A Probabilistic Plan Recognition Algorithm Based on Plan Tree Grammars

Christopher W. Geib[1] and Robert P. Goldman[2]

[1]University of Edinburgh, School of Informatics 2 Buccleuch Place, Edinburgh, EH8 9LW, United Kingdom, cgeib@inf.ed.ac.uk
[2]SIFT LLC, 211 N. First St. Suite 300, Minneapolis, MN 55401, rpgoldman@sift.info

February 10, 2009

Abstract

We present the PHATT algorithm for plan recognition. Unlike previous approaches to plan recognition, PHATT is based on a model of plan execution. We show that this clarifies several difficult issues in plan recognition including the execution of multiple interleaved root goals, partially ordered plans, and failing to observe actions. We present the PHATT algorithm's theoretical basis, and an implementation based on tree structures. We also investigate the algorithm's complexity, both analytically and empirically. Finally we present PHATT's integrated constraint reasoning for parametrized actions and temporal constraints.

1 Introduction

There is an increasing need for automated systems that understand the goals and plans of their human users. Applications that need such understanding include everything from assistive systems for the elderly, to computer network security, to insider threat detection, to agent based systems. As we develop such systems, we find that much of the early work on plan recognition made simplifying assumptions that are too restrictive for effective application in these domains. Some such simplifying assumptions include:

- The observed agent is only pursuing a single plan at a time.
- The observed agent's plans are totally ordered.
- *Failing* to observe an action means it will never be observed *or* the observer will see an arbitrary subset of the actual actions.
- The actions within a plan have no explicit temporal relations.
- The plan representation is purely propositional. That is, actions do not have parameters.[1]

While some of these limitations have been addressed individually, our new plan recognition system, PHATT, is the first system to provide a solution to all of these issues within a single framework. PHATT takes a very different approach to the problem of intent inference[2] from other systems, and it is this approach that allows it to address all of these issues at the same time.

Most, if not all, early work on plan recognition treated plans as patterns to be matched against data, rather than as recipes for actions to be executed. Unlike such approaches, PHATT is based on a model of plan *execution*, which more simply and elegantly captures key aspects of the plan recognition problem. The critical observation behind this

[1]For example, one may have an action that goes from home to the train station, and an action that goes from the train station to home, but not an action that moves between two arbitrary locations.

[2]Plan recognition is sometimes also referred to as "task tracking," "intent recognition" or "intent inference." We will use these terms interchangeably

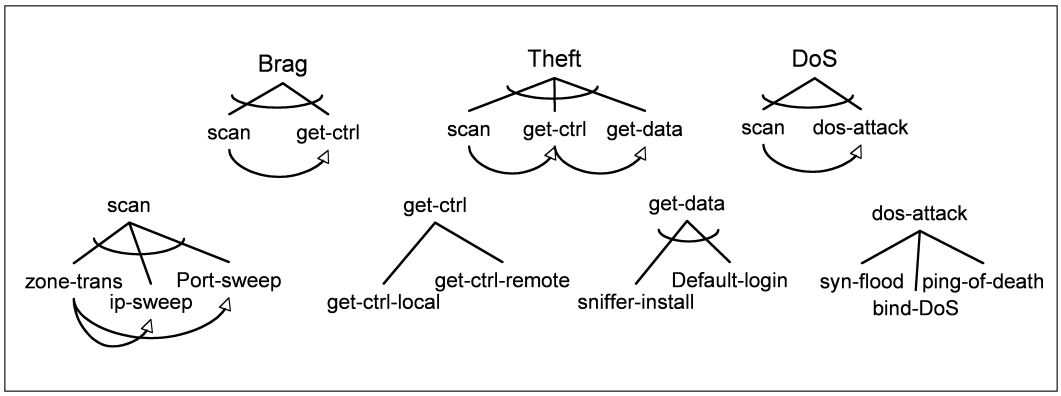

Figure 1: An example set of plans: In this figure, AND-nodes are represented by an undirected arc across the lines connecting the parent node to its children. OR-nodes do not have this arc. Ordering constraints in the plans are represented by directed arcs between the ordered actions. For example action **scan** must be executed before **get-ctrl** which must be executed before **get-data** to perform **Theft**.

approach is that goal driven agents will take those actions that are consistent with their goals and are enabled by the actions they have already taken. We call the set of actions that the agent could execute next, given their goals and the actions they have already performed, the *pending set*. Putting the execution of plans and pending sets at the center of our plan recognition model, we can build a stochastic model of plan execution and from it develop a probabilistic algorithm for recognizing plans.

Like much of the prior work we will be assuming that the agents being observed are not actively deceitful. Deceitful agents might attempt to dissemble, misdirect, or otherwise take actions to deliberately confuse an observing agent rather than directly to achieve goals. We will not be discussing how to address these issues here.

The rest of this paper follows the following structure. We first present an example domain and a short review of prior work, with particular attention to limitations of this previous work that PHATT addresses (Sections 2 and 3). Section 4 describes the intuitions behind the PHATT algorithm followed by the theoretical core of the paper in Sections 5 to 8. Section 5 formalizes plan libraries in terms of leftmost plan trees. Section 6 provides an abstract, top-down algorithm that closely parallels the generative model, providing a smooth transition to the probability model, and a first step to the actual implementation. Section 7 provides a formal probability model for use with the explanations produced using the plan trees. Section 8 gives a bottom-up algorithm that approximates the top-down algorithm and then discusses its implementation and limitations.

The rest of the paper covers evaluating the PHATT algorithm and extensions. Section 9 discusses the algorithm's formal complexity, while Section 10 covers empirical complexity results and some studies of the algorithm's scalability. Section 11 explains PHATT's use of variables and temporal constraints to improve its efficiency. Finally, Section 12 concludes this paper with a discussion of topics that we think are particularly interesting areas for future work.

2 Background

Most plan recognition algorithms require as input a plan library which implicitly specifies the set of plans that are to be recognized. PHATT [20, 22, 25] is based on a model of the execution of simple hierarchical plans [16]. In this framework, plan libraries are partially ordered AND/OR trees. AND-nodes represent *methods* for achieving a particular task: all of the children of an AND-node must be performed in order to perform the parent task. The children may be further constrained to be performed in a particular order (or one of a set of possible orders), by annotating them with pairwise ordering constraints.

As an example, Figure 1 displays a small example plan library taken from a computer network security domain. In this case, the attacker is motivated by one of three top-level AND-node goals: bragging (**Brag**) (being able to boast of his/her success to other crackers); theft of information (**Theft**); or denial of service (**DoS**) (attacking a network by consuming its resources so that it can no longer serve its legitimate objectives). Attackers who wish to achieve

bragging rights will first scan the target network for vulnerabilities (**scan**), and then attempt to gain control (**get-ctrl**). They are not motivated by exploiting the control they gain. On the other hand, attackers who wish to steal information will scan for vulnerabilities, get control of the target, and then exploit that control to steal data (**get-data**). Finally, an attacker who wishes to DoS the target need only scan to identify a vulnerability, and then carry out his DoS attack (**dos-attack**).

OR-nodes in the plan library represent places where the agent may choose one of a number of alternate methods to achieve a task. Only one of the children of an OR-node need be performed in order for the parent action to be achieved. For this reason, ordering constraints between the children of an OR-node are not allowed. For example, in Figure 1 the OR-node **dos-attack** has three possible children: synflood (**syn-flood**), bind DoS attack (**bind-DoS**), and the ping of Death (**ping-of-death**), but only one of them must be executed to perform a **dos-attack**. Since OR-nodes represent choices within the plan we will also refer to them as *choice points* for the plan, and will use these two terms interchangeably.

Our representation of plans as partially ordered AND/OR trees is similar to the Hierarchical Task Network (HTN) representation in Ghallab, Nau, and Traverso's recent textbook[24, p. 244–245], but does not take into account action preconditions and postconditions. Note that our example plan library, while displaying a variety of the phenomena that we will be interested in discussing, is not a full, up-to-date, or even realistic plan library for this computer security domain. This library simply illustrates the use of method decomposition (represented by AND-nodes), choice points (represented by OR-nodes), and ordering constraints between sibling actions. We will use this plan library as a running example throughout this article.

3 Previous Work in Plan Recognition

Attempts to perform plan recognition are almost as old as artificial intelligence itself, and over the years a large number of methods have been applied to plan recognition. Some of the methods used include rule-based systems, parsing (both conventional and stochastic), graph-covering, Bayesian nets, cost-based abduction, etc. Early approaches paid little attention to choosing between different explanatory hypotheses. Either the problem was not isolated as a separate problem of particular interest (as in early rule-based approaches), or it was finessed (as in graph-covering approaches). Our Bayesian approach addresses this issue directly (as did earlier Bayesian approaches). Many of the approaches achieved computational efficiency by limiting the expressiveness of their plans, particularly by imposing rigid assumptions about the type and number of the plans, or the ordering of their steps. Few of these approaches were able to take into account evidence from *failure* to observe actions; however this is critical for some domains[43].

Cohen, Perrault and Allen [12] distinguish between two kinds of plan recognition, *intended* and *keyhole* plan recognition. In intended recognition, the agent is cooperative; its actions are done with the intent that they be understood. For example, a tutor demonstrating a procedure to a trainee would provide a case of intended recognition. In keyhole recognition, the recognizer is simply watching normal actions by an ambivalent agent. These cases arise, for example, in systems that are intended to watch some human user imperceptibly, and offer assistance, appropriate to context, when possible.

The earliest work in plan recognition [49, 55] was rule-based; researchers attempted to come up with inference rules that would capture the nature of plan recognition. However without an underlying formal model these rule sets are difficult to maintain and do not scale well.

In 1986, Kautz and Allen (K&A) published an article, "Generalized Plan Recognition," [36] that framed much of the work in plan recognition to date. K&A defined the problem of keyhole plan recognition as the problem of identifying a minimal set of *top-level actions* sufficient to explain the set of observed actions. Plans were represented in a plan graph, with top-level actions as root nodes and expansions of these actions into unordered sets of child actions representing plan decomposition. To a first approximation, the problem of plan recognition was then a problem of graph covering. K&A formalized this view of plan recognition in terms of McCarthy's circumscription [41]. Kautz presented an approximate implementation of this approach that recast the problem as one of computing vertex covers of the plan graph [35]. This method is quite efficient, but exploits for its efficiency the assumption that the observed agent is only attempting one top-level goal at a given time. Furthermore, it does not take into account differences in the *a priori* likelihood of different goals. Observing an agent going to the airport, this algorithm views "air travel," and "terrorist attack" as equally likely explanations, since they explain (cover) the observations equally well.

To the best of our knowledge, Charniak was the first to argue that plan recognition was best understood as a specific case of the general problem of *abduction*, or reasoning to the best explanation [8]. Charniak and Goldman

(C&G) [10] argued that, viewing plan recognition as abduction, it could best be done as Bayesian (probabilistic) inference. Bayesian inference supports the preference for minimal explanations, in the case of equally likely hypotheses, but also correctly handles explanations of the same complexity but different likelihoods. For example, if a set of observations could be equally well explained by two hypotheses, theft and bragging being one, and theft alone being the other, simple probability theory (with some minor assumptions), will tell us that the simpler hypothesis is the more likely one. On the other hand, if as above, the two hypotheses were "air travel" and "terrorist attack," and each explained the observations equally well, then the prior probabilities will dominate, and air travel will be seen to be the most likely explanation. C&G used knowledge-based model construction (KBMC) [54] to build Bayesian networks expressing particular plan recognition (story understanding) problems, and then solved those networks for the posterior probability of explanations.

Previous systems did not handle failures to observe actions well. Some systems assumed that the set of actions observed was complete, thus the failure to observe an action required to achieve some objective, G, was sufficient reason to conclude that the agent was not trying to achieve objective G (this is effectively an application of the closed world assumption). Other systems simply treated the set of observations as an arbitrary subset of the set of actions actually executed. For C&G, this followed from their focus on plan recognition as part of story understanding[9]. In human communication, stories are radically compressed by omitting steps that the reader or hearer can infer based on explicitly-mentioned material and background knowledge. For example, reading the (not very interesting) story "Jack went to the supermarket. He paid for his groceries, and went home," the reader will assume the occurrence of several rather complicated steps in the plan for shopping. The reader will *not* assume that for some reason Jack did not locate the desired groceries and pick them up off the store shelf. In such cases, the plan recognizer must assume that it is "observing" only some subset of the actually-occurring events.

Observers in other situations often know that some actions have *not* been carried out and can make use of this knowledge. Consider the plan library in Figure 1. What would happen if one observed actions consistent with **scan** and **get-ctrl**? Assuming their a priori probabilities are the same, a plan recognition system should conclude that **Brag** or **Theft** are equally good explanations. However, as time goes by, if the system sees other actions without seeing actions that contribute to **get-data**, the system should become more and more certain that **Brag** is the right explanation and not **Theft**, because if **Theft** were the right explanation, sooner or later we would have seen some of the additional actions (those done in service of **get-data**).

Systems like those of C&G and K&A are not capable of reasoning like this, because they do not start from a model of plan execution over time. As a result, they cannot represent the fact that an action has not been observed *yet*. In general such systems take one of two solutions. First they can assert that the action has not and *will not* occur, or second they can be silent about whether an action has occurred — implying that the system has failed to notice the action, not that the action hasn't occurred. Neither of these solutions is very satisfying.

Parsing-based approaches to plan recognition promise greater efficiency than other approaches, but at the cost of making strong assumptions about the ordering of plan steps. Vilain [53] presented a theory of plan recognition as parsing, based on K&A's theory.[3] Vilain does not actually propose parsing as a solution to the plan recognition problem. Instead, he reduces limited cases of plan recognition to parsing in order to investigate the complexity of K&A's theory. The major problem with parsing as a model of plan recognition is that it does not treat partially-ordered plans or interleaved plans well. Both partial ordering and interleaving of plans result in an exponential increase in the size of the required grammar, issues which we have addressed in implementing PHATT. We will discuss the relationship of plan recognition to parsing further in Section 9.

More recently, Pynadath and Wellman (P&W) have proposed a plan recognition method that is both probabilistic and based on parsing. They represent plan libraries as probabilistic context-free grammars (PCFGs) and extract Bayes networks from the PCFGs to interpret observation sequences. Unfortunately, this approach suffers from the same limitations on plan interleaving as Vilain's. P&W also propose that probabilistic context-*sensitive* grammars (PCSGs) might overcome this problem, but it is difficult to define a probability distribution for a PCSG [47]. We will return later in the paper to discuss other differences between our probability and that of P&W.

There has been a large amount of very promising work done using Hierarchical Hidden Markov Models (HHMMs)[6], Conditional Random Fields (CRF) [38, 39, 52] and related approaches [27, 42]. These approaches offer many of the efficiency advantages of parsing approaches, but with the additional advantages of incorporating likelihood information and of supporting machine learning to automatically acquire their plan models. The first work that we know of in this area was provided by Bui[6] who has proposed a model of plan recognition based on a variant of Hidden Markov

[3]This was not the first attempt to cast plan recognition as parsing [51].

Models (HMMs). A similar HMM serves as a foundation for this work, and while Bui's work is based on a model of plan execution it does not address the case of multiple goals.

The work using CRFs and similar approaches under the title of *activity recognition* [27, 38, 40, 39, 42, 52] is very promising, but should be recognized as solving a different problem from the one addressed here. The early work in this area very carefully chose the term *activity* or *behavior recognition* to distinguish it from plan recognition. The distinction to be made between activity recognition and plan recognition is the difference between recognizing a single (possibly complex) activity and recognizing the relationships between a set of such activities that result in a complete plan. Much of the work on activity recognition can be seen as discretizing a sequence of possibly noisy and intermittent low-level sensor readings into coherent actions that could be treated as inputs to a plan recognition system.

Much of the work on activity recognition defines the problem as one of labeling each element of a sequence of observations with a single unstructured activity label. While such labels can be at varying degrees of abstraction, this process does not address how these activity labels should be combined to construct more complex structures representing larger plans like those produced by PHATT. The distinction between activity recognition and plan recognition is very similar to the distinction in the natural language processing (NLP) community between *tagging*, identifying part of speech tags with individual words (a task that CRFs have been shown to be very good at as [39] points out), and *parsing* which combines words with part of speech tags into whole sentences. While these two problems are related, they are distinct, as are the problems of activity recognition and plan recognition. Our work on PHATT is focused firmly on plan recognition rather than activity recognition.

Several researchers have been interested in using keyhole recognition to improve team coordination. That is, if agents in a team can recognize what their teammates are doing, then they can better cooperate and coordinate. They may also be able to learn something about their shared environment. For example, a member of a military squad who sees a teammate ducking for cover may infer that there is a threat, so that it also takes precautions.

Huber, et al. [31] present an approach to keyhole plan recognition for coordinating teams of Procedural Reasoning System (PRS) based agents. Their approach, like C&G, is based on KBMC. They developed an approach for automatically generating belief networks for plan recognition from PRS knowledge areas (hierarchical reactive plans). The most important difference between our work and theirs is that we obtain a simpler structure by working with the plan representation directly, instead of generating an intermediate representation (the belief network), with inhibitory links, etc. as they do. Further, it is not clear how they handle the interleaving of multiple plans and the development of plans over time.

Kaminka, et al. [34] also present an algorithm for keyhole recognition for teams of agents. This work has a number of nice properties: it is probabilistic, rooted in a model of plan execution, and considers the question of how to handle missing observations of state changes. However it differs from this work significantly in using a different model of plan execution and it assumes that each agent is only pursuing a single plan at a time. Finally their work differs from this work in devoting a great deal of effort to using knowledge of the team and its social structures and conventions to infer the overall team behavior.

