From Interaction Data to Plan Libraries: A Clustering Approach*

Mathias Bauer
German Research Center lor Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
66123 Saarbrucken, GERMANY
bauer@dfki.de

Abstract

Plan libraries are the most important knowledge source of many plan recognition systems. The *plan decompositions* they contain provide information about how a plan has to be executed to actually achieve its associated goals and be recognized by the system. This paper presents an approach to the automatic acquisition of plan decompositions from sample action sequences. In particular a clustering algorithm is introduced that allows groups of "similar" sequences to be discovered and used for the generation of plan libraries. Empirical tests indicate that these libraries can indeed be successfully used for plan recognition purposes.

1 Introduction

Plan libraries are the most important knowledge source of many plan recognition systems. They not only contain all possible types of plans (or goals) to be recognized by such a system—thus delimiting the search space of possible plan hypotheses—, but also represent the details of how these plans have to be executed to actually achieve their associated goals. These "recipes"—the so-called *plan decompositions*—are necessary to map observed actions onto the plan hypotheses available which is one of the central steps within the plan recognition process.

Although they are being widely used, the question of how to actually construct plan libraries that support the recognition process in an optimal way has only recently been investigated more thoroughly (e.g. [Lesh and Etzioni, 1996]). In [Bauer, 1998] a method was introduced that allows abstract plan decompositions to be generated from action sequences logged while observing test subjects interacting with an application system. What makes this approach attractive is the fact that it does not rely on the existence of formalized domain knowledge, although it can use possibly available information to improve its results.

* Supported by the German Ministry of Education, Science, Research, and Technology under grant ITW 9703.

This *join* procedure can be considered an instance of a supervised learning algorithm, that is, it requires its training instances to be labeled with their respective class memberships (the domain goals associated to the various action sequences in this case). Often, however, only unlabeled interaction data are available that are very tedious to classify by hand as they contain a huge number of irrelevant details like spurious actions. This paper extends the approach from [Bauer, 1998] for the acquisition of plan libraries by a clustering algorithm that determines sets of *similar* action sequences in unlabeled training data that can be forwarded to the *join* algorithm to compute plans to be used in the library. Analyzing these library entries and labeling them with the domain goals they seem to achieve—such that they can actually be used for plan recognition purposes—is then relatively easy as the small number of plans generated represents the *essential* aspects of the original action sequences only. Furthermore the resulting libraries can be expected to support the plan recognition process particularly well as they contain abstractions of actual users' behaviors rather than idealized plans designed by a knowledge engineer.

The rest of this paper is organized as follows. Section 2 reviews the basic notions introduced in [Bauer, 1998]. Sections 3 and 4 introduce the new clustering algorithm and present empirical findings, respectively. Finally, Section 5 discusses related work before Section 6 summarizes the results.

2 Joining Action Sequences

A *plan decomposition*—or simply, a *plan*—is a tuple (P, *Ap,Cp)* where P is a unique identifier, *Ap* is a set of actions—either concrete or abstract—, and *Cp* is a set of constraints representing additional details about the internal structure like temporal ordering of the elements of *Ap.* To optimally support plan recognition, a plan decomposition should not be too restrictive by containing irrelevant constraints that unnecessarily restrict the number of recognizable user interactions. On the other hand, being too "fuzzy" by leaving out too many details implies the danger of not being able to discriminate between competing hypotheses. A good compromise is to concentrate on those aspects of a plan—i.e. actions

and constraints describing its internal structure—that axe absolutely *necessary* to achieve the associated domain goal. Obviously a component of a plan is necessary just in case it occurs in *all* action sequences that form a valid instance of this plan. As it is hardly ever possible to enumerate all possible ways to actually carry out a plan, this notion of necessity has to be approximated by collecting a limited number of action sequences and determining what they have in common. These *training data* are obtained by fixing a set of domain goals to be covered by the plan library and observing a number of test subjects trying to achieve these goals. The action sequences executed are recorded, grouped according to common domain goals, and forwarded to the "join" algorithm as described in [Bauer, 1998]. That is, in contrast to the situation considered in Section 3, there exists a unique labeling of the input data with their associated domain goals.

