On plan recognition and parsing

John Maraist
SIFT, LLC*
jmaraist@sift.info

Christopher W. Geib
University of Edinburgh†
cgeib@inf.ed.ac.uk

Robert P. Goldman
SIFT, LLC
rpgoldman@sift.info

Probabilistic plan recognition systems based on weighted model counting all work roughly the same way: first they compute the exclusive and exhaustive set of models that explain a given set of observations; next they assign a probability to each model; finally they compute the likelihood of a particular goal by summing the probability of the explanatory models[1] in which that goal occurs. In this paper we discuss an optimization for the model-building first step: rather than retain the full tree-like structure of goals which have been partially observed, we can keep only the *frontier* of as-yet unobserved actions and unachieved subgoals. The system *Yappr* which we present here uses techniques familiar from parsing algorithms. We give an informal introduction to Yappr in Section 1, a more formal presentation in Section 2, and an analysis of Yappr's complexity in Section 3. In Section 4 we present experimental result showing the improvement realized from this technique, and then conclude with discussions of the algorithm's limitations, and future work.

1 Introduction

The starting point for our work is Geib and Goldman's system PHATT [2003; 2009]. PHATT models contain collections of partially expanded hierarchical plans [Ghallab *et al.*, 2004] for specific goals that account for the observed actions. PHATT builds these plans incrementally, from left to right, by adjoining leftmost, depth-first plan tree fragments to the existing plans. To extend an existing plan these leftmost trees are attached at the plan's *frontier*, in the position of unachieved subgoals at trees' leaves. There may be multiple ways to incorporate a single observation into an explanatory model: different ways of incorporating the observation yield multiple trees that could be adjoined into the model. In such cases, PHATT makes multiple copies of the original model, and each tree is inserted into its own copy of the plan structure expanding the set of explanations.

Here we make two observations about PHATT's operational behavior. First, this approach to plan recognition is reminiscent of recursive descent parsing algorithms [Steedman, 2000]: PHATT builds its trees in the same way a parser might label a tree root with the grammar's starting symbol, expand the tree with nodes corresponding to further nonterminals, finally producing a parse tree with nonterminals labelling branches and terminals labelling leaves. However, traditional parsing methods alone cannot address plan recognition. The most pressing reason for this is our commitment to allowing for the execution and recognition of multiple, concurrent, interleaved plans. Standard parsing algorithms like CKY [Younger, 1967] are designed for a single sentence at a time and usually assume that they will observe a complete sentence. Human language speakers do not interleave multiple sentences in the same way they interleave execution of plans. People regularly engage in such multitasking; our plan recognition algorithms should be capable of recognizing these situations and their component plans. Of course it is not surprising that existing parsing algorithms make such assumptions, but their reliance on these assumptions does make them inappropriate for our task.

The second observation about PHATT's operational behavior is that during explanation construction, the part of the plan's tree structure above the frontier is not used; only the frontier is required. Any approach which copies that structure will incur significant computational costs. It is the frontier (in combination with observed actions) that determines the new trees which can be added to the model; all additions to the explanation's structure are made at the frontier. Moreover, with a straightforward transformation of the plan library we will not need the tree structures even to extract the plan's root-level goals. Since all interactions with the explanation take place at the frontier, we can dispense with the tree structure, maintain only the tree frontier, and avoid unnecessarily creating and copying interior tree structure in the model's plans.

We will show that the frontier of the trees in an explanatory model can be represented as a string of nonterminals and ordering constraints from a grammar that defines the original plan library to be recognized. By careful precomputation, we can build a plan grammar that explicitly models only the frontiers and ordering constraints of the leftmost depth-first trees from the original plan library. We will then show how to use the new grammar to build explanatory models that maintain just the frontiers of the plans in the model. This will allow us to build and maintain equivalent but significantly smaller

*Smart Information Flow Technologies, 211 North First Street, Suite 300, Minneapolis, MN 55401-1480.

†School of Informatics, 10 Crichton Street, Edinburgh, EH8 9AB, Scotland.

[1] Since models in these systems explain the observations in terms of plans that are being executed, we will use the terms *model*, *explanation*, *explanatory model*, and *hypothesis* interchangeably.

model structures.

Intuitively, the approach we will argue for with Yappr has three high level steps:

1. Offline, precompile the plan library into a *plan frontier fragment grammar* (PFFG).
2. Use the resulting PFFG to build explanations for the observed actions. This process will look very much like a string rewriting or graph editing process.
3. Maintain the PFFG productions and the order in which they are used in building each of the explanations. This information will allow us to build more complete explanations in the case of queries for information beyond the root goals being followed.

Strictly speaking, this approach corresponds to *goal* recognition rather than *plan* recognition [Blaylock, 2005]: the latter calculates the complete plan structure being followed by the agent, as opposed to identifying only the top-level goals. We see Yappr as a middle road: the root goals being pursued are the most common probabilistic query for such systems. Therefore, we argue for an algorithm that builds restricted models that allow us to quickly answer this question while still allowing the system to reconstruct complete plan models if a probabilistic query is made about details not available in our restricted models.