Avrahami-Zilberbrand and Kaminka [1] have reported some of the most closely related work to our own. In order to draw these contrasts more clearly, we will return to discuss their work after we have provided more intuitions for our algorithm.

4 Intuition

PHATT takes a Bayesian approach to plan recognition. Our Bayesian reasoning, as is customary, is based on a stochastic, generative model of the phenomena to be reasoned about. For plan recognition, this requires building a stochastic model of the process of choosing and executing plans, and making observations of the executing plans. This model gives us the probability of a plan, $P(plan)$,[4] and the probability of observations given the plan, $P(obs|plan)$. We then "invert" the model to go from observations to hypotheses about the underlying plans, i.e., we reason from this model to the probability of plans given observations, $P(plan|obs)$, using Bayes' law. In this section we will provide the intuitions behind PHATT's model of plan execution and how the process is inverted to infer plans. A more detailed and formal treatment is given in the following sections.

The central idea behind the PHATT plan execution model is that plans are executed dynamically, and as a result the action an agent takes at each time step critically depends on the actions they have previously taken. Intuitively, at any moment, the executing agent might like to execute any of the actions that will contribute to one of his current

[4] We unpack what is meant by the probability of a plan later.

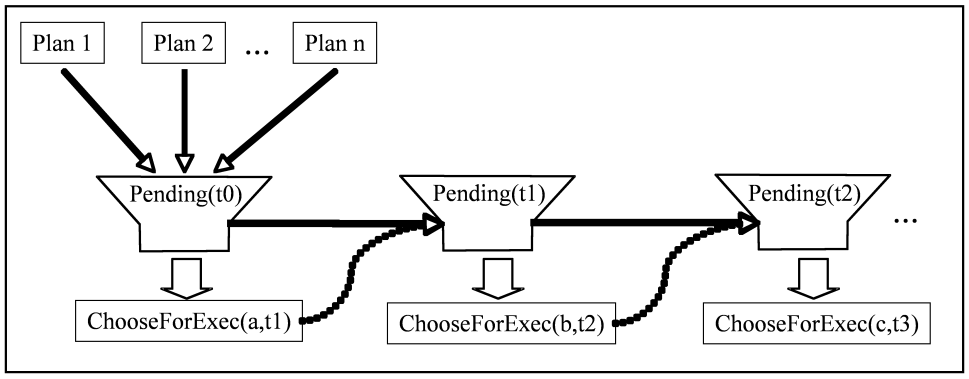

Figure 2: A simple model of plan execution: A given set of plans defines an initial pending set of actions that are enabled for execution in an initial state. At each time step, one action in the pending set is chosen for execution (ChooseForExec). The execution of the chosen action makes progress in executing the plans and enables further actions. To model this, at each time step we can generate a new pending set of enabled actions based on the previous pending set and the executed action

goals, but it can only successfully execute those that have been *enabled* by prior execution of predecessor actions in the plan. We call the set of actions that contribute to the agent's current plans and are enabled by its previous actions the *pending set*. These are the actions that are "pending" execution by the agent.

With this idea in mind, we can build a model of plan execution. First, an agent chooses a set of goals. To achieve these goals, the agent chooses a set of methods (commits to a set of choices at the OR-nodes in the plan library). Before the agent begins acting, there are a subset of actions in the plans that have no prerequisite actions. These actions form the initial pending set for the agent. From this initial set, the agent chooses an action for execution.

After the agent performs an action, the agent's pending set is changed. Some actions will be removed from the pending set (the action that was just executed, for example) and other actions may be added to the pending set (those actions that are enabled by the execution of the previous action). The agent will choose its next action from the new pending set and the process of choosing an action and building new pending sets repeats until the agent stops performing actions or finishes all of its plans. This process is illustrated in Figure 2.

We can probabilistically simulate this model of plan execution by sampling the agent's goals and plans, then repeatedly choosing elements from the resulting pending sets, generating future pending sets from which later actions are selected. We note this probabilistic model of plan execution is a HMM because the observer cannot see the agent's goals, their choices at the OR-nodes in the plans, or the pending sets.

To use this model to perform probabilistic plan recognition, we take the observations of the agent's actions as input and invert the generation process to build up an explanation for the observed actions. By hypothesizing goals and plans for the agent, and then stepping forward through the observation trace, we can generate a possible sequence of pending sets. When we reach the end of the set of observations we will have an assignment of each observed action to a hypothesized plan that achieves the one of the agent's hypothesized goals and a sequence of pending sets that is consistent with the observed actions. This collection of plan structures and pending sets is a single complete *explanation* for the observations.

We assume that our input plan library is augmented with probabilities, specifically prior probabilities of root goals, method choice probabilities, and probabilities for picking elements from the pending sets. In this paper, we will make some simplifying assumptions to simplify the derivation of these probabilities. It should be noted, however, that these simplifying assumptions are not essential to the functioning of PHATT, and most could be fairly easily relaxed. We provide a brief discussion of what happens when these assumptions are violated in Section 8.4.

On the basis of these probabilities, we can compute the probability of each explanation. Since we are taking a probabilistic approach, we want to compute the conditional probability of a particular explanation, exp, given a set of observations, obs. Using Bayes' rule, the conditional probability of the explanation is:

$$P(exp|obs) = P(exp \wedge obs)/P(obs)$$

We will exploit the equivalent formulation (assuming a mutually exclusive and exhaustive set of explanations):

$$\begin{aligned} P(exp|obs) &= P(exp \wedge obs) / \sum_i P(exp_i \wedge obs) \\ &= P(exp)P(obs|exp) / \sum_i P(exp_i)P(obs|exp_i) \end{aligned}$$

where the denominator sums the probability mass of all of the explanations to produce the probability of the observations.

In this work, an explanation will be allowed to contain multiple hypothesized goals (we will provide more formal definitions for these terms in the next section). Since we can compute the probability of a single explanation and the observations, if we build the complete set of possible explanations for the observations, and compute the probability for each, we can then compute a single goal's conditional probability by summing the probability mass of explanations that contain the goal:

$$P(goal|obs) = \sum_{\{exp_i | goal \in exp_i\}} P(exp_i|obs)$$

In previous work [25], we presented a direct implementation of the model, almost exactly as presented above. We did this using a prover for Poole's Probabilistic Horn Abduction (PHA) logic [44, 45]. We provided PHA rules that described the generative model above, and the PHA prover was able to use the formulation of the generative model, together with a plan library and a sequence of observations, to perform probabilistic plan recognition in a top-down manner, hypothesizing the set of all possible root goals and generating the pending sets and explanations as we have described. We discuss the top-down algorithm in Section 6.

However, to do this required assuming that there could be at most a single instance of each possible root goal. This provided the finite hypothesis space required by the top-down algorithm. This is not an unusual assumption to make in plan recognition and so was thought to be acceptable. However, in later applications we found this assumption too restrictive. Domains like computer network security regularly have multiple instances of the same root goal active at the same time [21].

To meet these needs, we have developed a bottom-up algorithm for PHATT. This algorithm performs essentially the same inference, but directly manipulates tree structures representing probabilistic explanations, rather than using resolution theorem proving. Being bottom-up it is not required to make the "single instance assumption." We will discuss this further after we have provided a more formal specification of both the top-down and bottom-up versions of the algorithm in terms of tree structures.

With these intuitions in hand, it is worth noting that Avrahami-Zilberbrand and Kaminka [1] take a similar approach, but differ in some subtle points. While they maintain the set of hypotheses in much the same manner as this work, instead of using a model of plan execution and pending sets as a foundation for their work, they check the consistency of observed actions against previous hypotheses. This allows them to solve some of the problems that we address, but will not allow them to recognize those tasks that depend critically on the pending set including handling negative evidence (*not* seeing actions). In order to compute probabilities for their explanations Avrahami-Zilberbrand and Kaminka have suggested the use of HMMs as an area for future work.

In the following sections, we will first formally define explanations and pending sets based on a plan library and tree structures. We will do this in a manner very similar to the definitions for context free grammars(CFGs) found in [28] but extend them to handle partial ordering of actions.

5 Formalizing Explanations

The foundation of any plan recognition system is a collection of plans to be recognized. These plans must be specified in a formal language. In this section we will first define the language for specifying plans in the form of a plan library and then provide a number of definitions of terms, tree structures, and algorithms that are built up from the plans defined in a plan library. The most critical of these structures is the set of generating trees constructed from a plan library that are used to build plan structures. This set of definitions will culminate in a formal definition for explanations and the specification of an algorithm for their generation based on the plans in a plan library. In each case, after the formal definitions we will provide intuitions to try to aid the reader.

5.1 Plan Tree Grammars

Definition 5.1 *We define a* plan library *as a tuple* $PL = \langle \Sigma, NT, R, P \rangle$ *where* Σ *is a finite set of* basic actions *or terminal symbols,* $NT = TNT \cup NNT$ *is a finite set of non-terminal symbols such that* $TNT \cap NNT = \{\}$*,* R *is a distinguished subset of "intendable root" non-terminal symbols* $R \subseteq NT$*, and* P *is a set of production rules of the form* $A \rightarrow \alpha : \phi$*, for* $A \in NT$*, where*

- *if* $A \in TNT$ *then* α *is a single terminal symbol* $\sigma \in \Sigma$*, and* $\phi = \{\}$*. In addition, we have the following conditions:*
  - $\forall A \in TNT$*:* $A \rightarrow \sigma_1 : \{\} \in P \wedge A \rightarrow \sigma_2 : \{\} \in P \vdash \sigma_1 = \sigma_2$*,*
  - $\forall A, B \in TNT$*:* $A \rightarrow \sigma : \{\} \in P \wedge B \rightarrow \sigma : \{\} \in P \vdash A = B$*, and*
  - $\forall \sigma \in \Sigma, \exists A \in TNT$*, such that* $A \rightarrow \sigma : \{\} \in P$*.*
- *if instead* $A \in NNT$ *then:*
  1. α *is a string of symbols from* NT^*
  2. $\phi = \{(i, j) | \alpha[i] \prec \alpha[j]\}$ *where* $\alpha[i]$ *and* $\alpha[j]$ *refer to the* i^{th} *and* j^{th} *symbols in* α*, respectively.*

Following traditional CFG based encodings for hierarchical plans, a plan library defines a set of production rules (P) that describe how a distinguished set of non-terminal symbols (R), representing the root goals for the plans to be recognized, can be expanded into sequences of other terminal and non-terminal symbols (NT). By repeatedly applying these rules to the non-terminals, a given root goal symbol can be reduced to a sequence that only contains terminal symbols (Σ). This sequence of terminal symbols represents the observable actions for one instance of the high level goal.

Our definition diverges from traditional CFGs in two ways. First, for every terminal in the grammar, plan tree grammars must have a distinguished non-terminal (captured in the set TNT) that maps uniquely to the terminal symbol. The set NNT captures the non-terminals in the grammar that do not uniquely map to a terminal. We will discuss this more after Definition 5.3.

Second, plan tree grammars have explicit ordering constraints in the production rules of the CFG, defined by a relation $\prec$. These constraints indicate when actions must be performed in a specific order. In traditional CFGs, the ordering of symbols within a production indicates a required ordering in the plan. In our grammars, all symbols on the right hand side of a production rule are assumed to be unordered unless the ordering relation for the production states otherwise. Our grammar formalism is similar to the work on ID/LP grammars [32] and other grammar formalisms that separate ordering constraints and decomposition. We will have more to say about this in Section 9.

Note that root goals, members of R, may appear on the right hand side of a production. That is, root nodes are those that are permitted to appear at the top of a derivation tree; they are not required to appear *only* at the top of a derivation tree.

To tie together the two representations we use in this paper, the production rules in this formulation of a plan library correspond to AND-nodes in the plan tree shown in Figure 1. OR-nodes in Figure 1 are captured when there is more than one production rule for a non-terminal. For example, the productions:

Theft→ **scan get-ctrl get-data**: {(1,2) (2,3)}
scan→ **zone-trans ip-sweep port-sweep**: {(1,2) (1,3)}
get-ctrl→ **get-ctrl-local**: {}
get-ctrl→ **get-ctrl-remote**: {}
get-data→ **sniffer-install default-login**: {}

would capture the first two levels of the plan for **Theft** shown in Figure 1 with the two rules for **get-ctrl** capturing the OR-node.

Definition 5.2 *Given a rule* $\rho = A \rightarrow \beta : \phi$*, we say* $\beta[i]$ *is a* **leftmost symbol (child) of** A **given** ρ *if* $\nexists j$ *such that* $(j, i) \in \phi$*.*

Intuitively, the set of leftmost symbols for a given rule are those symbols that must be first in any expansion of the non-terminal using the rule. No other action in the right hand side of the rule is ordered before these symbols by the rule's ordering constraints. Note that the definition does not require that there be a unique leftmost symbol of a rule. We denote the set of leftmost symbols of a rule ρ as $\mathcal{L}(\rho)$. We will use $\mathcal{R}(\rho)$ to denote the set of all symbols that are not leftmost of ρ. I.e., for $\rho = A \rightarrow \beta : \phi$, $\mathcal{R}(\rho) = \beta - \mathcal{L}(\rho)$, where $-$ is interpreted as set difference.

Definition 5.3 *Given a plan library $PL = \langle \Sigma, NT, R, P \rangle$, and a terminal symbol, $\sigma \in \Sigma$, we define a* **leftmost tree T, deriving** *σ, as a tree such that*

1. *every node in T is labeled with a symbol from $\Sigma \cup NT$.*
2. *every interior node in T is labeled with a symbol from NT.*
3. *if an interior node, n, in T, labeled A has children with labels $\beta_1, ...\beta_k$, then*
   - *$\exists \rho \in P | \rho = A \rightarrow \beta_1 ... \beta_k : \phi$,*
   - *node n is additionally annotated with ρ*
   - *no children of n labeled with symbols in $\mathcal{R}(\rho)$ have children.*
   - *at most one leftmost child of n has children of its own.*
4. *there is a single distinguished node in the frontier of T labeled with a terminal symbol, and this node is labeled σ. We call this the* **foot** *of the tree T and denote it* $\mathrm{foot}(T)$.

Leftmost trees are left branching trees whose frontier contains only a single terminal symbol. In the case of an intendable root non-terminal, this would be the tree capturing just the leftmost spine of the full expansion of the root non-terminal, ending at the specified terminal symbol. Note that it is the inclusion of the TNT set of non-terminal symbols in the definition of a plan library that allows us to guarantee that one can create leftmost trees with only a single terminal symbol in their frontier.

Leftmost trees correspond very closely to minimal, leftmost, depth-first derivation trees for a specific terminal for traditional CFGs. The only difference is that, in our grammars, the ordering relation defined for the plan library is used to determine which methods/non-terminals are leftmost.

We will use leftmost trees to build explanations and as elements of explanatory hypotheses. To do this, we first define a *generating* set of trees for a particular plan library and then define the process by which the trees are composed to produce derivations of sequences of observations.

Definition 5.4 *A set of leftmost trees is said to be* **generating** *for a plan library $PL = \langle \Sigma, NT, R, P \rangle$ if it contains all of the leftmost trees that derive an action in Σ rooted at a non-terminal in NT. We denote the generating set $\mathcal{G}(PL)$ and refer to its members as* **generating trees**.

To combine the trees in the generating set to build larger trees, we define *substitution*. This is the process whereby a frontier non-terminal in an existing tree is replaced with a leftmost tree that is rooted with the same non-terminal symbol while obeying the ordering constraints defined by the plan library.

Definition 5.5 *Given a tree, T_{init}, with a frontier, non-terminal node, m, let node n be m's parent node, and assume that n is labeled A and annotated with rule $A \rightarrow \beta_1 ... \beta_k : \phi$. Let m's label be β_i, and assume we are given a leftmost tree whose root is also labeled with β_i; call it T_{β_i}. We say T_{β_i} can be* **substituted for** *m* **in** *T_{init}* **resulting in** *T_{res} just in the case that $\forall j | (j, i) \in \phi$, the frontier of the sub-tree of T_{init} rooted at n's child labeled β_j only contains terminal symbols, and that T_{res} is the tree that is obtained by replacing β_i by T_{β_i}.*

In order for a tree to be substituted for a non-terminal in an existing tree, the portion of the original tree's frontier that precedes the substitution site must be completely expanded (contain only terminal symbols/basic actions). This guarantees that the ordering constraints contained in the original grammar are met. Aside from the partial ordering, and the requirement that the portion of the plan that precedes the substitution site must be fully expanded, our definition of substitution is the same as Joshi's definition of tree adjunction [33]. Note that the set of non-terminals, TNT, and the production rules that rewrite elements of TNT to single elements of Σ were included in the definition of a plan library to permit a uniform treatment of substitution and generating trees. To emphasize the fact that the majority of the trees

we will be dealing with will be built by repeated substitution, following work in natural language processing, we will refer to such trees as **derivation trees** to distinguish them from the original leftmost trees within the generating set. We will call such derivation trees **partial** if the tree's frontier contains non-terminals.

In the remainder of this discussion of the PHATT algorithm, there may be multiple identical (partial) trees, between which we may need to distinguish. Accordingly, will talk about particular *tree instances* that are distinguished from the more general tree definitions in the plan library by having a rigid designator associated with them for indexing purposes. Thus the following discussion will be couched in terms of introducing new tree instances, and substitution into particular tree instances.

