Controlling the Hypothesis Space in Probabilistic Plan Recognition

Froduald Kabanza and **Julien Filion**
Université de Sherbrooke
Sherbrooke (QC) J1K 2R1, Canada
{*kabanza, julien.filion*}@*usherbrooke.ca*

Abder Rezak Benaskeur and **Hengameh Irandoust**
Defence R&D Canada - Valcartier
Québec (QC) G3J 1X5, Canada
{*abderrezak.benaskeur, hengameh.irandoust*}@*drdc-rddc.gc.ca*

Abstract

The ability to understand the goals and plans of other agents is an important characteristic of intelligent behaviours in many contexts. One of the approaches used to endow agents with this capability is the weighted model counting approach. Given a plan library and a sequence of observations, this approach exhaustively enumerates plan execution models that are consistent with the observed behaviour. The probability that the agent might be pursuing a particular goal is then computed as a proportion of plan execution models satisfying the goal. The approach allows to recognize multiple interleaved plans, but suffers from a combinatorial explosion of plan execution models, which impedes its application to real-world domains. This paper presents a heuristic weighted model counting algorithm that limits the number of generated plan execution models in order to recognize goals quickly by computing their lower and upper bound likelihoods.

1 Introduction

The ability to understand the goals or plans underlying the behaviours of an observed agent, also known as plan recognition, is an important characteristic of intelligent behaviours. It is central for achieving situation awareness in many contexts, including understanding the intent of other agents we interact with. Examples of applications include user modelling [Lesh *et al.*, 1999; Conati and VanLehn, 1996], assistive technologies [Demeester *et al.*, 2008; Kautz *et al.*, 2003], malicious activity detection [Geib and Goldman, 2009; Jarvis *et al.*, 2004], and opponent intent recognition in video games [Synnaeve and Bessière, 2011].

One of the main plan recognition paradigms, commonly referred to as a plan-library based approach, is to assume that the observing agent is given a plan library describing the expected behaviours of the observed agent. The plan recognition problem is thus reduced to inferring both goals and plans from the plan library that explain the observed behaviour. Approaches of this kind mainly differ on their plan representation formalisms and their inference theories and techniques. Examples include approaches based on Hierarchical Task Networks (HTN) [Geib and Goldman, 2009; Avrahami-Zilberbrand and Kaminka, 2005], probabilistic grammars [Pynadath and Wellman, 2000], Bayesian networks [Synnaeve and Bessière, 2011], Hidden Markov Models [Bui *et al.*, 2002], and Markov logic [Sadilek and Kautz, 2010].

Goals and plans can be inferred from observations and a plan library by generating or counting an exhaustive set of mutually exclusive models of plan execution that are consistent with the observed behaviour. The probability of a given goal can then be computed as a proportion of models satisfying the goal; in other words, based on the weight of the model. Such an explicit plan execution model facilitates the handling of interleaved plans or plans geared towards achieving multiple goals [Geib *et al.*, 2008; Geib and Goldman, 2009]. However, this approach results in a combinatorial explosion of the goal and plan hypothesis space, which is intractable for most real-world applications.

Domain-specific knowledge could be used to prune or prioritize the hypothesis space in plan-library based approaches [Sukthankar and Sycara, 2008]. In this paper, we approach the problem from a different but complementary angle. We trade the slow computation of exact goal hypothesis probabilities against a quick computation of lower and upper bounds of goal hypothesis probabilities. As the computation of bounding intervals only requires a partial enumeration of plan execution models, our algorithm is able to provide early assessments of the observed agent's goals much faster than a standard model counting algorithm. The goal probability intervals shrink as processing time increases, and converge to the exact goal probability values. This is particularly useful when an agent needs to recognize the behaviours of another agent within deadlines. For example, an agent may be interested in reacting only to the goals of another agent provided that they have a probability above a given threshold. In such a case, our algorithm will provide a result as soon as the bounds cross this threshold.

In the next section, we discuss the syntax and interpretation of plan libraries. From that point, we explain the basics of a standard weighted model counting approach. This is mostly background from [Geib and Goldman, 2009], but we formulate it differently for a concise, but nevertheless precise description. We then explain the controlled generation of plan execution models. Finally, we discuss experiments and conclude with brief remarks.

2 Plan Library

A plan library is specified as a partially-ordered multiset context-free grammar (pomset CFG) [Nederhof *et al.*, 2003].

