Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial
ISSN: 1137-3601
revista@aepia.org
Asociación Española para la Inteligencia Artificial
España

Marchetta, Martín; Forradellas, Raymundo
Supporting Interleaved Plans in Learning Hierarchical Plan Libraries for Plan Recognition
Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, vol. 10, núm. 32, 2006, pp. 47-56
Asociación Española para la Inteligencia Artificial
Valencia, España

ARTÍCULO

Supporting Interleaved Plans in Learning Hierarchical Plan Libraries for Plan Recognition

Martín Marchetta*, Raymundo Forradellas

Facultad de Ingeniería, Universidad Nacional de Cuyo
Centro Universitario, (5500) Mendoza. Argentina
mmarchetta@fing.uncu.edu.ar, kike@uncu.edu.ar

Abstract

Most of the available plan recognition techniques are based on the use of a plan library in order to infer user's intentions and/or strategies. Until some years ago, plan libraries were completely hand coded by human experts, which is an expensive, error prone and slow process. Besides, plan recognition systems with hand-coded plan libraries are not easily portable to new domains, and the creation of plan libraries require not only a domain expert, but also a knowledge representation expert. These are the main reasons why the problem of automatic generation of plan libraries for plan recognition, has gained much importance in recent years. Even when there is considerable work related to the plan recognition process itself, less work has been done on the generation of such plan libraries. In this paper, we present an algorithm for learning hierarchical plan libraries from action sequences, based on a few simple assumptions and with little given domain knowledge, and we provide a novel mechanism for supporting interleaved plans in the input example cases.

Keywords: Plan Recognition, Intelligent Agent, Machine Learning, Computer Aided Process Planning.

1 Introduction

Software applications and their interfaces have reached a great complexity level. Applications assisting users in knowledge rich domains, such as Computer Aided Process Planning (CAPP) in make-to-order or assembly-to-order production strategies (our main application domain), are more and more common nowadays. This situation yields the need of more natural, powerful and easy interaction between humans and these applications.

Plan recognition techniques have been studied in many contexts, from natural language understanding [1], to interface agents [15]. These techniques are useful for many domains, and in many contexts, because the need of recognizing user's intentions from his behavior is present virtually whenever a human interacts with intelligent machines. The inferring of user's intentions is a key point for building intelligent agents more integrated in their environments, and more useful for their users. Assistance given by intelligent agents to their users is better if these agents understand their users' intentions, and plan recognition is a useful technique for this task.

Most existing plan recognition techniques, are based on the use of a plan library previously cre-

*Also CONICET

ated ([10], [11], [14], [7]). Until some time ago, these plan libraries were completely hand coded by human experts. In recent years, the automatic and semi-automatic generation of plan libraries has gained much importance, and some approaches, with different results, have been developed. These approaches automate some of the tasks that are needed, in order to build plan libraries. However, all of them require labeled examples, or requires much hand-coded knowledge what makes them very restrictive. As far as we know, the automatic acquisition of hierarchical plan libraries, from unlabeled example cases, has not been achieved yet.

In [16], some preliminary ideas and an algorithm for learning hierarchical plan libraries were presented. In this paper, we present a refinement of that work, which includes not only the capability of learning hierarchical plan libraries from unlabeled examples, but it also allows these examples to contain interleaved plans. In order to illustrate the proposed approach, a simple example from the flexible packaging industry domain described in [8], is given in this work.

The rest of this paper is organized as follows. The next section gives an overview of the motivation and objectives of this work. Section 3 details some of the issues that are addressed in this paper. Section 4 presents the algorithm for generating plan libraries. Section 6 presents a discussion of related work. Finally, section 7 presents the conclusions of this paper and the future work.

2 Motivation

Even when the algorithms presented in this work could be applied to plan recognition systems for many different application domains, our main interest is focused in CAPP domain (Computer Aided Process Planning). In this section we briefly describe some CAPP peculiarities, and some issues of current plan recognition systems that difficult their application to this domain.

