Comparing Plan Recognition Algorithms through Standard Libraries

Reuth Mirsky and Ran Galun and Ya'akov (Kobi) Gal
Ben-Gurion University of the Negev, Israel
{dekelr,ranga,kobig}@bgu.ac.il

Gal Kaminka
Bar-Ilan University, Israel
galk@cs.biu.ac.il

Abstract

Plan recognition is one of the fundamental problems of AI, applicable to many domains, from user interfaces to cyber security. We focus on a class of algorithms that use plan libraries as input to the recognition process. Despite the prevalence of these approaches, they lack a standard representation, and have not been compared to each other on common testbed. This paper directly addresses this gap by providing a standard plan library representation and evaluation criteria to consider. Our representation is comprehensive enough to describe a variety of known plan recognition problems, yet it can be easily applied to existing algorithms, which can be evaluated using our defined criteria.

We demonstrate this technique on two known algorithms, SBR and PHATT. We provide meaningful insights both about the differences and abilities of the algorithms. We show that SBR is superior to PHATT both in terms of computation time and space, but at the expense of functionality and compact representation. We also show that depth is the single feature of a plan library that increases the complexity of the recognition, regardless of the algorithm used.

Introduction

Plan recognition is a key AI problem that deals with reasoning about an agent's goals and plans according to a sequence of observed actions. Recent advancements have applied PR technologies to a variety of real world domains, including education [Amir and Gal, 2013; Uzan *et al.*, 2015], cyber security [Geib and Goldman, 2001; Bisson *et al.*, 2011; Mirsky *et al.*, 2017b] and more [Vered and Kaminka, 2017; Keren *et al.*, 2014].

A plan recognition algorithm allows an *observer* to reason about the goals and execution process of an agent, the *actor* given a plan library and a set of observed actions. It uses a partial sequence of observations and a plan library as input and outputs either a sequence of future steps or a plan [Bui, 2003; Blaylock and Allen, 2006; Wiseman and Shieber, 2014; Chakraborti *et al.*, 2017]. Although all of these problems have a lot in common, there is no single standard representation to allow comparison of these works.

The problem this paper addresses is the lack of standardized methods to compare between PR algorithms. We highlight the lack of standardization in the plan recognition community and propose a standardized representation, Plan Library Domain Description (PLDD), that will allow algorithms and domains to be evaluated on the same basic grounds. We also provide a list of criteria that should be considered for such an evaluation and the order they should be used. This work takes the first steps in this effort, by presenting the following contributions:

1. PLDD - A general, XML-based, plan library description that can be used as a standard for plan library representations.

2. A proposed list of criteria to evaluate different plan recognition algorithms and choose the most fitting one for a specific problem.

3. A use-case example that compares two leading algorithms based on the proposed comparison criteria.

For our comparison, we will use two plan recognition algorithms that represent two plan recognition families – SBR [Avrahami-Zilberbrand and Kaminka, 2005] and PHATT [Geib and Goldman, 2009]. PHATT and SBR were both designed to perform plan recognition using a plan library domain representation, but are fundamentally different. PHATT was inspired by parsing and natural language in order to allow comprehensive representation that can handle real-world qualities, while SBR was inspired by planning and its focus is on fast performance in order to be able to be integrated in a robot, for example.

This difference in focus translates into different abilities of the algorithms: SBR is faster and can give partial answers about the current state of the actor, but these answers might not be consistent with future observations. PHATT, on the other hand, is more comprehensive and flexible with the possible inputs it can process and the outputs it can produce, but its runtime is sometimes even exponentially slower than SBR.

This work provides a first thorough evaluation of these two algorithms, using a list of predefined evaluation criteria and a variety of domain settings. This evaluation provides interesting insights, both specific to the algorithms and general to plan recognition domain description.

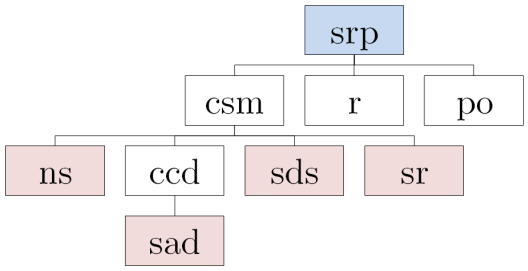

Figure 1: A plan describing the observation sequence ns, sad, sds, sr.

Related Work

There are several approaches to represent a domain in plan recognition. Some recent advent of work on plan recognition as planning takes as input a planning domain, usually described in STRIPS, a set of possible goals and selects one of the goals [Ramırez and Geffner, 2010; Sohrabi *et al.*, 2016; Pereira *et al.*, 2017; Freedman and Zilberstein, 2017; Shvo *et al.*, 2017; Vered and Kaminka, 2017; Masters and Sardina, 2017]. In this work, we focus on plan library-based plan recognition [Blaylock and Allen, 2006; Sukthankar and Sycara, 2008; Kabanza *et al.*, 2013; Geib, 2017]. Previous works used PHATT or SBR as baseline and showed aggregated improvements based on specific properties.