We are now in a position to formally define a *substitution set*. The substitution set will fulfill the role of the pending set we referred to in the introduction. Since we want our algorithm to support multiple root goals and even multiple instances of the same root goal, we define a substitution set relative to a given set of partially expanded derivation tree instances.

Definition 5.6 *Given a plan library, PL, and a set of partial derivation tree instances, D, representing plan instances from PL, we define the* **substitution set for** *D, represented as $PS(D)$, as a set of tree instances $T \in \mathcal{G}(PL)$ that can be substituted into some tree in D. Each tree instance in $PS(D)$ is indexed by the tree in D into which it is to be substituted, and the particular non-terminal in the tree for which it will be substituted. In the special case that $D = \{\}$ we define $PS(D) = \{\}$.*

The substitution set is a set containing an instance of each tree in $\mathcal{G}(PL)$ that could be substituted into some tree in D.

The pending sets we discussed earlier may be extracted from the substitution sets. While our initial discussion of pending sets was couched in terms of actions, the definition that we have produced is actually in terms of the tree structures that support the inclusion of a particular action in the agent's plan. For this reason, it is possible that there may be multiple tree instances within the substitution set that have a foot labeled with the same terminal action symbol. Such tree instances are made unique by being designed to substitute into different non-terminals in the derivation trees or by making different commitments about how to achieve the plan.

Our original pending set is nothing more than the set of terminal symbols occurring as the foot of some tree in a substitution set.

Definition 5.7 *Given a plan library $PL = \langle \Sigma, NT, R, P \rangle$, set of possibly partial derivation trees and D and a substitution set for D, $PS(D)$, we can define:*

$$\text{PendingSet}(D) = \{x | \exists t \in PS(D) \wedge x = \text{foot}(t)$$

For example, if PHATT has seen the partial input **zone-trans**, and is considering an explanation with the single root goal **Brag**, the substitution set would be $\{$**IP-SWEEP** $\rightarrow$ **ip-sweep**, **PORT-SWEEP** $\rightarrow$ **port-sweep**$\}$[5] From these we may extract the pending set $\{$**ip-sweep**, **port-sweep**$\}$. Since the pending set and the substitution set are so closely related, we will use the terms interchangeably, except where the difference is critical.

Finally, we formally define an explanation for a series of observations. Intuitively, an explanation for a set of observations is a pair that contains:

1. A forest of possibly partial derivation trees representing a set of plan instances with each observed action assigned to a terminal symbol and obeying the ordering constraints of the plan library. We note this is a *set* of derivation trees in order to explicitly support multiple root goals with interleaved plan execution.

2. The sequence of substitution sets that were used in the production of the particular derivation trees. As we will see in Section 7, the substitution sets will be used in computing the probability of an explanation so it will be helpful to explicitly carry them along in the definition.

Therefore:

Definition 5.8 *We define an* **explanation** *for a sequence of observations, $\sigma_1 ... \sigma_n$ as $\langle D_n, \{PS(D_0), ...PS(D_n)\}\rangle$. D_n is a possibly empty forest of possibly partial derivation trees, where the terminal leaves are the set $\sigma_1 ... \sigma_n$. $\{PS(D_0), ...PS(D_n)\}$ are the series of substitution sets used in the construction of D_n.*

[5] Note that **IP-SWEEP** is the member of NNT corresponding to the terminal **ip-sweep**, and similarly for **PORT-SWEEP**.

In the rest of this discussion we will often use the terms **possible** or **partial explanations** when we wish to emphasize that an explanation has not processed all of the observed actions. Keep in mind that it is possible for later observations to be inconsistent with earlier choices made in an explanation and therefore rule out the explanation for the entire set of observations. With this collection of definitions in hand we are now in a position to provide a formal top-down algorithm for computing an explanation for a set of observations.

6 Top-Down Algorithm for Building Explanations from Trees

In this section, we provide a nondeterministic algorithm for explanation generation. As with most nondeterministic algorithms, ours will give an idealized picture of a program which first guesses what the observed agent intends to do, and then verifies that hypothesis by matching observations against its hypothesis. This algorithm is very close to what was done in our earlier implementation [25]. While this algorithm is not practical, it very closely parallels the generative model, so it provides a good basis for the probability discussion which follows, and a bridge to the bottom-up PHATT algorithm.

We define the process of explanation-building in two stages: the construction of an initial hypothesis, and the progression of an hypothesis by incorporating a new observation into it. We want our algorithm to support multiple root goals and even multiple instances of the same root goal, therefore we define an *initial goal hypothesis* to be a set of root goal *instances*.

Definition 6.1 *An* **initial goal hypothesis***, D_0, is a set of instances of non-terminals in R with no ordering constraints between them.*

Since there are no ordering constraints between the elements of an initial goal hypothesis, they are all available immediately to have trees substituted for them.

Procedure 6.1 (Top-down explanation generation)

PROCEDURE $Explain(\{\sigma_1 ... \sigma_n\})$
  CHOOSE initial goal hypothesis D_0 *from* R*;*
  $E = \langle D_0, \{PS(D_0)\}\rangle$*;*
  LOOP FOR i = 1 to n DO
    CHOOSE $T_{new} \in PS(D_{i-1})$ *such that* $\text{foot}(T_{new}) = \sigma_i$*;*
    $D_i =$*Substitute*(T_{new}, D_{i-1})*;*
    $E = \langle D_i, \{PS(D_0)...PS(D_i)\}\rangle$*;*
  END LOOP;
RETURN E*;*

The top down procedure for finding a single explanation works by using the first CHOOSE operation to select the initial goal hypothesis. After then computing the substitution set for the set of goal instances, the algorithm loops through the sequence of observations using a second CHOOSE operation to select elements from the substitution set for substitution into the current set of derivation trees to incrementally produce the derivation trees (and pending sets) for the explanation.

We cannot over-stress the fact that *Explain* is a *nondeterministic* algorithm. The CHOOSE operations are nondeterministic choice operations. To resolve these operations for even a single explanation we would have to use search. Keep in mind that the selection of the initial goal hypothesis determines all of the goal instances being pursued by the observed agent, so this top-down algorithm is effectively a generate and test algorithm. The first CHOOSE operation is generating a hypothesis about the root goals, and the inner loop is testing that the observations conform to this hypothesis.

A naive search-based implementation of this algorithm faces a formidable search problem. To find even a single explanation, a large number of hypotheses both for possible goals as well as the plans being pursued to achieve them must be considered. Most of these will not account for the observed actions and will have to be abandoned. The handling of all of this search has been left implicit in the nondeterministic nature of the CHOOSE operators. Worse, since the number of goals is not a priori bounded, the space of explanatory hypotheses is theoretically infinite.

While not practical, the abstract top-down algorithm helps by providing a crisp definition of the search space, and will aid in the discussion of the probability model for the explanations that we will cover in the next section. After the discussion of the probability model we will return to discuss a bottom-up algorithm for building explanations incrementally based on observations that is much closer to the implementation in the PHATT system but does not align quite as cleanly with the probability model.

7 The Probability Model

Recall from Section 4 that PHATT must compute

$$P(exp \wedge obs) = P(exp)P(obs|exp)$$

for each explanation, exp, given the set of observations obs. We can see the first term, $P(exp)$, as the probability of agent having the hypothesized goals and plans, namely the derivation trees in the explanation. The second term, $P(obs|exp)$, is the probability that the observed actions are chosen from the associated sequence of substitution sets. We further break down the probability $P(exp)$ into two terms: one term for the probability that the agent has the hypothesized set of root goals, $P(goals)$, and a second term for the probability that the given set of plans is chosen to achieve the goals, $P(plans|goals)$. Note that we assume that the observed actions are conditionally independent given the goals. We also assume the probability of each root goal is independent from the other goals in the explanation. This results in the following formula:

Formula 7.1

$$P(exp \wedge obs) = P(goals)P(plans|goals)P(obs|exp)$$

The first term, $P(goals)$ is the prior probability of the set of root goal instances being adopted by the actor. In PHATT, the prior for each root goal is given in the plan library and we represent the probability of an agent adopting goal G as $P(G)$. In order to compute the probability of a set of goals we therefore take the product of each of their probabilities. We also note that modeling each of these goal selections as independent is often unrealistic, but see our discussion of this assumption later in this section.

To address the issue of multiple instances of the same goal, we define $P(G)$ as the probability that the agent adopts an instance of G and keeps sampling. Therefore, the probability that there will be exactly n instances of any G will be $P(G)^n(1 - P(G))$, a geometric distribution. This is almost certainly incorrect — our intuition is that the probability of multiple instances of a single goal goes down far more rapidly than this. However, in practice the oversimplification seems benign: the effect of prior evidence and the underestimated decline in probability are sufficient to give good results. The theory will accommodate more sophisticated probability models that make fewer independence assumptions, but this could add significant computational cost.

Letting $|G_{exp}|$ represent the number of instances of goal G in the explanation exp, we have the following formula:

Formula 7.2

$$P(goals) = \prod_{G \in goals} P(G)^{|G_{exp}|}(1 - P(G)) \prod_{G \notin goals} (1 - P(G))$$

which can be rewritten as:

Formula 7.3

$$P(goals) = \prod_{G \in goals} P(G)^{|G_{exp}|} \prod_{\forall G \in R} (1 - P(G))$$

Note that since the second term in this formula is a product over the set of all intendable root goals, R, it is actually a constant across all explanations.

The second term in Formula 7.1, $P(plans|goals)$, is the probability of the agent choosing a particular means to achieve a goal. In our model, this is determined by the choices for each of the OR-nodes in the derivation trees. Therefore, for each non-terminal/sub-goal for which there are multiple production rules, PHATT must have a probability that the given rule was used to expand the specific non-terminal. For example, in Figure 1, a cyber attacker could use a

syn-flood, **bind-DoS**, or **ping-of-death** for a denial of service attack. PHATT must have a distribution over how likely each of these possible attacks are given that the agent is going to commit a denial of service attack.

Typically, we have assumed that each production rule, $A \rightarrow \alpha : \phi$ in the plan library is equally likely given the agent is attempting to achieve the (sub-)goal on its left hand side. I.e., $P(A \rightarrow \alpha_j : \phi_j | A) = P(A \rightarrow \alpha_k : \phi_k | A)$ for all j and k. Therefore we will define $|A|$ for any symbol $A \in NNT$ as the number of rules in P that have A as their left hand side. This allows us to write the following formula:

Formula 7.4

$$P(plans|goals) = \prod_{A \in plans} 1/|A|$$

This product is defined over all the non-terminal OR-nodes in the plan forest, i.e., over all the choice points in the plan forest. Note, the uniformity assumption is *not* required by the framework. One could specify a non-uniform distribution over these choices in a case where some methods were more likely than others.

The third term $P(obs|exp)$, is the probability that a particular sequence (string) of actions will be executed by the agent when carrying out its plan. If there were only a single root goal and the plan's action were totally ordered, as in a conventional CFG, there would be a unique sequence for every plan, and this term would always be one or zero. However, since we have partial orders and multiple interleaved plans, this term is the probability that the observed sequence of actions was selected from the sequence of substitution sets.

To compute the probability that the actions are chosen from the substitution sets in order, we will assume that all of the actions within the substitution set are equally likely. Thus, for a particular substitution set at time k in explanation exp, the probability of any specific element of the set is given by $1/|PS_k|$. Again note that the uniformity assumption made here is not required. Any distribution could be used. This choice could even be conditioned on the state of the world, hypothesized root goals, and plans of the agent. This results in the following formula:

Formula 7.5

$$\begin{aligned}
P(obs|exp) &= P(obs_1|exp)P(obs_2|exp, obs_1)\ldots \\
&\quad P(obs_n|exp, obs_1, \ldots, obs_{n-1}). \\
P(obs_1|exp) &= 1/|PS(D_0)| \text{ if } obs_1 \in PS(D_0) \\
&= 0 \text{ otherwise.} \\
P(obs_i|obs_0, \ldots, obs_{i-1}) &= 1/|PS(D_{i-1})| \text{ if } obs_i \in PS(D_{i-1}) \\
&= 0 \text{ otherwise.} \\
P(obs|exp) &= \prod_{i=1}^{n} 1/|PS(D_i)|
\end{aligned}$$

With all of these formulas in hand we can now rewrite Formula 7.1 one last time as:

Formula 7.6

$$P(exp \wedge obs) = K \prod_{G \in goals} P(G)^{|G_{exp}|} \prod_{A \in plans} 1/|A| \prod_{i=1}^{n} 1/|PS(D_i)|$$

Where K is the constant $\prod_{\forall G \in R}(1 - P(G))$.

Remember, given this formula for the probability of each explanation, the conditional probability for a particular goal,G, can be computed by summing the probability mass for those explanations, exp_i that contain the goal and dividing by the total probability mass associated for all the explanations. That is:

Definition 7.1

$$P(G|obs) = \sum_{exp_i | G \in roots(exp_i)} P(exp_i \wedge obs) / \sum_{exp} P(exp \wedge obs)$$

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, occurs. Recall that the denominator is the the same as the prior probability of the observations, so it will usually be less than one.

The use of three different probabilities differentiates this work from work on probabilistic grammars[47, 53]. Most probabilistic context free (and sensitive) grammar (PCFG/PCSG) research has included the use of a single probability for each grammar rule to capture how likely it is that the given non-terminal is expanded using that grammar rule. This leaves out the term for the substitution sets making it more difficult for these other systems to address partially ordered plans, multiple concurrent plans, and even partial observability.

Simplifying assumptions PHATT makes a number of simplifying assumptions — mostly uniformity assumptions — about its probability parameters. One may ask about the effect of these assumptions. We do not have room here to address this issue experimentally, but there is a fair amount of evidence, from diagnostic applications, that Bayesian systems are robust to inaccuracy in their parameters (e.g., [46]) and additionally some analytic information about how to assess the sensitivity of conclusions to probability parameters[7]. We have drawn upon this research in the following assessment.

Before we discuss the PHATT's sensitivity to its assumptions and parameters, we would like to stress once again that the uniformity assumptions that PHATT makes are *not* inherent in the algorithm itself. These assumptions could be relaxed in most cases without significantly complicating PHATT. In the following discussion, together with discussing the effect of the simplifying assumptions applied to PHATT's probability parameters, we will also discuss how these could be relaxed.

To recapitulate, there are three kinds of probability parameters in PHATT, each of which is subject to simplifying assumptions: (1) the prior probabilities of goals; (2) the probability of choosing a method from among the alternative methods for a single (sub) goal; (3) the probability of choosing a specific action from the set of currently pending actions. We discuss each of these in turn, below:

Prior probabilities of goals PHATT's performance is sensitive to the prior probabilities of intendable root nodes to the extent that its input is ambiguous. E.g., in our example library (Figure 1), if there's a distribution that is not uniform over the three root goals, then while we see only **scan**-related actions, then we will be off to the extent that the priors are off. So if the prior probability of **Brag** was .2, and that of **Theft** and **DoS** were only .1, then the posterior probability given that we have seen only **zone-trans**, and ignoring the possibility of multiple plans, would be

$$P(\textbf{Brag}|\textbf{zone-trans}) = .5 \quad > \quad P(\textbf{Theft}|\textbf{zone-trans}) = P(\textbf{DoS}|\textbf{zone-trans}) = .25$$

However, once we get further observations, the problem will tend to correct itself. Indeed, if we rule out multiple-goal explanations, the appearance of any of the data-gathering or DoS-related actions would eliminate the **Brag** goal entirely. If we permit multiple-goal explanations, the posterior probability of **Brag** *and* **DoS**, say if we had seen a subsequence ending with **synflood**, would be approximately $(.2 * .1)/(.1 + .2 * .1) \approx .17$ versus $.1/.12 \approx .83$ for **DoS** alone.

The uniformity assumption for the prior probabilities of goals would be the easiest assumption to change in the PHATT implementation. Since the priors are a weighting factor applied to every explanation, PHATT would need next to no modification to accommodate non-uniform priors.

There are two other issues related to the uniform priors assumption for root goals. The first issue is our assumption that root goals are independent. For some goals, this is absurd: for example the pairs "brewing tea" and "making toast"; and "bungee jumping" and "brewing tea" are unlikely to be independent. The former are probably positively correlated, and the latter negatively. Positive correlations can be handled relatively easily by introducing new top-level goals that activate combinations of the original goals. For example, one might have an "afternoon tea" goal with a method that involved both brewing tea and making toast.

PHATT's model is much less friendly to negatively correlated goals. In order to accommodate them, one could make a more elaborate version of the approach for positively correlated goals. In this more elaborate version, one would specify probabilities for combinations of subgoals, that we call *contexts*. For example, one might have an

"extreme sports" context and a "quiet afternoon" context, that would probability information that tended to turn on (resp. off) goals like "bungee jumping" and off (resp. on) goals like "afternoon tea." Doing so would require slightly more machinery, but wouldn't change the probability computations very much, since the root goal probabilities need be computed only once. However, the bottom-up goal introduction method would need to be modified to interact with these context structures. One could determine on an application by application basis whether this kind of machinery was necessary – note that for this example there are unlikely to be a lot of action sequences that would cause PHATT to believe that the agent was actually both bungee jumping and brewing tea, so here the independence assumption is probably benign.

The second issue is our model for the probability of multiple instances of the same root goals, which is a hypergeometric distribution for each root goal. This likely overstates the probability of multiple instances of root goals. However, as we will see, the PHATT implementation doesn't consider such explanations unless primed by specific observations (see Section 8.1 for details). This limits this overestimate to cases where there is some evidence for the hypothesis.