2.1 Syntax

Definition 1. *A plan library is a tuple $L = \langle A, G, I, R\rangle$, where A is a finite set of action symbols (terminals), G is a finite set of goal symbols (non terminals), $I \subseteq G$ is a set of intendable goal symbols, and R is a set of goal-decomposition rules (production rules) of the form $g \rightarrow \tau$, where:*

- *$g \in G$;*
- *τ is a partially ordered string over the alphabet $(A \cup G)$, represented as a pair $[\beta, C]$, where:*
  - *$\beta \in (A \cup G)*$ is a string of goal and action symbols;*
  - *C is a set of ordering constraints of the form (i, j), meaning that the i^{th} symbol of β must precede the j^{th} symbol of β.*

A rule $g \rightarrow [Y_0, \ldots, Y_N, C]$ means that the goal g can be accomplished by achieving or executing each of the Y_i in any order consistent with constraints C. A set of rules with a common left-hand side g represents alternative choices for accomplishing goal g. The specification of a plan library also supports the use of preconditions and parameters for production rules and actions. However, for the sake of conciseness, we will not use them in this paper. A pomset CFG augmented with preconditions and parameters is equivalent to traditional HTNs used in planning [Ghallab *et al.*, 2004].

As an example, Figure 1 illustrates a plan library that is graphically displayed as an AND-OR tree. AND-nodes have an arc across the lines to their children, while OR-nodes do not have such an arc. The actions are leaf nodes and the goals are interior nodes. Each rule of the form $g \rightarrow [Y_0, ..., Y_N, C]$ is represented as an AND-node g along with children $Y_0, \ldots, Y_N$, such that an arrow from Y_i to Y_j accounts for a constraint (i, j) in C. A set of rules with a common left-hand side, $g \rightarrow [\beta_0, C_0], \ldots, g \rightarrow [\beta_N, C_N]$, corresponds to an OR-Node g along with children groups $\beta_0, \ldots, \beta_N$, where the children in β_i are linked according to C_i.

This example illustrates a plan library for the StarCraft real-time strategy (RTS) game. The example is partial and extremely simplified for illustration purposes. StarCraft players often follow rules of thumb, particularly during the opening phase. We can specify such rules in a plan library used by a game AI to recognize its opponent's goals and plans. Figure 1

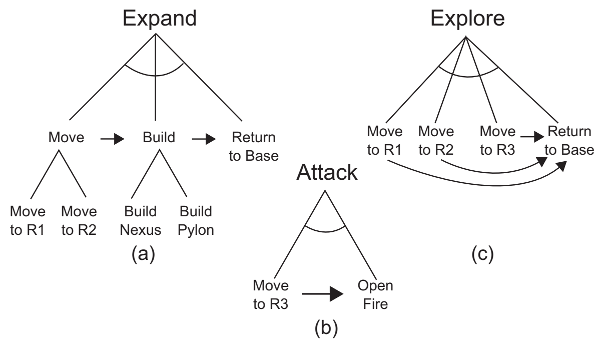


Figure 1: A simplified plan library for the StarCraft game

illustrates rules of thumb for a player progressing on a terrain divided into four regions, which are his own base, R1 (neutral), R2 (neutral) and R3 (opponent's base). The *Expand* rule is a recipe for expanding the player's territory by taking control of an additional region. The *Explore* rule is for gathering intelligence information from the environment. The *Attack* rule is for destroying the opposing force. Note that by executing strategic and tactical actions to play the game, players will influence each other's behaviour. Their plans can also involve actions that achieve deceptive behaviours. However, our algorithm does not address mutual influences or detect deception. These are topics for future investigation.

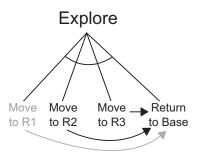
2.2 Interpretation

We assume that the observed agent behaves by planning and then executing planned actions. He proceeds by first choosing a goal that is a conjunction of intendable goals, generates a plan achieving the goal, and then executes the plan. We also assume that the observing agent has a plan library $L = \langle A, G, I, R\rangle$ conveying the expected behaviours of the observed agent. We further assume there are no conflicting interdependencies between intendable goals[1].

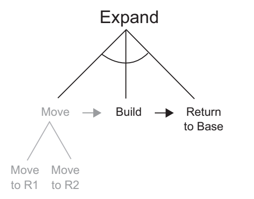
Under these assumptions, planning for a conjunctive goal amounts to choosing a set of production rules in the plan library that recursively expand each conjunct down into an action plan. The chosen rules for each conjunct form a complete derivation tree and the frontier of the tree is a partially ordered multiset of actions, that is, a plan $[\alpha, C]$, where α is the set of grammar symbols labelling the frontier, and C is a partial order progressed from rules used in the tree down to the frontier. The plan for the conjunctive goal is the frontier of the forest composed of trees. Specifically, if $[\alpha_i, C_i]$ are the plans of trees t_i for the conjuncts, the forest's plan is $[\alpha', C']$, where α' is the union of the α_i and C' is the union of the C_i. A complete derivation tree (or forest) has a frontier that consists only of action symbols, representing a complete plan for achieving the tree's root goal (or forest's conjunctive goal). A partial derivation tree (or forest) has a frontier made of action and goal symbols, that is, a partial plan for achieving the tree's root goal (or forest's conjunctive goal).