2 The Yappr system

Plan libraries

The foundation of any plan recognition system is a collection of plans to be recognized. These plans must be specified in a formal language.

Definition 1 *We define a* plan library *as a tuple* $\langle \Sigma, NT, R, P \rangle$ *where,* Σ *is a finite set of basic actions or terminal symbols,* NT *is a finite set of methods or nonterminal symbols,* $R \subseteq NT$ *is a distinguished set of intendable or root nonterminal symbols, and* P *is a set of production rules of the form* $A \rightarrow \alpha : \phi$ *where:*

- $A \in NT$.
- α *is a finite multiset of symbols from* $\Sigma \cup NT$.
- ϕ *gives a partial order on indices into* α.

The similarity between this definition and the usual formalism for context-free grammars (CFGs) is obvious. We have extended the traditional notion with multiple starting symbols and the explicit ordering constraint relation ϕ. These ordering constraints indicate when actions must be performed in a specific order. Traditional CFGs productions take α to be a string instead of a multiset, with ϕ given by the total order of the symbols in the string. Here we assume that the actions in α are unordered unless that ϕ states otherwise.

In a traditional CFG partial ordering is handled by replicating those rules that contain partial ordering, one production rule for each of the possible orderings for the actions. Since a system of the kind we are describing would be forced to consider all of the possible orderings for the actions, using a traditional CFG could result in a catastrophic increase in the computational cost of the algorithm.

This formulation is also subtly different than in traditional hierarchical task networks (HTNs) [Ghallab *et al.*, 2004]. Some formulations of HTNs allow arbitrary applicability conditions that must be true before a production can be used. The plan library defined here is strictly less expressive than these formulations of HTNs, but equivalent to HTN formulations without these additional conditions.

Definition 2 *Given a rule* $\rho = A \rightarrow \beta : \phi$ *We say* $\beta[i]$ *is a* leftmost child of A given ρ *if* $\nexists j$ *such that* $(j, i) \in \phi$.

We write $\mathcal{L}(\rho)$ for the set of all leftmost children given ρ; this set describe exactly those symbols that are required to be first in any expansion of the parent nonterminal by ρ. Note that this definition does not require that there be a unique leftmost symbol of a rule. We use $\mathcal{R}(\rho)$ to denote the set of all symbols that are not leftmost of ρ.

Definition 3 *Given a plan library* $\langle \Sigma, NT, R, P \rangle$ *we define a* leftmost tree T deriving α *as a tree such that:*

- *Every node in T is labeled with a symbol from* $\Sigma \cup NT$.
- *Every interior node in T is labeled with a symbol from* NT.
- *If an interior node n labeled A in T has children with labels* $\beta_1, ...\beta_k$, *then*
  - $\exists \rho \in P$ *such that* $\rho = A \rightarrow \beta_1 ...\beta_k : \phi$,
  - *Node n is additionally annotated with* ρ
  - *No children of n labeled with symbols in* $\mathcal{R}(\rho)$ *have children of their own.*
  - *At most one child of n labeled with a symbol in* $\mathcal{L}(\rho)$ *has children of its own.*
- *There is a distinguished node in the frontier of T labeled with the terminal symbol* α *that is leftmost for its parent. We call this the* foot *of the tree T.*

Leftmost trees correspond very closely to minimal, leftmost, depth-first, derivation trees for a specific terminal in traditional CFGs. In this case, the ordering relation defined for the plan library is used to determine which nonterminals are leftmost. We will use leftmost trees to build PFFGs. To do this, we first define a *generating* set of trees:

Definition 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 some basic action in* Σ *rooted at a method in* NT. *We denote the generating set* $\mathcal{G}(PL)$ *and refer to its members as* generating trees.

Finally, on the basis of these generating trees we can define the PFFG for the specific plan library.

Definition 5 *We define the* plan frontier fragment grammar *(PFFG) for the plan library* $PL = \langle \Sigma, NT, R, P \rangle$ *and its generating trees* $\mathcal{G}(PL)$ *as a tuple* $PFFG_{PL} = \langle \Sigma, NT', R, P' \rangle$ *where:*

- $NT' \subseteq NT$.
- P' *is the set of all production rules of the form*

$$pid : \langle a, B \rangle \rightarrow \alpha : \phi$$

for a tree $T \in \mathcal{G}(PL)$*:*

- *pid is a unique identifier for the production rule,*
- $a \in \Sigma$*; we say that a is the* foot *of T,*
- $B \in NT'$*; we say that B is the* root *of T,*

- α *is the finite multiset of symbols over* $\Sigma \cup NT$ *equal to the frontier of* T *with a removed.*
- ϕ *gives a partial order among the indices into* α.

Effectively these definitions use a particular plan library to produce the set of leftmost trees for library's plans, and then define a grammar that captures just the frontiers of the leftmost trees used in the construction of any of the original plans. In so doing, we precompile information about the structure and choices inherent in the plan without requiring the grammar be totally ordered. This way of thinking about the plan library allows us to maintain the state of the derivation of a particular explanation by just maintaining the frontier of the explanation along with its ordering constraints.