2.1 The CAPP Domain

CAPP is a set of activities related to the manufacturing management and planning. It includes tasks such as part routing design (i.e, definition of the individual manufacturing processes needed for the production of a part), machines and tools selection, bill of materials elaboration, and production scheduling [9]. Our interest is particularly focused on the creation of intelligent agents to interactively assist process planners in their tasks.

There are several factors that affect decisions of human process planners, such as experience, knowledge of manufacturing processes, design styles, etc. Additionally, different products to be manufactured can share some parts of their manufacturing process plans with other products. This situation gives opportunities to apply plan recognition techniques for creating intelligent agents capable of assisting human process planners in their design tasks. These kind of techniques could automatically learn the "know-how" from expert process planners, and then assist less experienced engineers in these tasks. Moreover, such systems could also help experienced process planners during their tasks, by inferring their intentions and helping them in several ways (creating partial plans on their behalf, giving them advising or detecting errors based in the knowledge stored in the plan library, etc).

2.2 Creating the Plan Library

Automatic generation of plan libraries is important for many reasons. Hand-coding plan libraries is expensive, tedious and error prone. It also requires intensive work of a domain expert (the user of the application), and a knowledge representation expert (the domain modeler). Moreover, a specific plan library is required for each domain, so the creation of these libraries is done many times, which is a big burden in complex domains.

There also exist domains where primitive events do not change frequently, while valid combinations of these elements do vary. This is the case of CAPP domain, where the basic manufacturing capabilities of a factory do not change frequently, while the combination of these capabilities in complex manufacturing processes do.

In these scenarios, the automatic learning of plan libraries, can significantly reduce the work of the knowledge representation expert when the plan libraries are created (e.g, when adapting the system to a new domain), as well as when they are maintained (e.g, when new manufacturing processes are created or not used anymore). In this work, we aim at learn a hierarchy of plans the user usually pursues (which changes frequently), only

using hand-coded information about the primitive events that could be observed (which does not vary frequently).

There exist in the literature some approaches to learning plan libraries. However, even when these techniques alleviate the burden of generating libraries for new domains, they do it with limited results or they require labeled examples. The methods for generating *expressive* plan libraries, still require human labeling and explicit training. Moreover, as far as we know, these works do not address explicitly the situation when the observed action sequences contain interleaved plans.

The main reason why interleaved plans appear in action sequences, is because humans frequently executes simultaneously different tasks. In the proposed application domain however, interleaved plans could appear more frequently when the system learns some concept that, even when exist, is not useful during the recognition process (such as some combination of manufacturing processes whose parts, even when observed in some action sequence representing a process plan, are not frequently related to each other).

A common example of the described situation is that where a product, in order to be produced, requires some special combination of manufacturing processes that is only required for that product. If the learning algorithm "remembers" all this particular cases, the plan recognizer accuracy could be reduced, as will be seen in following sections. The desired situation is that in which only the important manufacturing processes (and their subprocesses) are learned, discarding the rest of them.

The learning of hierarchical libraries from unlabeled examples that may contain interleaved plans, using primitive actions representations as the only hand-coded knowledge, is the central topic of this paper.

3 Plan libraries generation issues

A decomposition of an action (also called *event*, or *task*), is a set of simpler actions that must be done in order to accomplish it. The actions corresponding to a decomposition, can be either primitive or non-primitive. Besides, an action can have alternative decompositions, and all of them accomplishes it.

In order to infer such decompositions, a learning algorithm must be provided. Such an algorithm, must have the capability of identifying higher-level actions, based on the primitive ones observed in the unlabeled action sequences.

Another problem that must be addressed, is that of interleaved plans. Previous works assume that each action sequence observed, contains only actions related to one goal [2], or that there may be actions for more than one goal and the hierarchical decomposition is not inferred, but given by the trainer by means of labeled examples ([5], [6]). This could be true if the action sequences, are provided by an expert, or by somebody that is explicitly training the system, but it would be useful to have an algorithm that can operate even with interleaved plans and unlabeled examples.

4 Hierarchical plan libraries learning

In this paper, a similar model to that described in [10] and [11] is used for plan libraries representation. In the mentioned works, the representation language is first order logic, and a simplified graph representation is also proposed, which we adopt here since it is enough to illustrate the proposed learning mechanism of partial plan libraries.

