uninteresting) domain that admits a single observation trace of probability one!

With this model of plan recognition couched in terms of plan execution, PHATT is able to handle a number of problems that are critical to application to real world domains including: Multiple interleaved goals, partially ordered plans, the effects of context on the goals adopted, the effect of negative evidence or failure to observe ("the dog didn't bark in the nighttime"), missing observations, and observations of failed actions. We refer readers to [Geib and Goldman, 2001b; 2001a; Goldman *et al.*, 1999] for a more complete discussion of these issues.

3 Handling Partial Observability

We can build a conceptually simple treatment of partial observability on this approach to plan recognition. To do this, first, the probability that the observed agent has actually performed some action, but it was not observed, is quantified. That is, the probability that there is *no* observation, given a particular action: $Pr(\neg obs(a)|a)$. Second, we fold this probability into the formula for computing the probability of a given explanation. Third, because the resulting set of explanations is infinite, we modify the algorithm for generating explanations to eliminate any explanation in which the probability of failure to observe exceeds a user defined threshold. We will discuss each of these steps in more detail.

3.1 Quantifying the Probability of Not Observing

We are rarely equally likely to observe all actions in a domain. For example, the US Government is much more likely to be aware of underground testing of a nuclear device than it is to recognize a very slow port scan on one of their computers. Relative to the sensors available, some actions are harder to detect than others. That is, we are concerned with the *false negative rate* for a given sensor. This feature of the sensor can be learned by watching its long term performance and noting when it fails to report if there has been an actual occurrence. We assume that for each possible observed action, we will learn the long term false negative rate of our observation stream and use this as the prior probability that the action might not be observed when it is performed, $Pr(\neg obs(a)|a)$, or simply $Pr(Unobs_{\mathrm{a}})$.

3.2 Computing Probabilities

Having an explicit model of the probability that a particular action could be performed and not observed makes computing the probability of a given explanation and observations relatively easy. The following new definition for the probability of an explanation and observations extends Definition 1 with a final term to explicitly model the probability that all unobserved actions in the explanation are executed and not observed.

Definition 3 $Pr(exp, obs) =$

$$\prod_{i=0}^{I} Pr(G_i) \prod_{i=0}^{J} (1/|Ch_j|) \prod_{k=0}^{K} (1/|PS(k)|) \prod_{l=0}^{L} (Pr(Unobs_l))$$

Where the index L ranges over all of the actions in the explanation that are hypothesized as having been executed but not observed. There is no change in the formula for computing the conditional probability for a specific goal.

3.3 Changing the algorithm

Extending the definition of the probability of an explanation to cover unobserved actions is not enough. We must also address the question of how to generate explanations that make use of unobserved actions.

Section 2 describes an algorithm for building explanations that incrementally processes each of the observations to advance PHATT's pending set. In the work described here, we extended the algorithm for generating explanations to explicitly consider the possibility that some actions that not present in the observation stream were actually executed. Note that, given the assumption that the agent will not engage in actions that do not contribute to its goals, even unobserved actions must be within the pending set of the observed agent. This significantly constrains the set of actions that can be executed even unobserved. Thus, the following algorithm will not consider the possibility of unobserved actions that are inconsistent with the current pending set of the agent.

To produce a complete and covering set of explanations for the observations, explanation generation must be extended to consider the possibility that every action in the pending set may be executed next and not observed. This results in a two stage explanation generation algorithm. First, the algorithm considers the possibility that each of the actions in the pending set is performed and not observed. Second, following the original algorithm, it generates those explanations in which no unobserved action was performed but instead the next action in the observation stream was performed. This results in a significant expansion of the search space.

3.4 Thresholding the Production of Explanations

Without further mechanisms, the algorithm provided above will not terminate building explanations. While the original algorithm naturally terminates at the end of the observation stream, nothing in the algorithm specification we have provided states when the system must stop considering the possibility of ever more unlikely explanations with streams of unobserved actions.

To address this we require the user to provide a threshold value that determines how unlikely an explanation they are willing to accept, relative to unobserved actions. Specifically, the algorithm, removes any explanation where:

$$\prod_{l=0}^{L} Pr(Unobs_l) < Threshold$$

This allows the user to specify, for example, that they are only willing to accept explanations, such that ninety percent of the time the unobserved actions could have been done and not noticed.

Since the algorithm is generating explanations incrementally, thresholding the acceptable explanations provides a natural method for limiting the search for explanations, and provides a stopping criteria for the algorithm. At some point the addition of another unobserved action will drive the current explanation below the threshold. At this point we prevent the addition of this unobserved action and stop considering the addition of other unobserved actions. If all of the observations have been explained this explanation can be kept.