Earlier we observed that PHATT's operation resembled *recursive descent* parsing in particular, and correspondingly for these definitions to result in a finite PFFG we must bound any recursion among productions of the plan library. Otherwise the number of generating trees will be unbounded, resulting in an infinite PFFG. To ease our analysis of the complexity of the algorithm we will also assume without loss of generality that the initial plan library's production rules contain no epsilon productions [Aho and Ullman, 1992]. This does not present a significant issue as they can be removed before production of the PFFG using the traditional method for CFGs. Since the process of producing the PFFG cannot introduce epsilon productions, the resulting PFFG will not have them either.

Explanations

A compiled plan library is an input, along with observed actions, to plan recognition; we must also formalize the outputs of the process, the explanations.

Definition 6 *We define a (possibly partial)* explanation *for a sequence of observations* $\sigma_1...\sigma_n$*, given a plan library* $\langle \Sigma, NT, R, P \rangle$*, as a tuple* $\langle \sigma_{m+1}...\sigma_n, \alpha, \phi, PS \rangle$ *where:*

- $0 \leq m \leq n$*, and* $\sigma_{m+1}...\sigma_n$ *are those observations that have not yet been explained.*
- α *is the* explanation frontier*, a finite multiset of symbols from* $\Sigma \cup NT$ *representing the frontiers of the plans in the explanation.*
- ϕ *gives a partial order on indices into* α.
- $PS = \langle pid_1, \alpha[i_1] \rangle, \langle pid_2, \alpha[i_2] \rangle, ..., \langle pid_m, \alpha[i_m] \rangle$ *is a sequence of pairs of production identifiers and elements of* α*. PS records the specific productions that were applied to produce* α.

Definition 7 *We define the* pending attachment points *for an explanation* $e = \langle \sigma_i...\sigma_n, \alpha, \phi, PS \rangle$ *as* $\{\alpha_k : \alpha_k \in \alpha \wedge \nexists (j,k) \in \phi\}$ *and we denote this set as* $AP(e)$

Main algorithm

The main Yappr algorithm takes a PFFG and a sequence $\sigma_1...\sigma_n$ of observations as inputs, and generates the complete set of explanations for a given set of observations. We define the *initial explanation* for every problem as the tuple $\langle \sigma_1...\sigma_n, \{\}, \{\}, \{\} \rangle$. Starting from this base case, we maintain the set of explanations as we process each observation in turn.

There are three possibilities at each observation: 1) the observation removes a single terminal symbol from the explanation, 2) the observation adds nonterminals to the frontier of the explanation, and 3) the observation is the first action of a previously unobserved plan. The algorithm below produces the complete set of explanations for a given a set of observations and a PFFG. Each case is commented in the code.

PROCEDURE **Explain**($\{\sigma_1...\sigma_n, PFFG_{PL} = \langle \Sigma, NT, R, P \rangle\}$)
  $D_0 = \{\}$; E = Emptyqueue();
  Enqueue($\langle \sigma_1...\sigma_n, \{\}, \{\}, \{\} \rangle, E$);
  Process each observation in turn.
  FOR i = 1 to n DO
    Loop over all explanations.
    WHILE Nonempty(E) DO
      E' = Emptyqueue()
      $e = \langle \sigma_i...\sigma_n, \alpha, \phi, PS \rangle$ = Dequeue(E);
      Extend existing plans.
      FOR EACH $B \in AP(e)$;
        Remove terminals from the frontier.
        IF $B = \sigma_i$, THEN
          $\alpha' = \alpha - B$;
          ϕ' = UpdateRemoving(ϕ, B);
          Enqueue($\langle \sigma_{i-1}...\sigma_n, \alpha', \phi', PS \rangle, E'$);
        END IF;
        Expand the frontier.
        FOR EACH $pid : \langle \sigma_i, B \rangle \rightarrow \gamma : \psi \in P$
          $\alpha' = (\alpha - B) + \gamma$;
          ϕ' = UpdateAdding(ϕ, ψ);
          $PS' = PS + \langle pid, B \rangle$;
          Enqueue($\langle \sigma_{i-1}...\sigma_n, \alpha', \phi', PS' \rangle, E'$);
        END FOR LOOP;
      END FOR LOOP; *(extending existing plans)*
      Introduce new plans.
      FOR EACH $pid : \langle \sigma_i, C \in R \rangle \rightarrow \gamma : \psi \in P$
        $\alpha' = \alpha + \gamma$;
        ϕ' = UpdateAdding(ϕ, ψ);
        $PS' = PS + \langle pid, C \rangle$;
        Enqueue($\langle \sigma_{i-1}...\sigma_n, \alpha', \phi', PS' \rangle, E'$);
      END FOR LOOP;
    END WHILE LOOP; *(over previous explanation set)*
    Prepare for the next iteration.
    $E = E'$
    IF E is empty THEN fail;
  END FOR LOOP; *(over sequence of observations)*
RETURN E;

A small amount of further bookkeeping in each case ensures that the ordering constraints are kept consistent, and that constraints refering to a removed actions are either deleted or appropriately redirected to existing actions in the explanation. This is the task of the UpdateRemoving and UpdateAdding functions. Note that if none of these cases apply, the current explanation is inconsistent with the current observation and is pruned from the search.