Under this representation, a plan library is a graph containing *actions* (also called *events*) as its nodes. Actions are connected by 2 kinds of edges: thick gray arrows represent "is a" relationships (abstractions); thin black arrows represent "part of" relationships (decompositions). A special action called *End Event*, has a "is a" relation with all the "top-level" actions (actions executed for their own sake). The complete formal model supports the representation of further features (such as explicit representation of events orderings and arbitrary restrictions between them).

The algorithms proposed in this paper learn plan libraries only partially, because these additional (and necessary) features are not taken into account; despite of this, our internal representation model supports them (for example, the decompositions' model allows the specification of pairs of actions where the first one must be executed before the second one). These simplifications will

be relaxed in future works.

Some assumptions are made about the characteristics of action sequences observed and the plan libraries learned. In first place, we assume that each action sequence is complete (i.e, contains at least all the actions necessary to achieve every goal present on it), and does not contain spurious actions. For example, if the action sequence AS_1 contains actions for two interleaved plans P_1 and P_2, then it contains all the actions needed for achieving the goals of P_1 and P_2. Besides, we assume that if two different plans appear *always* together in the observations, then they may be considered as the same one (there is no evidence that allows the plan library learner to infer the division of the plans). Finally, if two actions shares parts of their decompositions, the hierarchy generated may be one that is equivalent (but not equal) to the real one.

The assumption that action sequences are complete and free of spurious actions is particularly important at this stage of the research. If the action sequences are incomplete, as well as if they contain spurious primitive actions (such as user's mistakes), the proposed algorithm will generate plan libraries that include these anomalies. Spurious actions will be included within the decomposition of the inferred non-primitive ones, and if some sequences are incomplete, non-primitive events with incomplete decompositions will be introduced in the plan library.

Additionally, if we assume that sooner or later the correct action sequence is observed (i.e, the action sequence that is complete and error free), the processing of this sequence will result in the inclusion of the correct non-primitive event into the knowledge base, so the consequence of processing action sequences with the mentioned anomalies will be that plan libraries learned could include correct hierarchical definitions of objectives and plans, and incorrect ones. Little work has been done on this topic, and some approaches propose very simple heuristics to filter out spurious actions (see [2]). The problem of avoiding incorrect learning of plan libraries, or their later correction will be addressed in future works.

4.1 Decompositions

A decomposition of a non-primitive event, is the set of parts of that event that must be done in order to accomplish it. In order to learn a hierarchical plan library, an algorithm for inferring such decompositions from the observed action sequences is needed.

A desirable feature of a hierarchical decomposition learning algorithm is it to be incremental, and to work in an on-line fashion, so the previously made inferences are used in subsequent steps, instead of storing the example cases and process (or re-process) them in batch.

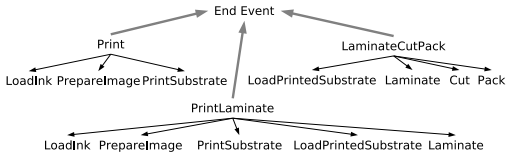

Figure 1. Simple plan library for packaging industry domain

Intuitively, whenever a set of new actions appears in an action sequence, this set of actions can be considered as a decomposition for some new non-primitive one that represents the observation. Consider the plan library presented in figure 1, as an example. Initially, the system has an empty library, only containing the *End Event*. Now, suppose that the system processes the sequence:

$$AS_1 = \{LoadInk, PrepareImage, PrintSubstrate\}$$

Because none of the actions have been seen before, the system creates a new composed action, named $Print'$ with $LoadInk$, $PrepareImage$ and $PrintSubstrate$ as its parts (we denote it with an apostrophe to show that the system will assign an arbitrary name to it).