[1]Handling conflicts in a conjunctive goal is a difficult problem for plan-library based approaches. A workaround is to replace a conflicting conjunctive goal by a single intendable goal, while ensuring that the corresponding production rules resolve the conflict.

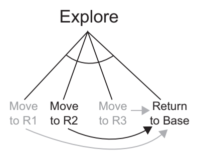
Figure 2 illustrates partial derivation forests for the plan library in Figure 1. Nodes are labelled either with goals or actions. A goal node g with children $Y_0, \ldots, Y_N$ means that the rule $g \rightarrow [Y_0, \ldots, Y_N, C]$ was chosen to achieve g, with C reflected by the links between the children. We will revisit this figure later when discussing the generation of goal and plan hypotheses.



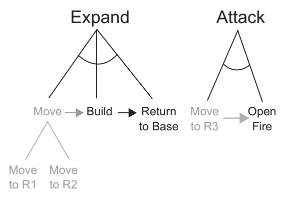
(a) h_0^1 for ⟨MoveTo R1⟩



(b) h_1^1 for ⟨MoveTo R1⟩



(c) h_0^2 for ⟨MoveTo R1, MoveTo R3⟩, derived from h_1^1



(d) h_1^2 for ⟨MoveTo R1, MoveTo R3⟩, derived from h_2^1

Figure 2: Examples of partial derivation forests

3 Basic Hypotheses Generation Algorithm

We assume the actions are observable—an assumption that can be relaxed as discussed in [Geib and Goldman, 2005], although at the expense of an additional exponential blow up in the hypothesis space. Thus, if an agent executes a plan in a library $L = \langle A, G, I, R \rangle$, then every sequence σ of observed actions is a prefix of a sequence of actions that is a particular ordering of a plan derived from a conjunctive goal. Ergo, we define a hypothesis for a sequence of observations σ as the goal and plan of a minimal derivation forest, such that the sequence of observations is a prefix of a total ordering of the plan.

Earlier complexity analysis of a plan-library based plan recognition approach suggests that the generation of hypotheses that explain a sequence of observations is at least an NP-hard problem [Vilain, 1991]. A bottom-up approach, while more complicated, is on average more efficient than a top-down approach because it only generates hypotheses that are relevant to the observations. It proceeds by simulating the execution of an exhaustive set of mutually exclusive hypotheses, in synchrony with the sequence of observations (σ). This is done by generating a tree of hypotheses, as illustrated in Figure 3. The two circled frontiers illustrate breadth-first and controlled exploration strategies. At this point, we discuss the breadth-first strategy.

For a sequence of observations σ, let us note σ_i the observation in position i and $\sigma_{[i:j]}$ the subsequence of observations from position i to j. Each hypothesis at level $i+1$ is an explanation for $\sigma_{[0:i]}$. The root node in the hypothesis tree is the empty hypothesis. The children h_c of a hypothesis h_p at level i are generated by taking into account all possible ways of explaining σ_i, from h_p. To understand how this is done, first note that each tree in a hypothesis has a multiset of enabled actions, that is, the multiset of actions having no precedence constraint in the tree's plan. We refer to this multiset as the pending set of the plan or, by extension, the pending set of the tree. These are actions pending execution by the observed agent assuming he is committed to the plan. A pending set for a hypothesis is the union of pending sets of the trees composing it.

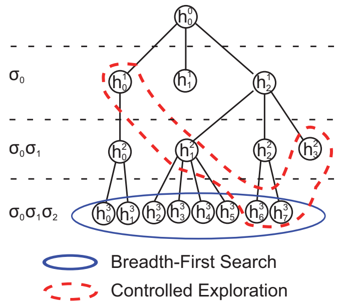


Figure 3: Illustration of a tree of hypotheses

Now, the procedure for expanding a hypothesis h_p into its children h_c, given the observation σ_i, is as follows:

(a) Consider all possible ways of choosing and then executing an action of h_p's pending set; each such possibility gives rise to a chid h_c. Specifically, for each tree t in h_p, if σ_i is in t's pending set, generate a child h_c differing from h_p by the fact that the action σ_i is trimmed from t.

(b) Check whether σ_i could be the start of a new goal's execution—that is, check whether σ_i could be a leftmost symbol of a plan for a new goal. For each such tree t, generate a child h_c from h_p by adding t with the action σ_i trimmed off.