Note the addition of $\langle pid, B \rangle$ to PS in the second and third cases. If required, we can use PS with the ordered list of observations to recreate the entire tree structure. Walking the

list of *pid*, nonterminal pairs and adjoining a copy of the tree structures from the plan library indicated by the *pid* at the specified nonterminal, will allow us to reconstruct the full plan structure underlying the explanatory model. Of course, implementations which do not require the full generality of queries which this log allows may simplify the representation of explanations appropriately, possibly further improving performance.

It is the fact that B is *removed* or *replaced* in α by γ, or that γ is added to α that makes this algorithm very much in the spirit of a string rewriting algorithm. The string of symbols, α, representing the explanation frontier are rewritten by the PFFG rules.

Keep in mind that each enqueue operation in the above code creates a duplicate of the explanation structure to allow the different rules to be applied separately. It is this copying of the explanation structures that is less time consuming with Yappr's structures than with the forest of trees that PHATT uses. We will return to discuss this in detail later.

Model counting and probabilibities

To complete the Yappr system, we must provide a mechanism for computing the probabilities of the explanations we generate, and of individual goals driving some subsequence of the observed actions. An explanation e for a sequence of observations $obs = \sigma_1 \ldots \sigma_n$ is a set of plans each built to achieve some multiset $G_0, ..., G_I$ of the known goals R. There may be multiple such explanations for fixed obs that differ in the goals, plans being pursued, or assignment of observations to plans. Leaving the observations implicit, we denote the complete and covering set of such explanations for a given set of observations as Exp, and the subset of Exp that make use of a particular goal G as Exp_G

We take a Bayesian approach to plan recognition, and so compute the conditional probability of a particular goal given the set of observations $Pr(G|obs)$. By Bayes' rule we have

$$Pr(G|obs) = \frac{Pr(G \wedge obs)}{Pr(obs)}$$

or equivalently

$$Pr(G|obs) = \frac{Pr(G \wedge obs)}{\sum_{\forall G' \in R} Pr(G' \wedge obs)}$$

where the denominator is the sum of the probability mass for all goals. However given our definition, we know that the plans in an explanation determine the goals in that explanation. Given that Exp is complete and covering, we can rewrite the previous equation as:

$$Pr(G|obs) = \frac{\sum_{e \in Exp_G} Pr(e \wedge obs)}{\sum_{e \in Exp} Pr(e \wedge obs)} \quad (1)$$

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.

Thus, if we can compute the probability of individual explanations for the observations, we can perform plan recognition by weighted model counting. We build a mutually exclusive and exhaustive set of possible explanations for the observations, compute the probability for each, and then sum the probability mass of explanations that contain a particular goal and divide by the probability mass of all of the explanations of the observations.

To use Equation 1 to compute the conditional probability for any particular root goal, we need to be able to compute the probability of each explanation and the observations. We use the following formula, which is very similar to the one used in PHATT:

Definition 8

$$Pr(exp \wedge obs) = \prod_{i=0}^{I} Pr(G_i) \cdot \prod_{i=1}^{n} \frac{Pr(rule_i)}{|ext(AP(exp_i))|}$$

where:

- $obs = \sigma_1 \ldots \sigma_n$.
- $Pr(G_i)$ *is the prior probability of the goals* G_i *being pursued; the set of pursued goals is easily extracted from the productions* PS *of the explanations.*
- $Pr(rule_i)$ *is the probabilistic contribution of any plan choices that are captured within the rule, precomiled with the PFFG. Recall that any any particular PFFG rule may actually encode the choice of multiple productions from the original plan library; the likelihood of the agent making each of these choices must be captured in the model's probability.*
- $ext(AP(exp_t))$ *is the set of rules applicable to an explanation immediately prior to observation t; it is just the set of all production rules that could be applied given the pending attachment points. The size of this set in the formula expresses the likelihood of choosing one such expansion.*

This definition assumes that $Pr(rule_i)$ is precomputed, and that the choice from $ext(AP(exp_i))$ is made uner uniform probability. In fact, modeling these decision making processes is a very complex problem, and nothing in our approach rules out more complex probability models for them. However, for simplicity and low computational cost in our comparisons, we have assumed that each of these choices is captured by a uniform distribution. In the former case, the probability for each PFFG rule is computed offline when the PFFG is created and associated with its respective rule, and so the corresponding calculation is just the multiplication of these precalculated constants. In the latter case, the bookkeeping for counting the possible productions is a simple extension of procedure **Explain**, whose details we elide. This allows the second term in the above equation to be computed by taking the product of the probabilities associated with each production used in the explanation.