When an observation contains parts of known non-primitive events as well as unknown primitive actions, they can be considered as the parts of a more general event that represents the entire sequence. For example, suppose that the system processes:

$$AS_2 = \{LoadInk, PrepareImage, PrintSubstrate, LoadPrintedSubstrate, Laminate\}$$

Here, events $LoadInk$, $PrepareImage$ and $PrintSubstrate$ are identified as a decomposition of $Print'$, so the algorithm will create a new composed action named $PrintLaminate'$, which has

Print′, *LoadPrintedSubstrate* and *Laminate* as its parts. In figures 2.a and 2.b, the states of the plan library after the first and second observation, are shown.

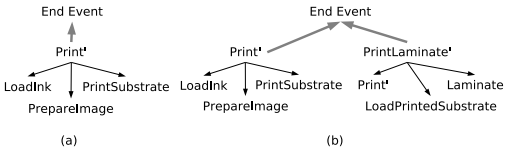

Figure 2. Plan library under construction

Now suppose that the action sequences were seen in the inverse order. After the first sequence, the system creates *PrintLaminate′*, with *LoadInk*, *PrepareImage*, *PrintSubstrate*, *LoadPrintedSubstrate* and *Laminate* as its parts. When the second sequence is processed, only a part of a known action is identified. In these cases, the system must figure out that in fact, *LoadInk*, *PrepareImage* and *PrintSubstrate* are part of *Print′*, which in turn is part of *PrintLaminate′*. Thus, the action taken by the system should be to create *Print′*, and modify *PrintLaminate′* as needed. Finally, the new action is added to the plan library, in accordance with the decomposition inferred, obtaining as before the state shown in figure 2.b.

Definition 1 (Complete decomposition). *A set of actions AS, containing all the parts of a non-primitive event E_c, and no other events, is a complete decomposition of E_c.*

Definition 2 (Partial decomposition). *A set of actions AS, containing some parts of a non-primitive event E_c, but not all of them, and no other events, is a partial decomposition of E_c.*

The base of the method described intuitively before, is the identification of unknown actions, complete decompositions and partial decompositions. In absence of further information, all new action sequences are considered to be a decomposition of a new non-primitive event. With the arrival of new example cases, the plan library is refined successively. These refinements are based on the *complete* and *partial decompositions* identified in the new observations.

A schematic algorithm, named HIDEL (HIerarchical DEcomposition Learner), for generating hierarchical plan libraries is shown in Figure 3.

```
HIDEL(AS , PLibrary)
Inputs:
   AS: A new action sequence
   PLibrary: The current plan library
BEGIN
   do
      CompDec = IDENTIFYCOMPLETEDECOMPOSITIONS(AS)
      for each decomposition Dec_i in CompDec
         remove Dec_i from AS
         add composedEvent(Dec_i) to AS
   while elementCount(CompDec) > 0

   PartDec = IDENTIFYPARTIALDECOMPOSITIONS(AS)
   for each decomposition Dec_i in PartDec
      if elementCount(Dec_i) > 1
         NewCompEvent = CREATEEVENT()
         add Dec_i to decomposition(NewCompEvent)
         remove Dec_i from AS
         add NewCompEvent to AS
         replace Dec_i with NewCompEvent in PLibrary

   if elementCount(AS) > 1
         NewCompEvent = CREATEEVENT()
         add AS to decomposition(NewCompEvent)
         add NewCompEvent to PLibrary as End Event
   else
         Event = getEvent(1,AS)
         if not(isSpecialization(Event, End Event))
            add Event to PLibrary as End Event
END HIDEL
```

Figure 3. HIDEL Schematic algorithm

We now briefly describe some functions present in figure 3. The function *composedEvent(seq)* returns the non-primitive event associated with the (complete or partial) decomposition *seq*; *getEvent(i,seq)* returns the *i-th* element of the action sequence *seq*; *decomposition(evt)* returns a reference to the list of events that are part of *evt*; finally, *isSpecialization(evt1, evt2)* returns *true* if *evt1* is a specialization of *evt2* (i.e, if evt1 "is a" evt2). HIDEL identifies complete and partial decompositions present in the action sequence being learned. The algorithm transforms the original action sequence while processes each complete and partial decomposition identified. The following operations are done:

a. Complete decompositions are identified. For each complete decomposition found, the corresponding non-primitive event's parts are replaced in the action sequence by the non-primitive event itself. This operation is repeated until no more complete decompositions are detected in the processed action sequence

b. For each partial decomposition identified, a new non-primitive event is created with the elements of the partial decomposition as its parts. The partial decomposition is then

the action sequence. Finally, all the occurrences of the partial decomposition, are replaced in the plan library by the new non-primitive event created.

