Automatic Generation of Plan Libraries for Plan Recognition Performance Evaluation

Giovani Farias, Lucas Hilgert, Felipe Meneguzzi, Renata Vieira, and Rafael H. Bordini
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Postgraduate Programme in Computer Science – School of Informatics (FACIN)
Porto Alegre, Brazil
giovani.farias@acad.pucrs.br, lucaswhilgert@gmail.com
{felipe.meneguzzi, renata.vieira, rafael.bordini}@pucrs.br

***Abstract*—Some plan recognition approaches represent knowledge about the agents under observation in the form of a plan library. Although such approaches use conceptually similar plan library representations, they seldom, if ever, use the exact same domain in order to directly compare their performance. For any non-trivial domain, such plan libraries have complex structures representing possible agent behavior, so plan recognition approaches often fail to be tested at their limits and only rarely are they compared with each other experimentally, leading to the need for a principled approach to evaluating them. In order to address this shortcoming, we develop a mechanism to automatically generate arbitrarily complex plan libraries; such plan library generation can be directed through a number of parameters to allow for systematic experimentation.**

I. Introduction

Plan recognition systems require a knowledge base that encodes the behavioral repertoire of an observed agent. The earliest plan libraries encoded recipes as collections of preconditions, subgoals, constraints, and effects [1]. Many different algorithms have been used to deal with plan recognition based, for example, on graph covering [2], Bayesian networks [3], and probabilistic state dependent grammars [4]. These methods typically use an individual plan library to represent the set of plans that are expected to be recognized. The sequence of observations are matched against this plan library to generate recognition hypotheses ranked according to some rating method. While an agent performs some task, the observation sequence acquired is matched against the plan library, and the obtained sequences are processed as plan recognition hypotheses. Applications can have complex multi-feature observations, which may present a high computational cost of matching these observations against all possible plan steps in the plan library.

Information about the complexity of the plan recognition algorithms is easily found [5]. However, there is no approach to automatically generate complex domain structures with a particular parameter set, supporting experiments to evaluate the performance of various plan recognition algorithms. Plan recognition libraries often have a complex structure, due to the large number of possible observation sequences that need to be encoded. Thus, it is important to have a mechanism for automatically generating these structures in order to evaluate the plan recognition algorithms under varying conditions. We present an approach for automatic generation of plan libraries as input for plan recognition algorithms. The generated plan libraries can be used as test suits in experiments for practical performance evaluation. The main contribution of this paper is the developing of a plan library generator, that allows the construction of such test suites, based on a set of parameters that will determine the plan library complexity. Thus, it is further possible to create and use the same information domain to directly compare the performance among various plan recognition approaches using different structures of plan libraries.

This paper is organized as follows. In Section II, we briefly survey about plan recognition. The definition of plan library and how plan recognition systems structure their knowledge base in Section III. In Section IV we describe the parameters used by the algorithm to create random plan libraries. The algorithm developed to generate plan libraries based on given parameters are described in Section V. Finally, we conclude the paper in Section VI.

II. Plan Recognition

Plan recognition can be defined as the task of recognizing the intentions of the user based on the available evidence, that is, user actions, explicit statements about intentions, and user preferences. Plan recognition research was initially defined in [6] using a rule-based approach, like other early work such as [7]. Kautz and Allen [2] developed one of the first logical formalization of plan recognition, providing the conceptual framework for much of the work in plan recognition up to the present. They defined the problem of plan recognition as finding a minimal set of top level actions adequate to justify the set of observed actions. Charniak and Goldman [8] argued that plan recognition is largely a problem of inference under conditions of uncertainty, and in addition to retrieving explanatory plans, a plan recognition system must also be able to select a hypothesis based on the likelihood of the explanation given the evidence.