One subtle point about the calculation of $|ext(AP(exp_i))|$ is worth mentioning. When a new goal is introduced it is necessary to modify this number for each preceding time point. Since our model is based on the assumption that the set of goals is determined before the actions are started, when a new goal is first observed it is possible that any of the lead actions for the new plan could have been done earlier. This means that when we add a new goal to an explanation we need to account for the possibility that it could have been performed earlier, and this requires modifying the recorded sizes of the pending attachment point sets. Note however, this is a $O(n)$ operation where n is the number of observed actions, and as

we will see, it is dominated by the cost of explanation construction.

3 Complexity analysis

Having given the Yappr algorithm, we still need to show that it will be more efficient than model construction based on tree adjunction. We will show this by considering the complexity of the model construction algorithm.

In Yappr, the complexity for generating a single explanation is $O(nlog(m))$ where n is the number of observations to be explained, and m is the length of the longest plan possible in the plan library. We argue for this in the following way. For a single explanation, for each of the n observations, a single PFFG rule is instantiated, the nonterminal is removed from the explanation, and the right hand side of the instantiated PFFG rules is inserted.

With efficient data structures the removal of the nonterminal and the insertion of the right hands side of the PFFG rule can be done in constant time, however the instantiating of the PFFG rule requires creating a copy of the rule. This process is dominated by the copying of the right hand side of a PFFG rules. The length of the right hand side of the any PFFG rule, corresponds to the depth of the original plan tree, and so costs $O(log(m))$ to copy and instantiate.

Note the $O(log(m))$ length of the rules holds even if there are multiple nonterminals at each level of the leftmost trees that generated the PFFG. To see this, let K to be the maximum length of any of the production rules in the initial library. This means that any individual level of one of the leftmost trees can have no more than K nonterminals and by extension the length of any rule in the PFFG can be no longer than $Klog(m)$. $K \leq m$ and for most domains $K << m$. This only expands the PFFG right hand by a constant and the depth of the original plan tree dominates this feature for all domains where $K << m$.

Given that the PHATT algorithm is adjoining leftmost trees, its complexity is not significantly different for this portion of the problem. To see this, consider that the significant difference between a single leftmost trees and a PFFG production rule is the addition of a $O(log(m)$ number of nonterminals that act as a spine for the attachment of the frontier symbols. We do note that the constant for the Yappr algorithm should be significantly smaller.

As we argued in the introduction, the significant savings for Yappr is in the smaller size of the data structure it uses to represent the plans in an explanation. In Yappr, as in PHATT, the creation of each explanation requires copying the explanation structure. Since in the worst case there can be an exponential number of explanations to consider [Geib, 2004] the size of the data structures to be copied is critical.

In PHATT each explanation is a forest of tree structures, in the worst case a single tree of $O(2^n - 1)$ nodes has to be copied for each of the explanations. Bounding the size of the Yappr data structure is a little more challenging. In order to copy an explanation in Yappr, we must copy both the model's frontier and the ordering constraints.

First, we consider the ordering constraints. If we let the size of the explanation frontier be m, then in the worst case there is an ordering constraint between each pair of actions in m, resulting in m^2 constraints that need to be copied. Given the absence of epsilon productions in the initial grammar, the size of the explanation frontier must be less than or equal to the number of observations. Therefore in the worst case, copying the ordering constraints would take $O(n^2)$.

Note that given the absence of epsilon productions the frontier itself can never exceed n elements in length making the copying of the frontier $O(n)$, and dominated by the constraint copying process. Therefore the worst case $O(n^2)$ sized copy operations for our algorithm represents a significant savings over the tree copying algorithm.

For less densely connected graphs the effective complexity of the Yappr data structure will be less that $O(n^2)$ and the resulting savings will be greater. Most promising for the Yappr algorithm, the greatest number of explanation copy operations occurs precisely when the actions in the plan are least ordered. Plans with completely unordered actions result in a very large number of possible explanations. However, in precisely these cases, the Yappr explanation copying operation reduces to $O(n)$ since only the frontier needs to be copied and there are no ordering constraints. Thus the greatest computational savings from the smaller size of the copy operations for Yappr occurs exactly in the most computationally expensive cases. Our experiments confirm this.

We also consider the cost of computing the probability of an explanation in the Yappr data structure. Inspection of Definition 8 shows that we have to perform operations on each of the rules used. These multiplications can be done in a single pass as can the computing the probability of the root goals. Thus, the cost of computing the probability for a single explanation in Yappr is $O(n)$, and is dominated by the construction of the explanations.

4 Experimental results

To strengthen and verify the complexity results of the previous section, we implemented Yappr in Allegro Common Lisp (ACL) and ran empirical studies to directly compare the runtime of the PHATT and Yappr systems on the same input data. Our experiments were conducted on a single machine, a dual AMD 2000+ (1666.780) running Kubuntu Linux 7.04. We used ACL's timing facilities to measure the CPU time of the algorithms alone, excluding garbage collection and system background process activity.