c. Finally, if the processed sequence does not correspond to a complete decomposition (i.e, if the processed action sequence has more than one event), a new non-primitive event is created with the actions in the sequence as its parts. The new non-primitive event is added to the plan library as an End Event. On the other hand, if the processed sequence has only one event (if it corresponds to a single complete decomposition), a check is made in order to know if the corresponding event was already an End Event, and if not a new abstraction relationship is created between them.

The general idea of the algorithm is to recognize known non-primitive events (present in the plan library), by observing their parts in the action sequences. Thus, when a complete decomposition is detected, it is replaced in the action sequence by the corresponding composed event, and the algorithm repeats this process until it reaches the "highest" non-primitive events whose components were observed. Partial decompositions, on the other hand, represent subsets of parts shared by more than one composed event (at least one already present in the plan library, and the new one represented by the action sequence currently being processed), so when they are detected, they are grouped as parts of a new composed event, so the partial decomposition (i.e, the shared subset of events) is replaced in the plan library by the new composed event created.

It is important to note that, if the parsed sequence corresponds to a complete decomposition and the corresponding non-primitive event is already an End Event, the plan library is not modified since it already contains the information provided by the observation. Thus, if no new combinations of primitive actions are presented to the algorithm, no further modifications are made to the plan library.

4.2 Interleaved plans

When an observed action sequence contains events of more than one plan, some special problems arise. Interleaved plans generate difficulties, because in absence of further information, a plan library builder can not distinguish between two interleaved plans, and an individual composed one. Interleavings of plans included explicitly in the plan library reduces the accuracy of the plan recognizer, since whenever one of the interleaved plans is seen in the user's behavior, the other one could be suggested as the next user's step, and because this could delay the recognition. Thus, the detection and removal of interleaved plans during the plan library learning process is a desirable feature.

To exemplify the interleaved plans problem during the learning process, consider again the abstract plan library shown in figure 1. Suppose that the action sequence

$$AS_1 = \{LoadInk, PrepareImage, PrintSubstrate, LoadPrintedSubstrate, Laminate, Cut, Pack\}$$

is observed. Without additional information, it is difficult to infer if the whole observation corresponds to a single plan, or if it corresponds to two interleaved plans (*Print* and *LaminateCutPack*), so the system creates $InterleavedPlans'$ to represent AS_1. Now suppose that the following sequences are processed:

$$AS_2 = \{LoadInk, PrepareImage, PrintSubstrate\}$$
$$AS_3 = \{LoadPrintedSubstrate, Laminate, Cut, Pack\}$$

After that, two new events are created (the first one has $LoadInk, PrepareImage$ and $PrintSubstrate$ as its parts, and the second has $LoadPrintedSubstrate$, $Laminate$, Cut, $Pack$). In addition, $InterleavedPlans'$ is updated. Figure 4.a and 4.b shows the plan library learned after AS_1, AS_2 and AS_3 are processed. As can be seen, the new observations did not solve the problem. If during the recognition process, the plan recognizer concludes that the user is trying to $Print'$, then it can consider the possibility that the user is trying to execute $InterleavedPlans'$, which in general is not true.

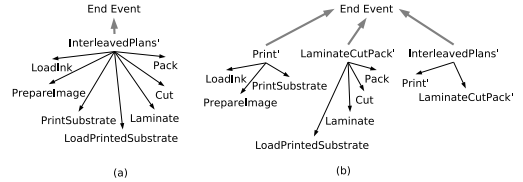

Figure 4. Interleaved plans problem

The problem of interleaved plans can be tackled in different ways, depending on which biases are imposed to the plan library representation. Biases imposed to plan libraries represent assumptions about what these plan libraries will contain. Thus, if some ambiguous case is processed, but one of the possible interpretations of the case contradicts the plan bias, then the ambiguity can be solved by discarding the contradictory interpretation (for more details about biases in machine learning, see [17]).