YAPPR [Geib *et al.*, 2008] can improve PHATTs runtime significantly if the user is not interested in complete plans, but rather in the goals of the actor and predictions about future actions. DOPLAR [Kabanza *et al.*, 2013] extended YAPPR using probabilistic reasoning to reach even better performance, at the cost of completeness. CRADLE [Mirsky *et al.*, 2017a] augmented PHATT with the ability to process PLs with parameters and proposed a set of pruning heuristics. Avrahami and Kaminka enhanced the basic SBR to handle interleaving of more than one plan and proposed an algorithm that reasons about the utility of the actor as part of the recognition process [Avrahami-Zilberbrand and Kaminka, 2007]. Sukthankar and Sycara [2008] based on SBR to deal with large domains and multiple agents.

Other notable works are ELEXIR [Geib, 2009], SLIM [Mirsky and Gal, 2016] and YR [Maraist, 2017] which present new approaches for plan library-based plan recognition algorithms.

All of the above works used a single existing algorithm as a baseline and showed improvements in specific properties, but they did not compare to more than one algorithm nor did they investigate the differences between the algorithms.

Previous papers surveyed methods for plan recognition, but did not try to run and evaluate the presented algorithms as well [Carberry, 2001; Sukthankar *et al.*, 2014]. This work is the first attempt to present empirical comparison of works in plan recognition.

Background

We start by mentioning briefly several basic concepts in the world of plan recognition, simplified for brevity. For a more detailed and formal description, we refer the reader to [Kabanza *et al.*, 2013]. In a plan recognition problem, there is an observer and actor, and the observer needs to infer the goals and plans of the actor. The observer is given a plan library describing the possible expected behaviors of the actor.

Definition 1 *(Plan Library) A plan library (PL) is a tuple $L = \langle B, C, G, R \rangle$, where B is a set of basic actions, C is a set of complex actions, $G \subseteq C$ is the goals the actor can achieve and R is a set of recipes that provide context of how complex actions can be decomposed into other actions, and in which order the constituent actions need to be performed.*

A *plan* for achieving a complex action $g \in G$ is a tree whose root is labeled with g. Each intermediate node is labeled with a complex action such that its children are a decomposition of the complex action according to a recipe. The order of the children must not collide with the ordering constraints enforced by the recipe. An *observation sequence* is an ordered set of basic actions that represents actions carried out by the actor. A plan p *describes* an observation sequence O iff every observation is mapped to a leaf in the tree. The actor is assumed to plan by choosing a subset of complex actions as intended goals and then carrying out a separate plan for completing each of these goals, using basic actions.

We will use a running example based on a real-world domain, an open-ended educational system called TinkerPlots. It is used world-wide to teach students in grades four through eight about statistics [Konold and Miller, 2004]. Using TinkerPlots, students build stochastic models and generate pseudo-random samples to analyze the underlying probability distributions. Our running example will use a TinkerPlots problem, called ROSA:

> There are 4 letters printed on cards, each card contains one letter: A,O,R,S. The cards are lined up in a row. After mixing the cards up, what is the probability that the cards would spell ROSA?

In order to solve this problem, a student must perform three subtasks: (1) create a sampler model (the complex action csm); (2) run the model (r); (3) plot the results (po). When accomplishing all subtasks successfully, the student is said to have solved the ROSA problem, which can be represented by the complex action - Solve ROSA problem (srp).

A teacher who wishes to understand if the student is solving the ROSA problem correctly, is trying to solve a plan recognition problem. We can model it such that the basic actions in the PL are the actions the student can perform in the software, such as add new sampler (ns), add device to sampler (sad), set number of draws in the sampler (sds) and number of repetitions (sr); srp, csm, r and po are some of the complex actions; and the recipes describe how complex actions are built from a sequence of other actions. For example, one of the recipes will state that srp is constructed from a sequence of csm, r and po. r must follow csm, as one must first create a sampler before it can be used to run tests, but po, creating a plot, can be executed at any time.