Let $AS_1, ..., AS_n$ be the set of action sequences belonging to some goal g. Each of these sequences consists of a set of temporally ordered actions $\hat{a}_1, ..., \hat{a}_{l_j}$ where $\hat{a}_i = a_i(o_{i1}, ..., o_{in_i})$. Here a_i is the *action type* this instance belongs to (like ls or cp in a UNIX context), and the various o_{ij} are constants representing the domain objects being manipulated (the action parameters).

Besides these action sequences the *join* procedure can make use of various types of domain knowledge without depending on their actual availability. As will become clear, a logical theory *D* containing general domain knowledge can be used to infer structural relationships among the elements of an action sequence. Furthermore, an *action type hierarchy* D_a and an *object type hierarchy* D_t representing abstraction relations among action and object types, respectively, will play a central role in the abstraction step of the *join* procedure. Given a set *X* of concepts from either D_a or D_t, the operation *msa* computes the *most specific non-trivial abstraction* of all members of *X* within the corresponding abstraction hierarchy. Note that the result is undefined if the root concept is the only one subsuming all of *X* (in this case the elements of *X* can be considered to have nothing in common). For a precise definition of *msa* the reader is referred to [Bauer, 1998].

In a first step, these action sequences are transformed into labeled graphs G_{AS_i} making the interrelationships among the constituents of AS_i explicit. While the action instances contained in AS_i form the nodes, there are two types of edges. An edge $\hat{a}_i \xrightarrow{<} \hat{a}_j$ represents the temporal order between both actions (in this case $\hat{a}_i$ occurs before $\hat{a}_j$), while an edge

$$a_i(o_{i1}, ..., o_{in_i}) \xrightarrow{\langle p,k,l \rangle} a_j(o_{j1}, ..., o_{jn_j})$$

represents the fact that relation *p* holds between the action arguments o_{ik} and o_{jl}. The resulting *action graph* then has the form $G_{AS_i} = \langle AS_i, T_i \cup S_i \rangle$ with a set of *temporal edges* $T_i = \{\hat{a}_i \xrightarrow{<} \hat{a}_j \mid i < j\}$ and a set of *structural edges* $S_i = \{\hat{a}_i \xrightarrow{\langle p,k,l \rangle} \hat{a}_j \mid D \models p(o_{ik}, o_{jl})\}$. If $D = \emptyset$, i.e. if virtually nothing is known about the domain, only the equality of two objects can be recognized and made explicit within G_{AS_i}.

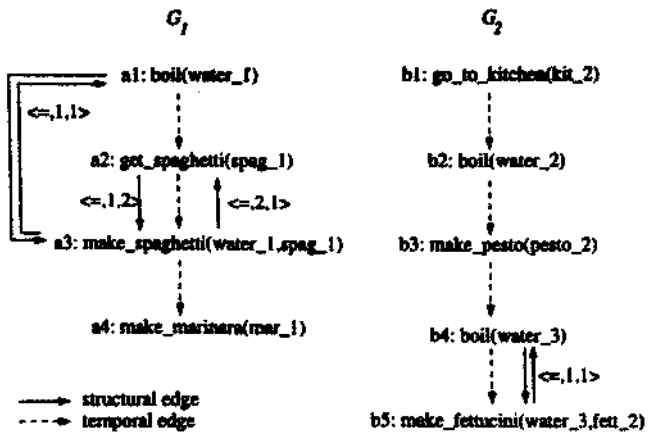

Figure 1: Two sample action graphs.

Given two action sequences in the form of action graphs $G_1 = \langle A_1, T_1 \cup S_1 \rangle$ and $G_2 = \langle A_2, T_2 \cup S_2 \rangle$, their common abstraction or "join" $G_1 \otimes G_2$ is computed as follows. Let $\hat{a}_i = a_i(o_{i1}, ..., o_{in}) \in A_i, i = 1, 2$. Then $\hat{a}_1 \otimes \hat{a}_2 = a(o_1, ..., o_n)$ where $a = msa(\{a_1, a_2\})$ and for all $1 \leq i \leq n$: $o_{1i} = o_{2i} = o_i$ is a constant o o_i s a variable of type $msa(\{type(o_{1i}), type(o_{2i})\})$.

