SLIM: Semi-Lazy Inference Mechanism for Plan Recognition

Reuth Mirsky and **Ya'akov (Kobi) Gal**
Department of Information Systems Engineering
Ben-Gurion University of the Negev, Israel
{dekelr@post.bgu.ac.il,kobig@bgu.ac.il}

Abstract

Plan Recognition algorithms require to recognize a complete hierarchy explaining the agent's actions and goals. While the output of such algorithms is informative to the recognizer, the cost of its calculation is high in run-time, space, and completeness. Moreover, performing plan recognition online requires the observing agent to reason about future actions that have not yet been seen and maintain a set of hypotheses to support all possible options. This paper presents a new and efficient algorithm for online plan recognition called SLIM (Semi-Lazy Inference Mechanism). It combines both a bottom-up and top-down parsing processes, which allow it to commit only to the minimum necessary actions in real-time, but still provide complete hypotheses post factum. We show both theoretically and empirically that although the computational cost of this process is still exponential, there is a significant improvement in run-time when compared to a state of the art of plan recognition algorithm.

1 Introduction

As intelligent systems become prevalent in our daily lives, our interactions can benefit greatly when these systems are able to understand our activities. This recognition problem can be roughly divided into two types of tasks: goal recognition and plan recognition. The goal recognition task can be viewed as a classification problem, where we receive a sequence of actions and need to label this sequence with a goal. For example, consider a case where we observe someone's actions in the kitchen. Depending on the granularity of our goals, we can recognize "baking" "baking a cake", "baking a chocolate cake", etc.

The plan recognition task, on the other hand, focuses on the complete hierarchy and dependencies between the observed actions. Given a sequence of observations, the plan recognition task is to provide with either a hierarchy of activities that explains the observations. Using the same kitchen example, a potential agent who performs plan recognition can hypothesize in real time about how the person is going to make a chocolate cake. This agent will potentially be able to assist the cook or prevent failures, while a goal recognition agent can only change the label from "baking a chocolate cake" to "failing to bake a chocolate cake".

This motivation can be seen in various domains: assist disaster response crews by recognizing the protocol they are using [Blaylock and Allen, 2006], advise about missing steps in the medical guideline a doctor is following [Charniak and Goldman, 2013; Ng and Mooney, 1992], assist to students learning via online software [Amir and Gal, 2013; Uzan *et al.*, 2015] and provide real-time response to a cyber attack according to its specific actions [Bisson *et al.*, 2011; Qin and Lee, 2004].

The plan recognition task is more costly than the goal recognition task. This is because it must output a complete hierarchy of plans that explain the observed actions. Existing online plan recognition algorithms can only process small sequences of actions, work on limited domains, or hinder the completeness of their outputted hypotheses [Wiseman and Shieber, 2014; Kabanza *et al.*, 2013]. Consider the baking scenario, where the cook starts with pouring flour and sugar - many cakes start with these steps, so keeping all possible hypotheses that explain the cook's actions will require a plan recognition algorithm to generate many different plans. In general, the number of hypotheses may grow very large even for a small number of observations [Mirsky *et al.*, 2016].

The main contribution of this paper is a new type of plan recognition algorithm that combines both bottom-up and top-down parsing processes called SLIM (Semi-Lazy Inference Mechanism). Rather than maintain complete plan hierarchies, SLIM constructs plans from the bottom up, according to a least commitment policy which adds the smallest plan fragments that explain the observations. We present a theoretical analysis showing that SLIM requires to consider less combinations than the state of the art. We then show empirically that the description of local behavior is easier to maintain in real-time and leads to significantly faster performances on a domain from the plan recognition literature [Kabanza *et al.*, 2013].

2 Related Work

Bui [2003] use particle filtering to provide approximate solutions to plan recognition problems. Ramirez and Geffner [2009] use a planning domain definition which allows an implicit representation of the possible plans. Suk-

thankar and Sycara [2008] propose heuristics (based on temporal constraints) to narrow the number of hypotheses to consider during recognition but require that the full plan library be generated in advance. Other works [Conati *et al.*, 1997; Katz *et al.*, 2007] use plan recognition algorithms to infer students' plans to solve problems in a simulated physics environment by comparing their actions to a set of predefined possible plans. None of these works was compared to each other and each was evaluated on domain-specific problems.

Shieber and Wiseman [2014] use a discriminative reranking approach in abductive settings in order to infer plans on the same data set; their work requires a training set and is not appropriate for online use. Other works which focused on the offline case of plan recognition, where the inference is performed after the agent completed execution [Amir and Gal, 2013; Uzan *et al.*, 2015]. However, these works cannot be of use if the recognition is needed in real-time.