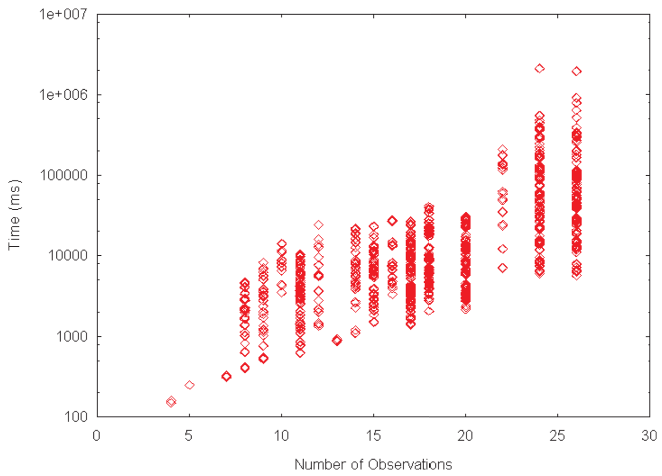


Figure 4: runtimes for Fully Observable Domain

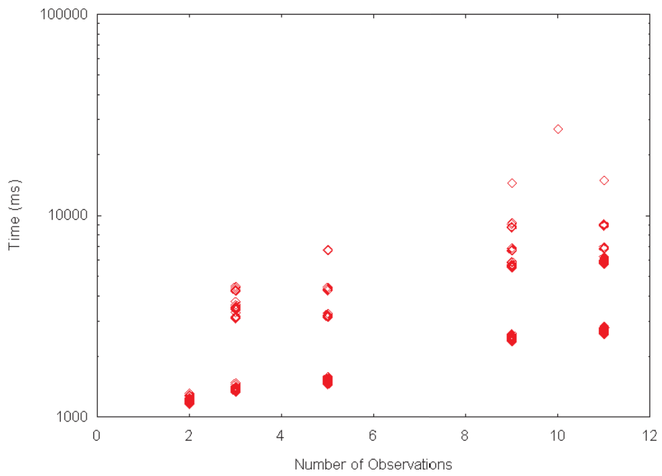


Figure 5: runtimes for partially Observable Domain

If there are still remaining observations the algorithm must attempt to explain the remainder without considering unobserved actions.

Notice that this approach allows the system to differentially consider actions on the basis of their false negative rates. That is, at the same threshold value, the system will consider explanations with more instances of actions that are more likely to have been performed and not observed than actions that are not likely to have been performed and not observed. For example, the system would more easily consider an explanation with multiple stealthy port scans (low probability of observation) than an explanation with even a single unobserved successful denial of service on one of its critical DNS servers (high probability of observation).

4 Implementation

To test this approach we have extended the PHATT algorithm with the changes described in the previous sections. All of the empirical results we will report were generated with our implementation of PHATT written in Allegro Common Lisp 6.2 on a 700MHz Pentium III.

4.1 Runtime Cost

To explore the runtime costs of this algorithm we chose two instances of plans from the plan library shown in Figure 1. These plan instances are interleaved, preserving the partial ordering constraints of each of the plans. We ran fifteen hundred such plan instances without considering the possibility of unobserved actions (i.e. the Threshold was set to 1.0) the majority of the plans had a runtime of under a second. The actual runtimes are shown in Figure 4.

Unfortunately, considering even the very most likely unobserved actions has a significant impact on the algorithm's runtime. By lowering the threshold to 0.75 all of the instances' runtimes were forced above a single second and all of the examples longer than twelve observations were in excess of 100 seconds. The actual runtimes are shown in Figure 5, with those test cases longer than 100 seconds left off the graph.

In all of these test cases the system was very accurate, correctly identifying the component plans with probability greater than .75. However it should be noted that the purpose of these tests was to examine the runtime requirements for the system. While we do not anticipate a significant increase in the runtime with more ambiguous plans, it should be noted that the plans in these test cases do not have a high degree of ambiguity. We are working on producing a thorough and complete evaluation of the sources ambiguity and its effects for this algorithm.

4.2 Threshold Effects

Figure 5 makes clear the cost of considering partial observability in domains. The significant increase in runtime is not really a surprise. By considering the possibility of unobserved actions, the algorithm we have described moves from considering only those elements in the pending set that are consistent with the next observed action to having to consider all of the elements in the pending set. Thus the search space of possible explanations has significantly increased.