Figure 2 illustrates the expansions of hypotheses into some of their children. Note that step (a) requires performing a leftmost derivation of h_p deep enough to derive its pending set. To efficiently check the membership of an action in a pending set, a bookkeeping optimization is to precompute actions that come first in a leftmost derivation of a plan for any goal. The actions that come first in a leftmost derivation of a plan for a given goal form the $First$ multiset for the goal (for a goal g, we denote this by $First(g)$). This is a generalization to pomset CFG of a $First$ set in traditional CFG top-down parsing [Sudkamp, 2006][2]. Note that only the frontier of the hypothesis tree is kept in memory.

4 Computing Exact Goal Probabilities

The probability of a hypothesis and a sequence of observations (noted $P(h \wedge \sigma)$) measures the likelihood that an agent decides to execute h's plan following an action interleaving

[2]Each element of a $First$ multiset corresponds to the concept of a foot for derivation trees in [Geib *et al.*, 2008].

that matches σ. This probability can be used to compute the probability of a goal given the observations (noted $P(g \mid \sigma)$), the probability of a plan given the observation, or the probability of a behaviour (set of plans) given the observations. Here we only show how to compute $P(g \mid \sigma)$. Using Bayes' rule, we can write:

$$\begin{aligned} P(g \mid \sigma) &= P(\sigma \mid g)P(g)/P(\sigma) \\ &= P(g \wedge \sigma)/P(\sigma) \end{aligned} \quad (1)$$

Let us note H_σ the exhaustive set of mutually exclusive hypotheses for the sequence of observations to date, σ, and $roots(h)$ the intendable goals of hypothesis h. We can rewrite the previous equation as:

$$P(g \mid \sigma) = \frac{\sum\limits_{h \in H_\sigma \mid g \in roots(h)} P(h \wedge \sigma)}{\sum\limits_{h \in H_\sigma} P(h \wedge \sigma)} \quad (2)$$

To compute $P(h \wedge \sigma)$, we need prior probability that the observed agent will commit to a set of intendable goals, $P(goals)$, the goal commitment model. We also need the probability that he will choose a particular rule to achieve a goal, that is, $P(rule \mid g)$, the planning model. Finally, we need the probability that he will execute an action given the current pending set, that is, $P(action \mid PendingSet)$, the interleaving model. These distributions could be learned from a domain. By default, we assume that the agent selects identical goal instances by following a geometric distribution. The probability of choosing the same goal is thus independent from the goals already selected. In other words, $P(goals) = \prod_{g \in I} P(g)^N(1 - P(g))$, where N is the number of instances of g in $goals$ and $P(g)$, given as input, is the *a priori* probability that an agent follows goal g at least once. For $P(rule \mid g)$, we assume a uniform choice distribution among applicable rules. For $P(action \mid PendingSet)$, we also assume a uniform choice distribution among enabled actions. As noted by [Geib and Goldman, 2009], these default distributions can be surprisingly effective in practice.

Let us note $rules(h)$ the set of rules appearing in h and $lhs(r)$ the left-hand side of rule r. It can be shown [Geib and Goldman, 2009] that:

$$\begin{aligned} &P(h \wedge \sigma) = \\ &K \prod_{g \in roots(h)} P(g) \prod_{r \in rules(h)} P(r \mid lhs(r)) \prod_{i=0}^{n} P(\sigma_i \mid PS(h, \sigma, i)) \end{aligned} \quad (3)$$

where $K = \prod_{g \in I}(1 - P(g))$ is the probability that an agent will not pursue more goals than he has already committed to, and $PS(h, \sigma, i)$ is the pending set of h's plan after executing the first i actions of σ.

Current model counting algorithms, notably YAPPR [Geib *et al.*, 2008] and PHATT [Geib and Goldman, 2009], enumerate all the hypotheses to compute the probability of a goal given the observations to date. Given the sequence of observations to date, $\sigma_{[0:i]}$, level $i+1$ of the hypothesis tree is entirely generated (see Figure 3), and Equation (2) is applied to obtain goal probabilities. However, this exhaustive enumeration of plan execution models quickly becomes intractable.

5 Controlling the Generation of Hypotheses

In many applications, plan recognition is involved to enhance the understanding of a situation in order to decide how to react. In such cases, an agent may decide to react to another agent's goal or plan, based on whether or not the assessed probability of the goal or plan is above a certain threshold. For example, the game AI in an RTS game could decide to defend an asset or not, depending on whether or not the probability of an attack against the asset is above some threshold. In that case, instead of assessing the exact probability or expected utility of goals, it is sufficient to compute lower and upper bounds for goal probabilities. If the lower bound is above the threshold, the goal is worthy of attention. If the upper bound is below the threshold, the goal is not worth consideration.[3] At this point, we explain how to compute goal-probability bounds.

