Empirical Analysis of a Probabilistic Task Tracking Algorithm

Christopher W. Geib
Honeywell Labs
3660 Technology Drive
Minneapolis, Minnesota 55418
Christopher.geib@honeywell.com

Steven A. Harp
Adventium Labs
100 Mill Place
111 Third Avenue
Minneapolis, Minnesota 55401
Steven.harp@adventiumlabs.org

Abstract

This paper presents an abductive probabilistic algorithm for task tracking/intent inference. It then describes experiments and analysis of the complexity of the algorithm showing a number of conclusions. The most interesting is that empirically the algorithm scales linearly in the number of plans within the plan library.

1. Introduction

We have been working on probabilistic task tracking/intent recognition motivated by real world problem domains including: computer network security, insider threat, and assisted care of the elderly. One of the most critical features for real world application for this technology is the speed of the algorithm. In an effort to understand the factors that effect the runtime of one algorithm, and to focus future theoretical analysis of the algorithm, we have done some exploratory empirical analysis. This paper reports on these experiments. This paper will first provide a description of the tested algorithm and the problems that it solves. It will then discuss exploratory experiments we conducted in order to understand the factors affecting its runtime and the conclusions drawn from them. While much of the results reported here are specific to the particular algorithm, there are some more general conclusions that are highlighted at the end of the paper.

2. The Model

Recent work in execution based plan/intent recognition [1,4, 5, 6, 13] has been based on a model of the execution of simple hierarchical plans. Figure 1 is an hierarchical plan library represented as partially ordered and/or trees that are given to the system to define the set of plans the system is expected to recognize. "And nodes", representing plan decomposition (all the children must be performed for the parent to be achieved) are represented by

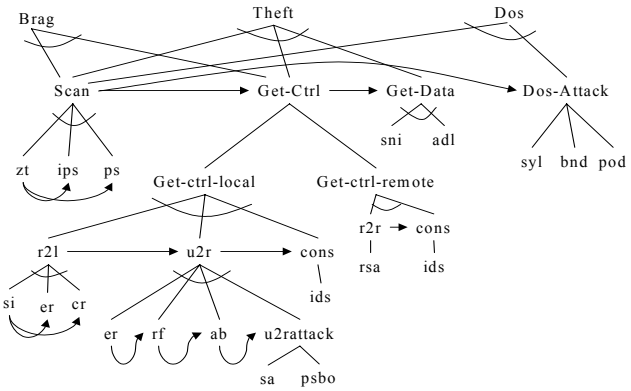


Figure 1: A Plan Library

an undirected arc across the lines connecting the parent node to its children. "Or nodes", which represent choice points in the plan (only one of the children must be performed to achieve the parent) do not have this arc. Directed arcs represent ordering constraints between plan actions. For example, in Figure 1, action **zt** must be executed before **ips** and ps.

In this approach to plan recognition we recognize that plans are executed dynamically and that an agent will only execute those actions in its plans that have been enabled by its previous actions. To formalize this, initially the executing agent has a set of goals and chooses a set of plans to perform to achieve these goals. The set of plans chosen determines a set of pending primitive actions that represent the possible first actions of the plans. The agent then chooses one of the actions in the set and executes it. The set of actions in the plan that are enabled by the executed action constitute the new pending set from which the next action will be chosen. The process of generating

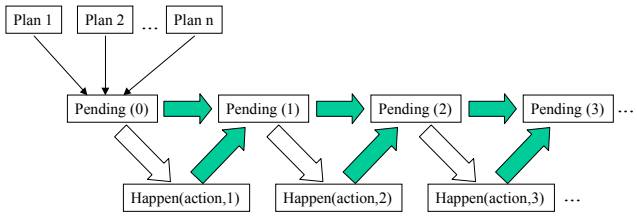


Figure 2: A Model of Pending Set Generation

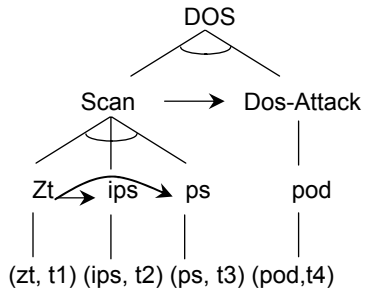
pending sets is illustrated in Figure 2.