In [10], a bias is imposed to the plan library such that an event can be a part of another one, or can be a "top-level event" (i.e, an event that is not part of another one, and that is executed for its own sake), but not both at the same time. If an observed event E is executed for its own sake (i.e, it is not executed for accomplishing a more general one), and then E is present in other observation along with other events, it follows that this last observation is an interleaving of plans, since the bias imposed to the plan library does not allow E to be an end event, and be part of another one at the same time.

Thus, if this bias is imposed to the representation of plan libraries, when an event that is part of a non-primitive one appears alone in a new observation, as an *end event*, it follows immediately that the previously inferred composed event was, in fact, an interleaving of more than one plan. The bias solved the ambiguity.

In the above example, when AS_2 is processed the composed event $Print'$ is created, and because it appears as executed for its own sake, it follows that $InterleavedPlans'$ contains, at least, two interleaved plans ($Print'$ and the rest of $InterleavedPlans'$), so a reasonable modification to the plan library would be to remove $Print'$ from $InterleavedPlans'$.

The bias mentioned above simplifies the solution of the problem of interleaved plans. However it is sometimes too strong, because in some cases it could be useful to allow an event to be a part of another one, and a top-level event at the same time (see [7]). This situation arises when an event is executed sometimes for its own sake, and sometimes to accomplish a more general task. If this is the case, another mechanism is required in order to avoid the ambiguity of interleaved plans.

In this last scenario, we propose the use of a probabilistic rule to infer if an event really exists as a concept, or if it corresponds to interlaved plans. For our proposal, we suppose that interleaved plans appear sporadically and not systematically. If they appear systematically, they are not considered as interleaved plans (although in the domain conceptually they may be). Thus, we want to calculate the probability that some action A' is in fact the interleaving of several plans, instead of a real non-primitive event. Let $N(A'_T)$ be the number of observations in which A' or any of its parts appears, and $N(A'_{int})$ and $N(A'_{\neg int})$ the number of action sequences where A' is and is not an interleaving of plans, respectively. Then, it holds that

$$N(A'_T) = N(A'_{int}) + N(A'_{\neg int}) \quad (1)$$

The total number of observations that involve A', is the sum of those that involves A' as a real concept, and those that involve it as an interleaving of plans. If the value of $N(A'_{int})$ is known, then the probability that A' is in fact an interleaving of plans can be calculated as

$$P(A'_{int}) = \frac{N(A'_{int})}{N(A'_T)} \quad (2)$$

The problem with this approach is that the system does not know the real plan library (since it is learning it from what it perceives), and we assume an unsupervised learning mechanism, so it can not, in principle, calculate the number of observations in which A' is an interleaving of plans. If we assume that A' is an interleaving of plans whenever one or more of its parts appears as end events, then $N(A'_{int})$ can be approximated as

$$N(A'_{int}) = N(A'_{1_e}) + N(A'_{2_e}) + \cdots + N(A'_{n_e}) \quad (3)$$

where the A'_is are the actions that have to be done in order to accomplish A', and $N(A'_{i_e})$ is the number of observations in which the action A'_i appears as end event. It is important to note that A'_is are the direct parts of A', i.e, they do not include the decomposition of the parts of A' (the parts of A' could be themselves non-primitive events with their own sub-parts, but when we mention the parts of A', we refer to its direct parts). Additionally, if we assume that A' is a real event when it appears explicitly (when all its parts appear together), then equation (1) can be written as

$$N(A'_T) = \underbrace{N(A')}_{N(A'_{\neg int})} + \underbrace{N(A'_{1_e}) + \cdots + N(A'_{n_e})}_{N(A'_{int})} \quad (4)$$