Figure 1 shows an example plan, describing the observation sequence ns, sad, sds, sr. The goal of the plan, srp,

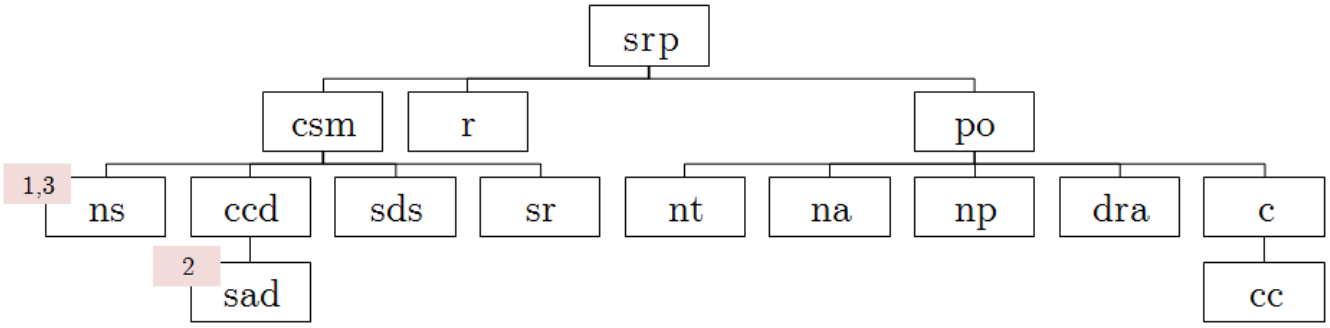

Figure 2: Example Plan from a Plan Library mapping by SBR.

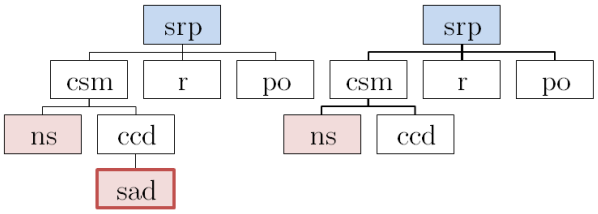

Figure 3: An example outputted explanation by PHATT.

is colored in blue and the student started executing the basic actions, colored in red, to achieve csm. Notice that the actions required for r and po are still missing. This means that this plan is incomplete. It is an important point, as the algorithms we wish to evaluate here are *online* algorithms, meaning that the can start process observation sequences online while the actor is still executing the plan.

Algorithms

We will test our standard representation and evaluation criteria on SBR and PHATT. Both algorithms can output the complete plans of the actor rather than just their goals. Both algorithms can output plans that describe incomplete observation sequences, allowing them to perform on-line inference. Moreover, both algorithms can handle partial ordering, multiple goals and bounded recursions. Recursion in a PL means that a recipe can have an action as a constituent of itself. Bounded recursion means that these recipes can only be used a predefine number of times.

While both these algorithms share many common traits, they represent different approaches for plan recognition with fundamental differences.

SBR is a descriptive algorithm, in the sense that it generates a single structure – a mapping of all the possible plans that can be generated from the PL. Given an observation, it traverses this structure and marks specific nodes upon it that can represent the observation, consistent with only the last observation. After a sequence of observations $O_1, \ldots, O_n$ were marked on the data structure, SBR can be requested to output the plans describing them. It traverses the structure once more and tries to collect consistent paths that can explain the observation sequence soundly.

PHATT, on the other hand, is a generative algorithm, in the sense that it builds possible explanations incrementally. Given a set of explanations E_{n-1} for an observation sequence $O_1, \ldots, O_{n-1}$ and a new observation O_n, PHATT generates branches with O_n as a leaf node, based on the recipes of the PL, and tries to attach these branches in all possible combinations to E_{n-1}.

To illustrate the different approach each algorithm is using to infer the actor's plans, consider the complete plan that can be created for the ROSA problem in Figure 2, and the observation sequence ns, sad, ns. This sequence means that a student created two samplers and then added a device to the first sampler. SBR will construct this complete plan once in the initialization stage, and every time it will process an observation, it will mark its timestamp on that single structure. Thus, the ns node will receive the timestamps 1 and 3, while the node sad will receive the timestamp 2.

PHATT, on the other hand, will construct a copy of a plan for each possible explanation to the observation sequence. Figure 3 shows one such explanation, where the first and third observations are described by the left plan, and the second observation is described by the right plan.

Plan Library Domain Description (PLDD)

In order to facilitate desired properties from several algorithms, we chose to represent the domain file in an XML format and create a language for PL descriptions, called Plan Library Domain Description (PLDD). The XML format was chosen as it can handle quite complex representations, in a fashion that is both readable to the human eye and easy to parse by a computer. It was designed to be robust, so it can be extended in the future to allow more properties, such as prior probability and effects of a basic action. Figure 4 contains a graphical representation of the DOM (Document Object Model) description of PLDD. It shows all the elements in a PL and their attributes.