Probabilities of methods given (sub)goals Here again, the effect of incorrectness in the assumptions will cause confusion to the extent that the observation sequence is otherwise ambiguous. For example, consider a case where PHATT has seen enough input to be considering two, equally likely, explanations, one for root goal A and one for root goal B, and each plan would next proceed to do either C or D. Under our uniformity assumption, PHATT can see either C or D and it will continue to treat both A and B as equally likely. However, consider the case where C is much more likely given the root goal A versus B. In this case the true posterior odds of A versus B would be proportional to the odds of choosing a C given A versus given B. For example, if the probability of choosing C given A was .9, but the probability of choosing C given B was only .1, then PHATT could be substantially off until and unless it made some further disambiguating observation.

Such a deviation from uniformity, could arise in our sample plan library if there were multiple ways to **get-ctrl**. Suppose that braggarts were more likely to be unsophisticated "script kiddies" prone to using gross exploits, where a data thief was more likely to take precautions to remain undetected. We have not considered problems in which this kind of deviation from uniformity happens, but as with non-uniform root goal priors, it would be quite easy to modify PHATT to take such probabilities into account. Once again, these probabilities are simple weighting factors in PHATT.

Probability distributions over substitution sets Recall that PHATT assumes that an agent is equally likely to execute any of its enabled actions. For many domains, this is going to be inaccurate: In some domains agents are likely to operate in a depth-first way, and persist in working on subgoals of a single goal until it is satisfied. On the other hand, in time-pressured domains with multiple simultaneous goals (consider a short-order chef, for example), the agent may time-slice to multitask. It is even possible for the specific goals, methods of achieving them, or even situational features like the weather to effect the selection of the next action for execution.

In cases where the true probability of the action selection is affected by these features, there is no question that PHATT's probabilities can be substantially off. However, it is not hard to extend PHATT to take these features into account. It is simply a matter of storing the relevant features within the explanation and then using them to compute the probabilities. Depending on the domain this more advanced modeling could be quite helpful or have a significant computational cost with little impact. It is an area for future work to identify when such more complex modeling will be worth while.

* * *

We have now provided formal definitions, a top-down algorithm, and a probability model for plan recognition based on a model of plan execution. However as we pointed out in Section 6 the top-down algorithm requires an exhaustive search over an infinite search space. To eliminate this problem in the next section we will outline a very similar bottom-up algorithm that only introduces root goal hypothesis as they are suggested by the observed actions circumventing the need for an initial goal hypothesis.

8 Practical Explanation Building

8.1 Bottom up explanations

By taking a bottom-up approach to explanation generation we can introduce goals to an explanation after observing an action that is consistent with it, rather than initially hypothesizing the set of root goals. Taking this approach requires an additional definition.

Definition 8.1 *We define the set of* **intendable trees** $\mathcal{I}$ *as the set of generating trees rooted at an intendable non-terminal symbol:*

$$\mathcal{I} = \{T \in \mathcal{G}(PL) | \, \text{root}(T) \in R\}$$

Where $\text{root}(T)$ *denotes the label of the root node of the tree* T.

Every tree in $\mathcal{I}$ derives the first action of a plan rooted in one of the distinguished intendable root non-terminals. Note that there may be multiple trees in $\mathcal{I}$ that have the same root and leftmost child, since the interior nodes of the tree may be different.

In the top down algorithm, we started the explanation-finding process by choosing an initial goal hypothesis. In the bottom-up algorithm, on the other hand, the initial hypothesis set will be empty, and we add new tree instances (copied from elements of $\mathcal{I}$) into the set of derivation trees as new goals, and associated plans, are suggested by the observed actions.

Procedure 8.1 (Bottom-up single explanation generation)

PROCEDURE $Explain(\{\sigma_1 ... \sigma_n\})$
  $E = \langle \emptyset, \{\emptyset\} \rangle$;
  LOOP FOR i = 1 to n DO
    $\langle DT, \{PS_0, ..., PS_{i-1}\} \rangle = E$;
    CHOOSE
      (1) $T_{new} \in PS_{i-1}$ *such that* $\text{foot}(T_{new}) = \sigma_i$;
        $DT_{new} = \text{Substitute}(T_{new}, DT)$;
        $PS_{new} = PS(DT_{new})$;
        $E = \langle DT_{new}, \{PS_0, ..., PS_{i-1}, PS_{new}\} \rangle$;
      OR
      (2) $T_{new} \in \mathcal{I}$ *such that* $\text{foot}(T_{new}) = \sigma_i$;
        $DT_{new} = DT \cup \{T_{new}\}$;
        $PS_{new} = PS(DT_{new})$;
        $\{PS'_0, ..., PS'_{i-1}\} = \text{BackpatchPS}(\text{root}(T_{new}), \{PS_0, ..., PS_{i-1}\})$
        $E = \langle DT_{new}, \{PS'_0, ...PS'_{i-1}, PS_{new}\} \rangle$;
    END LOOP;
  RETURN E;
END PROCEDURE;

PROCEDURE $BackpatchPS(R, \{PS_0, ..., PS_m\})$
  $T_{addition} = \{t \in \mathcal{I}$ *such that* $\text{root}(t) = R\}$;
  LOOP FOR i = 0 to m DO
    $PS'_i = PS_i \cup T_{addition}$;
    END LOOP;
  RETURN $\{PS'_0, ..., PS'_m\}$;
END PROCEDURE;

The bottom-up algorithm, Procedure 8.1, is a dynamic programming algorithm that always maintains a current set of derivation trees, based on the set of derivation trees from the previous iteration. A single set of derivation trees is

enough to build the explanations, but is not sufficient to support the probability computations: for this we need a full sequence of substitution sets. The substitution sets could be derived from a sequence of partial derivation trees, but maintaining only the sequence of substitution sets, provides PHATT a savings in time and memory.

The top-down algorithm (Procedure 6.1) intitially "guessed" the set of goals, and then expanded plan trees downward, so it featured two nondeterministic CHOOSE operations. However, in Procedure 8.1 all of the choices have moved to a single point in the algorithm, but this is a more complex choice. There are now two possibilities. First, an observed action can contribute to one of the goals that is already part of the derivation trees (case (1) for the CHOOSE operator), in the same way as the top-down algorithm. Second, an observed action could introduce an entirely new plan for an entirely new root goal (case (2) for the CHOOSE operator) by selecting T_{new} from $\mathcal{I}$.[6]

Backpatching an explanation While this algorithm introduces new goals into its hypotheses only as needed, *conceptually* these goals were there all along. That is, from the model's view the agent had these goals at the time the first action was selected from the initial substitution set, however they had simply chosen not to execute any of the actions that contributed to the plan for this goal.

Since the algorithm only adds goals as needed, when a new goal is introduced, the prior substitution sets in the explanation will not have contained trees for this plan. Thus, the prior substitution sets will be incorrect. To later use these substitution sets to correctly compute the probability of each explanation they must be backpatched to account for the presence of the new goal at the prior time points. To do this requires adding tree instances to the prior substitution sets for all of the intendable trees that have the new goal as their root. These amended substitution sets are computed by the call to the "BackpatchPS" function in case (2).

The cost of doing this is bounded by the number of observations and so is relatively inexpensive for the algorithm. We note that each such backpatch operation is a relatively simple set union and can be made very rapid by careful bookkeeping. The set of intendable trees for each root goal can also be identified and stored once before execution removing the cost of computing this set as well. Thus the cost of this operation can be reduced to $O(n)$ and as such is dominated by other costs in the algorithm. Keep in mind that this operation only happens when new root goals are introduced to an explanation. Thus, in the case where an agent has only a single goal this happens exactly once at the first observation.

An example Here is how PHATT would construct one of the explanations for the sequence of observations: {**zone-trans**, **ip-sweep**, **zone-trans**}. Figure 3 shows the construction of the derivation trees and the related substitution sets. We note that to save space in the figure we have represented the trees in the substitution sets as pairs made up of the tree's foot terminal and root instance symbols.

The first **zone-trans** observation introduces an instance of an intendable tree for the goal **DoS** which we have indexed with a one ("1"). We note that the introduction of the new root goal requires us to backpatch the initially empty pending set with the tree for the **zone-trans** observation and the **DoS** root goal. The requirement to backpatch is indicated by the asterisk next to the substitution set and results in the new substitution set for time t0:

- Pending(t0) = {(**zone-trans**, **DoS1**)}

The addition of this plan also adds two trees to the substitution set for the actions (**ip-sweep** and **port-sweep**) that are enabled as part of this plan. This same plan is then extended to explain the **ip-sweep** that is next observed.

Finally, when another **zone-trans** is observed the algorithm introduces a second instance of the goal **DoS**. This results in a number of additions to substitution set. The asterisk next to the final substitution set indicates that all of the previous partial explanations must be backpatched to include the initial actions for **DoS2**. Thus, after explaining all of the observations and two rounds of backpatching, the final explanation for the observations would contain the derivation trees shown in the third pane of Figure 3, and the following sequence of substitution sets.

- Pending(t0) = {(**zone-trans**, **DoS1**), (**zone-trans**, **DoS2**)}
- Pending(t1) = {(**ip-sweep**, **DoS1**), (**port-sweep**, **DoS1**), (**zone-trans**, **DoS2**)}
- Pending(t2) = {(**port-sweep**, **DoS1**), (**zone-trans**, **DoS2**)}
- Pending(t3) = {(**port-sweep**, **DoS1**), (**ip-sweep**, **DoS2**), (**port-sweep**, **DoS2**)

[6] It is possible for this new root goal to be another instance of a root goal that is already present in the set of derivation trees.

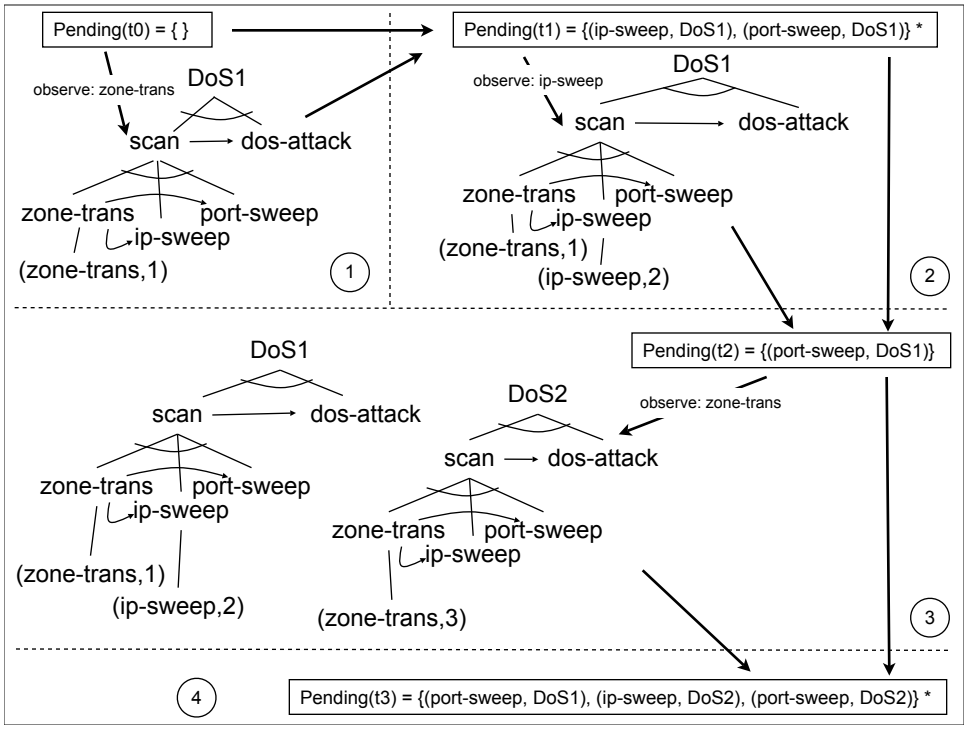

Figure 3: The process of building a single explanation for three observations along with associated pending sets. Starting in the upper left and moving clockwise, each pending set is grouped with the observation that follows it and the partial derivation trees that result from the observation.

Note the addition of the (**zone-trans**, **DoS2**) element to each of the prior substitution sets, and the addition of the (**zone-trans**, **DoS1**) element to the initial set as a result of backpatching.

We can compute the probability of this particular explanation by multiplying together the three terms described in Formula 7.1. First, we multiply the priors for the two root goal instances of **DoS**. Second, we multiply the probability for each of the choice points in the plans. In this case this term would be one since there were no choice points in these plans. Third and finally, we would multiply in the probability associated with each action being chosen for execution from the substitution set (given our uniformity assumption). If we assume $P(\textbf{DoS}) = .6$ this would result in the following:

$$P(example \wedge \{\textbf{ zone-trans}, \textbf{ip-sweep}, \textbf{zone-trans}\}) = (0.6 * 0.6) * 1.0 * (0.5 * 0.333 * 0.5) = 0.02999$$

Keep in mind that this is only one of the possible explanations for the observations, and that this process would be repeated for all of the explanations before we could compute the conditional probability for a given goal.

Completeness The top-down algorithm is clearly complete in the sense that it can generate all possible explanations for any observation trace. Completeness is a more complicated question for the bottom-up algorithm.

Where full observation sequences are concerned, the bottom-up algorithm is also complete. By "full observation sequence," we mean an observation sequence that contains all (and only) the set of actions generated by a set of plan trees. If Procedure 8.1 is given such a sequence as input, it can find the corresponding explanation.

A sketch of a proof is as follows: for each root goal, g, in any actual explanation for the observations, there exists a tree, $T \in \mathcal{I}$, and an observation σ_j such that $\text{root}(T) = g$ and $\text{foot}(T) = \sigma_j$. Processing the loop for σ_j will cause the second (2) clause of the choose operator in Procedure 8.1 to insert the T into the set of derivation trees. All further structure for the plan for g is added by substitution operations per Definition 5.5. Those substitutions are executed by the first (1) clause of the choose operator in Procedure 8.1 as the appropriate σ's are processed. Thus, for full observation traces, Procedure 8.1 is complete.

This notion of completeness is not necessarily the right one for *incremental* plan recognition. Ideally we might like the algorithm to be complete in the sense that given any observation sequence, p, the algorithm constructs explanations for all possible complete observation sequences, ω st $\omega = p \cdot \omega'$, for some ω'.[7] For previous approaches, such as K&A, that limit their hypothesis spaces to plans generated from a single root goal, a definition like this one makes sense. However, PHATT's explanation space is not finite, so we are not going to be able to realize this sense of completeness in any concrete implementation.

For example, consider the case where we have a single observation for of **zone-trans** from our original plan library (Figure 1). The bottom-up PHATT algorithm will consider one instance of each of the root goals (because **zone-trans** can be the initial action for any of these plans), meaning the set of hypothesized goals will be { **Brag** }, { **Theft** } and { **DoS** }. However, the single action is also consistent with an infinite number of other explanations, such as { **Brag**, **Theft** }, { **Brag**, **Theft**, **DoS** }, etc., in which the agent has multiple goals but we have yet to see any of their actions.

While Procedure 8.1 doesn't satisfy this very strong sense of incremental completeness, it provides a weaker sense of incremental completeness as follows: For any any observation sequence, p, and completion $\omega = p \cdot \omega'$, the algorithm will construct a partial explanation, E that can be completed to E' such that E' is an explanation for ω. This follows from the proof of completeness for full observation sequences.

We may strengthen the claim of incremental completeness as follows: Given an observation sequence p and completion $\omega = p \cdot \omega'$, the algorithm will find all explanations E that can be completed to E', an explanation for ω, if p contains at least one action contributing to each plan in E.

8.2 The Full algorithm

Finally we present pseudo-code for the full algorithm, it combines bottom up construction of all the explanations with the computation of probabilities for each explanation. Note that the first and second terms in Formula 7.6 (goals and choice node probabilities) are computed as the explanation is constructed, but the final term is computed in a final set of nested loops after all of explanations have been found.

Procedure 8.2 (Full algorithm)

[7] Where $\cdot$ is the concatenation operator.