Replacing (3) and (4) in (2), the probability that A' is an interleaving of plans can be approximated as

$$\begin{aligned} P(A'_{int}) &= \frac{N(A'_{int})}{N(A'_T)} = \\ &= \frac{N(A'_{1_e}) + N(A'_{2_e}) + \cdots + N(A'_{n_e})}{N(A'_T)} = \\ &= P(A'_{1_e}) + P(A'_{2_e}) + \cdots + P(A'_{n_e}) \quad (5) \end{aligned}$$

The value of $P(A'_{int})$ can be used in several ways:

1. Comparing directly $P(A'_{int})$ and $P(A'_{\neg int})$ to know how to interpret A' (as a "real" non-primitive event or as an interleaving of plans)
2. Using a threshold for $P(A'_{int})$, beyond which A' can be considered as an interleaving of plans
3. Using a threshold for the rate $\frac{P(A'_{int})}{P(A'_{\neg int})}$

Note that equation (5), gives a meassure of the probability that A' is in fact an interleaving of plans, that changes *only* when an action sequence that contains A' or any of its parts is processed.

In our experiments, when an interleaving of plans is detected, we remove it from the plan library. In the example shown in figure 4, the removal of $InterleavedPlans'$ will result in a partial plan library consistent with the real one (shown in figure 1).

5 Implementation

We have developed an implementation of the algorithms described in section 4, and some preliminary tests were carried out. The algorithms were implemented in Java, and some of the knowledge management required was built in SWI-Prolog [1]. The interaction between components built in Java and Prolog was implemented using the JPL package for SWI-Prolog, which is a bidirectional Prolog/Java interface.

[1] http://www.swi-prolog.org/

The experiments we conducted were based on small datasets of simulated data. We provided the unlabeled examples one by one to the algorithm, and we evaluated the resulting changes in the plan library. The results obtained are similar to those expected, but we need bigger datasets of real data in order to get more significant and reliable results. These topics are further commented in section 7.

6 Related Work

Some work has been done for learning plan libraries, with different results. One of the first approaches for acquiring plan libraries automatically, was to generate all possible goals and plans for a given domain, using some bias to allow only valid goals and plans ([13], [12]). This approach has some drawbacks. First, it requires the specification of predicates for building goals (predicates are used as "building blocks" for constructing goals). This is restrictive since the designer must in advance, determine the possible sub-goals (predicates) the user could need to satisfy in the specific domain.

A second drawback is that it needs biases, in order to generate valid goals and plans. The determination of biases is not an easy task, and must be performed by an expert. If the biases are too restrictive, some valid plans could be left out of the plan library, making impossible the completeness of the recognizer. If the biases are to weak, invalid plans and goals could be included in the library, thus reducing the soundness of the recognizer. In our approach, we replace the use of biases by the use of example cases of the plans actually executed by the observed agent.

Other type of approaches are based on the acquisition of plan libraries from example cases. In [2], a method for generating abstract plans from *labeled examples* is presented. This method generates abstractions of a set of action sequences that achieves some goal. In [3], a clustering approach is presented, that can be used to group action sequences that corresponds to the same goal. This eliminates the need of goal annotation for examples. These works use some knowledge about an abstraction hierarchy of actions, and generates non-hierarchical plan libraries.

In [5] and [6], a technique for generating hierarchical task models is presented. These task models are generated using a set of labeled examples. In particular, decompositions are not inferred, but given by the trainer.

Another approach is to avoid the use of a plan library for plan recognition. In [18] and [19], a method is presented for plan recognition, that does not use an explicit plan library. Instead, it uses a kind of planning algorithm in order to explain the actions and effects observed. This approach requires the specification of the valid goals that the observed agent could be pursuing.

Some concepts mentioned in this work were pointed out in [4]. That early work, shares some of the objectives of this work, and a conceptual approach to address the automatic generation of plan libraries, for plan recognition in intelligent interface agents was proposed. An alternative plan library representation (also named *task model*) is assumed, that is more suitable for interface agents domain, and some preliminary ideas for adapting the task model with new observations were proposed.

In [16], some ideas for acquiring hierarchical plan libraries from unlabeled examples were presented. Alternative decompositions for non-primitive events, and some considerations about interleaved plans were exposed, and a preliminary version of HIDEL algorithm was proposed.

