Plan Libraries for Plan Recognition: Do We Really Know What They Model? (Position Paper)

Robert P. Goldman
SIFT LLC
211 N. First St. Suite 300
Minneapolis, MN 55401, USA
rpgoldman@sift.info

Froduald Kabanza and **Philipe Bellefeuille**
Département d'informatique
Université de Sherbrooke
Sherbrooke, QC J1K 2R1, Canada
{kabanza, philipe.belefeuille}@usherbrooke.ca

Abstract

In this paper we explore challenges related to the engineering of plan libraries for plan recognition. This is an often overlooked problem, yet essential in the design of any real world plan recognizers. We mainly discuss challenges related to the evaluation of equivalence between plan libraries. We explain why this is a potential source of ill-conceived plan libraries. We consider an existing well-established probabilistic plan recognizer as vehicle for our discussion, using the formalism of probabilistic hierarchical task networks to represent plans. We propose avenues for exploring solutions to those challenges within that framework.

Introduction

In plan recognition, although formal representations are used to model behaviours of the observed agent, to date little attention has been paid to the knowledge engineering of plan libraries. This is in striking constrast to the area of formal methods, where experts know the importance of tools for analyzing the equivalence of program models. In state-based transition models, for example, theoretical tools such as equivalence by simulation, stuttering equivalence, and others, have been invented to make it possible to identify subtle differences in the behavior of programs that may appear equivalent at the surface but that differ in practice (Baier and Katoen 2008).

When designing a plan library, an important issue for the knowledge engineering is to determine when two plan structures are equivalent. It is possible for two plan libraries to look equivalent on the surface, while hiding subtle semantic differences that have a large effect on the outcome of the plan recognizer. While addressing such debugging problems for a plan library concerning a realtime strategy (RTS) video game, we ended up raising broader questions related to the knowledge engineering of plan libraries.

One question concerns equivalence between plan libraries. In other words, when can one plan library replace another without affecting the outcome of the plan recognizer? Intuitively, two plan libraries should be equivalent if they model the same behaviours. This becomes subtle when one consider that a plan library models to some extent internal choices or decisions made by the observed agent and that the modeled behaviour may be only partially observable.

Plan representation formalisms are for the most part judged based on their expressive power. From a knowledge engineering standpoint, we argue that another important quality for a plan representation formalism is *modeling sensitivity*, that is, the extent to which a given plan representation is susceptible to differences in semantics that are difficult to detect from their syntax.

The notions of plan equivalence and modelling sensitivity convey some of the difficulties knowledge engineers face when designing a plan library for a real world application. To initiate the discussion on these issues, we discuss the use of sampling from the probability distribution implicitly specified by the plan library (and some background assumptions). With such sampling, the knowledge engineer can answer questions about the equivalence and sensitivity between plan libraries.

Tools have already been investigated supporting the construction of planning domains (McCluskey, Liu, and Simpson 2003). Such tools are aimed at the knowledge engineering of the *planning domain* but not the *plan library*. When specifying a planning domain, the focus is on modeling actions or operators in terms of their preconditions, effects, related resources and constraints. In contrast, a plan library as required by a plan recognizer, is a specification of abstract plans, conveying a richer structure than just actions or operators, in terms of hieararchy of subgoals and/or subplans, and temporal relationship between them, the lower level of the hierarchy being the actions or operators. As there are modeling issues that are specific to plan libraries, we argue that appropriate corresponding knowledge engineering methods should be investigated.

To fix a context, here we consider plan libraries modeled using probabilistic hierarchical task networks (PHTN). However, the key points of our discussion can be generalized to different plan representation formalisms.

Motivating Example

Real Time Strategy (RTS) games involve multiple opposing forces, each consisting of a team of one or more players, and each player controlling a number of military units. Units are typically recruited from structures that the player builds using different resources, as the game advances. The victory conditions can be different from one scenario to another, but usually involves destroying some or all of your opponents' structures or controlling strategic points for a certain period of time. As armies build up on each side, initial skirmishes can quickly turn into all-out wars.