In this model, for each possible valid (explains all the observations) explanatory hypothesis (set of goals, chosen plans, and observed actions), there is at least one corresponding series of pending sets. As an example, in the plan library of Figure 1, the three root goals, **Brag**, **Theft**, and **DOS**, share the same initial action **zt.** Therefore there are three possible initial pending sets each corresponding to an explanation of **zt** by a different hypothesized goal: {(zt, DOS)}, {(zt, Theft)} and {(zt, Brag)}. After we observe **zt**, each of these pending sets will contain both **ips** and **ps** but not **zt**. Given a particular set of observations, the pending sets will differ between hypotheses only after the completion of the **Scan** subtask. For example, for the **Brag** and **Theft** hypothesis, after the **Scan** subtask the pending sets will be {(si, Brag), (rsa, Brag)} and {(si, Theft), (rsa, Theft)} respectively. For the **DOS** hypothesis the pending sets will be {(syl, DOS), (bnd, DOS), (pod, DOS)}.

We have used this model to perform probabilistic plan recognition. We take the observations of the agent's actions as an execution trace and find the conditional probability of each of the root goals given the observed series of actions. To provide some intuition for the probabilistically inclined, the sequence of pending sets can be seen as a Markov chain, and the addition of the action executions makes it a hidden Markov model.

3. The Algorithm

The algorithm under study, called the Probabilistic Hostile Agent Task Tracker (PHATT), is a three-stage process. First it computes the complete and covering set of possible *explanations;* second it computes the probability of each of the explanations. Third it computes the conditional probability of the given goal on the basis of the probability of the explanations. In the following section, we formally define explanations and then discuss these processes. Space prevents us from a complete discussion of this approach; we refer readers to [4, 5, 6, 7] for more discussion.



PS(t0) = {(zt, DOS)}
PS(t1) = {(ips, DOS), (ps, DOS)}
PS(t2) = {(ps, DOS)}
PS(t3) = {(syl, DOS), (bnd, DOS), (pod, DOS)}

Figure 3: An Explanation

3.1 Building the Set of Explanations

We define an *explanation* as a forest of plan instances with assignments of the observed action instances to the plans and the associated pending set for each time step. As such a single explanation represents all of the information that would be produced during a single internally consistent execution simulation using the pending set method described above. For example, given the plan library in Figure 1, the only possible explanations for the observed actions: {(zt, t1), (ips, t2), (ps, t3), (pod, t4)} is given in Figure 3. Note that we will be using an integer time notation in this work.

Building a single explanation requires that two steps be performed for each of the observed actions: 1) updating the pending sets, 2) adding the appropriate data structures to represent the plans, actions and goals. By relatively simple preprocessing of the plan library we can store the sets of actions enabled by a given action. This allows us to reduce the time for updating the pending sets to a negligible amount proportional to the number of actions enabled by the executed action. By only adding the minimal tree structure required to attach each observation to the explanation we can reduce the complexity of adding a single observation to an explanation to O(log(n)) runtime where n is the number of observed leaf actions in the plan. Thus, building a single explanation should have an O(m n log(k)) runtime where m is the maximum number of actions enabled by any action, n is the number of observed actions and k is maximum number of leaf actions in any plan in the library.

Of course, since the algorithm must consider all of the possible explanations, this relatively small runtime will be multiplied by the number of possible explanations resulting in a more significant runtime. It is important to note that this process does no probabilistic reasoning. It is simply building the complete set of explanations logically consistent with the observations of the agent's actions.

3.2 Computing the Probability of an Explanation

Once we have the complete and covering set of explanations, PHATT establishes the probability of each of them. It computes this probability by considering three kinds of probabilistic information for each of the goals in each explanation. First, PHATT considers the prior probability of the actor adopting the root goal. This is assumed to be a given with the plan library and can be tailored to the specific agent being observed.