PROCEDURE $ExplainAndComputeProb(\{\sigma_1 ... \sigma_n\})$
  %% Initialize the data structures.
  $D_0 = \{\}$; $E = \text{Emptyqueue}()$;
  $\text{Enqueue}(\langle D_0, \{PS(D_0)\}, 1, 1\rangle, E)$;
  %% Loop over all the observations.
  LOOP FOR i = 1 to n DO
    %% Loop over all the explanations in the queue.
    WHILE $\text{Nonempty}(E)$ *DO*
      $E' = \text{Emptyqueue}()$
      $\langle DT, \{PS_0, ..., PS_{i-1}\}, \textit{prob-roots}, \textit{prob-choices}\rangle = \text{Dequeue}(E)$;
      %% Consider all the existing plans the observation could extend.
      LOOP FOR EACH $T_{new} \in PS_{i-1}$ *such that* $\text{foot}(T_{new}) = \sigma_i$;
        $DT_{new} = \text{Substitute}(T_{new}, DT)$;
        $PS_{new} = PS(DT_{new})$;
        $\textit{local-prob-choices} = \textit{prob-choices} * \text{PTS}(T_{new})$;
        $\text{Enqueue}(\langle DT_{new}, \{PS_0, ..., PS_{i-1}, PS_{new}\}, \textit{prob-roots}, \textit{local-prob-choices}\rangle, E')$;
      END FOR EACH LOOP ;
      %% Consider all the new plans the observation could introduce.
      LOOP FOR EACH $T_{new} \in \mathcal{I}$ *such that* $\text{foot}(T_{new}) = \sigma_i$;
        $DT_{new} = DT \cup \{T_{new}\}$;
        $PS_{new} = PS(DT_{new})$;
        $\{PS'_0, ..., PS'_{i-1}\} = \text{BackpatchPS}(\text{root}(T_{new}), \{PS_0, ..., PS_{i-1}\}$
        $\textit{local-prob-choices} = \textit{prob-choices} * \text{PTS}(T_{new})$;
        $\textit{local-prob-roots} = \textit{prob-roots} * P(\text{root}(T_{new}))$;
        $\text{Enqueue}(\langle DT_{new}, \{PS'_0, ..., PS'_{i-1}), PS_{new}\}, \textit{local-prob-roots}, \textit{local-prob-choices}\rangle, E')$;
      END FOR EACH LOOP;
    END WHILE;
    $E = E'$
  END LOOP;
  $\textit{result} = \emptyset$;
  %% Compute the probability of each explanation.
  WHILE $\text{Nonempty}(E)$ *DO*
    $\langle DT, \{PS_0, ..., PS_n\}, \textit{prob-roots}, \textit{prob-choices}\rangle = \text{Dequeue}(E)$
    $\textit{local-prob-pend} = 1$;
    LOOP FOR i = 1 to $n - 1$ *DO*
      $\textit{local-prob-pend} = \textit{local-prob-pend} * 1/|PS_i|$;
    END LOOP;
    $\textit{prob-hypoth} = \textit{prob-roots} * \textit{prob-choices} * \textit{local-prob-pend}$;
    $\textit{result} = \textit{result} \cup \{\langle DT, \{PS_0, ..., PS_n\}, \textit{prob-hypoth}\rangle\}$;
  END WHILE;
RETURN result;

The functions Emptyqueue, Nonempty, Enqueue and Dequeue are the standard functions used for queue data structures. Elements of the queues are four-tuples, $\langle DT, \{PS_0, ..., PS_i\}, \textit{prob-roots}, \textit{prob-choices}\rangle$, where the first element, DT, is the set of derivation trees, the second is the sequence of substitution sets, $\textit{prob-roots}$ is the product of the probabilities of the root goals, and $\textit{prob-choices}$ is the product of the probabilities of the choices.

As in the bottom up case, the function BackpatchPS, updates each of the prior substitution sets by adding to them those trees that should have been present if the goal introduced by T_{new} had been known earlier. The function PTS gives the probability associated with the choice nodes in T_{new}. Since the choice probability for a tree is a constant for each tree, it can be computed once offline. Once the set of all of the explanations has been built, the final nested loops iterate over each explanation computing the probability contribution of each of the substitution sets. Note that this final loop doesn't include a contribution from PS_n since the action that will be selected from this substitution set has not yet been observed. Finally, the root node probabilities, probabilities of the choices, and the probabilities of drawing from the pending sets ($\textit{local-prob-pend}$) are multiplied together and returned as the probability of the explanation.

8.3 Implementation Differences

The implementation of PHATT follows the algorithm above closely, but there are several variations made to enhance efficiency. PHATT, does not maintain the complete substitution sets since only the sizes are actually needed, a significant time and memory savings. PHATT caches the intendable trees in order to avoid recomputing them, a significant space-for-time trade-off. PHATT also leaves out probabilities of "missing" goals — goals that don't appear in an explanation; this is simply a constant factor in our explanations. Finally, PHATT allows only bounded recursion in its explanations. We discuss each of these issues below.

Storing only the sizes of the substitution sets Given the uniformity assumptions that we make in computing the likelihood that a particular action is chosen from the substitution set, all that is needed for this computation is the size of the pending set. Thus rather than maintaining the complete substitution sets, the implementation only stores and backpatches the sizes of these sets.

Caching the intendable trees Procedure 8.2 has two separate inner loops: one to search for extensions to the known trees, and one to search for new trees that could be added. Actually PHATT combines the pending set with $\mathcal{I}$ to form a single set. This has a number of benefits.

First, PHATT can search over both spaces using a single loop and a common treatment of the process of extending an explanation. This greatly simplifies the control of the algorithm at the small cost of clarity. Second, PHATT can continuously update the substitution set, $PS(D_i)$, for its current derivation trees, D_i, rather than computing the pending set from scratch each time. This enables PHATT to more rapidly find the set of possible insertion points for any observed action. To aid in this, PHATT precomputes, and maintains the intendable trees as a fixed set of trees, added to the pending set, that contains a single instance of every element of $\mathcal{I}$:

Definition 8.2 (Core hypothesis set)

$$\mathcal{PS}_{core} = \mathcal{I}.$$

As a simple example of generating one explanation, following the PHATT algorithm with the plan library shown in Figure 1, one of the explanations found by this algorithm for the sequence of observations:

$$\{(\mathbf{zone-trans}, 1), (\mathbf{ip-sweep}, 2), (\mathbf{zone-trans}, 3), (\mathbf{port-sweep}, 4), (\mathbf{ping-of-death}, 5)\}$$

is shown in Figure 4. An expression (**action**, t) indicates that **action** was performed at time t; we use counting numbers for time indexes. In Figure 4, rather than displaying the actual generating trees that would be in the hypothesis set, each element of the hypothesis set is represented by a pair that contains the leftmost child of the generating tree and the root goal of the tree into which the element is to be substituted. Note the presence of the core hypothesis set, PS_{core} in the initial substitution set before plan recognition begins.

The presence of $\mathcal{PS}_{core}$ as a subset of all the pending sets should affect the probability computations, since the size of the pending set is a crucial term in computing the probability of a explanation. However, since the size of $\mathcal{PS}_{core}$ is a constant, very simple bookkeeping allows us to address this. All that is required is subtracting the size of $\mathcal{PS}_{core}$ from the size of each pending sets before using them in computing the probability of the explanation.

Approximating goal probabilities Rather than using Formula 7.3, PHATT drops the terms for the probability of of goals *not* in the explanation, $\prod_{\forall G \in R}(1 - P(G))$.[8] Keep in mind that this term is a constant across all of the explanations and therefore doesn't change the *relative* likelihood of any goal.

Bounded recursion In enumerating the substitution sets, we must be able to enumerate the complete set of leftmost trees. It follows from this that the PHATT implementation cannot support unbounded recursion within the plan grammar. This is not to say that recursive definitions are not possible within the formalism, rather that any recursion must be only up to a fixed depth. This should not represent a problem in practice since bounded agents in general will not have arbitrarily deep plans. For example, in our demonstration domain no recursion was required.

[8] Note that the original PHA specification and implementation of our theory (reported in [25]) did *not* make this simplifying assumption. That was one of the reasons it was impractically slow.

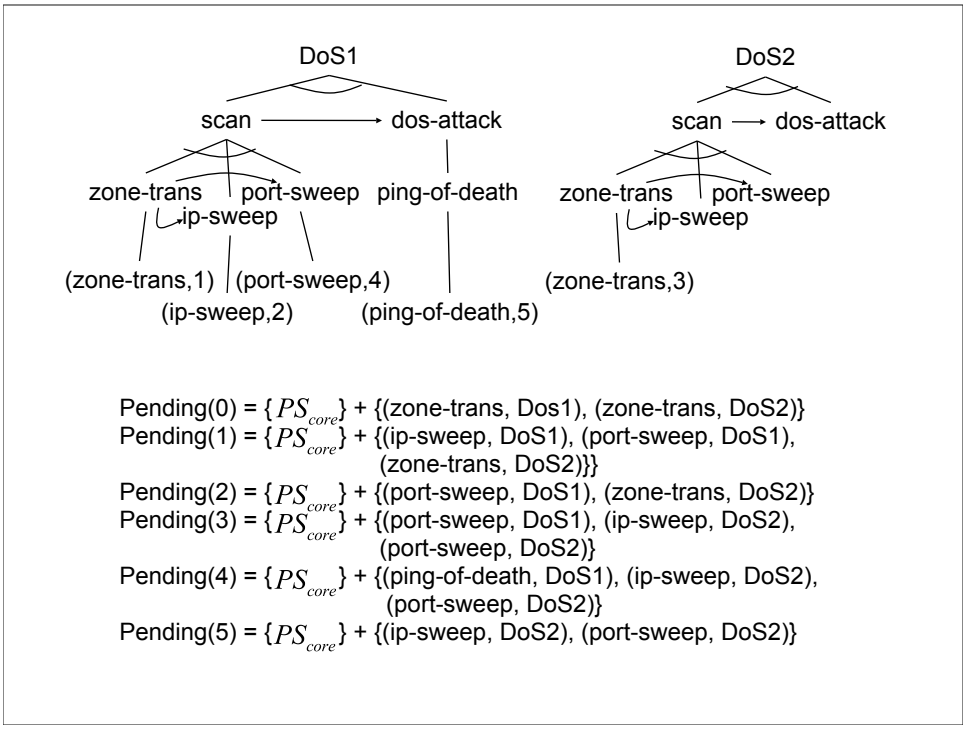

Figure 4: An explanation including derivation trees and substitution sets for the set of observations (**zone-trans**,1), (**ip-sweep**,2), (**zone-trans**,3), (**port-sweep**,4), (**ping-of-death**,5) making use of PS_{core} to simplify explanation construction.

8.4 PHATT addressing issues in plan recognition

Having completed our discussion of the PHATT algorithm and its implementation, this section gives a final brief discussion of some issues around plan recognition and the PHATT algorithm that are easier to discuss after the algorithm.

Accuracy As part of verifying the correctness of our implementation, we have conducted simple studies of the algorithm's accuracy. In all of these tests the algorithm performed as expected. PHATT was able to correctly identify the present root goals and compute the correct conditional probabilities give the system assumptions. This is true both for single and multiple root goal situations across a wide number of different kinds of plans. Section 10 of this paper will detail a number of different kinds of plan structures and detail a number of parameters that we have explored for considering the algorithm's runtime. For each of the parameter settings that we discuss there, we first verified the system's ability to accurately draw conclusions for those problem settings.

Multiple root goals Going beyond simply handling multiple root goals to handling multiple instances of the same root goal is unusual for plan recognition systems. In fact, many applications [13, 30] do not allow a user to have more than one goal at a time let alone multiple instances of the same goal. However for many real world domains this is simply an unacceptable assumption to make.

Consider the cyber security domain from Figure 1. In the real world, it is common for a determined cyber attacker to launch multiple different attacks against a single host, and even multiple instances of the same attack, to achieve a single goal. This is done for a number of reasons: diversity of target susceptibility, attack success likelihood, and to create confusion. Thus, in this domain, it is very common to see multiple instances of the same goal being pursued by different, very similar, or even identical instances of plans. For example, the explanation presented in Figure 4 must explain a second observation of **zone-trans** at time 3 that is consistent with a second **DoS** goal. Any complete algorithm for plan recognition must consider the possibility that there are multiple interleaved instances of this goal being pursued by a single agent at the same time. Most previous work has discounted the possibility that a single agent

could be pursuing multiple instances of the same root goal at the same time. However, in many domains of interest this is simply not a valid assumption.

Partially Ordered Plans As we pointed out in Section 5, the formal underpinnings of the language used for PHATT's plan libraries makes partial order explicit in the language and as a result does not require the system to perform an exponential "unfolding" of the grammar to cover all of the possible orderings for plans. PHATT also takes pains within the generation of the pending sets for a particular explanation to enforce the ordering constraints for each particular plan. This allows PHATT to take very seriously the idea of partial ordering within plan execution while still maintaining the expressiveness of the plan language. The efficiencies that result from this cannot be underestimated as we will see in Section 9.

Negative Evidence People regularly take a failure to observe actions consistent with a hypothesized goal as evidence that the goal is not being pursued. However previous work in plan recognition has not captured this intuition. With this model of plan recognition couched in terms of plan execution, PHATT is able to partially address this problem. In fact, this naturally falls out of the way in which the probabilities are computed for the pending set term for each explanation.

For example, suppose we have a plan library with three intendable roots A, B, and C. Further we assume that B is defined in the plan library by the production $B \rightarrow AD : (1, 2)$. This means that any observed plan for A could also be a plan for B. Any sequence of actions achieving A are a *proper prefix* of a plan for B and D represents the remaining actions in B not in A.

Now suppose we have a set of observations that we want to explain, and it is actually the case that the agent has either A, C or B, C as their set of goals. Any explanations that account for these actions with a root goal of A have identical assignments of observations to explanations with root goal B up until A is completed. Now consider computing the probability of the two explanations. After we have seen the last action in A, the pending set of the A, C must be smaller than the pending set for the B, C explanation, since the A, C pending set can only have actions for C while in the case of B, C the actions that make up D are still in the pending set. Making our uniformity assumption, the larger size of the pending set lowers the probability of the B, C explanation each time an action that is not in D is observed. Thus, as more and more actions are observed without seeing an action that contributes to D, the conditional probability of our favored explanation, A, C, will increase. If we do subsequently see an action that contributes to D then A, C can be ruled out as a hypothesis since it can not explain the action. Thus the B, C explanation, no matter how unlikely must be correct and will have its conditional probability increase.

Although PHATT can handle negative evidence as outlined above, PHATT still must have seen at least *some* action that contributes to a plan in order to consider it. The handling of negative evidence we have described here can only take place in the context where we have first seen some evidence for the plans in question. Thus the system does correctly handle the case we sketched above where the complete plan A is a proper prefix of another plan is treated correctly, but will not explicitly hypothesize the absence of goals that it has no evidence for.

Overloading Actions Some prior work on plan recognition has allowed an action to contribute to more than one plan. In contrast PHATT assumes that each action must contribute to only one goal. It would be relatively easy to extend the explanation building process to allow a single observation to be explained by multiple elements of the substitution set. In this case rather than selecting a single element, the algorithm would have to explore all of the subsets of trees that could account for the observed action. This would significantly increase the size of the search space. Another, more significant, problem is that we do not have a probability model for these situations. Identifying an appropriate probability model for this case is an open research question.

Hostile Agents As we mentioned in the introduction, PHATT assumes that the observed agents are not actively hostile to the inference of their goals and plans. We have done work on partially observable domains using PHATT[20], that would be applicable to recognizing when a hostile agent has hidden some of their actions. However, a great deal more work is required for a full treatment. Clearly, if a hostile agent is attempting to distract the observer, the set of root goals will not fulfill our independence assumptions. The agent would carefully choose sets of goals designed to obfuscate the true goals of the agent. This will require addressing the uniformity and independence assumptions both for the root goals and for the selection of actions from the pending sets. This is a very exciting area for future work.

Plan Language Expressiveness While the use of temporal constraint based reasoning and typed variable systems are both well studied areas in computer science, these technologies have not always been brought to bear within the languages used for plan library specifications. In many applications it is critical to reason about the temporal relations between actions within in a plan. Plan library engineering is facilitated by the use of plan variables to form method and goal schemas. We have included these as extensions to PHATT; they are discussed in Section 11.

9 Complexity

In this paper we present both analytic and empirical results on the complexity of the PHATT algorithm. Previous results in the complexity of plan-recognition demonstrate that the PHATT algorithm must be at least NP-Hard[35, 53]. PHATT must be at least as hard as approaches like Kautz' which use heuristic methods to find a single approximately optimal[9] explanation for a set of observations. However, if we also ask PHATT to compute the posterior probability of explanations, given the set of observations, then there is an additional NP-complete computation to compute the posterior probabilities. This is essentially a Bayes net probabilistic inference problem, which has also been shown to be NP-complete[14].

9.1 Finding an Explanation

First, we show that finding an explanation for an observation trace, in our framework, is NP-complete.

Explanation-finding is in NP: In Section 6, we provided a nondeterministic algorithm for finding an explanation for an observation trace. We can show that this algorithm is in NP as follows: The outer loop of the algorithm is executed once for every observed action. Accordingly, we need only show that all of the actions in the loop can be executed in polynomial time. These operations are:

1. Choose an element of the substitution set whose foot matches the observation.
2. Substitute the new tree into the partial explanation and
3. Update the substitution set.

The complexity of step 1 is proportional to the number of internal nodes in the tree, and a constant, k, that captures the maximum number of rules for a single non-terminal. Substitution is only possible at internal nodes of the tree, so there can be no more than $2nk$ choices. Step 2 requires a traversal of the original tree to make a copy, and then a traversal of the substituted tree, to insert a copy into the copy of the original tree. This may be done in time linear in the size of the tree. At worst, the tree will have fewer than $3n$ nodes: n nodes for the NNT, n leaf nodes and n internal nodes.[10] Finally, updating the substitution set (step 3) is done by walking the frontier of the tree into which we just substituted. The size of the frontier is always smaller than n, the number of leaves in the full tree at the end of the algorithm. Ergo the algorithm is in nondeterministic polynomial time.

Explanation-finding is NP-hard We can see that explanation-finding is NP-hard by a reduction from 3-dimensional matching [18, pp. 50–53]. 3-dimensional matching is defined as follows: we are given three sets, W, X and Y, and a set of triples, M, each of which specifies an acceptable three-way marriage: i.e., each triple in M is $\langle w, x, y\rangle, w \in W, x \in X, y \in Y$. A solution is a subset of M that covers W, X and Y exactly. A 3-dimensional matching problem may be translated into a plan recognition problem as follows: The sets W, X, Y are translated into a string, with the elements of W first, then the elements of X and then the elements of Y. For each triple, $\langle w, x, y\rangle \in M$, we have a grammar rule $s \rightarrow w, x, y : w \prec x, x \prec y$, where s is an intendable root. The translation is linear in the size of the input problem, and it can readily be seen that we can find an explanation for this problem if and only if there exists a corresponding 3-dimensional matching.