Players in RTS games need to be constantly aware of the behaviours of their opponents to anticipate and counter their future moves. Correctly predicting the adversary's actions is key to deciding winning moves, whereas incorrectly identifying it often puts the player in a dire position. In modeling the plan library for an RTS domain, one can distinguish between strategic and tactical plans. Strategic plans dictate what kind of units the player will produce, whether he will play aggressively, or defensively. Tactical plans dictate how units are deployed and used. These two types of plans are not independent; the strategy a player chooses will affect the composition of his army and therefore will affect how he will use it on the tactical level. It is useful to separate them, however, because the cues used to recognize strategic plans are for the most part different from the ones used to recognize tactical plans. Recognizing these plans separately does not prevent using the output of one plan recognizer as input to the other.

Figure 1 illustrates an oversimplified strategic plan library for StarCraft, representing possible openings for the Zerg faction. The trees in this figure are task decomposition trees, which are partially-ordered AND/OR trees. As is customary, AND nodes are indicated by subtrees crossed with an arc, so that economic-opening-a is done by performing all of its children: recruit 5 workers, recruit overlord, recruit 3 workers, etc. OR nodes are subtrees without arcs across them, so that build may be done *either* by doing 12 pool *or* 12 hatch. Cross edges indicate temporal orderings. All of the sub plans in Figure 1 are totally ordered.

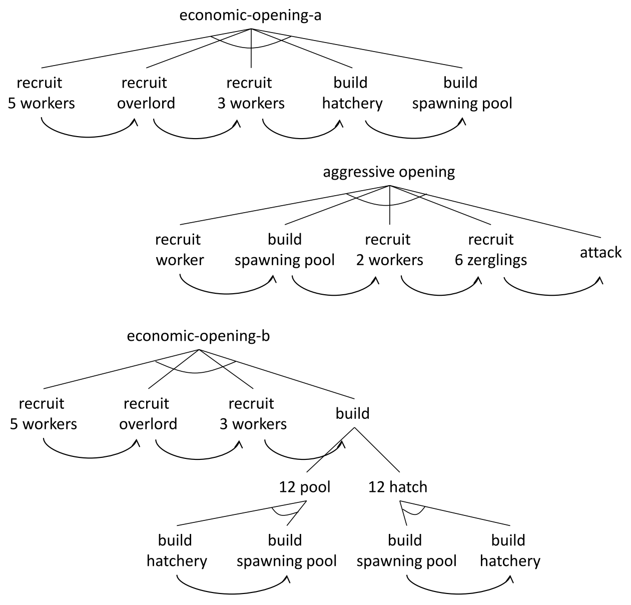

Figure 1: Simplified strategic plans for StarCraft

To make the discussion concise, the rest of the paper will be illustrated by artificial and much simpler plan libraries like those in Figure 2. They model the basic issues related to or-branch choices and action interleaving that are encountered in the StarCraft plans, while being more terse. Note that $G_1 - a$ shows partial ordering, a must be done before b and c, but b and c may appear in either order.

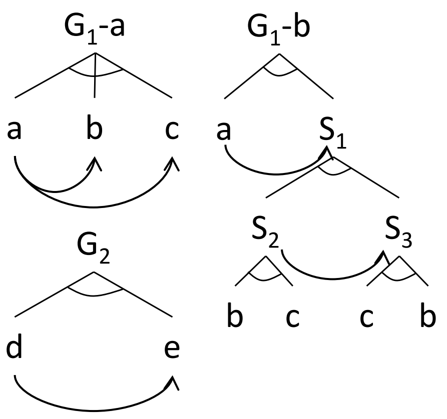

Figure 2: Simple artificial plan libraries. $L_1 = \{G_1 a, G_2\}$, $L_2 = \{G_1 b, G_2\}$.

Our interest in these issues was triggered when we encountered the following issue: Consider the following two plan libraries, L_1, that consists of the HTNs $G_1 a$ and G_2 from Figure 2, and L_2, that consists of $G_1 b$ and G_2. Note that $G_1 a$ and $G_1 b$ generate the same strings of symbols – a followed by b and c in either order. One may jump to the conclusion that the probability of seeing b after seeing a followed by d would be the same for both librairies. If the result is different, then there must be a bug somewhere. However, according to the probability model proposed for PHATT (Geib and Goldman 2009) these two libraries generate quite different distributions: in particular with L_1 the probability of seeing b after having seen a followed by d is $1/3$, whereas the result is $1/4$ with L_2.