Geib and Goldman presented two probabilistic online recognition algorithms – The first is PHATT [Geib and Goldman, 2009], that builds all possible plans incrementally with each new observation. This approach maintains all possible hypotheses for matching future unseen actions by the agent. While this approach is complete, it is very slow. The second, YAPPR [Geib *et al.*, 2008] was devised to be more efficient, but at the cost of symbolic representation of the plans. This approach outputs a distribution over the agent's inferred goals. Kabanza et al. [2013] extended this approach to compute lower and upper bounds on the goal hypothesis probabilities and prune the search space of possible plans. Again, this algorithm only outputs a distribution over the set of goals. Another work by Geib [2009] suggests the use of combinatory categorial grammars (CCGs). This representation is more compact at the cost of expressiveness, as it outputs probabilities on different categories instead of complete plans.

Avrahami and Kaminka's SBR [2005] suggested intelligent tree-based representation for plan recognition and an algorithm that traverses the trees and log actions in a manner that is temporally consistent with the observations. Upon query request, the logged actions can be used to construct the possible hypotheses at the time of the query. This delayed construction causes the algorithm to perform only minimal commitments about matching actions to the grammar. This work was extended by Fagundes et al. [2014] to provide an anytime expectation of time needed to recognize the plan, and by Avrahami and Kaminka [2007] to reason about the cost of The late commitment of SBR makes it extremely efficient in terms of time, but at the cost of space. In order to be able to provide with a hypothesis when needed, the algorithm must explicitly generate all paths that were possibly taken. SLIM was inspired by both PHATT's incremental approach and by SBR's lazy commitment.

Other previous works tried to reduce runtime complexity of the plan recognition process by giving up on the more specific details of the plan or hindering completeness [Geib, 2009; Geib *et al.*, 2008].

3 Background

We follow the definition by Geib and Goldman, in which we assume that the plan recognition algorithm is given a plan library which can implicitly describe any expected behaviors of the observed agent. We simplified their definition for clarity.

Definition 1 (Plan Library) *A plan library is a tuple* $L = \langle \Sigma, NT, G, R \rangle$*, where* Σ *is a finite set of terminal letters,* NT *is a finite set of non-terminal letters,* $G \subset NT$ *is the goal actions and* R *is a set of production rules of the form* $A \rightarrow \alpha : \phi, p$*, where:*

1. $A \in NT$
2. α *is a string of symbols from* $(\Sigma \cup NT)*$
3. $\phi = \{(i, j) \mid \alpha[i] \leq \alpha[j]\}$ *where* $\alpha[i]$ *and* $\alpha[j]$ *refer to the i-th and j-th symbols in* α*, respectively.*
4. p *is the prior probability for using this rule to execute* A*.*

This definition follows traditional PCFG based encoding for hierarchical plans [Pynadath and Wellman, 2000]. This representation is also similar to fundamental planning formalisms such as Hierarchical Task Networks (HTNs) [Ghallab *et al.*, 2004] but does not include action preconditions and effects. Σ represents the basic actions that can be observed. NT represents complex actions, which are different levels of abstract actions that can be expanded into sequences of both basic and complex actions. G is a subset of NT, representing the highest level of abstraction to describe a plan – by the goal it tries to accomplish. Each $r \in R$ is a production rule, describing how a complex action (the left side of the rule) can be accomplished by a sequence of both basic and complex actions (the right side of the rule). The tuples described in ϕ are explicit ordering constraints, similar to ID/LP grammars [Shieber, 1984]. A *plan* for achieving a complex action $c \in NT$ is a tree whose root is labeled by c, and each parent node is labeled with a complex action such that its children nodes are a decomposition of its complex action into constituent actions according to one of the production rules in R. The temporal constraints of the production rules enforce the order that the actions can be observed. In this paper, we use the terms "plan" and "tree" interchangeably.

A plan is said to be *complete* if all its leaf nodes are labeled with basic actions. Incomplete plans include complex level actions that have not been decomposed using a production rule. These *open frontier* nodes represent actions that are expected to be carried out by the agent in the future [Geib and Goldman, 2009].

For example, consider the following plan library:

Basic actions $\Sigma = \{a, b, c\}$

Complex actions $NT = \{X, A, B, C\}$

Goals $G = \{X\}$

Production rules Three rules for A,B,C of the form: $A \rightarrow a \mid \emptyset$ and a rule for achieving the goal $X \rightarrow A, B, C \mid \{(1, 2)\}$. The ordering constraint in this rule means that A must be achieved before B.