Each PL contains a list of letters, divided to terminals which are the basic actions, and non-terminals which are the complex actions, as described earlier. Each letter has a name which describes what this letter represents, and a more compact id to use in recipes. Figure 5 shows an example for letter description. It shows the beginning of the Non-Terminals list and two actions: one is the srp action, and the other is

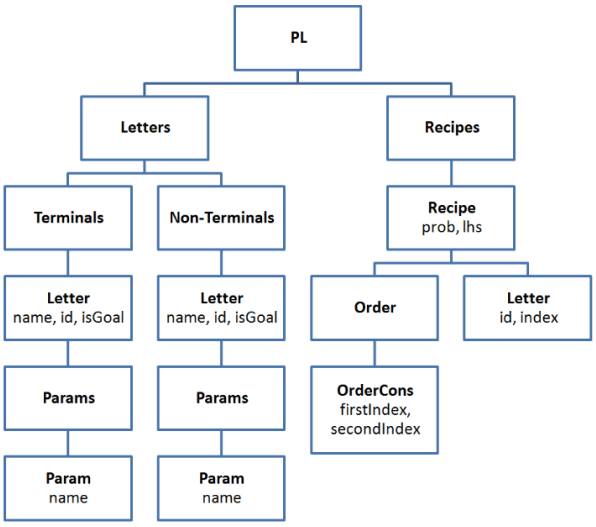

Figure 4: Structure of a plan library XML representation.

```
<Letters>
    <Non-Terminals>
        <Letter goal name="Solve Rosa Problem" id="srp"></Letter>
        <Letter name="Create Sampler Mechanism" id="csm">
            <Params>
                <Param name="s">
            </Params>
        </Letter>
```

Figure 5: An example to the XML representation for letters.

csm. Solve ROSA Problem, srp, is a possible goal, so it has the method "goal" in it. "Create Sampler Mechanism", csm, is an action with a parameter representing the sampler's id, which is labeled with s.

The recipes of the PL are represented as a list as well, where each recipe contains several properties: lhs is the id of the letter in the left hand side, and $prob$ is the probability of the recipe. The probability of a recipe, $prob$, is the probability of choosing this recipe to execute its complex action from all the possible actions. More formally, Given a recipe r^* and $R_{lhs^*} \subseteq R$, a set of all recipes with the same left hand side lhs^* as r^*, $prob^*$ is the probability of choosing r^* from R_{lhs^*}. Each $letter$ element in a recipe represents a letter from the recipe's right hand side. Since XML does not have inherent support of ordered elements, a letter's index is used to refer to that letter in the ordering constraints. $order$ is a list of ordering constrains, where the two indexes represent two actions that must come in a specific order. The action in the $secondIndex$ must follow an action in the $firstIndex$.

Figure 6 shows an example of a recipe, describing how the complex action labeled srp can be executed by performing 3 other actions - csm, r and po. The first OrderCons enforces that the action with the index 1, csm, must come before the action with the index 2, r. The action po can be executed at any execution order with csm and r.

```
<Recipes>
    <Recipe prob=0.3 lhs="srp">
        <Order>
            <OrderCons firstIndex="1" secondIndex="2">
        </Order>
        <Letter id="csm" index="1">
        <Letter id="r" index="2">
        <Letter id="po" index="3">
    </Recipe>
```

Figure 6: An example to the XML representation for recipes.

Methodology

Using the standard representation above, different algorithms can receive the same input and be evaluated on the same grounds. Before measuring any numerical value, it is crucial to make sure that the output of the algorithms is the same as well, and that they are trying to solve the same problem. After this is set, there are several types of criteria that can be used to evaluate the algorithms in terms of efficiency, robustness and more. The following list is a proposal for the order of the evaluations and tests to perform.

1. Problem definition. Each algorithm focuses on different problems with different features. For example, some algorithms only output the goals of the actor. Others output the goal and a prediction about future actions, but no plan decomposition. In order to compare the algorithms, we must first make sure that they try to solve the same problem. In this work, we chose two of the most comprehensive plan recognition algorithms, that can output the complete plans of the actor.

2. Abilities. While we require all compared algorithms to use the same problem definition and output similar results, we know that every algorithm is designed differently, to solve different challenges. This is why we also wish to allow qualitative evaluation of the various abilities of the algorithms. This part of the comparison is also meant to allow each algorithm to highlight its novelty.

3. Runtime. While runtime measurements are practical, it might not be enough for a thorough evaluation. However, as shown in our empirical work, it can provide with some insights when the run times are significantly different between compared algorithms. We divided this evaluation into two measures – evaluation of runtime and number of function calls. The runtime measure is divided to initialization runtime, that can be executed offline before the actor begins to act, and processing runtime, which is the actual recognition in real-time.

4. Space. Since this measure depends heavily on the implementation, we tried to provide with a more general metric, which is the number of nodes in the plans each algorithm creates. This measure is also good as a sanity check, as there is a lower bound to it, which is the number of nodes in all of the plans that should be outputted. Any algorithm can be evaluated in comparison to this ideal number.

Plan Libraries

Using PLDD as our standard plan library representation, we were able to run and compare implementations of PHATT and SBR. In order to evaluate the impact of several PL properties, we used a PL generator [Kabanza *et al.*, 2013] that can be configured to output PLs that vary in several features which are known to affect the explanation set size [Geib and Goldman, 2009]:

Number of Goals Representing the number of different goals an actor might pursue at the same time.