Two of our co-authors (Kabanza and Bellefeuille) became convinced that there must be a bug somewhere, and the issue bothered them for a long time. The question for a long time was: Why are the probabilities different if the plan libraries are equivalent? After they consulted the PHATT co-author, he too got trapped by the cognitive bias for a while, before being able to identify what made the models diverge: while they are *linguistically* equivalent, they are not *probabilistically* equivalent. In the following section of the paper we will briefly introduce the probabilistic plan recognizer PHATT and the probability model it assumes. Then we will return to this seeming paradox.

PHATT: A Bayesian plan recognition system

PHATT (Geib and Goldman 2009) is a Bayesian plan recognition system. At the core of PHATT is a probabilistic model of how action sequences are generated by an agent that is following some unobserved plan taken from a plan library that is known by the observer (Goldman, Geib, and Miller 1999). The approach is Bayesian in that the plan library (with some ancillary assumptions) describes a probability distribution over the possible action sequences. This probability distribution is *inverted*, using Bayes' law to identify the plans that underly a given action sequence.

Plan recognition systems in the PHATT family (which also include the original, unnamed system (Goldman, Geib, and Miller 1999) and the heavily optimized Yappr (Geib, Maraist, and Goldman 2008)), assume a very simple HTN model, as partially-ordered AND/OR trees or, equivalently, as plan grammars. Plan grammars are similar to context free grammars (CFGs), with terminals for the observable actions. Unlike CFGs, the right hand sides of plan grammar rules are unordered sets, rather than ordered sequences. Rules can be annotated with ordering constraints that specify that (the expansion of) one symbol must precede (the expansion of) another. For example, a plan grammar rule, $S \rightarrow a, b$ could generate either the string a, b or the string b, a. With an added ordering constraint, $S \rightarrow a, b; b < a$, could only generate the string b, a.

The generative model of plan execution that PHATT assumes is as follows:

1. The agent first chooses a set of one or more top level goals to pursue.
2. For each of these top level goals, the agent chooses a decomposition (see, e.g., (Ghallab, Nau, and Traverso 2004)) that reduces it to a partially-ordered set of primitive actions.
3. The agent executes these (interleaved) plans, at each time point choosing a single action from the set of *enabled* actions that constitutes the agent's *pending set*. An action is *enabled* once all of its predecessors have been performed.

The choices in this process are modeled probabilistically. Typically, for lack of more information, we have modeled these choices uniformly, assigning an equal probability to each possible outcome. However this is in no way an inherent feature of the model; had one good reasons to choose different probabilities, they could easily be incorporated.

Given a plan library $\mathcal{P}$ and a goal $\mathcal{G}$, with a slight abuse of notation, we can describe PHATT model for computing the probability of a sequence of observations $\sigma = \sigma_1 ... \sigma_n$ as follows:

$$\begin{aligned} p(\sigma) &= p(\sigma_1 ... \sigma_n) && (1) \\ &= p(\sigma_1 ... \sigma_n | \mathcal{G}, \mathcal{P}) p(\mathcal{G}, \mathcal{P}) && (2) \\ &= p(\sigma_1 ... \sigma_n | \mathcal{P}) p(\mathcal{P} | \mathcal{G}) p(\mathcal{G}) && (3) \end{aligned}$$

σ is a sequence of actions (1). The probability of an action sequence is conditional on the agent's goals ($\mathcal{G}$) and plans ($\mathcal{P}$) (2). The agent's actions are directly conditional only on the plans, and the plans only on the goals (3). Finally, each step is independent of its predecessors, given the state of the pending set (π), and the initial pending set is conditional on the plans:

$$\begin{aligned} p(\sigma_n | \sigma_1 ... \sigma_{1-n}, \mathcal{P}\mathcal{G}) &= p(\sigma_n | \pi_n) \times && (4) \\ & \quad p(\pi_n | \sigma_1 ... \sigma_{1-n}, \mathcal{P}, \mathcal{G}) \\ p(\pi_1 | \mathcal{P}, \mathcal{G}) &= p(\pi_1 | \mathcal{P}) p(\mathcal{P} | \mathcal{G}) && (5) \end{aligned}$$

Prior work has shown how to invert this generative model, given a particular string of observations, σ to recover quantities of interest such as $p(G|\sigma)$, the probability that the agent is pursuing some particular goal, G, given that we have observed the string of actions, σ. Several other queries of interest can be answered in the same way.