Each of the three plans in Figure 1 are incomplete plans for achieving the complex action X. Nodes with dashed outline

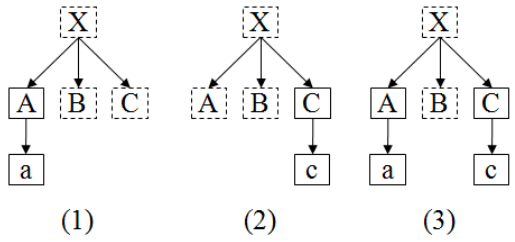

Figure 1: Example Plans

represent actions that have not yet been realized. Leaves with dashed outline represent open frontier nodes. As expected, the nodes in valid plans are always consistent with the rule's ordering constraints ϕ. This means that these plans are valid only if there isn't any constraint (i, j) in the rule if the i-th node is dashed and the j-th is not. Notice that the node for X is dashed as well, since in neither of these plans was X fully achieved yet.

An *observation sequence* is an ordered set of basic actions that represents actions carried out by the observed agent. A plan *describes* an observation sequence if each observation is mapped to a leaf node in the plan. The observed agent 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 the example from Figure 1, plan (a) explains the observation a, plan (b) explains c and plan (c) explains a, c.

An agent may execute more than one plan intermittently, and the observations explained by a given plan may not be contiguous. A *hypothesis* for a given observation sequence is a set of plans. Each of the plans in a hypothesis satisfies one of the goals that the agent is carrying out. For example, a possible hypothesis for the observation sequence a, c can be plan (c) in Figure 1, while another hypothesis can be a set of plans (a) and (b), which together explain the sequence a, c.

We assume that each hypothesis is assigned a probability representing the belief that this is the agent's intended set of plans, given the set of observations. This probability is calculated using the production rules' probabilities, in a similar fashion to Geib and Goldman [2009].

Both PHATT and the newly presented SLIM perform incrementally – at each step n, the algorithm gets a new observation σ_n and tries to combine it with the hypotheses that explain $\sigma_1, \ldots, \sigma_{n-1}$. The difference between the algorithms is in the construction process.

At the first observation σ_0, PHATT constructs a partial tree where the root is labeled with a goal $g \in G$ and a path in the tree from G to the observation. It is required that all leaves in the tree are open frontier nodes labeled with non-terminal symbols in NT, except the leaf representing the observation itself which is a terminal symbol in Σ. This leaf is called the *leftmost child* of the sub-tree, while the partial tree is referred to as a *leftmost tree*. The leftmost child represent the next possible action that can be added to the plan. PHATT attempts to combine each new observation σ_i to each hypothesis in the set of previous hypotheses, either by connecting the partial tree to an existing plan, or as a new plan in the hypothesis. We formalize this intuition using the following definitions:

Definition 2 (leftmost child) *Let T be a plan and let α_0 be an internal node in T with children $\langle \alpha_1, \ldots, \alpha_{i-1}, \alpha_i, \alpha_{i+1} \ldots, \alpha_n \rangle$. We say that α_i is a* left-most child *of node α_0 if:*

1. *α_i is an open frontier in T.*
2. *$\exists r \in R \mid \alpha_0 \rightarrow \alpha_1, \ldots, \alpha_{i-1}, \alpha_i, \alpha_{i+1} \ldots, \alpha_n : \phi$ such that for every child node labeled α_j, such that $j \neq i$, it holds that α_j is not in the open frontier of T.*

Definition 3 (leftmost tree) *A* leftmost tree *deriving an observation σ_t is a plan with frontier $\langle \alpha_1, \ldots, \sigma_i, \ldots, \alpha_n \rangle$ such that*

1. *$\sigma_i \in \Sigma$ and $\alpha_j \in NT$ for all $j \neq i$; and*
2. *For any child node labeled α_j and its parent α_k in the path from σ_i to the root, the node α_j is a leftmost child of α_k.*

In Figure 1, plan (1) has two leftmost children which are B, C, plan (2) has one leftmost child which is A and plan (3) has one leftmost child which is B. Notice that although there are two nodes in the open frontier in plan (2), only the leftmost child A can be considered for expanding this plan. Only plans (1) and (2) are leftmost trees, since plan (3) has two leftmost children.