That is, the abstract representation of the common information contained in two action instances $\hat{a}_1$ and $\hat{a}_2$ is a new action instance the type of which is the *msa* of both action types—if defined—and the arguments of which are either a concrete domain object represented by a constant *Oi* (if both actions had the same object as a parameter in the same place) or a newly introduced variable the type of which is determined from the types of the originally occurring domain action arguments. If $D_a = \emptyset$, $\hat{a}_1$ and $\hat{a}_2$ can only be joined if $a_1 = a_2$.

A temporal or structural edge is considered to be common to both G_1 and G_2 iff the corresponding action nodes to which the edge is incident could be joined using the above criterion. That is,

$$\hat{a}_{11} \xrightarrow{<} \hat{a}_{12} \quad \otimes \quad \hat{a}_{21} \xrightarrow{<} \hat{a}_{22} \quad = \quad \hat{a}_1 \xrightarrow{<} \hat{a}_2$$

if $\hat{a}_{11} \otimes \hat{a}_{21} = \hat{a}_1$ and $\hat{a}_{12} \otimes \hat{a}_{22} = \hat{a}_2$ and undefined otherwise. The join of two structural edges with identical labels is defined analogously.

Summarizing, the join of two action sequences represented as action graphs G_1 and G_2 is $G_1 \otimes G_2$ = (A, *TuS)* where

$$A = \{\hat{a}_{1i} \otimes \hat{a}_{2j} \mid \hat{a}_{1i} \in A_1, \hat{a}_{2j} \in A_2\}$$
$$T = \{e_{t1} \otimes e_{t2} \mid e_{t1} \in T_1, e_{t2} \in T_2\}$$
$$S = \{e_{s1} \otimes e_{s2} \mid e_{s1} \in S_1, e_{s2} \in S_2\}$$

In many cases the above result $G_1 \otimes G_2$ does not yet represent a plan decomposition with the desired properties as some actions may be used several times to create a new action node. A maximum subgraph of $G_1 \otimes G_2$ in which each action of G_1 and G_2 is joined at most once with another action is called a *valid join.* The set of all valid joins can be ordered using the measure

$$deg_r(\langle A, T \cup S \rangle) = w_a \cdot |A| + w_p \cdot |PA| + w_t \cdot |T| + w_s \cdot |S|,$$

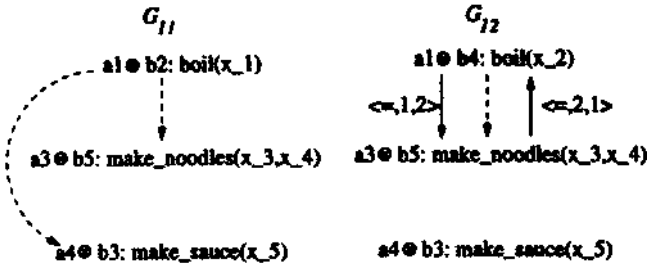

Figure 2: The valid joins of G_1 and G_2.

called the *degree of restrictiveness.* $PA \subseteq A$ is the set of *primitive actions* contained in A, i.e. those that do not abstract another domain action in the abstraction hierarchy D_a. If $D_a = \emptyset$, *PA* and *A* are obviously identical. The *non-negative* numerical weights w_a, w_p, w_t, and w_s assess the relative influence of the various components of a plan decomposition. As the *most restrictive* valid join, i.e. the one containing the maximum amount of details regarding plan execution (and thus maximizing deg_r) is the one to be used in the experiments described in Section 4, it will be identified with $G_1 \otimes G_2$ throughout the rest of this papa:. Obviously, the resulting action graph can be easily retransformed into a plan decomposition tuple of the form (P, A_P, C_P).