While the user defined threshold clearly has an effect on the size of the explanation search space, there is not a simple relation between them. The branching factor of the search space at each explanation critically depends on how many and what kind of unobserved actions have already been hypothesized.

To consider the worst case, Geib [2004] has shown that even in completely observable domains, the branching factor of the search space of explanations for this sort of algorithm, is exponential for plans that share a set of initial actions that are unordered with respect to each other. Consider such a case in a partially observable domain where the user defined threshold is sufficient to allow the initial actions to be performed unobserved. There will be an exponential increase in the size of the search space (caused by the initial unordered actions) until the threshold allows the algorithm to filter more unobserved actions.

Conversely, in the best case, very high threshold values will all but prevent the algorithm from considering unobserved actions and thereby adding to the algorithm's runtime. Thus setting the threshold parameter is critical to achieving reasonable runtimes and mitigating the effects of the increased search space.

4.3 Setting the Threshold

While we cannot provide a formula to compute an optimal threshold value for a domain, experience has shown us three rules that can help in setting the threshold.

1. The base line PHATT algorithm has some variability in its runtime. To get a feeling for this, if it makes sense for the domain, we suggest running the new algorithm the first time with a threshold high enough to prevent the considering of unobserved actions. This will not only provide a base line for understanding the runtime of the domain, but will also provide some understanding of the plans that may be identified without partial observability.
2. Offline analysis of the plan library can reveal helpful settings for the threshold. Consider the case of a plan that is known to have an action whose execution is not observed ninety percent of the time. To even consider this plan the threshold must be set below ninety percent. This can be generalized for sequences of unobserved actions the user is interested in making sure are considered. Thus if the user wants to be sure to consider the possibility of a particular sequences of actions being performed and not seen, the probability of this can be computed and used to set the threshold.
3. In a similar way, sets of unobserved actions the user would like to rule out can be used to set a lower bound for the threshold. If for domain reasons a particular set of actions is deemed too unlikely to occur and not notice, the probability of this can be computed and used to set a lower bound.

While these are only rules of thumb, using them has proved highly effective in identifying reasonable threshold values for the simulated domains on which we have tested the system.

5 Conclusions

This paper has presented a complete, theoretically sound, formal and tested algorithm for handling plan recognition in partially observable domains. We have shown, that dealing with partially observable domains does significantly increase the runtime of the algorithm, and for some domains, some threshold values will be computationally prohibitive. However, our experience suggests that for some domains this algorithm will be a viable solution method, and that the key to this lies in the structure of the domain and setting the threshold for unobserved actions appropriately. Previous work on this algorithm [Geib, 2004] suggests that the complexity of the algorithm is not as tightly coupled to the size of the state space as work on AMMs[Bui *et al.*, 2000]. Therefore, we are continuing this work, by attempting to specify the structure of the domains for which this represents an efficient solution method. We see this work as an effective alternative to other approaches that may scale in a different manner making it applicable for some domains.

References

[Bui *et al.*, 2000] Hung H. Bui, Svetha Venkatesh, and Geoff West. Policy recognition in the abstract hidden markov model. In *Technical Report 4/2000 School of Computer Science, Curtin University of Technology*, 2000.

[Charniak and Goldman, 1993] Eugene Charniak and Robert Goldman. A bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.

[Conati *et al.*, 1997] Cristina Conati, Abigail S. Gertner, Kurt VanLehn, and Marek J. Druzdzel. On-line student modeling for coached problem solving using bayesian networks. In *Proceedings of the Sixth International Conference on User Modeling*, 1997.

[Erol *et al.*, 1994] Kutluhan Erol, James Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task network planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS 94)*, pages 249–254, 1994.

[Geib and Goldman, 2001a] Christopher W. Geib and Robert P. Goldman. Partial observability in collaborative task tracking. In *Proceedings of the AAAI 2001 Fall Symposium on Collaborative task Tracking*, 2001.

[Geib and Goldman, 2001b] Christopher W. Geib and Robert P. Goldman. Probabilistic plan recognition for hostile agents. In *Proceedings of the FLAIRS 2001 Conference*, 2001.

[Geib, 2004] Christopher W. Geib. Assessing the complexity of plan recognition. In *Proceedings of the AAAI, 2004*, 2004.

[Goldman *et al.*, 1999] Robert P. Goldman, Christopher W. Geib, and Christopher A. Miller. A new model of plan recognition. In *Proceedings of the 1999 Conference on Uncertainty in Artificial Intelligence*, 1999.

[Pynadath and Wellman, 2000] David Pynadath and Michael Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-'00)*, pages 507–514, 2000.