Interestingly, goal-probability bounds can be computed more efficiently than exact probabilities, without exhaustively enumerating all hypotheses. That is, by not fully expanding hypotheses to match the sequence of observations, we lose the ability to compute $P(g \mid \sigma)$ precisely, but we can still estimate its lower and upper bounds. This idea is behind our new algorithm, the Decision-Oriented PLAn Recognizer, or DOPLAR. This algorithm expands the hypothesis tree by picking, each time, the hypothesis that weighs most in the computation of $P(g \mid \sigma)$. The contribution of a hypothesis h to $P(g \mid \sigma)$ is measured by $P(h \wedge \sigma)$, therefore the hypothesis that has the highest value $P(h \wedge \sigma)$ is always selected for the next expansion. This results in an unevenly expanded hypothesis tree, as illustrated in Figure 3.

To explain how the lower and upper bounds for $P(g \mid \sigma)$ are calculated, we need to introduce the notion of a weight for a hypothesis. For an unevenly expanded hypothesis tree, let us note F the set of hypotheses on the frontier of the tree. For a sequence of observations σ and hypothesis h, let us also note $DH(h, \sigma)$ all the descendants of h down to level $|\sigma|$ of the hypothesis tree; in other words, the descendants of h that explain the sequence σ. If h is at level $|\sigma|$, then $DH(h, \sigma) \equiv \{h\}$. We define the weight of a hypothesis h with respect to a sequence of observations σ as follows:

$$weight(h, \sigma) = \sum_{h_i \in DH(h, \sigma)} P(h_i \wedge \sigma) \quad (4)$$

An exact calculation of the weight of h requires that all of its descendants be generated down to level $|\sigma|$. Given that the principle of DOPLAR is to avoid an exhaustive exploration, it instead computes the lower and upper bounds of $weight(h, \sigma)$ based on h, not its descendants. If a hypothesis h is at level $|\sigma|$ (i.e., h explains σ), then its $weight(h, \sigma)$ is precisely $P(h \wedge \sigma)$. Thus, both lower and upper bounds are equal to $P(h \wedge \sigma)$. The most interesting case is when h is at

[3]More generally, agents may react to goals based on the expected utility thresholds instead of probability thresholds. It would not be difficult to compute goal-utility bounds in our approach.

a level strictly less than $|\sigma|$. For this case, the lower bound is 0, because $DH(h, \sigma)$ might be empty. The upper bound is given by the following lemma.

Lemma 1. *For a hypothesis h explaining $\sigma_{[0:i]}$, with $i \leq n$, $weight(h, \sigma_{[0:n]})$ is bounded above by $ub(weight(h, \sigma_{[0:n]}))$, where:*

$$ub(weight(h, \sigma_{[0:n]})) = P(h \wedge \sigma_{[0:i]}) \prod_{k=i+1}^{n} (1 + \sum_{g \in I | \sigma_k \in First(g)} P(g)) \quad (5)$$

A proof sketch for this lemma is as follows. First consider the basic case, $i = n - 1$, that is, when the hypothesis is one observation away from explaining σ. In order to define an upper bound on $weight(h, \sigma_{[0:n]})$, we must consider the two steps used to expand a hypothesis (Section 3). When expanding h using elements of the pending set (step a), it can be shown that, if all potential expansions (h_{pe}) of h match the sequence σ, then $P(h \wedge \sigma_{[0:i]}) = \sum_{h_{pe}} P(h_{pe} \wedge \sigma)$. However, some of the children might not match the next observation, therefore the contribution of those hypotheses to $weight(h, \sigma_{[0,n]})$ will be at most $P(h \wedge \sigma_{[0:i]})$ or $weight(h, \sigma_{[0:i]})$. Similarly, when expanding h by incorporating new goals (step b), it can be shown that, for each intendable goal g, the sum of $P(h_c \wedge \sigma)$ over all the children h_c of h generated to account for g will not exceed $P(h \wedge \sigma_{[0:i]})P(g)$. Furthermore, a goal g for which σ_{i+1} is not an element of $First(g)$ will not produce any children for h and can be ignored for this step. This reasoning can be repeated inductively to obtain Lemma 1.

For a working hypothesis set F (i.e., the frontier of the hypothesis tree explored by DOPLAR), let us note $T(F)$ the set of all fully expanded or terminal hypotheses in F, in other words, the hypotheses at level $|\sigma|$. We also note $NT(F)$ the set of partially expanded or non-terminal hypotheses in F.