The complexity arguments for top-down explanation-finding carry over to the bottom-up explanation finding algorithm, and thus also apply to the implemented algorithm. Examination of the NP-hardness result above shows that it must apply to bottom-up explanation-finding as well; the explanations that correspond to 3-D match solutions are bottom-up explanations.

[9] In terms of the size of the explanation.

[10] We forbid epsilon productions and productions that simply rewrite one non-terminal into another.

Relation to parsing Viewing PHATT's explanation generation as parsing helps us to identify why the above problem is difficult to solve deterministically. Here Vilain's results on plan recognition as parsing[53] are helpful. There are two aspects of PHATT's explanation grammars that make them difficult to parse. The first is the use of partial orders in the rules. These allow us to encode the equivalent of a factorial number of context-free productions in a single rule. Barton has shown that parsing with such rules in NP-complete[17]. A sketch of the argument is as follows: context free grammars (CFGs) with rules annotated with partial orders subsume unordered CFGs (UCFGs). Vertex cover can be reduced to UCFG parsing. The worst case for the recognition problem is the minimally-constrained one, since that leads to the largest explosion in the number of rules to be considered.

Above and beyond the issue of partial ordering in rules, we have the ability to interleave the strings generated by different root nodes in the grammar. Nederhof, Satta, and Shieber analyze the complexity of parsing in CFGs augmented with a "shuffle" operator[32]. They show that, while the shuffle makes the worst-case complexity of the parsing task exponential, it is $O(|P| \cdot k \cdot g \cdot q \cdot n^3)$. The n^3 factor is the conventional CFG parsing complexity and $|P|$ is the size of the grammar. q is the size of the state space of the automaton parsing each rule and k is the maximum width of the shuffling performed by the grammar. g is another term that measures the amount of memory that must be added to the parser in order to handle the interleaving, and can be reduced to q and k giving us: $O(|P| \cdot k \cdot \left(\frac{q\frac{e}{2}}{k}\right)^k \cdot q \cdot n^3)$. Shuffling width accumulates additively down the parse trees as more subtrees are shuffled together. Nederhof, *et al.* argue that in most parsing applications k should be a relatively small constant. However, in the worst case, for a plan library, k may be $O(dl)$ for d the depth of the grammar and l the maximum number of subtasks on the right-hand size of a rule. As with partial ordering, the problem becomes more difficult when the plan library is relatively *un*constrained temporally, and more top-level goals are mixed together in a single trace.

9.2 Explanations and their Probability

We have shown above that the problem of finding an explanation for a set of observations is NP-complete, and that the worst cases arise when a relatively unconstrained plan library requires us to maintain a large set of hypotheses. Recall from Definition 7.1 that PHATT's job of finding the conditional probability of a given goal is even more difficult, since doing so requires computing the full set of explanations.

We know that, in general, the problem of computing posterior probabilities for a Bayes net is #P-complete [14]. Unfortunately, the general result applies even to the special purpose computations that PHATT must make. Since we know finding a single explanation (top-down or bottom-up) is NP-complete, if we can show that PHATT's probability computation problem is capable of counting the solutions to the 3D-matching problem (without an explosion in the size of the encoding), then we have shown the probability computation problem to be #P-complete. This is easy to do. We can engineer the probabilities on the rules for the 3D-matching problem to ensure that all of the solutions are equally probable. In this case, the reciprocal of the probability of any one of the explanations will tell us the number of solutions to the 3D-matching problem, so the problem is #P-complete.

Some plan recognition researchers have attempted to ease the computational burden by trying to find only a single best, or possibly approximately best, explanation (e.g., a MAP solution)[11, 48, 26]. Unfortunately, even finding a MAP assignment to the variables of a belief network is NP-hard[14, 50].

9.3 Explanation Combinatorics

If PHATT is to find all explanations, then a critical complexity consideration is the rate at which the set of explanations grows, as a function of the input set of observations and the plan library. In particular, we must identify features of the plan library that cause the number of possible explanations to increase with the input length, and give bounds on these effects.

Given our algorithm for generating explanations, a single observation can increase, decrease, or leave unchanged the number of explanations.[11] That is, an observation is either inconsistent with some current explanation(s), in which case that explanation (or those explanations) can be pruned, and number of explanations decreases; or the observation is consistent with all of the explanations under consideration, in which case the number of explanations remains the same or increases. The critical question then is how many elements of the pending set have the observed action as their foot. It will be helpful to define a few terms for this discussion:

[11] We will talk about "the number of explanations," rather than the more exact but cumbersome "number of explanatory hypotheses under consideration."

Definition 9.1 *We define the* **attachment points for an observation σ in an explanation** $\langle D_n, PS(D_0), ...PS(D_n)\rangle$ *as the set of nonterminal instances A for which there exists at least one tree T in $PS(D_n)$ such that* $\text{root}(T) = A$ *and* $\text{foot}(T) = \sigma$.

Attachment points are the nonterminal symbols in the explanation where a new tree could be added into the explanation to account for the next observation. We will call this process of adding a single observation at an attachment point *explaining the observation*.

To explore how many new explanations can result from a single observation we will consider the case of a single observation and a single attachment point and then generalize to multiple attachment points. We will use $T^i_{\sigma,B}$ to refer to a generating tree with foot terminal symbol σ and root nonterminal B. It is possible for there to be more than one such tree and so we superscript the tree itself with i.

As we have said, when σ is observed, the algorithm must create an explanation for each $T^i_{\sigma,B}$ in the substitution set. Thus, if there are N such elements in the substitution set, $T^1_{\sigma,B}, ..., T^N_{\sigma,B}$, that will explain action σ as contributing to B, then N explanations will result from extending the single initial explanation. Keep in mind that if there were no attachment point for the observation, then the explanation could be discarded as being inconsistent and the number of explanations would decrease.

The question we must ask is how large can N be? In general, the number of such trees depends on the grammar. One cause of multiple elements of the substitution set that share a common root and foot pair are cases where recursion happens at the beginning of a plan. In these cases, we can imagine situations that require us to build trees that share the same root and leaf symbols but differ in the number of times a recursive production is invoked within the tree.

As a result, to build the generating trees for our plan grammar, we have bounded the depth of recursion we allow in the grammar. This has the effect of limiting the number of generating trees for the plan library. In this case, if the bound we have placed on the recursion is m, then a single explanation could be extended in exactly m ways, one for each of generating trees addressing the recursion.

Another reason for multiple trees that share a common root and foot pair has to do with the presence of OR-nodes in the plan library. Since an OR-node captures the fact that there are multiple ways to expand a given nonterminal, in the worst case we can construct plan libraries where in all of the alternative expansions share the same first action.

For any given plan library there will be some OR-node with the largest number of alternatives. We denote this number of alternatives by $MaxOrBF$. Since in the worst case each of the OR-nodes in a plan's expansion could have $MaxOrBF$alternatives, the number of such trees, and the resulting size of N for an specific root and leaf pair is bounded above by $MaxOrBF^{MaxD}$, where $MaxD$is the maximum length of any path from a root to a leaf in the original plan library. (Note that by definition $MaxD < m$ if recursive plans are contained in the plan library.)

Since $MaxOrBF^{MaxD}$ bounds N for the pairing of any observed leaf with any particular nonterminal in the plan, we can now give a bound on the overall size of the substitution set in light of the following definition:

Definition 9.2 *We define $|AP_{\sigma,exp}|$ as the number of attachment points in the current explanation, exp, for the current observation.*

Note that $|AP_{\sigma,exp}|$ includes both nonterminals that are part of existing plans as well as nonterminal roots that introduce new plans. In this case $|AP_{\sigma,exp}| * MaxOrBF^{MaxD}$ bounds the size of the substitution set and therefore the growth in the number of explanations with each observation. This confirms our intuitions that the number of explanations can grow quite rapidly in the worst case.

These worst-case results suggest that we must turn away from universal claims about PHATT's performance and focus on the actual performance encountered in particular cases. In the following section, we change our focus and turn to empirical evaluation of PHATT on a number of test cases, manipulating several key problem parameters.

10 Empirical Complexity and Scalability Results

We have conducted a series of experiments, based on our Common LISP implementation of the PHATT algorithm, designed to allow us to understand the most critical factors determining the runtime of the PHATT algorithm. Our initial hypothesis was that while the number of roots in the plan-library might have a large effect on the runtime of the algorithm, in fact, we believed that other features of the plan library would have more impact. Our results did verify this hypothesis. While the number of root plans had a measurable effect on the system's runtime, our experiments also

Factor	Description	Levels
order	Types of action ordering constraints	total, one, partial, unord, last
depth	Plan depth	3, 4, 5, 6
method-BF	Method branching factor	3, 4
choice-BF	Choice point branching factor	3, 4
roots	Number of root goals	10, 100, 200, 400, 600, 800, 1000

Figure 5: Experimental Factors

showed that partial ordering within the plan, especially at the beginning of the plan, has a far more dramatic effect on runtime.

10.1 Experimental Design

Our experiments measuring the runtime for our Allegro Common LISP 7.0 implementation of the PHATT algorithm were conducted on a Sun Sunfire-880 with 8Gb of main memory and 4 750-MHz CPUs, which afforded a large number of replications (1000). Note that measured CPU time (msec) was exclusive of any time used by the operating system or by other processes on the computer.

We identified five features of plan libraries that we believed might have a significant effect on the runtime of the PHATT algorithm: The kind of inter-action ordering constraints in the plans, the depth of the plans in the plan library, the number of intendable roots in the plan library, the branching factor for methods, and the branching factor for choice points. The following are the detailed definitions for each of the experimental factors

- **order**: This is an indication of how many and what type of ordering constraints exist between the actions in the methods (AND-nodes) in the plan library. A graphical representation of these is shown in Figure 6.
  - *Total*: the actions are totally ordered. Each action has a single ordering constraint with the action that precedes it.
  - *One*: each plan has a designated first action. All other actions in the plan are ordered after it but are unordered with respect to each other.
  - *Last*: each plan has a designated last action. All other actions in the plan are ordered before it, but are unordered with respect to each other.
  - *Partial*: Each action may have a single ordering constraint. This constraint orders the action after one other randomly chosen action in the definition. Cyclic orderings are prevented at generation. This means that methods can vary from being totally ordered to completely unordered. This was specifically included to approximate real world plan libraries. In most cases actions will be neither totally ordered nor completely unordered. Such a plan will never have more ordering constraints than the totally ordered case.
  - *Unord*: All of the actions are unordered with respect to each other.
- **depth**: This is a measure of the depth of the plan trees in the plan library. In these plan trees choice points (OR-nodes) and methods (AND-nodes) alternate levels. In all cases the root is defined as an OR-node.
- **roots**: This measures the number of plan root nodes in the plan library at 10, 100, 200, 400, 600, 800, and 1000 roots respectively.
- **method-BF**: This determines the number of actions (branching factor) at a method definition (AND-node).
- **choice-BF**: This determines the number of actions (branching factor) at an choice point (OR-node).

These factors and values are summarized in Figure 5. The discussion of each experiment will document which of the features was a tested factor and which were held constant.

All the actions in the plan libraries were unique. Thus, once an action is observed there is actually no ambiguity about what root intention the action must contribute to. The system can't recognize or leverage this fact, and therefore this does not inherently reduce the runtime of the algorithm. However, in general plans from such libraries will have

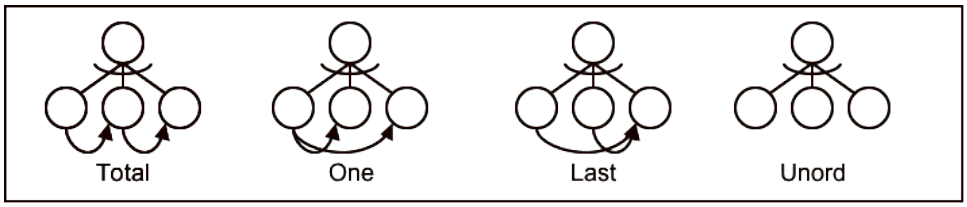

Figure 6: Graphical representations of the different ordering cases used in our experiments: total, one, last, and unord. Note the absence of the partial case. Each order partial plan was randomly built with each child action having at most one constraint pointing to another child node. Cycles in a order partial plans were prevented at construction.

lower runtimes than plan libraries with greater ambiguity in the plans since more ambiguous plans will have more explanations.

Keep in mind that the reduction in ambiguity does not rule out the possibility of more than one instance of a given plan. Therefore we chose to make this simplification to allow us to make several inferences about the space of possible explanations and the effects of various factors on the algorithm's runtime. We will return to discuss this later.

For each experiment, a separate plan library was generated for the experimental conditions. To test each plan library we generated a test data set containing one thousand test cases.To generate a test case three (possibly duplicate) roots were selected at random from the plan library. For each of these roots, a legal plan and linearization of the plan were generated following the plan library. The three complete plan instances were then randomly interleaved maintaining the ordering constraints of the individual plans, resulting in a single test case.

To run a single test, PHATT was started and loaded the plan library. Then for each test case, the internal clock was started, and PHATT was presented with the observed action sequence; after processing the sequence PHATT computed the probability distribution over the root goals. At this point, the clock was halted and the CPU time measured and recorded for the test case. There were test cases where the runtime of the algorithm registered as zero. In these cases, one millisecond was listed as the runtime for the test case.

10.2 First Experiment

We first explored how the algorithm's average runtime scales with the depth of the plans in the plan library. We collected runtimes under the following conditions: The **order** factor fixed at *Total*, the **method-BF** factor fixed at four, the **choice-BF** fixed at three, the **depth** factor varying from three to six, and the number of **roots** at ten, one hundred, and one thousand. Figure 7 shows the resulting average per observation runtime in msec vs. the plan tree depth. Given the log scale, the runtimes show a clear exponential trend across plan libraries with ten, one hundred and one thousand root nodes. Since we know that the runtime for building explanations depends on the size of the plans and the size of the plan depends exponentially on the depth of the tree and its branching factor, this result is not surprising.

We held the **method-BF** and **choice-BF** factors constant at four and three respectively in all of the remaining experiments. We felt this was acceptable since all of our test plans are complete trees, and in the limit the effect of these branching factors is dominated by the depth of the tree. While this exponential relationship should be kept in mind when working with the algorithm, our practical experience suggests that it may not be a significant problem. Our application experience suggests that the depth of hierarchical plans in most real world applications is limited to a relatively small value.

10.3 Second Experiment

Next we collected runtimes in a full factorial experiment for the **order**, **depth**, and **roots** factors across all of their values. Figure 8 plots average per observation runtime measured in msec on a log scale against the number of root goals for each of the **order** factor with the **depth** fixed at four. Note that data for **depth** three and five was also collected but showed the same trends we will discuss here for the **depth** four case. We note that in each case three complete plans were interleaved and presented to the system. None of the experiments reported here were run with only partial plans. However, since we are reporting average per observation runtimes, these figures should be representative for the average case for partial plans as well.

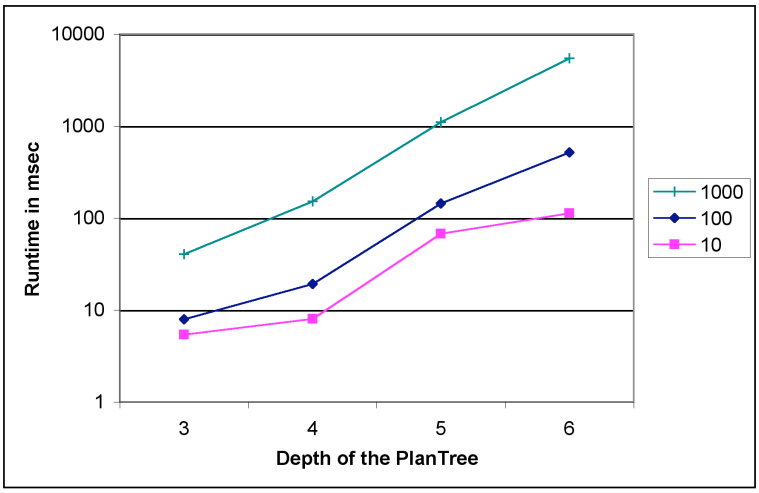

Figure 7: Average Runtime vs. Plan Depth

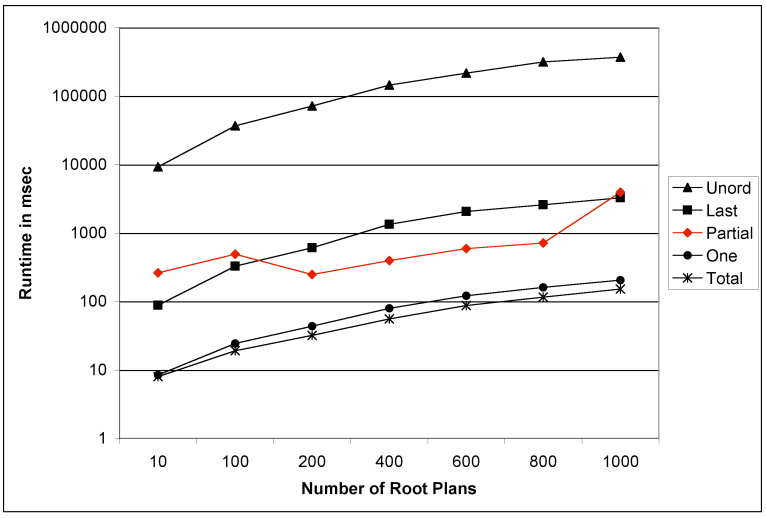

Figure 8: Average Per Observation Runtime with Plan Depth Fixed at Four: Note the runtimes decrease as the ordering within of the plans increases and move earlier within the plan. Also note a very significant increase for the completely unordered case.