Second, since an “or node” represents a possible choice point for the agent, PHATT must be able to compute the probability that the agent will choose the given expansion for the plan. Without loss of generality, we have assumed that each method is equally likely given its parent goal. PHATT easily supports the insertion of any other

distribution in this computation, however for ease, and local computability we have made this assumption.

Third, for each pending set, PHATT computes the probability that the observed action is the one chosen next for execution. Again in our test implementation of PHATT we have made a uniformity assumption. As with the probability for "or nodes" this is not required, and any other distribution could be used in its place.

On the basis of these three probabilities, PHATT computes the probability of a given explanation by multiplying together the priors for each goal, the probability of the expansion choices at each "or node", and the probability of the observed actions being chosen from the pending sets. Thus, we compute the probability of a given explanation using the following formula:

$$\Pr(\exp, obs) = \prod_{i=a}^{I} \Pr(G_i) \prod_{j=0}^{J} \left(1 \Big/ |\text{Choice}_j| \right) \prod_{k=0}^{K} \left(1 \Big/ |\text{PS}_k| \right)$$

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

3.3 Computing Conditionals for Goals

The algorithm now has a set of explanations that is, by construction, an exclusive and exhaustive covering of the set of observations, and has computed a probability for each explanation. On the basis of this information, PHATT computes the conditional probability of a specific goal by summing the probability mass associated with explanations that have the goal present and dividing it by the total probability mass of all the explanations. Thus the conditional probability of a specific goal is given by:

$$\Pr(g \mid Obs) = \frac{\sum_{e}^{Exp_g} \Pr(e, Obs)}{\sum_{e}^{Exp} \Pr(e, Obs)}$$

where the denominator sums the probability of all explanations for the observations, and the numerator sums the probability of explanations in which goal *g* occurs. This approach to probabilistic reasoning is motivated by Poole's Independent Choice Logic. [12]

A common misconception is that the denominator in this step should always be one, making this division unnecessary. In fact, the probability mass of the covering set of explanations almost never sums to one as a result of multiple independent root goals and choices within an explanation.

3.4 Related Work

There has been significant other work in probabilistic plan recognition [1, 2, 8, 11, 13]. Unlike this approach, much of this previous work [1, 2, 8] has been based on Bayesian networks and as such is subject to the well known complexity results for Bayesian reasoning.

The approach we are analyzing here is most similar in spirit to the work of Pynadath and Wellman [13]. However our probabilistic model is more complex making use of probabilities both for the root actions and the pending sets. It is this more explicit modeling of the plan execution process that enables the algorithm to handle a number of features critical to real world domains that are not treated by other systems: multiple interleaved goals, partially ordered plans, failure to observe ("the dog didn't bark"), missing observations, observations of failed actions, and noise actions that don't contribute to a goal. We must refer the interested reader to [4, 5, 6, 7] for a more detailed discussion of these issues.

4. Runtime Experiments

We have done some analysis of the correctness and effectiveness of the algorithm, however in this work we were focused on understanding those features of the problem space that effect the algorithm's runtime. While some aspects of the PHATT are amenable to theoretical analysis a complete theoretical analysis of the algorithm would be very difficult. However, key issues with the algorithms complexity can be recognized by empirical analysis. Therefore, we have conducted a series of exploratory experiments designed to allow use to understand the most critical factors determining the runtime of the PHATT algorithm. Our initial hypothesis was that while intuitively the number of roots in the plan-library might be assumed to have a large effect on the runtime of the algorithm that, in fact, other features of the plan library would have more impact. The experiments were also designed to be a starting point for more theoretical analyses to follow.

4.1 Experimental Design

Our experiments measuring the runtime for the PHATT algorithm were conducted entirely in situ 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. The experimental factors and levels used in the experiments are summarized in Chart 1.

For each experimental condition, a single plan library was generated. The discussion of each experiment will

Factor	Description	Levels
Order	Types of ordering constraints between actions	total, one, partial, unord, last
Depth	Plan depth	3, 4, 5, 6
BF	Method and Choice node branching factor	3, 4
Roots	Number of root goals in the plan library	10, 100, 200, 400, 600, 800, 1000

Chart 1: Experimental Factors

document which of the features was a tested factor and which were held constant.