Depth Representing the depth of the plan library. In the generator, this value is set to be the depth of all plans, but in other PLs, it is set as the depth of the deepest plan that can be created in the library.

Alphabet Size Representing the number of basic actions in the PL.

Or Branching Factor This is the number of different ways a complex action can be decomposed into a sequence of constituent actions.

And Branching Factor This is the number of constituents which decompose a complex action.

The generator outputs a plan library which is constructed from partially ordered AND/OR trees. We then translated these trees to a set of recipes: an AND node is translated into a single recipe, with the same ordering constraints as the partial order of the AND node's children. An OR node labeled with an action c is translated into a list of recipes with c as their left hand side, such that each recipe represents one option that can be executed to achieve c.

As a baseline, we used the same configuration as [Kabanza *et al.*, 2013]. In the baseline PL, the ordering constraints are set to about about 1/3rd of the possible orderings, the number of possible goals is set to be 5, the depth is set to 2, the alphabet size to 100, the AND branching factor is set to 3, and the OR branching factor is set to 2.

Using PLDD, we were able to run and compare implementations of PHATT and SBR on a set of these synthetic PLs. We measured both the runtime of the algorithms, divided into initialization time and process time, the number of created nodes in each algorithm and the number of basic function calls.

Empirical Results

Problem Definition

As stated before, both algorithms are complete and should be evaluated when they output the same plans. However, some fundamental properties of the algorithms cause them to output different explanations given the same input.

Consider the TinkerPlots PL, which has the following properties: its And-branching factor is 5, Or-branching factor is 2, an alphabet size of 32 and a depth of 3. This PL is highly recursive, and the recipes can create an unbounded number of permutations of plans. While PHATT has the ability to create plans of unlimited depth, SBR cannot output plans that were not created during initialization. Due to this difference, the algorithms output a significantly different set of explanations: for a sequence of mere 4 observations, SBR outputted 2 explanations, while PHATT outputted 6000.

Another difference is that PHATT has the ability to handle interleaved plans. While both algorithms can reason about an actor that executes more than one plan, only PHATT allows the actions of the plans to interleave. For example, when cooking a dinner, the actor might make a pasta dish and a salad dish. While the first action will be to start boiling water for the pasta, the next action might be chopping vegetables for the salad. In order to allow this, PHATT is required to keep track of all open plans, which means it keeps a copy of every plan in every explanation rather than using a single representation for all possible plans.

Even when the evaluated PLs do not allow interleaved plans, it can contain observation sequences that can be described by such plans. In the following example, based on our TinkerPlots example domain, the given observation sequence is: ns, ns, sad. This means that a student created two samplers and then added a device to the first sampler. Figure 3 depicts one of the explanations that are outputted from PHATT for this observation sequence. The left plan is incomplete when the second ns is executed, and then we come back to it with the action sad (outlined in red). PHATT outputs this explanation (among others), while SBR does not output it. Other than that, all other explanations appear both in PHATT and SBR's outputs.

Notice that this difference is only apparent when plans interleave - if one plan is complete and only then another begins, or if we don't return to the first plan after starting the second one - there is no difference between the algorithms. However, an instance will possible interleaving only appeared twice in 800 runs, both of which were in the PL with increased depth. In these cases, SBR outputted 3 and 4 explanations. PHATT outputted 4 and 5 explanations, respectively. In each case, the additional explanation had two plans with interleaved actions.

Abilities

The major difference between these two algorithms is SBR's inherent capability to provide an answer to a query about the actor's current state (Current State Query), without the need to reason back through all of the previous observations. For example, in the TinkerPlots domain, we might want to understand whether the student is currently working on the ROSA problem, or performing an action that biases from any solution. In such a scenario, we would like to highlight to the student or the teacher that the student is doing something wrong, regardless of their previous actions.

The ability to output only the current state gives the algorithm real-time responsiveness, but at the expense of consistency with future actions: the set of returned states might not be consistent with respect to future observations, and in order to find the consistent paths, a new plan traversal process is needed. In PHATT, on the other hand, the ability to query about the current state is not inherent in the algorithm, and can only be calculated by computing complete explanations and then eliciting from then the current possible states.