All of our experiments began by generating a plan library to be recognized. These libraries were represented as partially ordered and/or trees that are similar to HTNs [Ghallab *et al.*, 2004]. In this case and-nodes in the tree correspond to HTN methods and or-nodes correspond to the presence of multiple methods for a single goal in the plan. For all plans the root node was defined to be an or-node, and the plan alternated layers of or-nodes and and-nodes.

There are a number of features that define a plan library:

- *Root goals*: The number of top-level root goals.
- *Plan depth*. The depth of each plan tree in the library. Our plan libraries are made up of alternating layers of and-nodes and or-nodes, we define a depth of one to mean each plan has one layer of or-nodes followed by

one layer of and-nodes. A depth of two means there are two such two-ply layers.

- *And-node branching factor.* The number of children for each and-node. Note this factor when combined with the plan depth determines the length of each plan in the library. For example, a plan with depth two and and-node branching factor of three has nine (3^2) observable actions.
- *Or-node branching factor.* The number of children for each or-node. Since or-nodes represent alternatives in the plan, this factor has no effect on the length of plan instances but along with the plan depth and and-node branching factor determines the number of possible plans for each root goal in the library.
- *Order.* Within each and-node the order factor determines if and how the siblings are related via causal ordering links. We will examine four possible ordering conditions.
  - *Total*. Children of an and-node are totally ordered. Each child action has a single ordering constraint with the child action that precedes it.
  - *First*. The children of each and-node have a designated first action. All other sibling actions are ordered after it but are unordered with respect to each other.
  - *Last*. The children of each and-node have a designated last action. All other sibling actions are ordered before it, but are unordered with respect to each other.
  - *Unord.* The children of each and-node are completely unordered with respect to each other.

It is this last feature, order, that we treat as an experimental factor here. We hold the rest of these features constant with the following values:

Root goals	100
Plan depth	2
And-node branching factor	3
Or-node branching factor	2

For each of the different values of the order factor, we generated 100 data points by randomly selecting one root goal from the plan library and generating a plan for that root goal that obeyed the rules and ordering constraints in the plan library. Thus, each data point was nine observations long and contained no noise. Each such list of observations was then presented in turn to both Yappr and PHATT to compute the conditional probabilities for the goals. To provide the most equivalent runtime environments possible for each invocation of each algorithm, we triggered a full garbage collection beforehand. For each run we imposed a thirty second timeout on the problem. The runtimes for each test case are presented in Figure 1, and summarized in Table 1.

ACL's timing facility cannot resolve times of less than ten msec. This resulted in the system reporting a runtime of zero milliseconds for several cases in the Total and First orderings. In cases of a zero reported runtime, we have instead depicted a runtime of five milliseconds. This was done to more accurately reflect the reality of the process taking some non-zero amount of time.

Order	Algorithm	Mean	Std. dev.
Unord	Yappr	15728	78.4
First	Yappr	6.6	2.31
	PHATT	19.3	2.92
Last	Yappr	20.3	7.54
	PHATT	192.7	49.1
Total	Yappr	7.40	3.77
	PHATT	40.7	60.5

Table 1: Summary of run times (in milliseconds).

Across all tests Yappr outperformed PHATT (in some cases by almost an order of magnitude). These results clearly demonstrate the gains available by using the Yappr approach. We discuss the results for the individual tests in turn.

As we described in the previous section, Yappr saves the most on its copy operations when the frontier has the fewest ordering constraints. This can be seen clearly in Figure 1(a) which graphs the runtimes for plans with the Order factor = Unord. In these cases, even though PHATT and Yappr must compute the same number of explanations, the savings from the smaller models allows Yappr to solve problems PHATT is simply incapable of. In none of these cases was PHATT able to solve the problem within the thirty second timeout bound, or indeed within larger bounds of several minutes, while Yappr was able to solve all of them in under seventeen seconds. While this difference is not as profound in more ordered cases, it is present across all of the values of Order.

Consider the Order = Last case in Figure 1(b). Here Yappr outperforms PHATT by almost an order of magnitude. For clarity, in all of these figures we have added a line representing the average runtime for each system. Note that Yappr has a lower variance than PHATT. The relatively high variance of both systems is caused by considering models with multiple instances of the root goals that are discarded when the final action is seen.

Next, consider the Order = first case in Figure 1(c). While we see a drop in runtime below one second for both systems, Yappr's runtime still is less than half that of PHATT, and displays a smaller variance than PHATT. Finally, in totally ordered libraries shown in Figure 1(d) we see the same trends. Note that in this final case the runtimes on the y-axis is plotted on a log scale to accommodate the large standard deviation of the PHATT runtimes. Yappr again outperforms PHATT and has a smaller variance.

All of these experiments provide confirmation of our claims for the Yappr approach. Across the tested ordering constraints the experiments show the Yappr approach yields faster results and lower variance.