Order: This is an indication of how many and what type of ordering constraints exist between the actions in the methods in the plan library.

- *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 un-ordered. This was specifically included 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.

Plan Library Depth: This is a measure of the depth of the plan trees. In these plan trees "or nodes" (choice points) and "and nodes" (method expansions) alternate levels. In all cases the root is defined as an "or node" and levels alternate as they go down.

Number of 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 an "and node" (method definition).

Choice BF: This determines the number of actions (branching factor) at an "or node" (choice node).

Note that 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. This does not inherently reduce the runtime of the algorithm, and does not rule out the possibility of more than one instance of a given plan. However, this does allow us to make several inferences about the effect of various factors on the algorithm's runtime. We will return to discuss this later.

For each experiment and each plan library/experimental condition within the experiments, 1000 test cases were generated. To generate a test case three unique roots were selected at random. For each of these roots a legal plan and linearization of the plan was generated following the plan library. The three component plans were then randomly interleaved maintaining the ordering constraints of the individual plans.

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. This time was recorded for the condition. 1 msec was added to any runtime that registered as zero. The files containing the cpu times were imported into S-Plus for data analysis, analysis of variance and visualization.

4.2 First Experiment

We first explored how the algorithm's average runtime scales with the depth of the plans in the plan library. To look at this we collected runtimes under the following conditions: Order: fixed total. Depth: varying 3 – 6. Method BF: fixed 4. Choice BF: fixed 3. Roots: varying 100 and 1000. Figure 4 shows the results. Given the log scale, the runtimes show a clear exponential trend across plan libraries of 100 and 1000 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. While we are doing further study of the effect of the branching factors, given this result, we have held the method and choice branching factors constant at 4 and 3 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 the branching factors is dominated by the depth of the tree.

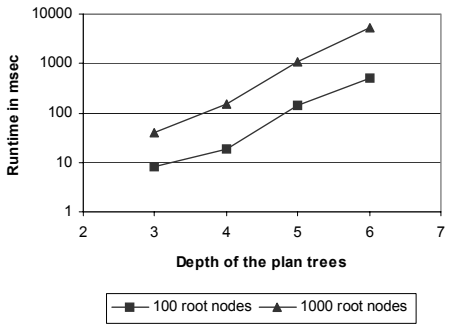


Figure 4: Average runtime vs. depth

This result should be kept in mind when working with the algorithm, but it may not be a significant problem since

the depth of HTN plans in most real world applications is limited to a relatively small value. (i.e less than 10) [9, 10].

4.3 Second Experiment

Next, we collected runtimes in a full factorial experiment for: Order: varying total, one, last, partial, unord. Depth varying 3, 4. Method BF: fixed 4. Choice BF fixed 3. Roots: varying 10, 100, 200, 400, 600, 800, 1000. Figure 5 plots runtime on a log scale against the number of root goals for each of the test conditions.

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 order partial, depth 4 runtimes do have a dip in the between 200 and 800 the overall trend is still linear. We believe this dip in the runtimes to be cause by the natural variability of the complexity of the partially ordered plans, and are examining this effect.

The graph also shows that the type of ordering constraint 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 order 'total' are not so obvious.

Given the similarity of 'one' and 'total', we were very interested in determining if there really is a 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 first experiment to test these contrasts between these pairs of order levels. The results shown in Table 1 imply significance at the 0.01 level, with order 'one' level being having greater runtimes than order 'total' level. Note: diffs greater than lwr are considered significant.

	diff	lwr	upr
one-total	0.1748367	0.1706773	0.1789961
unord-partial	5.0036438	4.9994845	5.0078032

Table 1: Tukey HSD test of Order Contrasts

It should be noted that the story is more complicated than suggested by this test, since the interactions in the model are significant.

Reconsidering the PHATT algorithm in light of these results suggests that the difference between the 'one' and 'total' order levels is caused by maintaining larger pending 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 pending set. In 'total' cases, there is always only a single next action enabled and in the pending set. This means that while multiple explanations are not a possibility for either case, the size of the average pending set will be larger for order 'one' cases than for order 'total' cases. Computing and maintaining these larger pending set 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.