Example: Consider the action graphs G_1 and G_2 from the cooking domain as depicted in Figure 1. For sake of simplicity, only structural and *direct* temporal edges connecting subsequent actions are shown. That is, temporal edges like the one between **a1** and **a3** are left out. Action arguments like "spagJT refer to particular domain objects. D_a contains information about the abstract actions make-pasta (subsuming make .spaghetti and make Jettucini) and make jsauce (subsuming make-pesto and mak_marinara). The action graphs G_{11} and G_{12} as depicted in Figure 2 represent the two valid joins contained in $G_1 \otimes G_2$. Both are partially ordered plans (i.e. the temporal order among the actions is only partially defined) made up of one concrete and two abstract actions each. They mainly differ in the inclusion of either al ⊗ b2 or al ⊗ b4 (including both of them simultaneously would violate the above-mentioned "valid join" condition). The number of temporal and structural edges is an immediate consequence of this choice (cf. the rule on how to join edges). Using the numerical weights $w_a = w_p = w_t = 1$ and w_s = 2—a combination that proved to yield reasonable results in experiments—, the degree of restrictiveness of G_{11} becomes 6 while the value for G_{12} is 9. As a consequence the latter would be the representative for $G_1 \otimes G_2$.

3 The Clustering Algorithm

The approach to the acquisition of abstract plan descriptions presented in the previous section relies on the training data being labeled with their respective classes, i.e. their associated domain goals. Often, however, all we can get is an unlabeled collection of interaction data. In such a situation it is necessary to first identify groups of "similar" action sequences that can be forwarded to the *join* algorithm. The members of such a group are expected to also achieve similar domain goals. The resulting abstract plans—that are reduced to their essential components—can then be classified by a domain expert, thus providing the basis for plan recognition. This section introduces a clustering method based on a *frequent set analysis* of the action types occurring in the training data.

Assume a set of action sequences S= $\{AS_1, ..., AS_n\}$ are given. The crucial idea is to identify sets of action types that co-occur frequently and join only those action sequences containing all of them. The approach taken here is similar to the *cluster mining* algorithm introduced in [Perkowitz and Etzioni, 1998] that is used to detect groups of Web sites that are frequently visited in the same session. The algorithm works as follows.

1. Let *A* = $\{a_1, ..., a_m\}$ be the set of all action types occurring in S. Construct a complete graph (i.e. each pair of nodes is connected by an edge) with nodes $a_1, ..., a_m$ and label each edge connecting two nodes a_i and a_j with the numerical value $min(P(a_i|a_j), P(a_j|a_i))$ which represents the minimum co-occurrence probability of ai and a, in the sequences currently in *S*.
2. Determine all maximum size cliques within this graph containing only edges the label of which is $\geq cf$.[1] The effect of the parameter $cf \in]0, 1]$, the "clustering factor", will be discussed below.
3. For each clique *C*—which represents a set of action types all of which co-occur frequently in the training data—determine the set *S(C)* of action sequences containing each of its members.
4. Compute the join of each *maximum size cluster S(C)*—it will be used as one library entry—, remove the members of *S(C)* from S, and repeat this procedure starting from step 1 until $S = \emptyset$.

The higher the value of *cf,* the smaller the size of the cliques of action types co-occurring with probability $\geq cf$ becomes. Consequently the size of clusters of action sequences containing these cliques grows (the smaller a set of action types, the higher the probability to find an action sequence containing it), and the number of clusters—and thus, plans in the library—decreases. Simultaneously, the plans become less complex, i.e. the deg_r values go down as each plan represents the aspects common to a larger set of action sequences. A small value for *cf* has the opposite effect of producing many clusters each of which consists of few action sequences only and yields a plan with high deg_r value.