Theorem 1. *Given a working set F over the sequence σ, we have the following lower bound on the posterior probability of a goal g:*

$$P(g \mid \sigma) \geq \frac{\sum_{h \in T(F) | g \in roots(h)} P(h \wedge \sigma)}{\sum_{h \in F} ub(weight(h, \sigma))} \quad (6)$$

A proof of this theorem hinges on the observation that, for a working set F over the sequence σ and a goal g, $P(g \mid \sigma)$ will be at its minimum value when $P(\sigma)$ is maximized and for each hypothesis h_i in $NT(F)$, $DH(h_i, \sigma)$ contains no hypothesis having a tree rooted on g. Indeed, in such a case, $P(g \wedge \sigma)$ is the sum of the weight of all hypotheses in $T(F)$ that are rooted on g. Furthermore, using Lemma 1, $P(\sigma)$ can be overestimated by summing $ub(weight(h_i, \sigma))$ for all $h_i \in F$.

Theorem 2. *Given a working set F over the sequence σ, the posterior probability of a goal g is bounded above as follows:*

$$P(g \mid \sigma) \leq \frac{\sum_{h \in T(F) | g \in roots(h)} P(h \wedge \sigma) + \sum_{h \in NT(F)} ub(weight(h, \sigma))}{\sum_{h \in F} ub(weight(h, \sigma))} \quad (7)$$

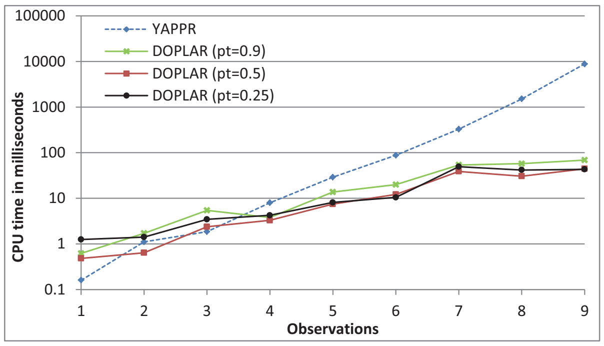
This theorem follows from the observation that $P(g \mid \sigma)$ will be at its maximum value when, for each hypothesis h_i in $NT(F)$, the set $DH(h_i, \sigma)$ contains only hypotheses where the observed agent has g as one of his goals, and where $weight(h_i, \sigma)$ is maximized. Using Lemma 1, we can overestimate the contribution of a partial hypothesis h_i to $P(g \wedge \sigma)$ with $ub(weight(h_i, \sigma))$. $P(g \wedge \sigma)$ can then be overestimated by the sum of $ub(weight(h_i, \sigma))$ for each $h_i \in T(F)$ and of the weight of all hypotheses in $T(F)$ that have a plan for g. In such a case, $P(\sigma)$ is at least the sum of $ub(weight(h_i, \sigma))$ for each h_i in F.

6 Experiments

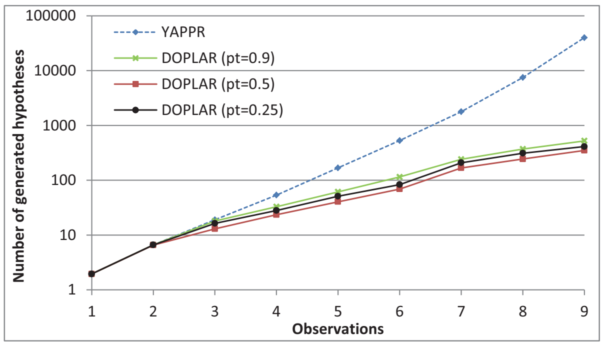
We implemented DOPLAR in Common Lisp as an extension to YAPPR, a weighted model counting based plan recognizer that uses an HTN plan representation [Geib *et al.*, 2008]. We tested both algorithms on artificial and simple StarCraft plan libraries, using GNU CLISP 2.48 on a quad-core Intel Core 2 Q8200 (2.33GHz) processor, running Windows 7. The tests were aimed at verifying the time and space taken by DOPLAR to recognize goals, using YAPPR as a baseline. We tested DOPLAR's two different stopping criteria: probability threshold (threshold), or error on the probability's uncertainty (error), which is the difference between the upper and lower bounds.

The artificial plan libraries were randomly generated by considering a number of plan library features that contribute to the explosion of the hypothesis space: the number of intendable goals, the nesting level of grammar rules (depth), the length of the right-hand side of grammar rules (branching factor), the number of rules sharing a left-hand side (OR-choice), the partial ordering constraints on the right-hand side of rules (constraints), and the number of actions. The grammar structure of the plan libraries is best explained by considering its AND-OR tree representation. The level of the AND-OR tree alternates between AND-nodes and OR-nodes, with the number of intendable goals set to 10, the depth to 4, the branching factor to 3, the OR-choice to 2, and with 100 unique actions. For each test, a plan library was generated with the structure just described, with terminal symbols randomly chosen among the 100 actions and the ordering constraint for each rule chosen randomly (an ordering constraint has a 33% chance of being added between any pair of symbols on the right-hand side of a rule). A sequence of observations derivable from an intendable goal was then randomly generated. Finally, the plan library and the sequence of observations were used as input for DOPLAR and YAPPR.