4.4 The Cost of Multiple Explanations

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 unordered PHATT is unable to conclude that a subset of the actions must all be part of the same plan instance. Thus the system maintains explanations that are consistent with all the possible subsets of actions contributing to different plan

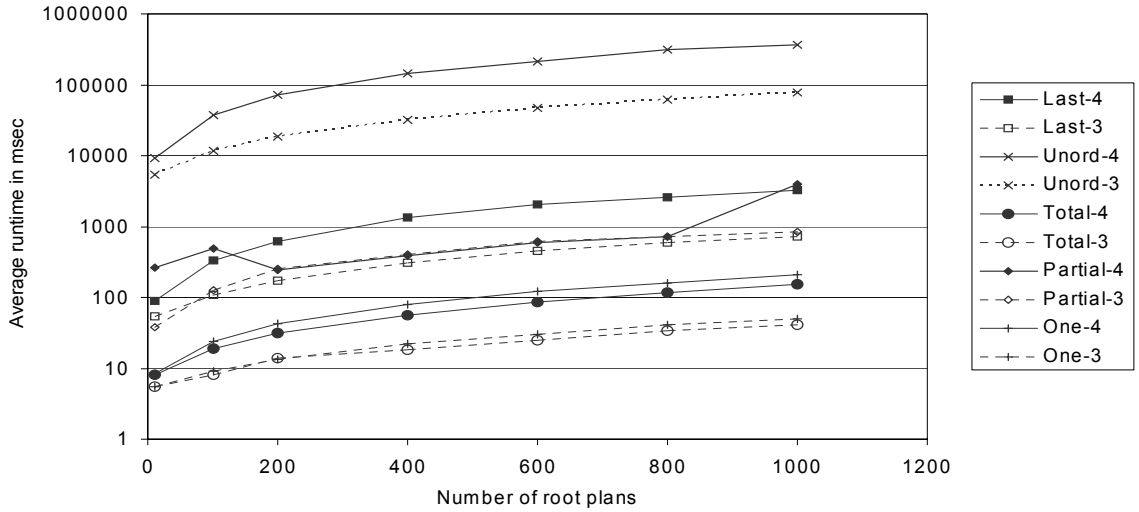


Figure 5: Average Runtimes

instances including the possibility that each action contributes to a separate plan instance. This contrasts sharply with the 'total' and 'one' levels which only have a single explanation.

The hypothesis that multiple possible first actions in a plan increases the number of maintained explanations and is the cause of the increased runtime is confirmed by the runtimes produced for the order 'last' test cases. The 'last' test cases have the same number of ordering constraints as the 'one' test cases and positioned such that 'last' and 'one' test cases have the same average size of pending sets. Thus, the increased runtimes of the 'last' cases must be a result of the larger number of explanations.

This in fact has lead us to other theoretical analysis that shows that "partial orderness" within a plan library can be more damaging to the runtime of plan recognition than repetition of actions in multiple plans. In [3] we have shown that the increased number of explanations that result from partial ordering within plans can more than dominate the relatively small increase in the number of explanations that can result from actions occurring in more than one plan. This alleviated our concerns about the unique action assumption that we have made in this analysis.

4.5 Early Closing of Plans

Inspection of the runtimes for the order last cases from the previous experiment revealed an interesting relationship. There was a significant gap between a cluster of test cases that had the worst runtimes and the rest of the test cases. 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, based on an actual test case, for three plans a, b and c:

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

note that the last three actions of the series are the final actions of each of the respective plans. We will call these *late closing* test cases.

The test case with the next worst runtime had closed 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 made a ten second difference in the algorithm's runtime and suggested a final experiment.