There are additional properties that both algorithms keep, and every additional algorithm will have to keep as well if

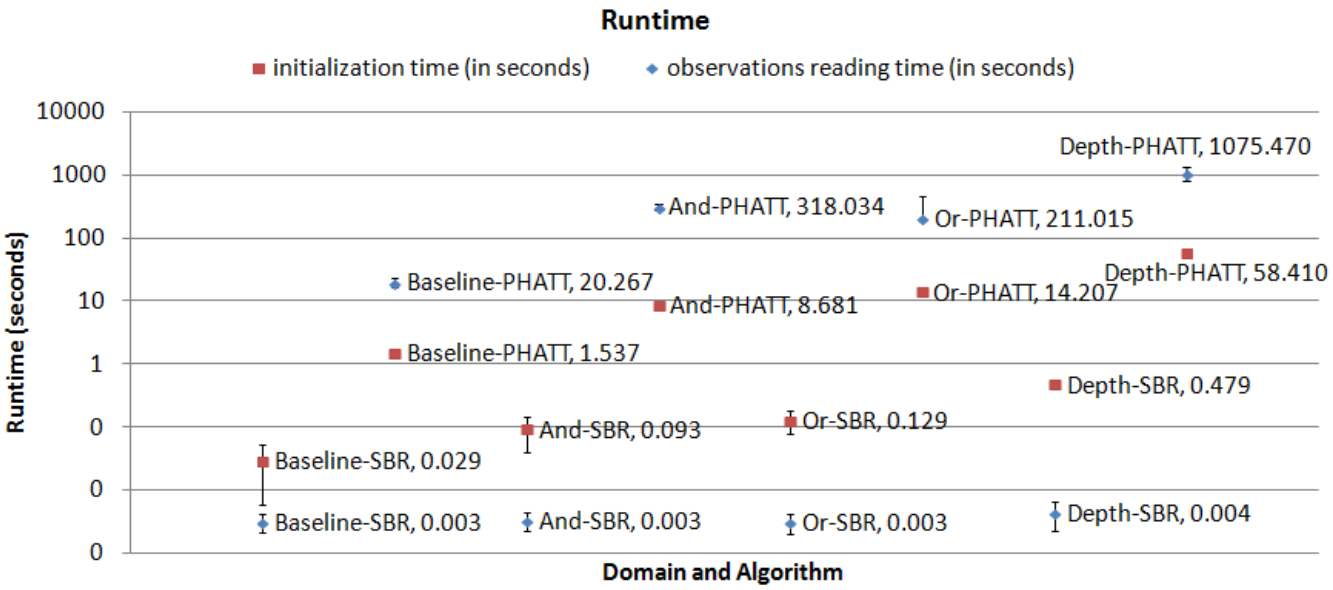

Figure 7: A comparison of runtime (in seconds).

it is to be compared to them: Both algorithms can handle partial ordering of the actions, which is a compact way to represent a sequence of actions that can be performed in several permutations. Both can output the complete plans of the actor rather than just their goals. Both algorithms can also reason about observation sequences that execute more than one plan, although as described above, only PHATT allows these plans to interleave.

Additional important property that was added to PHATT in successive work is the ability to reason about domains with parameters [Mirsky *et al.*, 2017a]. Such a PL description allows a more compact representation, focused on real-world domains. For example, in the TinkerPlots domain it allows us to label each action with an id of the sampler, thus we do not need to have a different action per sampler, but rather a single action that can be bounded to a specific sampler with an id parameter. While allowing a more compact PL representation, this ability has a large impact on runtime, since with every new observation, the algorithm needs to make sure that the parameters of the observations agree with the parameters of previous ones. Thus, there is a need to propagate the parameter values across the plans.

Runtime

In this subsection, we present the evaluated runtime of the algorithms on the various PLs. We stress than this comparison is only meant to provide with general notion of the algorithms' abilities – SBR was implemented using Java, and PHATT was implemented in Python. Both algorithms were tested on the same commodity i-7 computer. Figure 7 shows the runtime of 800 different instances both in PHATT and SBR. Each point represents an average of 100 instances using a single algorithm and single PL settings, with error bars representing the standard deviation of these runs. The points in the graph are also divided into the initialization time and the observation processing time. The y axis shows the runtime in log scale of seconds.

As seen in the figure, SBR is always faster than PHATT in an order or magnitude or more. It is interesting to see that most of SBR's effort is put in creating the initial data structure upon which it performs the traversal, as it needs to read and create a complete map of all possible plans once, and after that it just traverse this structure. PHATTs runtime, on the other hand, is mostly invested in the observation processing, as it generates the plans per explanation.

Additional point is that SBR's observation processing is robust to different PL settings. While PHATT's observation processing does depend on the PL type, SBR is indifferent to it – after it builds the complete map of plans, the observation reading and processing time is not significantly different, even in the PL with the large depth value, which takes the longest for SBR to process.

Another point that rises from this graph is that the type of the PL have the same impact on both algorithms – for example, the PL with increased depth is the hardest to process for both algorithms. In PHATT the depth is significant because the algorithm generates the plans incrementally, meaning that the depth of the branches that needs to be added is larger. In SBR, it means that the tree traversal takes longer.