Resolving the paradox

Let us return to the seeming paradox that we introduced earlier. The two grammars of Figure 2, which appear intuitively to be equivalent, actually generate different probability distributions.

For a worked example, see Figure 3, which shows the different probabilies that L_1 and L_2 assign to the sequence $abcdef$. In the notation of Figure 3, $p(d(x)|X)$ is the probability of drawing the terminal/action x from the pending set X. For example, the first thing that happens in generating the string $abcdef$ is to draw the action a out of a pending set that contains the actions a and d, and the probability of this event is $p(d(a)|\{a, d\})$. For the purposes of this discussion, we assume uniform probability distributions: $p(d(x)|X) = p(d(y)|X)$ for all pending sets X and all distinct actions x and y in X. Similarly, the chance of choosing any node for an OR is equal, so $p(S_2|S_1) = p(S_3|S_1)$.

The mystery deepens when we see that both L_1 and L_2 generate the same action sequences for single goals. That is

$$p(abc|G_1, L_1) = p(acb|G_1, L_1) = \\ p(abc|G_1, L_2) = p(acb|G_1, L_2)$$

and, of course,

$$p(def|G_2, L_1) = p(def|G_2, L_2)$$

The key to the seeming paradox is to recall that there are three components to the PHATT probability model. The first is the choice of goals. This component of the model is not involved here — we have limited ourselves to discussing the probabilities of different action sequences *conditioned on a known set of goals*. The second component is the decomposition of goals to partially-ordered sets of actions, given the top-level goals. We have seen above that this is not the issue at hand, since the two libraries generate the same partially ordered sets in isolation. Our attention, then, must turn to the third component of the model — the way actions are drawn from the pending set, conditioned on that pending set's composition (it is populated by these partially ordered sets of actions).

Let us return to the example where we first see a, followed by d. With the "flat"plan library, we are certainly in a state in which the next action will be chosen from the pending set $\{b, c, e\}$, so, if the distribution is uniform, the chance of seeing any one of these symbols next is $\frac{1}{3}$. On the other hand, if we have the hierarchical library, then the agent is in one of two possible states/pending sets, either $\{b, e\}$ or $\{c, e\}$ and now we can clearly see that, assuming uniformity again, the chance of e is $\frac{1}{2}$ whereas the chance of either b or c is $\frac{1}{4}$.

We may get a clearer sense of the mechanism, by examining an even simpler plan library, with only a single top-level

$$\begin{aligned}
p(abcdef|G_{1a}, G_2, L_1) &= p(d(a)|\{a,d\})p(d(b)|\{b,c,d\})p(d(c)|\{c,d\}) \\
&\quad p(d(d)|\{d\})p(d(e)|\{e\})p(d(f)|\{f\}) \\
&= 1/2 * 1/3 * 1/2 \\
&= 1/12
\end{aligned}$$

$$\begin{aligned}
p(abcdef|G_{1b}, G_2, L_2) &= p(abcdef|G_1 - b, G_2, S_2)p(S_2|G_{1b}) \\
&= p(abcdef|P_1, P_2, S_2, G_2)/2 \\
&= p(d(a)|\{a,d\})p(d(b)|\{b,d\})p(d(c)|\{c,d\}) \\
&\quad p(d(d)|\{d\})p(d(e)|\{e\})p(d(f)|\{f\})/2 \\
&= p(d(a)|\{a,d\})p(d(b)|\{b,d\})p(d(c)|\{c,d\})/2 \\
&= 1/2 * 1/2 * 1/2 * 1/2 \\
&= 1/16
\end{aligned}$$

Figure 3: L_1 and L_2 assign different probabilities to $P(abcdef|G_1, G_2)$.