Some of the works mentioned above automate some of the tasks that are needed in order to build plan libraries, and others propose methods for automatically generating plan libraries that are limited in expressiveness (notable exceptions are [16], and the early work presented in [4]). However, as far as we know, the automatic acquisition of hierarchical plan libraries, from unlabeled example cases, has not been achieved yet, and very little work has been done in supporting interleaved plans in action sequences used for learning.

7 Conclusions and future work

An algorithm for learning hierarchical plan libraries from unlabeled example cases was presented in this paper. A mechanism for working with interleaved plans was also provided. The presented approach gives a first approximation towards the automatic generation of expressive plan libraries from example cases, in knowledge rich domains.

We have implemented the algorithms reported in this paper, and we have conducted some small simulated experiments in order to test them. Our experiments were carried out using small datasets of simulated data, and the first results we obtained are very promising. We consider however, that further experiments are needed in order to get more significant results.

We conclude that, even when the ideas presented in this paper are promising, future work must include validation of the algorithms with real data from a knowledge-rich domain (such as CAPP), and some additional features must also be taken into account: abstractions of actions ("is a" relations), the inferring of actions orderings, actions with parameters, the inferring of restrictions on these parameters, and support of spurious and repeated actions.

References

[1] J. Allen. Recognizing intentions from natural language utterances. *Computational Models of Discourse, The MIT Press*, 1983.

[2] M. Bauer. Towards the automatic acquisition of plan libraries. *13th European Conference on Artificial Intelligence*, 1998.

[3] M. Bauer. From interaction data to plan libraries: A clustering approach. *16th International Joint Conference on Artificial Intelligence*, 1999.

[4] V. Eyharabide and A. Amandi. Automatic task model generation for interface agent developing. *6th Argentine Symposium of Artificial Intelligence*, 2004.

[5] A. Garland, N. Lesh, C. Rich, and C.L. Sidner. Learning task models for collagen. *AAAI Fall Symposium*, 2000.

[6] A. Garland, N. Lesh, and C.L. Sidner. Learning task models for collaborative discourse. *Workshop on Adaptation in Dialogue Systems*, 2001.

[7] R.P. Goldman, C.W. Geib, and C.A. Miller. New model of plan recognition. *15th Conference on Uncertainty in Artificial Intelli-*

[8] F. Ibañez, D. Diaz, and R.Q. Forradellas. Scheduling for flexible package production. *IEPM*, 1:385–400, 2001.

[9] S. Kalpakjian and S.R. Schmid. Manufacturing engineering and technology. *Addison-Wesley. 4th ed.*, 2002.

[10] H. Kautz. A formal theory of plan recognition. *Ph.D. Thesis. University of Rochester*, 1987.

[11] H. Kautz. A formal theory of plan recognition and its implementation. *Reasoning about plans, chapter 2, Morgan Kaufmann*, 1991.

[12] N. Lesh. Scalable and adaptive goal recognition. *Ph.D. thesis, University of Washington*, 1998.

[13] N. Lesh and O. Etzioni. Scaling up goal recognition. *5th International Conference on Principles of Knowledge Representation and Reasoning*, 1996.

[14] N. Lesh, C. Rich, and Sidner C.L. Using plan recognition in human-computer collaboration. *7th International Conference on User Modeling*, 1999.

[15] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, 1994.

[16] M. Marchetta and R. Forradellas. A new model for automatic generation of plan libraries for plan recognition. *3rd International Conference on Production Research - Americas' Region (ICPR-AMERICAS' 2006)*, 2006.

[17] T. Mitchell. Machine learning. *McGraw Hill*, 1997.

[18] M. Yin, W. Gu, and Y. Lu. Incorporating goal recognition into human-machine collaboration. *3rd International Conference on Machine Learning and Cybernetics*, 2004.

[19] M. Yin, H. Ou, W. Gu, Y. Lu, and R. Liu. Fast probabilistic plan recognition without plan library. *3rd International Conference on Machine Learning and Cybernetics*, 2004.