Example: Assume the training set contains the following action sequences (for sake of simplicity, only the action types contained are listed and both action parameters and constraints are omitted); $AS_1 = \{a_1, a_2, a_3\}, AS_2 = \{a_1, a_3\}, AS_3 = \{a_1, a_4\}$. With the value *cf* = 1.0 the trivial clique {ai} is identified that

[1] A *clique* is a complete subgraph of a graph, i.e. every two nodes must be connected by an edge in a clique. While this problem is known to be NP-complete, the removal of ail edges with label < *cf* usually reduces the graph to a tractable size.

leads to the cluster $\{AS_1, AS_2, AS_3\}$. For *cf* = 0.6, the algorithm successively detects the cliques $\{a_1, a_4\}$ and $\{a_1, a_3\}$ with associated clusters $\{AS_3\}$ and $\{AS_1, AS_2\}$. The results of joining the members of these clusters are

$$AS_1 \otimes AS_2 \otimes AS_3 = \{a_1\}, \quad (\text{for } cf = 1.0) \text{ and}$$

$$AS_1 \otimes AS_2 = \{a_1, a_3\}, \quad AS_3 = \{a_1, a_4\} \quad (\text{for } cf = 0.6).$$

These results correspond to what could be expected after the above discussion.

4 Experiments

A total of 53 action sequences from the UNIX domain containing up to 24 steps and belonging to 6 different domain goals formed the basis of the experiments reported about in this section. Typical goals were the search for files with certain properties or status checks for printers. For unbiased results, no additional domain knowledge was used, i.e. $D = D_a = D_t = \emptyset$. The objective of the experiments was to determine the impact of varying clustering factors on the clusters of action sequences computed and the feasibility of plan libraries created from them for plan recognition purposes. To this end the following experiments were performed using a 10-fold cross validation strategy[2] with the value of the clustering factor ranging from 0.1 to 1.0 in steps of 0.1.

1. Compute a clustering of the training data as described in Section 3.
2. For each resulting cluster compute the join of all its elements as described in Section 2.
3. TVy to recognize the test action sequences using the plan library containing all plans created in step 2.

Remarks: 1. The plan computed in the second step is labeled to belong to the prevailing class among the action sequences contained in the cluster. In case of a tie—i.e. two or more classes occur with the same (highest) frequency—a corresponding number of copies of this plan is included in the plan library, each marked with one of these classes.[3] Labeling the plan library entries—usually the task of a domain expert—is an important prerequisite for the plan recognition process. The plan recognizer's task consists in identifying all plans within the library that are completely instantiated by the current action sequence—i.e. the sequence contains all of the plan's actions and satisfies all of its constraints—and assigning their respective classes (as *plan hypotheses)* to this sequence.

2. Note that a plan library does not necessarily contain just *one* plan for each class (goal).

3. To produce unbiased results in the third step, a purely consistency-based, incremental plan recognizer was used that applied no focusing heuristic to discard unwanted plan hypotheses. Instead, consistency of observed actions with the plan decompositions was the only criterion to decide whether or not to maintain a hypothesis.

As a comparison, alternative clusterings ware computed using Cobweb [Fisher, 1987]. Cobweb incrementally constructs a hierarchy of clusters containing "similar" instances. Finally each node of the tree represents a *concept,* i.e. an intensional description of the class of instances subsumed in terms of the attribute values occurring within this class. While the leaves of this hierarchy contain the various instances, the root represents the most general class containing all instances encountered during the learning process. As Cobweb expects its input to be described in terms of attribute values, each action sequence was encoded as a vector of 39 Boolean values, each indicating the (non-) occurrence of one particular command within the sequence. To enable a fair comparison with the clustering procedure described in Section 3, a particular class of concepts was selected from the Cobweb hierarchies the subsumed instances of which form a partition of the training set while containing a "reasonable" number of elements.

Remark: The leaf concepts would have been inappropriate as they only contain one instance each, while very abstract concepts (i.e. those being close to the root of the hierarchy) increase the danger of producing "bad" clusters containing instances from many different classes.

A class *C* (represented as a node in the tree) is said to be "appropriate" iff

- *C* is not a leaf,
- none of the siblings of *C* is a leaf, and
- none of the descendants of *C* (as a node in the tree) is appropriate.