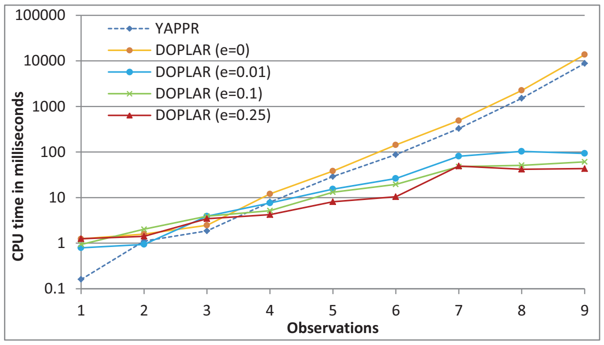
Figure 4 shows the results of the experiments for a sequence of 9 observations. The CPU times and number of hypotheses are plotted on a logarithmic scale. For each observation, the plotted y-value is the average over 100 tests, each involving a randomly chosen library and observation sequence. In Figure 4(a), DOPLAR computes goal-probability bounds up to the indicated threshold. Figure 4(b) shows the corresponding number of generated hypotheses. DOPLAR always performs significantly better than YAPPR except for the initial observation where DOPLAR's overhead outweighs the expansion of the initial empty hypothesis. The more ob-



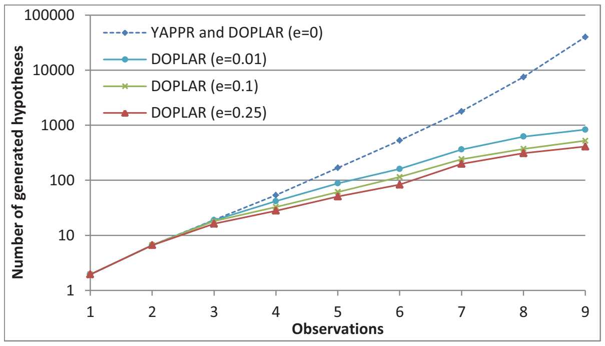
(a) Average CPU time for YAPPR vs. DOPLAR with probability threshold (pt)



(b) Average number of generated hypotheses for YAPPR vs. DOPLAR with probability threshold (pt)



(c) Average CPU time for YAPPR vs. DOPLAR probability interval (e)



(d) Average number of generated hypotheses for YAPPR vs. DOPLAR probability interval (e)

Figure 4: Comparison of DOPLAR and YAPPR. The x-axis is the sequence of observations (i is the i^{th} observation of the sequence). For each observation, the value (y-axis) is an average over 100 tests. The y-axis is plotted on a logarithmic scale.

servations, the larger the set of hypotheses, and the better the performance of DOPLAR compares to that of YAPPR. Assuming that an agent reacts to the goals of another agent depending on whether their probability is above a given threshold, it would be able to react much faster if it were using DOPLAR rather than YAPPR.

Figures 4(c) and 4(d) compare YAPPR to DOPLAR for errors on the uncertainty of goal probability computed by DOPLAR. The figures show that the exploration space can be efficiently controlled by tolerating some precision error on the goal probability using DOPLAR rather than trying to reach an exact probability using YAPPR. If DOPLAR is set to exhaustively generate all hypotheses (i.e., an error of 0), it generates the same hypotheses as YAPPR. However, as Figure 4(c) shows, the CPU time of YAPPR is slightly lower than DOPLAR's because of the overhead incurred by maintaining lower and upper goal-probability bounds. With much more complex plan libraries (obtained by increasing one of the random plan library generation parameters), YAPPR could not return any result in situations where DOPLAR was able to compute goal-probability bounds for different error values.

Tests on simplified plan libraries for the StarCraft domain produced results similar to those of the artificial domain.

7 Conclusion

We have introduced an anytime plan recognition algorithm based on weighted model counting. As for any other plan-library based approach, the learning of plan libraries would be a significant complement to this approach.

Previous extensions to the standard weighted model counting approach to handle temporal constraints [Geib and Goldman, 2009] and goal abandonment [Geib and Goldman, 2003] remain compatible with our approach. The method handling of unobserved actions suggested in [Geib and Goldman, 2005] would also be applicable to our approach, albeit it is intractable in practice.