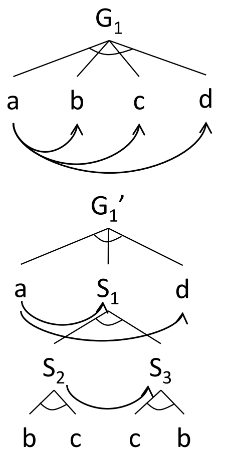

Figure 4: Simple plan library.

Sequence	G_1 prob.	G_1' prob.
abcd	.083	.062
abdc	.083	.062
acbd	.083	.062
acdb	.083	.062
adbc	.083	.125
adcb	.083	.125
dabc	.250	.250
dacb	.250	.250

Table 1: Table showing exhaustive generation from library of Figure 4.

goal, shown in Figure 4. For this simple example, we can exhaustively enumerate the set of strings generated with but a single goal, and we present the results in 1. Here it is the partially-ordered node d that causes the divergence.

Language equivalence and probabilistic equivalence

The examples we have discussed above show that it is critical to distinguish between two notions of equivalence when reasoning about probabilistic HTNs. We say that two plan libraries are *language equivalent* if they generate the same set of action sequences (i.e., equivalent if we ignore the probabilities). We say that they are *probabilistically equivalent* if they are language equivalent and each sequence is generated with the same probability with both libraries.

We argue that when working with PHATT and assuming a uniform probability distribution, a cognitive error bias is created, leading a knowledge engineer to mistakenly believe that linguistically equivalent plan libraries will also be probabilistically equivalent. While modeling plan librairies for the StarCraft domain, Kabanza and Bellefeuille had two linguistically equivalent plan librairies, yielding different goal recognition probabilities under the PHATT model. It took us time to realize that this was normal since the librairies were not in fact probabilistically equivalent, albeit language equivalent.

The divergence betwen linguistic and probabilistic equivalence is problematic because it makes it hard for a knowledge engineer to specify correct plans. How does one know when G_1 is the correct model and when G_1' is the one that should be used? One problem comes from the fact that the uniformity hypothesis hides the probabilities in the structure. If one was to explicitly represent the probabilities in the model, it might alleviate this problem somewhat. A more substantial problem is that the probability distributions are a function of both the plan library, which the knowledge engineer constructs, and the pending set/interleaving model, which the knowledge engineer may not perceive at all. Indeed, as we have seen in our first example, a knowledge engineer who explores the probability distributions with only single plan trees, may in some case fail to even encounter the issue (although the second example shows that even simple plan libraries may bring the issue to the fore).

One thing that makes this worrisome is that one may inadvertently generate different probability distributions when simply trying to provide a more "tidy" plan library. This is the case in our first example. This is particularly bothersome for a community like the computer science community, in which one often encapsulates bits of behavior to provide information-hiding. It seems wrong that such encapsulation should involve changing the probability distributions.

In the next section we discuss some preliminary directions we have considered towards giving plan library coders a better sense of the implications of their modeling decisions.

Exploring the distributions

To evaluate probabilistic equivalence between two libraries it is tempting to try getting inspiration from formal method approaches to defining equivalence between state transition systems (Baier and Katoen 2008). Many notions of equivalence between state transition systems exist, including bisimulation equivalence, trace equivalence, and stutter equivalence. Grossly, two state transition systems (or finite state machines) are bisimilar if each state in one system has a corresponding state in the other system exhibiting the same stepwise behaviours as the former state; in other words, each outgoing transition from one state is mimicked by a corresponding outgoing transition from the other state. Two systems are trace equivalent if they exhibt the same execution sequences (traces). The notion of stutter equivalence accounts for internal (invisible) transitions. More specifically, two systems are stutter trace equivalent if their traces only differ by internal transitions. A stutter bisimulation is defined likewise.