The clustering used in the experiments then corresponds to the collection of sets of instances subsumed by the appropriate classes.

The analysis of the empirical findings has to take into account several different aspects the first of which concerns the clusters detected.

How "good" are the clusters? Obviously, an optimal cluster contains only elements belonging to the same class, i.e. action sequences achieving the same domain goal. The degree of "disorder" within a cluster *C* can be measured by the *entropy e(C)* defined as

$$e(C) = -\sum_c p_c \log_2 p_c \qquad (1)$$

where c ranges over the possible classes (goals) an instance (sequence) can belong to and p_c is the relative frequency of instances belonging to class *c* among all instances contained in C. This numerical value can be used to quantitatively assess how well a cluster is "sorted" (note that e(C) assumes its minimal value of zero when all instances contained in *C* belong to the same class). In order to allow the results of experiments with varying numbers of classes to be compared, we use a *relative* measure of entropy by dividing the values obtained by (1) by the entropy of the original set of instances (action sequences) such that a value close to 1.0 corresponds to

[2] That is, the set of action sequences was randomly partitioned into 10 subsets of about equal size. In each of the 10 repetitions of the experiments, 9 of these subsets were used as training data whereas the respective remaining subset formed the set of test data.

[3] Of course the class memberships of the training data were not used during the clustering process.

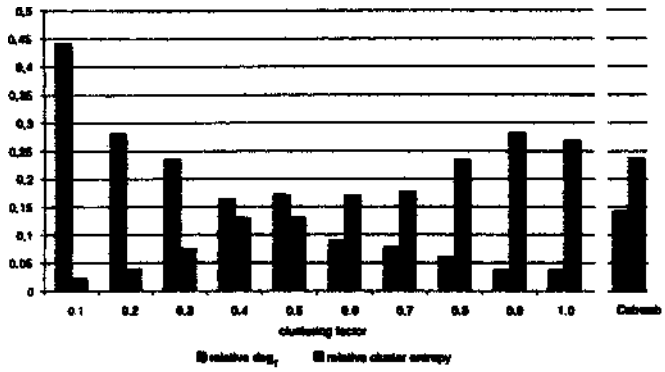

Figure 3: Relative deg_r and cluster entropy.

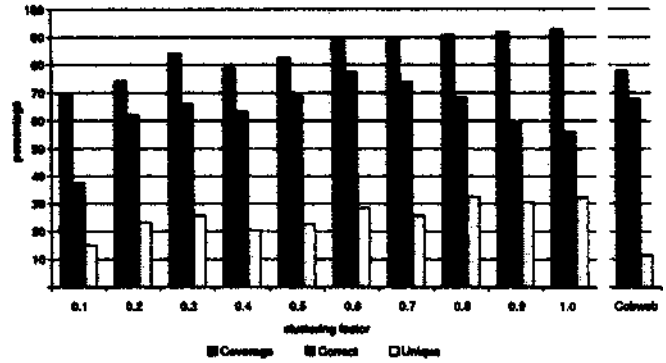

Figure 4: Results of plan recognition experiment - I.

a random selection of sequences from the training data. Figure 3 depicts the average values of this relative entropy measure along with the average values of the *relative degree of restrictiveness.* The latter corresponds to the deg_r values of the plan library entries created divided by the average deg_r value of the original action sequences contained in the training data.[4] As could be expected, an increasing value of cf yields clusters with high entropy values (the larger the cluster, the higher the probability to include sequences from more than one class) and the complexity of the plans produced—measured by deg_r—decreases. It can be seen that the *cf* values between 0.4 and 0.7 offered the best compromise between reducing the complexity of plan descriptions—without being over-restrictive or trivial—and maintaining a relatively low cluster entropy.

The results for Cobweb indicate that even the small clusters used in this comparison (most of them contained only 2 or 3 action sequences) have a relatively high entropy, i.e. these clusters are not "well sorted". As a consequence, joining the action sequences contained in these clusters produced an average of 4 "void" plans containing no actions or constraints in each repetition of the experiment, i.e. the sequences within such a cluster were highly incompatible with each other. [5] These plans were removed from the library before starting the plan recognition process. The relative deg_r and entropy values when ignoring void plans are 0.175 and 0.22, respectively. The high deg_r values result from the small number of elements per cluster.