Since plan recognition is the process of inferring an agent's plan, based on observations of its interaction with its environment, a plan recognition system must have a mechanism that is capable of inferring agent intentions by observing the agent's actions in the environment. Thus, this mechanism, from a given set of observations, retrieves one or more hypotheses about the agent's current plan of action. The knowledge used by this mechanism to infer plans is domain dependent, and therefore is commonly specified beforehand for each specific domain. This domain dependent information is usually encoded as two kinds

of inputs for the recognizer: a sequence of actions from the observed agent, and a set of plans and goals. In other words, the inputs to a plan recognizer are generally a set of goals the recognizer expects the agent to carry out in the domain, a set of plans describing the way in which the agent can reach each goal, and a sequence of actions observed by the recognizer. The plan recognition process itself consists in inferring the agent's goal, and determining how the observed actions contributes to reach it. The set of plans form a plan library and can include preconditions, effects, and subgoals. For a good overview of plan recognition in general, see Carberry [1], and for the most recent research in the field of plan, intent, and activity recognition, see Sukthankar et al. [9].

III. Plan Library

Most plan recognition systems require a knowledge base that encodes, into recipes, the ways in which agent goals can be achieved. A plan library is a knowledge base that codifies in some way the agent's beliefs concerning how the agent can reach each particular goal in the domain. Plan recognition systems have a plan library as an input, so several representations of agent plans have been used to approach this problem, and various methods applied to infer the agent's intention. These methods can be grouped in two main categories: symbolic and probabilistic approaches. Symbolic approaches aim at narrowing the set of candidate intentions by eliminating those plans that cannot be explained by the actions that the agent performs. The most used representation for symbolic approaches are plan hierarchies and consistency graphs. Probabilistic approaches explicitly represent the uncertainty associated with agent plans and allow a probabilistic ranking of the agent intentions mainly making use of Bayesian Networks [10] and Markov Models [11]. Most symbolic and probabilistic approaches are domain independent and can lead to accurate predictions provided the plan library is complete (for symbolic approaches) or provided the probabilities are correct (for probabilistic approaches). These approaches normally have the disadvantage of considering all the possible plans in the plan library given the observations. However, if observations so far can not distinguish between a set of possible intentions, probabilistic approaches can find the most probable one, while symbolic approaches can not select between them and have to wait for a single consistent explanation. Symbolic approaches are very sensitive to noisy actions, as the plan recognizer could wrongly exclude a plan (from the hypotheses explaining the observed behavior) if an unexpected action occurs in the middle of the execution of a plan.

Many plan recognition systems structure their plan libraries as an Hierarchical Task Network (HTN) [12], [13] to define the set of plans they are expected to recognize, in which goals are the root nodes and the observed actions are directly mapped to the leaf nodes. An *attachment point* in an HTN tree is a point in which an observation can be assigned to an action not observed yet, while *shared leaders* are action prefixes in the plan library that are common to different plans with different goals (root nodes). Typically, a plan library has a single dummy root node where its children are *top-level plans* and all other nodes are simply *plan steps*. In the library, sequential edges specify the expected temporal order of a plan execution sequence and vertical edges decompose plan steps into sub-steps. The library has no hierarchical cycles. However, plans may have a sequential self-cycle, allowing a plan step to be executed during multiple subsequent time stamps. Each agent action generates a set of conditions on observable features that are associated with a plan. When these conditions are met, the observations match a particular plan step. A complete algorithm for plan recognition must consider all coherent explanations for a given set of observations. Goldman et al. [14] assume that a plan library is made up of tasks structured in an hierarchical way, in which task nodes could represent goals, methods, and primitive actions. Similarly to Brown [15], the plan library could be viewed as a partially ordered AND/OR tree, in which the AND nodes are methods, connecting all action steps or sub-tasks needed to achieve the parent task, and the OR nodes are other isolated sub-tasks.

IV. Plan Library Generator – Parameters

The algorithm developed in this work was created in order to enable systematic analysis and performance comparison between several plan recognition algorithms given the variety of possible plan libraries. Thus, the *Plan Library Generator* (see algorithms in Section V) generates plan libraries based on various given parameters as follows:

Number of top-level plans (np): value that represents the branching factor of the root node, in other words, the number of children for the root node. This refers to the number of different independent top level plans in the plan library.

Depth (dt): corresponds to a measure of the depth of the plan trees. The depth of the plan library, from the root, which determines the number of *plan steps* that an instance of a plan (i.e., a complete path from a *top level* node to a *bottom level* node) contains.

Number minimum of branches (mi): represents the number minimum of branches that all nodes (other than root) must have. This value must belong to the interval $[1; ma]$.