The first thing one notices about this data is that the algorithm is scaling linearly in the number of plan roots in the plan library. This is validated across three orders of magnitude and is very encouraging for our use of it in large domains. Note that while the results for the *Partial* runtimes do have a dip between 200 and 800 the overall trend is still linear. We believe this dip in the runtimes to be caused by the natural variability of the complexity of the partially ordered plans, and are examining this effect.

The graph also shows that the Order factor has a profound effect on means. Unordered plans exhibit the highest means; partially ordered plans also have relatively high means. However, the difference between **order** *one* and *total* is not so obvious. Given the similarity of *one* and *total*, we were very interested in determining if there is a significant statistical difference between *total* and *one* levels of **order**, or for that matter, between *partial* and *unord* since this would tell us a great deal about the effect that ordering constraints have on the runtime of the algorithm. The Tukey HSD method (within the analysis of variance) was used on the data from the second experiment to test these contrasts, and verified that all of the lines in Figure 8 represent statistically significant differences in the algorithm's runtime.

Our analysis of the PHATT algorithm in light of these results shows that the difference between the *one* and *total* **order** levels is caused by maintaining larger substitution sets. In **order** *one* cases, after the initial action for a plan is seen all of the other actions are enabled and are added to the substitution set. In *total* cases, there is always only a single next action enabled and in the substitution set. This means that while multiple explanations are not a possibility for either case, the size of the average substitution set will be larger for **order** *one* cases than for **order** *total* cases. Computing and maintaining these larger substitution sets causes the increase in runtime. In the next section, a similar line of reasoning will allow us to explain the significantly higher runtimes of *unord* and *last* **order** levels.

10.4 The Cost of Multiple Explanations

Given our previous discussion, it is not surprising that examination of the PHATT output for data points in the **order** *unord* cases shows that they have a large number of explanations. Since all of the actions are not ordered, PHATT's algorithm is unable to conclude that any particular subset of the actions must all be part of the same plan instance. Remember also that PHATT will create and maintain multiple instances of the same root goal in a single explanation. Thus, in the case of unordered plans the system must maintain explanations that are consistent with all the possible subsets of actions contributing to different plan instances. This even includes the possibility that each action contributes to a separate plan instance. This contrasts sharply with the *total* and *one* levels. In these cases, the ordering of the actions only license a single explanation. In fact, in both cases, since there is a unique first action, there is only ever a single explanation for the observations. The difference between these two cases is accounted for by the larger substitution sets that must be maintained by the *one* levels after the first action is observed.

Our hypothesis that multiple possible first actions in a plan increases the number of maintained explanations and is the cause of significant increases in runtime is also confirmed by the runtimes produced for the **order** *last* test cases. In this case, a large set of explanations will collapse to a single explanation once the final action is observed. Keep in mind that all but one of the actions in these plans are unordered with respect to each other, and all of these actions are required to be executed before the final action. Thus, once the system sees the final action, the only consistent explanation is that all the observations contribute to the same single root goal. As a result they exhibit much higher runtimes than **order** *total* or *one* test cases.

The *last* test cases have the same number of ordering constraints as the *one* test cases removing that variable as possible cause. Further, the constraints are positioned such that *last* and *one* test cases have the same average size of pending sets again removing another possible cause. Since the runtimes for *last* are significantly greater than *one*, we conclude the increased runtimes of the *last* cases must be a result of the larger number of explanations produced for plans with this causal structure.

10.5 Early Closing of Plans

Close inspection of the raw runtimes for the **order** *last* cases from the previous experiment reveal an interesting relationship. There is a significant gap between the cluster of test cases that had the worst runtimes and the rest of the test cases. (See Figure 9.) Inspection of the worst test cases showed that they all shared a common property. In each case, the final three actions of the test were the final three actions of each of the component plans in the test case. For example, consider the following abstract ordered observation stream for three plans a, b and c that each have four steps:

```
{a1, b1, c1, a2, b2, c2, b3, c3, a3, c4, b4, a4}
```

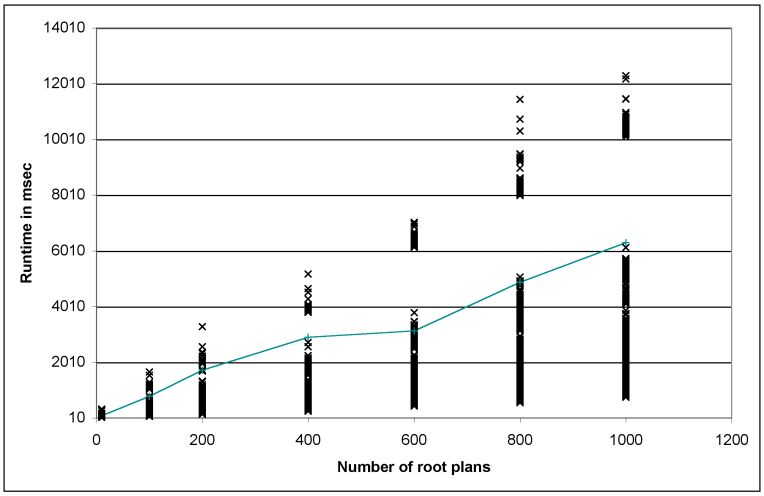

Figure 9: Raw Total Runtimes for Order Last, Depth 4 Test Cases. Note the line connects the average runtime for each of the test cases. A significant gap in the runtimes appears (the open space in each column of results just above the average runtimes) once the number of plans in the test set increases above 400.

note that the last three actions of the series, c4,b4,and a4, are the final actions of each of the respective plans. We will call plans with this property *late closing* test cases. The test case with the next worst runtime had closed at least one plan, one step earlier. In our example, this would be equivalent to swapping actions c4 and a3. We call these cases *early closing* test cases. This small change in the observation stream seemed to be making up to a several second difference in the algorithm's runtime and suggested a final experiment.

10.6 Third Experiment

To determine if early closure of plans caused the reduction in the number of explanations and thereby reduced runtimes, we collected runtimes for test cases that follow the five abstract test cases shown below:

1. a1, b1, a2, b2, b3, a3, c1, c2, c3, b4, a4, c4
2. a1, b1, a2, b2, b3, a3, c1, c2, b4, c3, a4, c4
3. a1, b1, a2, b2, b3, a3, c1, b4, c2, c3, a4, c4
4. a1, b1, a2, b2, b3, a3, b4, c1, c2, c3, a4, c4
5. a1, b1, a2, b2, b3, b4, a3, c1, c2, c3, a4, c4

In each test case, notice that the b4 action moves one time step earlier in the observation sequence. Specifically we took an individual late closing case from the **order** *last*, depth 4, 1000 plan library and generated five observation streams by moving the action corresponding to b4 earlier in the observation stream. Each of these test cases was presented to PHATT 8 times and the runtimes for each were averaged. Figure 10 graphs the average total runtimes for each of the test cases.

As the final action for the 'b' plan moves closer and closer to the beginning of the observation stream, the runtime for the test case drops. Given our previous discussion about the impact of unordered actions in a plan, the cause of this is relatively clear. Since these are instance of the *last* **order** level, the first three actions of each plan are unordered with respect to each other. As a result, PHATT cannot assume that all of the actions for a particular goal/plan contribute to a single instance of that plan. However, since all the actions are ordered before the final action of each plan, once PHATT sees the final action of a plan it can throw out any explanation that does not have all four of the observed actions for that plan contributing to a single instance. Thus, when PHATT is presented with the b4 action it can eliminate any explanation that involves more than a single instance of the plan for b. This results in a radical reduction in the number

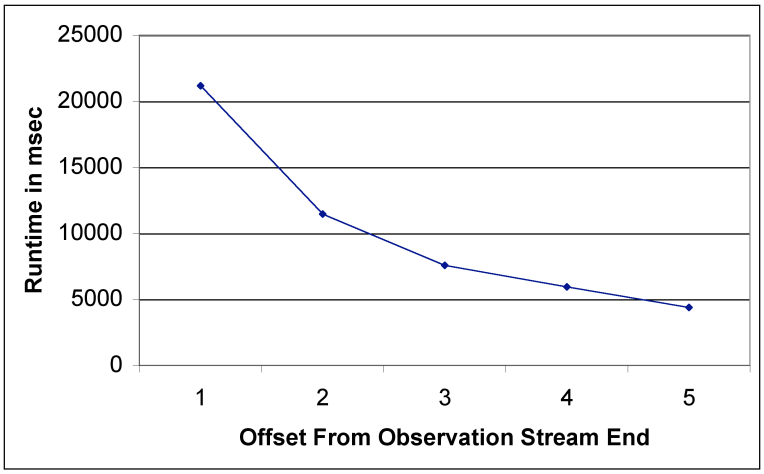

Figure 10: Average Total Runtimes for Early Plan Closing Test Cases

of possible explanations for the observation and a subsequent reduction in the overhead associated with keeping these explanations active.

In summary then, the major conclusions that we can draw from this set of experiments.

- The average runtime for the algorithm is scaling linearly in the number of roots in the plan library.
- The feature of the plan library that has the most significant effect on the algorithm's runtime is the ordering constraints within the plan library, followed by the number of roots in the plan library, followed by the actual depth of the plan trees.
- Plan libraries without ordering constraints represent an upper bound or worst case for the ordering factor.
- Plans with early ordering constraints result in significantly reduced runtimes over plans that have a number of unordered initial actions.
- Ordering constraints, even at the end of plans can significantly reduce the algorithm's runtime.
- Maintenance of a large number of possible explanations is a significant cost to the algorithm.

We are continuing our empirical analysis of the PHATT algorithm. We are conducting more experiments to further study the effect of plan branching factors and incomplete trees on the algorithm's runtime. While this further study is needed, the current results are very promising for the application of the PHATT algorithm.

11 Extensions for Real World Use

Thus far, all the plans we have discussed are propositional. However, propositional representations are too limiting for real world applications. Therefore we extended PHATT's action's with typed arguments and temporal constraints. This section will briefly outline these extensions. For this discussion we will use the action specification language used in our implementation. This language is equivalent to the definitions provided in Section 5. For example, in Section 5 we defined the first level of the plan for **Theft** as: **Theft**→ **scan get-ctrl get-data**: {(1,2) (2,3)}. In the language we will use here, this would be written with the obvious mapping as:

```
(def-method '(Theft) '((scan) (get-ctrl) (get-data)))
(def-precond 2 '(1))
(def-precond 3 '(2))
(endm)
```

This representation will allow us to more easily annotate actions plans with additional arguments and constraints.

11.1 Typed Action Variables

PHATT supports a limited form of typed action arguments. This requires extending the representation language for the plan library with variables, and defining a method for handling binding and propagation of their values within in the plan recognition algorithm. Consider adding typed arguments to our above example:

```
(def-method '(Theft (file :type :data-file) (location :type :computer))
    '((scan location)
      (get-ctrl location)
      (get-data file location))
(def-precond 2 '(1))
(def-precond 3 '(2))
(endm)
```

This definition now extends our previous example by specifying a typed variable for the specific file being stolen and computer it is being stolen from. We also note that the use of a common name within a definition indicates co-reference of the argument. Thus, the same location must be used in all three sub-actions of the definition.

Rather than supporting a full-blown first order representation, these variables function as argument place holders supporting value propagation and co-reference. The values for these arguments will be bound by the process of explaining an observation and are always assumed to be existentially quantified. In a probabilistic model like the one used by PHATT, supporting universal quantification of such arguments would require significant extra probabilistic machinery. Such a system would have to build a probability distribution over the set of all possible assignments to the variables, greatly increasing the algorithm's runtime. Instead, variables are bound when the first action that refers to them is added to the explanation. These bindings propagate upwards to actions at higher levels within the plan library enabling the binding of variables at lower levels to determine the bindings for actions that have not yet been observed.

For example, **scan** has one argument *location*. When PHATT builds the explanation structure for a specific observation of **scan** action, it binds the *location* variable and propagates the bindings up the explanation to the **Theft** action. Given that *location* has been bound, any observation of a **get-ctrl** action that involved a different binding for *location* would be incompatible with the previously observed **scan** action and can therefore be pruned. To aid this process, PHATT requires any arguments to an action that are to be propagated up the explanation must occur in at least one of the sub-actions of the parent action's definition.

Adding types to the PHATT variables enables further pruning of explanations. Consider our example, it has restricted the *location* argument to being of type *computer* and *file* to be of type *data-file*. These type restrictions prevent PHATT from using this plan to recognize cases of theft that involve stealing computer applications or don't happen on a computer network at all.

PHATT extends any type system by supporting simple negation of types. This allows the domain designer to specify the type of object that can not be bound to a specific variable. This is done through the keyword *:not* in the type definition. This type negation might seem to be of limited value since it only removes a single possible type from consideration. However, when coupled with the bottom up binding variables it can be quite valuable.

Variables in PHATT are bound by specific observations, and each observation is generated by a specific sensor that imposes limits on the possible values for the argument. This means the arguments to the observation will only ever fall within a limited range of values. This limits the scope of the action argument to a subset of types making type negation a much stronger tool. Consider the following case:

```
(def-method '(Theft (file :type :not :application) (location :type :computer))
    '((scan location)
      (get-ctrl location)
      (get-data file location))
(def-precond 2 '(1))
(def-precond 3 '(2))
(endm)
```

In this case the observation of a **get-data** action is assumed to only ever report an identifier for a file from a computer as the first argument. Since the action arguments within the explanation are bound by explaining the observation of the **get-data** action this argument is thereby constrained to only ever be a file on the computer. By negating the type application, the already limited set of electronic files can be further restricted to the set to non-application files.

We have implemented these ideas by augmenting PHATT with a very flexible argument type framework. The plan library designer specifies both the types and the system for computing type equivalence or subsumption. We have tested this framework by implementing three different type systems in PHATT: 1) A simple set of fixed types that use equality testing for subsumption, 2) a hand-built type hierarchy and subsumption test, and 3) the FaCT[29] system. All of the type systems we have been successfully integrated in PHATT. However, while each of these type systems have been fully implemented and extensively tested (with excellent results), we have not formally evaluated the additional runtime or memory requirements imposed by the use of the variable and type systems.

Typed arguments can allow PHATT to prune explanations, and may result in significant reductions in PHATT's search space. Further, with careful bookkeeping, the cost of the binding and propagating variables can be done with very minimal (in many cases constant) cost. This means typed argument may enable significant reductions in PHATT's overall runtime. However, this depends critically on the strength and complexity of the type system used and may vary considerably, depending on specific features of the plan library. Thus the magnitude of the savings that results from this can vary greatly depending on the representational choices made in the construction of the plan library and the cost of the subsumption tests for the type system.

To summarize:

- All of the variables within PHATT plan libraries are assumed to be existentially quantified and must be bound by an observation.
- Propagation of a variable binding starts at observations and propagates upwards.
- The system does not support explicit non-codesignation constraints. That is, while in PHATT all uses of the same variable name in a definition are assumed to refer to the same entity, the system has no method for explicitly stating that two variables with different names cannot refer to the same entity.
- While the system does support type negation, type variables and type variable codesignation are currently not supported. Especially in domains with hierarchical argument types, there are cases where it would be helpful to be able to specify that an action's arguments must have the same type but not be concerned with the actual type instance.

11.2 Temporal Constraints

Temporal constraints can be very effective in pruning the space of possible explanations. Many tasks have simple temporal relations between their subtasks, and considering explanations that violate these bounds is unreasonable. To allow PHATT to prune possible explanations based on these kinds of temporal constraints the Interval Constraint Engine (ICE) [3, 4, 5] has been integrated with PHATT.

ICE is a solver for Simple Temporal Problems (STP)[15]. A STP is a temporal constraint problem in which all of the constraints have the form $T_i <= T_j + C$, where T_i and T_j are timepoints. At it's heart ICE is a Bellman-Ford shortest path algorithm[2, 37] with incremental maintenance of two spanning trees, giving tightest bounds on earliest start/latest end times. The complexity of this algorithm is $O(mn)$ where m is number of vertices, and n is the number of edges in the temporal graph. However it has an incremental mode allowing for the on-line addition of constraints, a critical need for the evolving explanations built by PHATT. In practice, ICE can handle incremental updates in near constant time making it an excellent choice for integration with PHATT. In order to use ICE in PHATT, the allowed temporal constraints had to be restricted to only those that would result in a STP. To this end PHATT only supports two kinds of temporal constraints: overall duration, and inter-sibling constraints.

A duration constraint captures the overall duration for a single action. For example, for most people, taking more than an hour to make lunch would be unusual. We can code this into the rules for recognizing "normal lunch events", and if an agent violates this requirement, PHATT should not recognize this as a normal lunch event. It may be that the lunch plan has been abandoned or it may be a special occasion, but in either case PHATT should not consider this a normal lunch event. This kind of limitation can be captured by a simple restriction on the duration of the lunch action that requires that it not take longer than an hour to make lunch. Note that if duration constraints are placed on actions that have sub-actions this can actually force the sub actions to overlap. Consider the case of an action that has three sub-actions where the parent action is constrained to take no more than 3 time units, if any of the actions takes longer than one time unit the sub-actions must be overlapped in time or the explanation is inconsistent.