The second—and in this context even more important—question is *how good are the plan libraries produced?* To answer this question, the abovementioned plan recognition experiment was conducted the results of which are depicted in Figures 4 and 5. Here *coverage* stands for the percentage of action sequences for which the plan recognizer produced a non-empty result, *correct (unique)* is the percentage of sequences recognized correctly (correctly *and* uniquely[6]). *Plan hypotheses assigned* measures the average number of plans "recognized" by the system (that is, assigned to a single action sequence in the test data), while *library entries* measures the number of plans within the library.

To summarize the results, a clustering factor of about 0.6 yielded the best results for this domain (other domains may require different values for *cf).* Here almost 90% of all action sequences where covered, more than 77% were correctly recognized, and for more than 28% this result was even unique. The results for the libraries generated using the Cobweb clustering (78%, 68%, and 11%) are similar to those for the clustering factor 0.4. Note, however, that the degree of ambiguity measured by the number of plan hypotheses assigned is significantly higher for the Cobweb-based libraries than for all other competitors. With *cf* = 0.6, for example, each action sequence is assigned at most two hypotheses while the average in the Cobweb case is almost 2.5 (out of 6 different classes). Additionally, for ef = 0.6 the plan recognizer worked about ten times as fast as for *cf* = 0.1 or *cf* = 0.2 which is a direct consequence of the large number of library entries of high complexity *(deg_r* values) in these cases, while the Cobweb-based libraries slowed down recognition even more. The smaller libraries obtained for very high *cf* values speed up this process even more, but the recognition results are relatively poor.

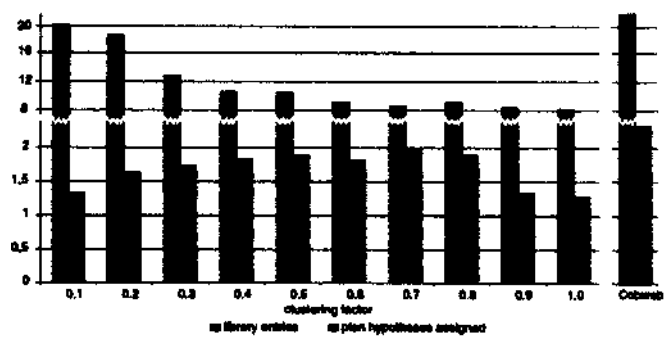

Figure 5: Results of plan recognition experiment - II.

5 Related Work

Abstracted versions of previous problem solutions have been studied both in the fields of AI planning (e.g.

[4] All values computed with $w_a = w_p = wt = 1$ and $ws = 2$.

[5] Note that the clustering method introduced in Section 3 never lead to void plans.

[6] Note that in many cases more than one plan hypothesis from the library proved compatible with the current action sequence—cf. Figure 5.

[Minion, 1985]) and programming by demonstration (e.g. [Cypher, 19933). In the former case they are reused for efficient planning from second principles, in the latter they enable a system to autonomously perform certain tasks on behalf of their users. As in both cases the aim is to generate an *executable* procedure, exact characterizations of actions (including preconditions and effects) as well as general domain laws are required. In contrast to this, the joining of action sequences as presented in this paper aims at producing *recognizable* structures and does not presuppose any domain knowledge.

In the plan recognition context, the explanation-based learning (EBL) procedure described in [Mooney, 1988] and the machine-learning inspired approach presented in [Lesh and Etzioni, 1996] have to be mentioned. In the first case the objective is to detect the general plan underlying a short narrative. As EBL is a very knowledge-intensive method, rich domain knowledge has to be available in order to produce generic structures that can be used to analyze similar, unknown stories. In the latter case the goal and plan libraries are only *implicitly* defined by a set of goal predicates and actions and corresponding *biases* that allow the search space to be enumerated. As a consequence, only certain classes of domain goals can be handled and the system requires complete planning knowledge.