Number maximum of branches (ma): represents the number maximum of branches that all nodes (other than root) can have. This value must be greater or equal to mi. The actual number of branches from a node is randomly chosen from the interval [mi, ma] whenever a new node is created.

Number of features (fs): defines the number of observable features, in the domain, available to be attached with a given plan step. Features are properties associated with the action of a given plan step, which need to be observed by the plan recognition algorithms for it to recognize the execution of that particular plan step.

Number of features per node (fn): defines the number of features associated with each top level plan and plan step. As a restriction, the value of the *number of features* (fs) has to be equal to or greater than the *depth* (dt) of the tree multiplied by the number of features attributed to each individual node (fn). In other words, there has to exist *at least* fn distinct features for each plan step node of an instance of a plan.

Conditions (mv): defines the number of values that can be associated with the features (recall that features, with particular value conditions, allow the identification of particular actions being executed by the observed agent). This value must be greater or equal to 0. Thus, assuming $mv = 2$ implies that all features will be multivalued assuming the values 0, 1 or 2 (i.e., only integer values in the interval [0,mv] can be assigned to the features).

Sequential edges (sq): value in the interval [0, 1], which determines the probability in which a branch can be created as sequential type. Thus, $sq = 0$ means that all branches will be decomposition type, as well as, $sq = 1$ means that all branches will be sequential type. Regarding feature distribution, it is important to emphasize that the same feature cannot be assigned with different values to a node and its respective decomposition children (as they represent a specialization of the parent). This way, when a *decomposition* node is created, it automatically inherits the features (as well as the values attributed to them) from its parent node.

Duplication (pd): represents the percentage of top level plans that are duplicated in order to generate ambiguous paths. The duplicated plan is not exactly equal to the top plan from which it was generated. The last leaf plan step in a duplicated plan is made different to establish some distinction between them. For example, $pd = 0.2$ means that 20% of top level plans are duplicates of others. So, approximately 40% of top level plans are not unique, presenting some difference only on the last leaf.

The size of the plan library is mainly determined by the number of top-level plans (np), the interval composed by the number minimum and maximum of branches $[mi, ma]$, and by the depth (dp). All plans in the plan library have an unique identification. Plans that present the same set of associated features and the same value of these features are considered equal, that is, they will match given the same set of observations. The ambiguity of the plan library influences the amount of distinct plans are fitting given a sequence of observations. The ambiguity is determined by the duplication parameter (pd) (larger implies more duplicated plans, and thus increased ambiguity), by the number of features (fs) (less features tend to decrease the possibility of distinction between plans), by the conditions value (more values associated to the features enable greater differentiation between plans that use the same set of features), and the number of features per node (fn) (greater number will cause more variety in plans).

V. Plan Library Generator – Algorithm

The generation of the *plan library* is conducted as described in Algorithm 1. The algorithm execution starts with

Algorithm 1 Generate Tree

Input: Number of top-level plans np, Depth dt, Number minimum of branches mi, Number maximum of branches ma, Sequential edges sq, Number of features fs, Number of features per node fn, Conditions mv, Duplication pd
Output: Node n

```
1: rt ← create root node
2: up ← getUniquePlans(np, pd)
3: tp ← createTopLevelPlans(up)
4: for all p in tp do
5:     createBranches(id, fs, pf, cd) ▷ obtained by p
6:     addChildNode(rt, p)
7: end for
8: duplicatePlans(pd)
9: return Node rt
```

the creation of *root* node of the *plan library* (Line 1), which is responsible for connecting all agent plans. This node is created as a *decomposition* node and no features are assigned to it. Next, the algorithm determines the number of distinct plans that will be created (Line 2). This number is based on number of top-level plans (np) and in the percentage of plans which will be duplicated (pd). For example, if np is set to 10 and pd to 10%, then 9 distinct plans will be created and the last one will be obtained through the copy of one of the previously created plans. This is important for evaluating plan recognition algorithms that typically keep track of sets of potential plans being executed until some disambiguation is possible, which is harder when many similar plans exist in the plan library. The next step consists in creation of the *top-level nodes* (Line 3) which corresponds to the agent plans (e.g., plans "*p1*" and "*p2*" in Figure 1). These nodes are created as simple *decomposition* nodes to which no *features* are assigned. After creation of *top-level plans*, the next step (Lines 4-7) consists in creation of their respective branches. This creation is conducted as described in Algorithm 2. Finally, after the creation of individual *top-level plans*, the algorithm selects (Line 8) the ones that will be duplicated (if a duplication percentage has been set). Note, however, that features values of bottom-level nodes from an plan copy are changed in order to distinguish it from the original plan, whenever plans are duplicated.