It is interesting to note that algorithms exhist for computing those equivalence relations, that is, establishing whether two transition systems are equivalent. Trying to adapt such algorithms to PHTNs pose two main challenges. Firstly, PHTNs convey more than transitions between steps. In addition, they convey hierarchical abstractions. With respect to state transition models, they are better compared to state charts (Harel 1987). Secondly, PHTNs include a probabilistic model. For the time being, we do not see how to overcome these two issues. We are instead investigating an approach based on the sampling of the probability distributions for PHTNs.

Since the PHTNs implicitly represent a probability distribution over a set of action sequences, to evaluate different plan libraries we must compare the probability distributions to which they give rise. Indeed, we have written a very simple, exhaustive generator to allow us to explore the distributions corresponding to the very simple grammars in discussed in this paper. This very simple tool allows us to enumerate the probability distribution for a given plan library and a given set of top level goals.[1] We have seen an example of the output of this system in Table 1.

[1]Without loss of generality, it could do the same task for any finite number of top level goals, simply by adding a top-level terminal that can rewrite to the "real" goals.

Exhaustive enumeration will not, of course, scale up to realistic plan libraries, since generation involves interleaved permutations. We expect that realistic plan libraries will require us to *simulate* the distributions rather than enumerate them exhaustively. Samples from the distribution will be inexpensive to generate, so this seems like a plausible strategy.

The ability to explore these distributions through simulation is unlikely to be a sufficient engineering tool, however. Realistically large plan libraries are going to give us distributions whose event space will be too large to visualize and compare effectively (except for interesting small cases such as small numbers of known top level goals). Instead, we will need to be able to use summary measures of similarity. We suspect that measures of cross-entropy (Wikipedia 2009) between two distributions (estimated by simulation) are likely to be useful for quantifying divergences:

$$H(p,q) = \sum_x p(x) \log q(x)$$

Note also that we can use conditional cross-entropy to measure divergences in specific, interesting parts of the probability space.

Ideally, we will not simply be comparing plan libraries against each other, but also against corpora of observed action sequences. If we have a substantial corpus, we can measure the entropy of a given plan library with respect to a distribution estimated from the corpus. We can also compare different plan libraries based on how well they model the corpus, by measures such as comparing the conditional probability of the corpus given each of the plan libraries.

Measuring the conditional probability of corpora given different plan libraries may not be an adequate measure of the incremental performance of a recognizer. Incremental performance may be of the greater interest. Typically one wishes to observe some partial action sequence in order to be able to interact with it either positively (as in intelligent help systems) or negatively (as in an AI opponent for a game). Given an unlabeled corpus, one may still use a plan recognizer like PHATT, with a generative model, to recognize the prefix of an action sequence, and then either find the conditional probability of the suffixes, or measure accuracy of predictions of next actions or some other prediction.

Conclusion

In this paper we have illustrated some knowledge engineering challenges working with a probabilistic HTN model for plan recognition. We have discussed the sources of these challenges, notably in interactions between different parts of the model. We have specifically shown how the model underlying PHATT and Yappr gives rise to the these challenges because of the way it combines the modeling of decomposition planning and interleaving. We have proposed some steps for exploring the implications of knowledge engineering decisions that we plan to explore in future work.

References

Baier, C., and Katoen, J.-P. 2008. *Principles of Model Checking*. MIT Press.

Geib, C. W., and Goldman, R. P. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence* 173(11):1101–1132.

Geib, C. W.; Maraist, J.; and Goldman, R. P. 2008. A new probabilistic plan recognition algorithm based on string rewriting. In Rintanen, J.; Nebel, B.; Beck, C.; and Hansen, E., eds., *Proceedings of the International Conference on Automated Planning and Scheduling*, 91–98.

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

Goldman, R. P.; Geib, C. W.; and Miller, C. A. 1999. A new model of plan recognition. In Laskey, K. B., and Prade, H., eds., *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*, 245–254. S.F., Cal.: Morgan Kaufmann Publishers.

Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Science of Computer Programming* 8(3):231–274.

McCluskey, T. L.; Liu, D.; and Simpson, R. M. 2003. GIPPO I: HTN planning in a tool-supported knowledge engineering environment. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 92–103.

2009. Cross entropy. Wikipedia entry.