Another evaluation method we present is the total number of function calls used. This measure might not stand on its own as an important evaluation metric, but it can complete the time measurement to provide some more insights. Figure 8 shows the total number of function calls used both in the PHATT and SBR runs. It is consistent both with the complexity of the PL, and with the runtime of the algorithms. First, it is clear that the baseline PL required less function calls than some of the variant domains. Second, SBRs run-

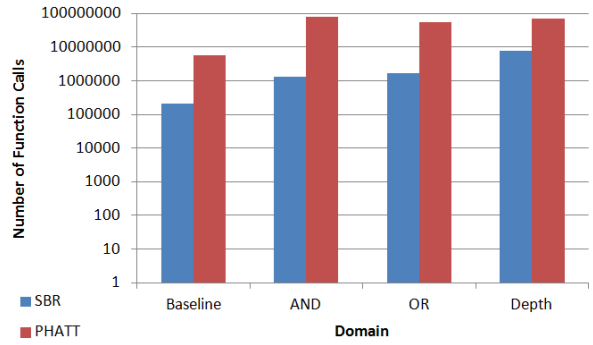

Figure 8: A comparison of function calls.

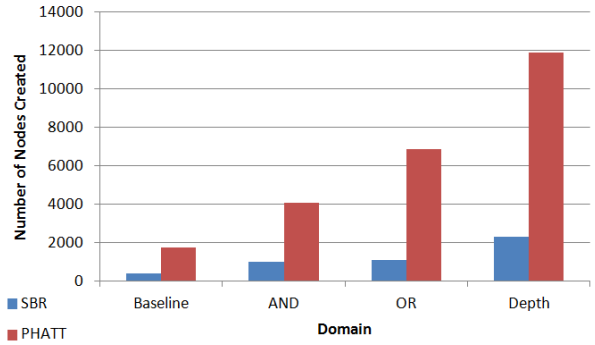

Figure 9: A comparison of space.

time is an order of magnitude or more faster than PHATT, and the gap in the number of function calls is similar.

Space

SBR was designed to save time – during observation processing, it only requires to traverse the plan library, while PHATT requires building new trees. The cost is that SBR spreads the complete PL in the beginning of the run. Theoretically, we would expect such a layout to consume more memory than PHATT. However, as seen in Figure 9, PHATT generated more nodes than SBR in the empirical tested PLs. This can be attributed to the fact that the more observations processed, the more trees that PHATT produces. It does so to consider every possible combination of adding a new observation into an existing explanation. This overhead was addressed in previous works, such as in the YAPPR algorithm [Geib *et al.*, 2008]. In this work, most of the bookkeeping of the plans is shed from PHATT, so that it only keeps the goal nodes and future actions that were yet to be seen in incomplete plans. However, the cost of this improvement is that the complete plans can no longer be outputted.

Discussion

In this paper we presented a first standard representation of plan libraries, PLDD. It is used as a standard of evaluation for plan recognition algorithms. Using this representation, we presented a checklist of criteria that should be considered when comparing plan recognition algorithms. We used this technique to evaluate, both theoretically and empirically, two known plan recognition algorithms. This evaluation sheds light both on the algorithm themselves, and on plan recognition domains in general.

The difference in approaches of SBR and PHATT can be translated into different outputs, as the evaluation using the TinkerPlots PL showed. In order to allow a fair evaluation, at least one of the algorithms will need to be fundamentally changed. This domain highlights the importance of standardizing the library description and expected behavior.

In most of the tested measures, SBR outperformed PHATT. Although SBR was implemented in Java and PHATT in Python, Java/Python speed differences are of linear factor [Prechelt, 2000, 2003], and similar in terms of memory consumption. Even allowing for such differences, PHATT is by far slower than SBR, and uses more space.

SBR is more robust, faster, and can give partial answers about the current state of the actor, but at the expense of the soundness of these partial answers. PHATT, on the other hand, is more comprehensive and flexible with the possible inputs it can process. It has advanced capabilities, such as inference of interleaved plans and the inherent ability to output prediction about future actions that were yet to be seen. These future actions are represented as the leaves in the generated plans that are not bounded to any observed action, like the second ccd node in Figure 3. These inherent abilities can be quite useful, and can justify using PHATT over SBR for certain applications.

Another contribution that emerges from this paper is a general evaluation of PL impact on the recognition process, regardless of the used algorithm. It was shown that these features of the PL are algorithm-dependent, meaning that some PL properties have a direct impact on the efficiency of the recognition. For example, shallower PLs are easier to handle than PLs with lower And- or Or- branching factor, regardless of the algorithm we will choose to use later. This result also serve as a motivation for works that try to design the domain such as [Keren *et al.*, 2017].

Future Work

We believe that this work opens many opportunities for other levels of evaluation: We would like to evaluate more algorithms and see how they can be compared using PLDD-defined PLs. We would also like to evaluate existing domains from the literature by running them on a variety of algorithms, to receive more general insights about plan recognition representation. Another interesting research direction is to apply this standardization to other plan recognition models, such as the PDDL-based works. This will also require to perform fundamental changes, such as adding states and effects to PLDD.