Algorithm 2 Create Branches

Input: Id id, Number of features fs, Parent Features pf, Current Depth cd
Output: Node n

```
1: nb ← getNumberOfBranches(mi, ma)
2: se ← getNumberOfSeqEdges(nb, sq)
3: rf ← getRandFeatures(fs, fn)
4: nd ← getNewNode(id, nb, se, cd, rf)
5: if pf not empty then
6:     getParentFeatures(nd, pf)
7: end if
8: if cd = dt then
9:     return Node nd
10: end if
11: ct ← 1
12: for i = 0 to sq do
13:     ti ← getNodeId(ct)
14:     sc ← createBranches(ti, fs, ∅, cd + 1, fn)
15:     addSequentialChildren(nd, sc)
16:     ct ← ct + 1
17: end for
18: for i = 0 to (nb − sq) do
19:     ti ← getNodeId(ct)
20:     sf ← getFeatureSubset(rf, fs)
21:     sc ← createBranches(ti, fs, sf, cd + 1, fn)
22:     addSequentialChildren(nd, sc)
23:     ct ← ct + 1
24: end for
25: return Node nd
```

Algorithm 2 describes the creation of plan-step nodes and their respective branches. It receives as input the textual identification of a *new node* (id), the *number of features* (fs), the subset of *features* assigned to its *parent node* (pf) and, finally, the *current depth* (cd), i.e., depth level in which the node is going to be created. Some of the parameters such

as *number of features per node* (fn), *number minimum of branches* mi, *number maximum of branches* ma and *sequential edges* (sq) are assumed as global (see Algorithm 1). The algorithm starts by determining the number of branches to be created for the new node (Line 1) and how many of them will be set as *sequential* branches (Line 2). Both information have to be defined before the creation of the new node as they help to determine its type. If the number of sequential edges (sq) is equal to the total number of branches (nb), the new node is going to be created as an *action* node (also referred as "*leaf node*"). Otherwise, the node is going to be created as a *decomposition* node. In next step (Line 3), the algorithm generates the subset of features to be assigned to the new node. This subset, of size as defined by fn, is extracted from the feature set (fs) received from the parent node; those features receive values randomly extracted from the previously defined interval ([0,mv]). Using the subset of features, and the information previously determined, the algorithm creates the new node (Line 4). However, before the branches of the new node are created, the algorithm checks if the current level has reached the expected depth (Line 8). If the expected depth has been reached, the new node is returned and the creation of the plan path is completed. If the depth has not been reached yet, the algorithm goes to the next step, which is the creation of the next level of the tree.

The next stage of algorithm execution is creation of *sequential* branches of node (Lines 12-17). It starts by defining the textual identifications of nodes to be created (controlled by ct). After that, the method is recursively called until all the sequential nodes are created. All nodes created in those recursive calls are added to the list of sequential children of the new node (Line 15). After the creation of the sequential branches, the algorithm uses a similar approach to create the *decomposition* branches of the new node (Lines 18-24). The main difference between the creation of sequential and decomposition branches is the distribution of features among nodes. While in creation of sequential nodes the construction of the feature subset (Line 3) is based on the whole feature set (fs), in creation of decomposition nodes the creation is based on a subset of fs, which eliminates features already used by parent nodes. Such elimination is conducted in line 20 of the algorithm. As previously explained, the same *feature* can not be attributed to a node and its (decomposition) children with different values, so the child node inherits the features (and its respective values) from parent node (Line 5). In Figure 1, the nodes of the tree represent plans (first level) and plan steps (second level and below) of the *plan library*, and the edges of the tree represent the relations between them. The "root" node is not considered as a plan, being used only as a way of connecting the various plans. Figure 1 shows sequential links represented by dashed arrows and decomposition links represented by solid arrows. For instance, there is a decomposition link between *p2* and *ps2_2*, and a sequential link between this *ps2_2* and *ps2_2_1*. The top-level plans are *p1* and *p2*. For clarity, Figure 1 does not show the set of conditions on observable features associated with plan steps.