5 Limitations

Yappr's algorithm is optimized to be able to answer queries about the root goals being pursued in a set of plans. The model building process only returns the current explanation

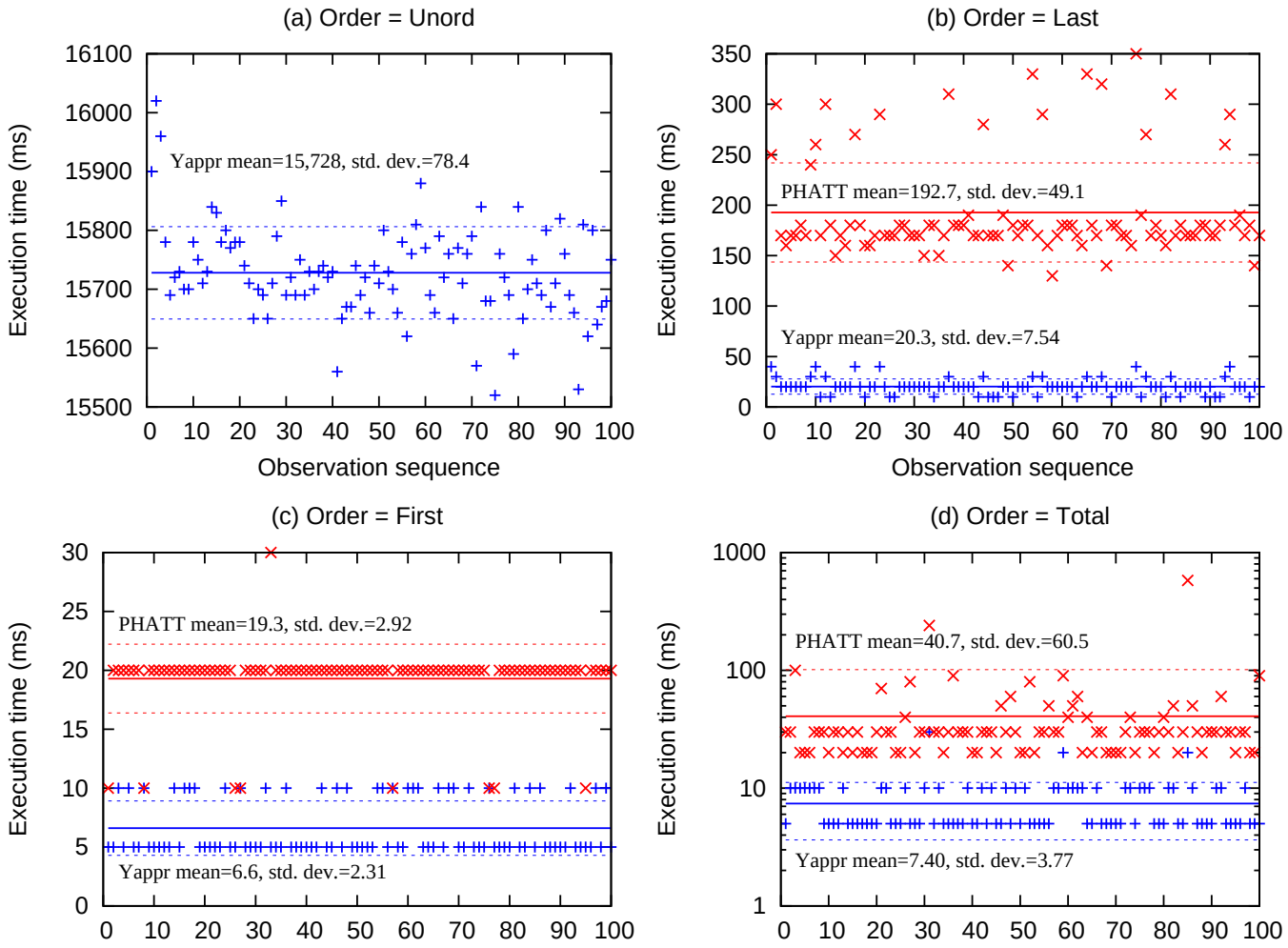

Figure 1: Runtimes comparing Yappr and PHATT. All graphs plot time in milliseconds on the vertical axis. Each Yappr and PHATT runtime is plotted as a plus-sign or X respectively. Solid lines show mean runtime with dotted lines show one standard deviation from the mean. Note in Graph (a) only Yappr's performance is shown. PHATT did not terminate on examples from this library, even allowing timeouts of several minutes.

frontier and the set of rules used to create it. From this we can query for the root goals, but not more complex queries.

For example, suppose we want to know the probability that a particular method was used to achieve a goal. With PHATT's complete plan tree representations it would be relatively easy to select those explanations that had this structure by walking the plan trees of each of the explanations. In order to do the same thing with Yappr, the full plan structures must be reconstructed from the set of observations and the list of applied rules.

Given access to the generating trees for the original grammar, reconstructing a single explanation with n observations and m being the length of the largest plan will take $O(nlog(m))$ time. This follows from our previous argument about the complexity of explanation building. Of course, again the critical question is how many explanations need to be reconstituted. We can imagine domains where there is very little copying of the explanations and very few explanations are pruned as being inconsistent with the observations. In such domains, if these more complex queries are required, it remains to be seen if Yappr would still outperform PHATT.