Acknowledgments

This research was partially funded by the Cyber@Ben-Gurion center. R.M. is a recipient of the Pratt fellowship at the Ben-Gurion University of the Negev.

References

O. Amir and Y. Gal. Plan recognition and visualization in exploratory learning environments. *ACM Transactions on Interactive Intelligent Systems*, 3(3):16:1–23, 2013.

D. Avrahami-Zilberbrand and G.A. Kaminka. Fast and complete symbolic plan recognition. In *International Joint Conference of Artificial Intelligence (IJCAI)*, volume 14, 2005.

D. Avrahami-Zilberbrand and G.A. Kaminka. Incorporating observer biases in keyhole plan recognition (efficiently!). In *AAAI*, volume 7, pages 944–949, 2007.

F. Bisson, F. Kabanza, A. R. Benaskeur, and H. Irandoust. Provoking opponents to facilitate the recognition of their intentions. In *AAAI*, 2011.

N. Blaylock and J. Allen. Fast hierarchical goal schema recognition. In *National Conference on Artificial Intelligence*, volume 21, page 796. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

H.H. Bui. A general model for online probabilistic plan recognition. In *Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

S. Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1):31–48, 2001.

T. Chakraborti, S. Sreedharan, Y. Zhang, and S. Kambhampati. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In *IJCAI*, 2017.

R.G. Freedman and S. Zilberstein. Integration of planning with recognition for responsive interaction using classical planners. In *AAAI*, pages 4581–4588, 2017.

C.W. Geib and R.P. Goldman. Plan recognition in intrusion detection systems. In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX'01. Proceedings*, volume 1, pages 46–55. IEEE, 2001.

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

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

C. W. Geib. Delaying commitment in plan recognition using combinatory categorial grammars. In *IJCAI*, pages 1702–1707, 2009.

C.W. Geib. Partial observability in grammar based plan recognition. In *Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR)*, 2017.

F. Kabanza, J. Filion, A. R. Benaskeur, and H. Irandoust. Controlling the hypothesis space in probabilistic plan recognition. In *IJCAI*, pages 2306–2312, 2013.

S. Keren, A. Gal, and E. Karpas. Goal recognition design. In *ICAPS*, 2014.

S. Keren, L. Pineda, A. Gal, E. Karpas, and S. Zilberstein. Equi-reward utility maximizing design in stochastic environments. *HSDIP 2017*, page 19, 2017.

C. Konold and C. Miller. *TinkerPlots Dynamic Data Exploration 1.0*. Key Curriculum Press, 2004.

J. Maraist. String shuffling over a gap between parsing and plan recognition. 2017.

P. Masters and S. Sardina. Cost-based goal recognition for path-planning. In *AAMAS*, pages 750–758. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

R. Mirsky and Y. Gal. Slim: Semi-lazy inference mechanism for plan recognition. In *International Joint Conference of Artificial Intelligence (IJCAI)*, 2016.

R. Mirsky, Y. Gal, and S.M. Shieber. Cradle: An online plan recognition algorithm for exploratory domains. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(3):45–1, 2017.

R. Mirsky, Y. Gal, and D. Tolpin. Session analysis using plan recognition. *arXiv preprint arXiv:1706.06328*, 2017.

R.F. Pereira, N. Oren, and F. Meneguzzi. Plan optimality monitoring using landmarks and planning heuristics. In *Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR)*, 2017.

Lutz Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23–29, 2000.

Lutz Prechelt. Are scripting languages any good? a validation of Perl, Python, Rexx, and Tcl against C, C++, and Java. *Advances in Computers*, 57:205–270, 2003.

M. Ramırez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*. Citeseer, 2010.

M. Shvo, Sohrabi S., and S.A. McIlraith. An ai planning-based approach to the multi-agent plan recognition problem. In *Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR)*, 2017.

S. Sohrabi, A.V. Riabov, and O. Udrea. Plan recognition as planning revisited. In *IJCAI*, pages 3258–3264, 2016.

G. Sukthankar and K. P. Sycara. Hypothesis pruning and ranking for large plan recognition problems. In *AAAI*, volume 8, pages 998–1003, 2008.

G. Sukthankar, C.W. Geib, H.H. Bui, D. Pynadath, and R.P. Goldman. *Plan, Activity, and Intent Recognition: theory and practice*. Newnes, 2014.

O. Uzan, R. Dekel, O. Seri, and Y. Gal. Plan recognition for exploratory learning environments using interleaved temporal search. *AI Magazine*, 36(2):10–21, 2015.

M. Vered and G.A. Kaminka. Heuristic online goal recognition in continuous domains. In *IJCAI*, pages 4447–4454, 2017.

S. Wiseman and S. Shieber. Discriminatively reranking abductive proofs for plan recognition. In *ICAPS*, 2014.