Inter-sibling temporal constraints constrain the temporal distance between the beginning or the end time points of two sibling actions. Consider the case of starting a car by turning the ignition key and depressing the gas. If the gas is

not depressed within a short proximity of the beginning of the key turning action the car will not start. The battery will wear down and no amount of gas will start the car. In this case, a temporal constraint is needed between the start times of the two sibling actions. Again, if both of these actions are seen, but they are not in the correct temporal relation PHATT should not consider any explanation of the actions that ascribes them to a car starting goal.

By specifying these temporal constraints on the basis of the beginning and end time for the actions, for ordered actions, we may also be implicitly defining a maximum time duration for one or both of the actions. Consider the case of two sibling actions α and β that are sequentially ordered. Now if we add a constraint such that β_{end} must be within 5 time units of α_{begin} then both α and β must each individually be less than 5 time units as well as their sum.

Both duration and inter-sibling constraints are representable as:

$$timePoint_1 <= timePoint_2 + C$$

where the time points in the case of a duration constraint are the begin time and end time of a single action, and in the case of an inter-sibling constraint they are begin or end times from two actions that are siblings in an action definition. It is this restriction that allows us to enforce that our temporal constraints will form a STP.

Thus with temporal constraints restricted to form an STP we know that we can make use of ICE's near constant time processing to efficiently prune explanations that violate the temporal restrictions defined within the plan. Our experience with the ICE integrated into PHATT suggests that while it does have a measurable effect on the algorithms runtime it does not dominate the effects of other domain features we discussed in Section 10.

12 Future Work and Conclusions

We have presented a plan recognition algorithm, the PHATT algorithm, that exploits a Bayesian model. The Bayesian model clarifies a number of knotty issues in plan recognition, including reasoning to a best explanation, using negative evidence, agents with multiple concurrent goals. We have presented an implementation of the PHATT algorithm that uses precomputed explanation trees for greater efficiency, and have given analytical and experimental complexity results. We have also discussed enhancements to this algorithm to incorporate temporal and type constraint management techniques.

Our work opens up many interesting directions for future investigation. We think the following seven areas are particularly interesting:

Goal abandonment Most real agents are not infinitely persistent. In the face of failures, or even simply through inattention, they will often abandon goals. Most previous plan recognition systems have not taken this into account. We have developed a technique, based on Bayesian reasoning about model mismatch, for identifying goal abandonment[23]. This approach makes use of the same kind of decrease in probability we discussed for negative evidence to identify when a goal has been abandoned. We are continuing to explore the impact this approach has on the runtime of the system.

Partial observability In many applications, it is impossible to see all of the actions that an agent carries out. There may be some actions that are simply never seen. For example, many factories are partially automated. In such factories, some control actions can be performed through a computer console, and such actions are readily observed. However, other tasks must be performed by field operators turning valves on and off with wrenches, and these actions will not be (directly) perceived by the plan recognition system. In other cases, we may have noisy sensors, so that the observation sequence will be a stochastic function of the actual action sequence. For example, if we track an agent's motion using GPS or wifi triangulation, we may lose track when the agent moves indoors or leaves the area of the wireless access points. For this reason we have begun to look at the problems of partial observability[19].

As a matter of theory, it is trivial to extend our model to handle partial observability. So far, we have considered the set of observations to be a complete and accurate trace of the action sequence, so that we have

$$P(exp, obs) = P(exp)P(obs|exp)$$

We may augment this with an observation model to give:

$$P(exp, obs) = P(exp)P(acts|exp)P(obs|acts)$$

(assuming that the observation probabilities depend only on the action to be observed).

While the theoretical extension is straightforward, implementing the theory has raised a number of difficult issues. Even simple deterministic models of observations, such as our example of the field operator actions, can cause the set of hypotheses to explode in a way similar to plan abandonment. For stochastic models, we need to develop procedures for inverting the observation models that take into account context (currently active explanatory hypotheses). Work on layered HMM approaches will be helpful here. However, as the localization example illustrates, the approach to inverting the observation model must be sensitive to the model's specific structure — in the case of wifi localization, for example, particular characteristics of geometry, radio performance, etc.

Influence of state Our model has not discussed the effect of state on plan recognition. However, obviously the state of the world can have a profound effect on the goals adopted, the plans used to achieve those goals and even the order in which actions are selected from the pending sets. We are interested in a number of simple ways in which state variables could be used to influence the probabilistic models of these system level features. We are also interested in ways in which observed state changes can be used to infer the performance of unobserved actions.

Failures State modeling will be necessary in order to do plan recognition in contexts where actions and methods can fail to achieve their desired ends. In addition to incorporating state effects, we will need a more sophisticated model of the agents' internal state. So far, we have been able to treat the agents as if they have chosen the complete decomposition of their plan at time zero. Now we will need a more complex model that can incorporate choices that occur later (in order to be sensitive to state at the time of method choice). We must also include in our models how the agents will react when they perceive one of their methods to have failed. We are currently working on a semantic web domain in which agents need to gather information about resource availability and suitability before they commit resources. Execution traces in this domain will necessarily include cases where an agent attempts to verify that resource r is suitable for use in a goal, only to find that it is not, and have to change to a new method based on some other resource, r'.

Hostile agents and intended recognition It may seem odd that we have grouped together reasoning about hostile agents and intended recognition, since the agents in these two areas have exactly opposite objectives. However, the two applications have one important thing in common: they raise game-theoretic concerns. That is, in both cases the agent executing the plan will be reasoning about the reasoning done by the plan recognizer. Hostile agents will try to turn that reasoning on itself to force it to the wrong conclusions, whereas in intended recognition, the agent will be trying to force recognition to the right conclusions, by using conventional methods, explicitly eliminating alternatives from consideration where there is ambiguity, etc. We have done a great deal of work in the area of hostile agents (notably in computer security)[19], but not using game-theoretic considerations. Instead, we have assumed that the hostile agents do not do *specific* reasoning about their opponents, but only try standard gambits to elude detection, etc. For this domain (where, indeed, many of the "opponents" are scripts, rather than actual intelligent agents), the simplifying assumption works fairly well, but for game-playing, etc., the approach is obviously insufficient.

Learning As do many other AI approaches, PHATT suffers from a knowledge engineering bottleneck. It is difficult to build and maintain plan libraries for recognition and to reliably assess their probability parameters. Fortunately, as with other probabilistic systems, PHATT is not terribly sensitive to the priors except in extreme cases such as our terrorism versus air travel example (and these are gross effects). However, the library construction problem is still a grave one. There has been a great deal of work on training HMMs for applications such as speech recognition and natural language processing, and some of this work has begun to be adopted in plan recognition as well. The PHATT algorithm shows the connection between good old-fashioned plan recognition, with its expressive models, and HMMs. We hope that this will lead to techniques for learning more expressive models of the type used by PHATT.

Reasoning about types, quantification, and equality In Section 11 we have discussed our use of types in handling parametrized actions. However, PHATT is limited to extremely simple uses of parametrization, notably those that can be treated simply as filtering, and in which variable bindings are simply propagated by unification. In some cases, we need more complicated reasoning. For example, if we have a plan for air travel that involves taking a train to the airport, we might have a constraint that the `destination-of` the train trip must be equal to the `train-station-of` the `starting-airport-of` the air travel. Note that the introduction of the `train-station-of` function means

that this constraint cannot directly be handled by simple symbol unification. Charniak and Goldman did extensive work on this kind of equality reasoning, in the context of plan recognition for story understanding [9]. Not only does equality reasoning arise in such problems, but we may find that the prior probability of plans must be conditioned on the values assigned to their parameters. Consider, for example, a plan library in which there is an action for "going." "Going" to Antarctica, for most of us, is going to be a much lower probability event than "going" to the nearest supermarket. Problems of quantification also arise in domains like the semantic web, and in softbots, where there may be variables drawn from non-finite (or only practically infinite) domains of quantification, for example files in a computer, which can be created and destroyed. Our current work on planning for the semantic web is leading us to examine these issues again.

Acknowledgments

This article was supported by DARPA/IPTO and and the Air Force Research Laboratory, Wright Labs under contract number FA8650-06-C-7606, and based upon earlier work supported by DARPA/ITO and the Air Force Research Laboratory under Contract No. F30602-99-C-0077.

References

[1] D. Avrahami-Zilberbrand and G. A. Kaminka, "Fast and Complete Symbolic Plan Recognition," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2005.

[2] R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[3] M. Boddy, "Temporal Reasoning for Planning and Scheduling: Lessons Learned," in *Advanced Planning Technology*, A. Tate, editor, AAAI Press, May 1996.

[4] M. Boddy, J. Carciofini, and G. Hadden, "Scheduling with Partial Orders and a Causal Model," in *Proceedings of the Space Applications and Research Workshop*, Johnson Space Flight Center, August 1992.

[5] M. Boddy and R. Goldman, "Empirical Results on Scheduling and Dynamic Backtracking," in *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation for Space*, 1994.

[6] H. H. Bui, S. Venkatesh, and G. West, "Policy Recognition in the Abstract Hidden Markov Model," *Journal of Artificial Intelligence Research*, vol. 17, pp. 451–499, 2002.

[7] H. Chan and A. Darwiche, "When do numbers really matter?," *Journal of Artificial Intelligence Research*, vol. 17, pp. 265–287, September 2002.

[8] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*, Addison-Wesley, Reading, MA, 1987.

[9] E. Charniak and R. P. Goldman, "Plan Recognition in Stories and in Life," in *Uncertainty in Artificial Intelligence 5*, M. Henrion, R. Schachter, and J. Lemmer, editors, pp. 343–351, Elsevier Science Publishing Co.,Inc., New York, NY, 1990.

[10] E. Charniak and R. P. Goldman, "A Bayesian Model of Plan Recognition," *Artificial Intelligence*, vol. 64, no. 1, pp. 53–79, 1993.

[11] E. Charniak and S. E. Shimony, "Cost-Based Abduction and MAP Explanation," *Artificial Intelligence*, vol. 66, no. 2, pp. 345–374, 1994.

[12] P. R. Cohen, C. R. Perrault, and J. F. Allen, "Beyond Question Answering," in *Strategies for Natural Language Processing*, W. Lehnert and M. Ringle, editors, pp. 245–274, Lawrence Erlbaum Associates, 1981.

[13] C. Conati, A. S. Gertner, K. VanLehn, and M. J. Druzdzel, "On-Line Student Modeling for Coached Problem Solving Using Bayesian Networks," in *Proceedings of the Sixth International Conference on User Modeling*, 1997.

[14] G. F. Cooper, "The Computational Complexity of Probabilistic Inference Using Bayesian belief Networks," *Artificial Intelligence*, vol. 42, pp. 393–405, 1990.

[15] R. Dechter, I. Meiri, and J. Pearl, "Temporal Constraint Networks," *Artificial Intelligence*, vol. 49, no. 1, pp. 61–95, 1991.

[16] K. Erol, J. Hendler, and D. 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)*, pp. 249–254, 1994.

[17] J. G. Edward Barton, "On the Complexity of ID/LP Parsing," *Computational Linguistics*, vol. 11, no. 4, pp. 205–218, 1985.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W.H. Freeman and Company, New York, 1979.

[19] C. Geib, "Plan Recognition," in *Adversarial Reasoning*, A. Kott and W. McEneaney, editors, Chapman and Hall/CRC, 2006.

[20] C. W. Geib and R. P. Goldman, "Partial Observability in Collaborative Task Tracking," in *Proceedings of the AAAI 2001 Fall Symposium on Collaborative task Tracking*, 2001.

[21] C. W. Geib and R. P. Goldman, "Plan Recognition in Intrusion Detection Systems," in *Proceedings of DISCEX II, 2001*, 2001.

[22] C. W. Geib and R. P. Goldman, "Probabilistic Plan Recognition for Hostile Agents," in *Proceedings of the FLAIRS 2001 Conference*, 2001.

[23] C. W. Geib and R. P. Goldman, "Recognizing Plan/Goal Abandonment," in *Proceedings of IJCAI 2003*, 2003.

[24] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*, Morgan Kaufmann Publishers, Inc., 2004.

[25] R. P. Goldman, C. W. Geib, and C. A. Miller, "A New Model of Plan Recognition," in *Proceedings of the 1999 Conference on Uncertainty in Artificial Intelligence*, 1999.

[26] J. R. Hobbs, M. E. Stickel, D. E. Appelt, and P. A. Martin, "Interpretation as Abduction.," *Artificial Intelligence*, vol. 63, no. 1–2, pp. 69–142, 1993.

[27] A. Hoogs and A. A. Perera, "Video Activity Recognition in the Real World," in *Proceedings of the Conference of the American Association of Artifical Intelligence (2008)*, pp. 1551–1554, 2008.

[28] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, 1979.

[29] I. Horrocks, "Using an Expressive Description Logic: FaCT or Fiction?," in *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR '98)*, 1998.

[30] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse, "The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users," in *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998.

[31] M. J. Huber, E. H. Durfee, and M. P. Wellman, "The Automated Mapping of Plans for Plan Recognition.," in *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 344–351, 1994.

[32] M. Jan Nederhof, G. Satta, and S. M. Shieber, "Partially Ordered Multiset Context-Free Grammars and ID/LP Parsing," in *Proceedings of the Eighth International Workshop on Parsing Technologies*, pp. 171–182, Nancy, France, April 2003.

[33] A. Joshi and Y. Schabes, "Tree-Adjoining Grammars," in *Handbook of Formal Languages, Vol. 3*, pp. 69–124. Springer Verlag, 1997.

[34] G. Kaminka, D. Pynadath, and M. Tambe, "Monitoring Teams by Overhearing: A Multi-Agent Plan-Recognition Approach," *Journal of Artificial Intelligence Research*, vol. 17, no. 1, pp. 83–135, 2002.

[35] H. Kautz, *A Formal Theory of Plan Recognition and its Implementation*, PhD thesis, University of Rochester, 1991.

[36] H. Kautz and J. F. Allen, "Generalized plan recognition," in *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI-86)*, pp. 32–38, 1986.

[37] J. L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, 1962.

[38] L. Liao, D. Fox, and H. Kautz, "Learning and Inferring Transportation Routines," in *Proceedings of AAAI 2004*, 2004.

[39] L. Liao, D. Fox, and H. Kautz, "Extracting Places and Activities from GPS Traces Using Hierarchical Conditional Random Fields," in *International Journal of Robotics Research*, volume 26, pp. 119 – 134, 2007.

[40] L. Liao, D. Fox, and H. A. Kautz, "Location-Based Activity Recognition using Relational Markov Networks," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pp. 773–778, 2005.

[41] J. McCarthy, "Circumscription — A form of non-monotonic reasoning," *Artificial Intelligence*, vol. 13, pp. 27–39,171–172, 1980.

[42] J. Modayil, T. Bai, and H. Kautz, "Improving the Recognition of Interleaved Activites," in *Proceeding of the Tenth Interantional Conference on Ubiquitous Computing*, 2008.

[43] M. Peot and R. Shachter, "Learning From What You Don't Observe," in *Uncertainty in Artificial Intelligence,Proceedings of the Fourteenth Conference*, pp. 439–446, San Francisco, CA, 1998, Morgan Kaufmann Publishers.

[44] D. Poole, "Logic Programming, Abduction and Probability: a top-down anytime algorithm for estimating prior and posterior probabilities," *New Generation Computing*, vol. 11, no. 3–4, pp. 377–400, 1993.

[45] D. Poole, "Probabilistic Horn Abduction and Bayesian networks," *Artificial Intelligence*, vol. 64, pp. 81–129, November 1993.

[46] M. Pradhan, M. Henrion, G. Provan, B. D. Favero, and K. Huang, "The Sensitivity of Belief Networks to Imprecise probabilities: an experimental investigation," *Artificial Intelligence*, vol. 85, no. 1–2, pp. 363–397, August 1996.

[47] D. Pynadath and M. Wellman, "Probabilistic State-Dependent Grammars for Plan Recognition," in *Uncertainty in Artificial Intelligence,Proceedings of the Sixteenth Conference*, pp. 507–514, 2000.

[48] E. Santos, Jr., "A Linear Constraint Satisfaction Approach to Cost-Based Abduction.," *Artificial Intelligence*, vol. 65, no. 1, pp. 1–28, 1994.

[49] C. Schmidt, N. Sridharan, and J. Goodson, "The Plan recognition problem: an intersection of psychology and artificial intelligence," *Artificial Intelligence*, vol. 11, pp. 45–83, 1978.

[50] S. E. Shimony, "Finding MAPs for Belief Networks Is NP-Hard," *Artificial Intelligence*, vol. 68, no. 2, pp. 399–410, 1994.

[51] C. L. Sidner, "Plan parsing for intended response recognition in discourse," *Computational Intelligence*, vol. 1, no. 1, pp. 1–10, 1985.

[52] D. L. Vail and M. M. Veloso, "Feature Selection for Activity Recognition in Multi-Robot Domains," in *Proceedings of the Conference of the American Association of Artifical Intelligence (2008)*, pp. 1415–1420, 2008.

[53] M. Vilain, "Deduction as Parsing," in *Proceedings of the Conference of the American Association of Artifical Intelligence (1991)*, pp. 464–470, 1991.

[54] M. P. Wellman, J. S. Breese, and R. P. Goldman, "From Knowledge Bases to Decision Models," *Knowledge Engineering Review*, vol. 7, no. 1, pp. 35–53, 1992.

[55] R. Wilensky, *Planning and Understanding*, Addison-Wesley, 1983.