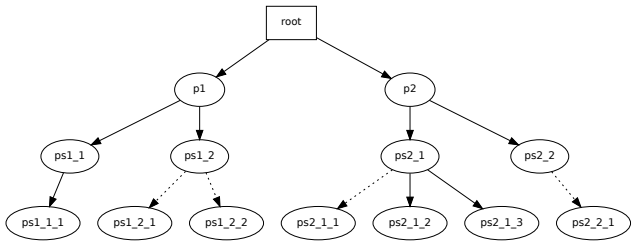

Fig. 1. Example of a plan library tree created by the *Plan Library Generator* with $np = 2$, $dt = 3$, $mi = 1$, $ma = 3$, $sq = 0.5$, $fs = 3$, $fn = 1$, $mv = 2$, and $pd = 0$.

VI. Conclusion

In this paper, we have developed an approach that allows principled performance evaluation for plan recognition algorithms. A plan library generator was created to generate complex structures based on a number of parameters that will determine the complexity of the plan library. Thus, a unique representation of an information domain can be used to compare the efficiency of several plan recognition algorithms. The performance of plan recognition algorithms is directly related to the structure and size of the plan library, as well as to the set of observations given to the plan recognition system.

References

[1] S. Carberry, "Techniques for plan recognition," *User Modeling and User-Adapted Interaction*, vol. 11, no. 1-2, pp. 31–48, Mar. 2001.

[2] H. A. Kautz and J. F. Allen, "Generalized plan recognition," in *AAAI*, T. Kehler, Ed. Morgan Kaufmann, 1986, pp. 32–37.

[3] H. Bui, "A general model for online probabilistic plan recognition," in *In Proc. of the International Joint Conference on Artificial Intelligence (IJCAI*, 2003, pp. 1309–1315.

[4] D. V. Pynadath and M. P. Wellman, "Probabilistic state-dependent grammars for plan recognition," in *In Proceedings of the Conference on Uncertainty in Artificial Intelligence, UAI2000*. Morgan Kaufmann Publishers, 2000, pp. 507–514.

[5] C. W. Geib, "Assessing the complexity of plan recognition," in *Proceedings of the 19th National Conference on Artifical Intelligence*, ser. AAAI'04. AAAI Press, 2004, pp. 507–512.

[6] C. F. Schmidt, N. S. Sridharan, and J. L. Goodson, "The plan recognition problem: An intersection of psychology and artificial intelligence." *Artif. Intell.*, vol. 11, no. 1-2, pp. 45–83, 1978.

[7] R. C. Schank and R. P. Abelson, *Scripts, plans, goals and understanding : an inquiry into human knowledge structures*, ser. The Artificial intelligence series. Hillsdale, N.J: L. Erlbaum, 1977.

[8] E. Charniak and R. P. Goldman, "A bayesian model of plan recognition," *Artif. Intell.*, vol. 64, no. 1, pp. 53–79, 1993.

[9] G. Sukthankar, R. P. Goldman, C. Geib, D. V. Pynadath, and H. H. Bui, Eds., *Plan, Activity, and Intent Recognition: Theory and Practice*. Elsevier, 2014.

[10] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[11] H. H. Bui and et al., "Hierarchical hidden markov models with general state hierarchy," in *Proceedings of the 19th national conference on artificial intelligence*, 2004, pp. 324–329.

[12] K. Erol, J. A. Hendler, and D. S. Nau, "Umcp: A sound and complete procedure for hierarchical task-network planning," in *AIPS*, K. J. Hammond, Ed. AAAI, 1994, pp. 249–254.

[13] K. Erol, J. Hendler, and D. S. Nau, "HTN planning: Complexity and expressivity," in *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 2)*, ser. AAAI'94. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1994, pp. 1123–1128.

[14] R. P. Goldman, C. W. Geib, and C. A. Miller, "A new model of plan recognition," in *UAI*, K. B. Laskey and H. Prade, Eds. Morgan Kaufmann, 1999, pp. 245–254.

[15] S. M. Brown, "A decision theoretic approach for interface agent development," Tech. Rep., 1998.