4.6 Third Experiment

To determine if early closure of plans and the reduction in the number of explanations reduced runtimes, we collected runtimes for the following abstract observation streams:

Case1: {a1, b1, a2, b2, b3, a3, c1, c2, c3, b4, a4, c4}
Case2: {a1, b1, a2, b2, b3, a3, c1, c2, b4, c3, a4, c4}
Case3: {a1, b1, a2, b2, b3, a3, c1, b4, c2, c3, a4, c4}
Case4: {a1, b1, a2, b2, b3, a3, b4, c1, c2, c3, a4, c4}
Case5: {a1, b1, a2, b2, b3, b4, a3, c1, c2, c3, a4, c4}

To do this we took the plan library for the order 'last', depth 4, 1000 root goal case and generated five observation streams that correspond to the above cases. Figure 7 graphs the runtimes for each of the test cases.

Notice that 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. 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

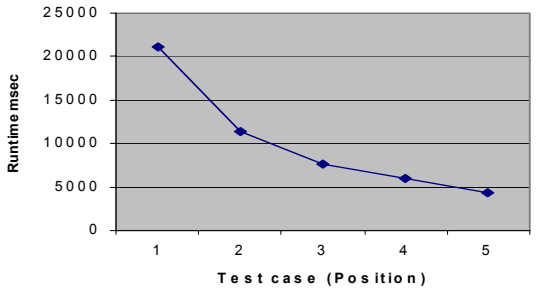


Figure 7: Runtimes for Early Plan Closing

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. For example, 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.

5. Conclusions, and Future Directions

Several things are evident 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.
- The way in which the constraints are organized has a significant impact on the algorithm's runtime.
- 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. This work has identified inefficiencies in the implementation of the PHATT algorithm that can be removed and will make the algorithm run even faster. Further we are conducting more experiments on the algorithm to further study of the effect of plan branching factors, the effects of incomplete trees, and reasons for differences in the variance of the algorithm. While this further study is needed, these initial results are very promising and provide a much needed grounding for the application of the PHATT algorithm.

6. References

[1] Bui, H., 2003, "A General Model for Online Plan Recognition," *Proceedings of IJCAI-2003.*

[2] Conati, C., Gertner, A., VanLehn, K., Druzdzel, M., 1997, "On-Line Student Modeling for Coached Problme Solving Using Bayesian Networks", *Proceedings of the Sixth International Conference on User Modeling* (UM-97).

[3] Geib,. C., 2004, "Assessing the Complexity of Plan Recognition", *Proceedings of AAAI-2004.*

[4] Geib, C., and Goldman, R., 2003, "Recognizing Plan/Goal Abandonment," *Proceedings of IJCAI-2003*.

[5] Geib, C., and Goldman, R., 2001a, "Probabilistic Plan Recognition for Hostile Agents," *Proceedings of the FLAIRS01 Conference*, Key West.

[6] Geib, C. and Goldman, R., 2001b, "Plan Recognition in Intrusion Detection Systems," *Proceedings of the DISCEX-II Conference*.

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

[8] Horvitz, E., Breese, J., Heckerman, D., Hovel, D., Rommelse, K., 1998, "The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users" *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98).*

[9] Kutluhan, E., Hendler, J. and Nau, D., 1994a, "UMCP: A Sound and Complete Procedure for Hierarchical Task Network Planning," *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS-94)*, Chicago.

[10] Kutluhan, E., Hendler, J. and Nau, D., 1994b, "HTN Planning: Complexity and Expressivity," *Proceedings of the National Conference on Artificial Intelligence (AAAI-94)*, Chicago.

[11] Patterson, D., Liao, L., Fox, L., and Kautz, H., 2003, "Inferring High-Level Behavior from Low-Level Sensors", in *Proceedings of the Fifth International Conference on Ubiquitous Computer (UBICOMP'03*).

[12] Poole, D., 1997, "The Independent Choice Logic for Modeling Multiple Agents Under Uncertainty," *Artificial Intelligence*, 94(1-2):7-56.

[13] Pynadath, D., and Wellman, M., 2000, "Probabilistic State-Dependent Grammars for Plan Recognition." In *Proceedings of the Conference on Uncertainty in Artificial Intelligence(UAI-00),* pp. 507-514.

[14] Sanghai, S., Domingos, P., Weld, D., 2003, "Dynamic Probabilistic Relational Models", *Proceedings of IJCAI-2003.*