Future work will aim for more complex plan libraries for StarCraft and comparison between our approach and approaches that do not require plan libraries, but rather rely only on primitive actions, such as [Lisý *et al.*, 2012; Ramírez and Geffner, 2010].

Acknowledgments

This work was supported by a grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada. We are grateful to Robert Goldman and John Maraist for making available their implementation of the YAPPR algorithm. We are also indebted to Philipe Bellefeuille and Francis Bisson for insightful comments on drafts of this paper.

References

[Avrahami-Zilberbrand and Kaminka, 2005] D. Avrahami-Zilberbrand and G. A. Kaminka. Fast and complete symbolic plan recognition. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 653–658, 2005.

[Bui *et al.*, 2002] H. H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract Hidden Markov Model. *Journal of Artificial Intelligence Research (JAIR)*, 17:451–499, 2002.

[Conati and VanLehn, 1996] C. Conati and K. VanLehn. Probabilistic plan recognition for cognitive apprenticeship. In *Proceedings of the 18th Annual Conference of the Cognitive Science Society (CogSci)*, pages 403–408, 1996.

[Demeester *et al.*, 2008] E. Demeester, A. Hüntemann, D. Vanhooydonck, G. Vanacker, H. Van Brussel, and M. Nuttin. User-adapted plan recognition and user-adapted shared control: A bayesian approach to semi-autonomous wheelchair driving. *Autonomous Robots*, 24(2):193–211, 2008.

[Geib and Goldman, 2003] C. W. Geib and R. P. Goldman. Recognizing plan/goal abandonment. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1515–1517, 2003.

[Geib and Goldman, 2005] C. W. Geib and R. P. Goldman. Partial observability and probabilistic plan/goal recognition. In *Proceedings of the IJCAI Workshop on Modeling Others from Observations (MOO)*, 2005.

[Geib and Goldman, 2009] C. W. Geib and R. P. Goldman. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, 117(11):1101–1132, 2009.

[Geib *et al.*, 2008] C. W. Geib, J. Maraist, and R. P. Goldman. A new probabilistic plan recognition algorithm based on string rewriting. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 91–98, 2008.

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

[Jarvis *et al.*, 2004] P. Jarvis, T. F. Lunt, and Myers K. L. Identifying terrorist activity with AI plan recognition technology. In *Proceedings of the 19th Conference on Artificial Intelligence (AAAI)*, pages 858–863, 2004.

[Kautz *et al.*, 2003] H. A. Kautz, O. Etzioni, D. Fox, D. Weld, and L. Shastri. Foundations of assisted cognition systems. Technical report, Department of Computer Science, University of Washington, Seattle, March 2003.

[Lesh *et al.*, 1999] N. Lesh, C. Rich, and C. L. Sidner. Using plan recognition in human-computer collaboration. In *Proceedings of the 7th International Conference on User Modeling*, pages 23–32, 1999.

[Lisỳ *et al.*, 2012] V. Lisỳ, R. Pıbil, J. Stiborek, B. Bošanskỳ, and M. Pechoucek. Game-theoretic approach to adversarial plan recognition. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, pages 546–551, 2012.

[Nederhof *et al.*, 2003] M.J. Nederhof, S. Shieber, and G. Satta. Partially ordered multiset context-free grammars and ID/LP parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies*, pages 171–182, 2003.

[Pynadath and Wellman, 2000] D. V. Pynadath and M. P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 507–514, 2000.

[Ramírez and Geffner, 2010] M. Ramírez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the 24th Conference on Artificial Intelligence (AAAI)*, pages 1121–1126, 2010.

[Sadilek and Kautz, 2010] A. Sadilek and H. Kautz. Recognizing multi-agent activities from GPS data. In *Proceedings of the 24th Conference on Artificial Intelligence (AAAI)*, pages 1134–1139, 2010.

[Sudkamp, 2006] T.A. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science*. Pearson international edition. Addison Wesley, 2006.

[Sukthankar and Sycara, 2008] G. Sukthankar and K. Sycara. Hypothesis pruning and ranking for large plan recognition problems. In *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*, pages 998–1003, 2008.

[Synnaeve and Bessière, 2011] G. Synnaeve and P. Bessière. A Bayesian model for plan recognition in RTS games applied to StarCraft. In *Proceedings of the 7th Arti?cial Intelligence and Interactive Digital Entertainment International Conference (AIIDE)*, pages 79–84, 2011.

[Vilain, 1991] M. Vilain. Deduction as parsing: Tractable classification in the KL-ONE framework. In *Proceedings of the 9th Conference of the American Association of Artificial Intelligence (AAAI)*, pages 464–470, 1991.