The IPAM algorithm presented in [Davison and Hirsh, 1998] is a *knowledge-free* method to construct a probabilistic model of action patterns contained in action sequences that is used to predict future actions. This is in contrast to [Yoshida and Motoda, 1995] where information about 10 relationships among UNIX commands is exploited to produce a probabilistic predictive model. Unlike IPAM, this *graph induction* method needs relatively few training data to successfully make predictions. Both approaches do not explicitly mention plans, but try to detect regularities in user behaviors.

Clustering algorithms on the basis of *shared subsequences* (e.g. [Zaki *et al.,* 1998]) are not able to abstract from the observed temporal relations among the actions in the training sequences. As a consequence they are not feasible for the task of generating abstract plans for plan recognition libraries.

6 Conclusion and Future Work

This paper introduced a clustering method that allows groups of similar action sequences to be detected that can be used to generate plan library entries, even in the complete absence of domain knowledge. The empirical findings indicate that the libraries produced by this clustering algorithm in combination with the *join* algorithm can indeed be successfully used for plan recognition purposes. At least for certain values of the *clustering factor* the results were better than those produced by using Cobweb to compute a clustering of the training data. It is important to note that the library entries generated are not necessarily executable "recipes", but structures that can be used to recognize the underlying domain goal. The recognition process is particularly well supported as these planrreflect the actual user behavior represented by the interaction samples used in the training data (including suboptimal and faulty behavior) while hand-coded libraries often concentrate on optimal or at least correct plans. Another application of the clustering/joining procedure is the generation of alternative decompositions for one single class in situations where the training data are already labeled with their respective class memberships, but contain strongly diverging ways of achieving the same goal.

Future work will include the application of the methods presented to additional domains and the investigation of the impact of various degrees of domain knowledge that could be used to improve the quality of plans and support the domain expert labeling the library entries. Furthermore, comparisons with other clustering algorithms will be performed.

References

[AAAI, 1998] *Proc. of the 15th National Conference of the American Assoc, for Artificial Intelligence,* 1998.

[Bauer, 1998] M. Bauer. Acquisition of Abstract Plan Descriptions for Plan Recognition. In AAAI [1998], pages 936-941.

[Cypher, 1993] A. Cypher, editor. *Watch what I do: programming by demonstration.* MIT Press, 1993.

[Davison and Hirsh, 1998] B. Davison and H. Hirsh. Probabilistic Online Action Prediction. In *Proc. of the 1998 AAAI Spring Symposium on Intelligent Environments,* 1998.

[Fisher, 1987] D. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning,* 2:139-172, 1987.

[Lesh and Etzioni, 1996] N. Lesh and O. Etzioni. Scaling up goal recognition. In J. Doyle et al., editors, *5th Conference on Principles of Knowledge Representation and Reasoning,* pages 178-189, 1996.

[Minton, 1985] S. Minton. Selectively Generalizing Plans for Problem-Solving. In *Proc. of the 9th International Joint Conference on Artificial Intelligence,* pages 596-599,1985.

[Mooney, 1988] R.J. Mooney. Explanation-Based Learning of Plans for Plan Recognition. In D. McDermott et al., editors, *Proc. of the AAAI-88 Workshop on Plan Recognition,* 1988.

[Perkowitz and Etzioni, 1998] M. Perkowitz and O. Etzioni. Adaptive Web Sites: Automatically Synthesizing Web Pages. In AAAI [1998], pages 727-732.

[Yoshida and Motoda, 1995] K. Yoshida and H. Motoda. CLIP: concept learning from inference patterns. *Artificial Intelligence,* 75:63-92, 1995.

[Zaki *et al.,* 1998] M Zaki, N. Lesh, and M. Ogihara. PlanMine: Sequence Mining for Plan Failures. In R. Agrawal et al., editors, *Proc. of the 4th International Conference on Knowledge Discovery and Data Mining,* pages 369-374,1998. AAAI Press.