Yappr is only able to recognize the set of plans that are encoded in the plan library that is initially provided. However, this does not mean Yappr is unable to generate an explanation for observation streams that contain such "unknown" plans. To do this, we include productions in the plan library that allow each action to be done as a root goal. This allows Yappr to build explanations in these cases, while still treating these

more complex explanations as less likely than explanations with fewer root goals for known plans.

6 Related work

We have discussed the relationship of Yappr to PHATT in detail, but there is a large amount of other related work. Huber *at al.* [1994] give an early approach to compiling plans into HTN-like structures for plan recognition. Geib and Goldman [2009] give a more complete survey of plan recognition algorithms and their complexity. We are not the first to suggest that plan recognition can be done by a process similar to probabilistic parsing. Pynadath and Wellman [2000] proposed the use of probabilistic context-free grammars for plan recognition. Yappr addresses a number of issues that are not addressed by Pynadath and Wellman, including cases of partially ordered plans and multiple interleaved plans.

The idea of maintaining only a subset of parse trees has also been used before in parsing. Most common parsing algorithms for context free grammars, including CKY [Younger, 1967], do not maintain an entire parse tree but instead only maintain the derived nonterminals of the grammar. However, to the best of our knowledge, this approach has never been used in plan recognition to reduce the overhead of maintaining explanatory models.

Our deployment of Yappr to the SPDR security system [Haigh *et al.*, 2009] provided insight into Yappr's usability and limitations. Our top-down use of PFFGs restricts the form of plan library rules in a manner similar to that placed on grammars in top-down parsing. We also found that some transformation of plan libraries was required for acceptable system performance: when several decompositions of goals or sub-goals all begin with a common prefix of actions, the space of explanations can grow quickly as all of these explanations must be separately maintained until a distinguishing action comes several relevant observations later. When this effect applies to several goals being executed concurrently, Yappr's performance degrades. In SPDR we addressed these difficulties by transforming the rules to factor common prefixes above disjunctions, but this introduction of internal symbols can make intermediate states difficult for the human monitor to understand. Parsing techniques may again be applicable: Leermakers *et al.* [1992] have considered compact storage of parse tree forests in their treatment of recursive ascent parsers; we hope to adapt their techniques both to relaxing the allowable forms of grammars, and avoiding obscuring transforms via more compact explanation storage. Geib [2009] has developed a new algorithm exploring some of these ideas.

7 Conclusion

In this paper, we have argued that explicitly representing tree-based explanation in plan recognition is needlessly costly. Instead of using tree fragments and adjunction to construct models for plan recognition, we have formalized the idea of plan frontier fragment grammars and shown how they can be used to build models in a manner similar to string rewriting. We have then provided a complexity argument for this approach and shown its improved performance over the PHATT system.

Acknowledgments

This material is based on work supported by (DARPA/U.S. Air Force) under Contract FA8650-06-C-7606 and the EU Cognitive Systems project PACO-PLUS (FP6-2004-IST-4-027657) funded by the European Commission.

References

[Aho and Ullman, 1992] A. V. Aho and J. D. Ullman. *Foundations of Computer Science*. W.H. Freeman/Computer Science Press, New York, NY, 1992.

[Blaylock, 2005] N. Blaylock. *Toward Tractable Agent-based Dialogue*. PhD thesis, University of Rochester, 2005.

[Geib and Goldman, 2003] C. Geib and R. P. Goldman. Recognizing plan/goal abandonment. In *Proceedings of IJCAI 2003*, pages 1515–1517, 2003.

[Geib and Goldman, 2009] C. Geib and R. P. Goldman. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, 2009. In press, doi:10.1016/j.artint.2009.01.003.

[Geib, 2004] C. Geib. Assessing the complexity of plan recognition. In *Proceedings of AAAI-2004*, pages 507–512, 2004.

[Geib, 2009] Christopher W. Geib. Delaying commitment in plan recognition using combinatory categorial grammars. In *Proc. IJCAI-09*, 2009.

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

[Haigh *et al.*, 2009] J. T. Haigh, S. A. Harp, R. O'Brien, J. Gohde, J. Maraist, and C. N. Payne. Trapping malicious insiders in the SPDR web. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*, pages 1–10, 2009.

[Huber *et al.*, 1994] Marcus J. Huber, Edmund H. Durfee, and P. Wellman M.˙ The automated mapping of plans for plan recognition. In *Proc. 1994 Distributed AI Workshop*, pages 137–152, 1994.

[Leermakers *et al.*, 1992] R. Leermakers, L. Augusteijn, and F. E. J. Kruseman Aretz. A functional LR parser. *Theoretical Computer Science*, 104:313–323, 1992.

[Pynadath and Wellman, 2000] D. Pynadath and M. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 2000 Conference on Uncertainty in Artificial Intelligence*, pages 507–514, 2000.

[Steedman, 2000] M. Steedman. *The Syntactic Process*. MIT Press, 2000.

[Younger, 1967] D. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 2(10):189–208, 1967.