When introducing σ_i into an existing hypothesis, PHATT considers two possibilities:

1. Adding the observation as a new plan in the hypothesis, which requires to construct a leftmost tree that derives σ_i with a root in G, the set of goals.
2. Updating an existing plan in the hypothesis with the observation, which requires to construct a leftmost tree deriving σ_i with a root which matches one of the open frontier nodes of an existing plan.

A disadvantage of this process is that each plan is explicitly represented in the hypothesis and the algorithm must maintain, for each hypothesis in the set, all possible plans that explain the observations seen thus far. Even for a small number of observations, the size of a plan in the corresponding hypothesis may be very large. For example, if the baker is taking out eggs, the algorithm should recognize that the next step is finding the whisk, no matter which type of cake is being prepared. However, PHATT must keep a separate hypothesis for each possible recipe - from chocolate cake to red velvet - and keep those deep plans through all next steps.

In order to reduce this cost, we propose that the incremental construction of a plan does not commit to a specific path between a goal and an observation until it is specifically needed.

4 The SLIM Algorithm

The motivation for SLIM derives from the need to infer the agent's plans in real-time. In many cases, the goal the agent is pursuing is less important, but rather how the observed actions combine together locally. Consider a known use case from the cyber security domain of an advanced persistent threat (APT) – an agent wishes to take use of resources and

exploit weaknesses over time. We wish to use plan recognition to detect and response to such attacks – at any given point in time, we do not need to know exactly what attack is being devised on the long run, but rather which local protocols are used (identity theft, permissions sharing, etc.) in order to respond to them specifically. In such scenarios, PHATT's production of complete paths to the root causes unnecessary overhead. SLIM works similarly to PHATT, but instead of producing a complete path to a possible root for each observation, it simply creates a 1-depth tree with the observation as a leaf. Each such tree is called a fragment.

Definition 4 (fragment) *A* fragment *for an action σ is a leftmost tree deriving σ with a depth of 1.*

Given a set of fragments for observation σ_n, SLIM tries to combine each new fragment with the hypothesis $h \in H_{n-1}$, either by adding it as a standalone fragment (the function COMBINEINDEPENDENTLY) or by fusing it with other plans (the functions COMBINEASCHILD and COMBINEASSIBLING).

Definition 5 (Fusion) *A plan p can be* fused *with a fragment f if (1) there exists a node o in the plan's open frontier; (2) the root of f is the same action as o; (3) there are no ordering constraints preventing from o to be executed next.*

For example, consider plan 2 from Figure 1 – although both A and B are in the plan's open frontier, a fusion can be performed only with a fragment whose root node is A, since B must be executed after A. We extend the definition of fusion to a hypothesis and fragment, where the one of the plans in the hypothesis can be fused with a fragment. A hypothesis in SLIM is a set of fused fragments which describe the observations. A *local hypothesis* is a set of fused fragments.

The SLIM algorithm includes two processes, one that constructs local hypotheses from the bottom-up, and one that constructs complete hypotheses using a top-down approach. The resulting improvement of SLIM in comparison to PHATT, is that it does not need to carry long paths to the root unless it is specifically asked for by the top-down process. It is similar to the SBR algorithm [Avrahami-Zilberbrand and Kaminka, 2005], which only keeps the "current state" unless it is required to complete the paths. However, it does commit (in each hypothesis) to one small fragment of the plan, thus making it only a semi-lazy algorithm, where SBR is defined lazy and does not commit to a path until it is necessary. In order to make sure that all ordering constraints are preserved, each fragment also keeps a timestamp, in a similar way to SBR. This timestamp will help in deciding whether two fragments that were developed individually can be combined in such a way that the ordering constrains will not be damaged.

4.1 Bottom-Up Inference

A main contribution of the SLIM is its process of deciding when two fragments are related and can be fused into one plan. After seeing $\sigma_1, \ldots, \sigma_{n-1}$, SLIM holds a local hypothesis set H_{n-1} for describing the observations so far. When seeing a new observation, σ_n, SLIM calls CREATEFRAGMENTS to generate a set of fragments for σ_n. Then SLIM

function SLIM BOTTOM-UP(H_{n-1}, σ_n)
  $H_n \leftarrow H_n \cup$ COMBINEDIRECTLY(H_{n-1}, σ_n)
  $F_n \leftarrow$ CREATEFRAGMENTS(σ_n)
  for all f $\in F_n$ **do**
    $H_n \leftarrow H_n \cup$ COMBINEASCHILD(H_{n-1}, f)
    $H_n \leftarrow H_n \cup$ COMBINEASSIBLING(H_{n-1}, f)
    $H_n \leftarrow H_n \cup$ COMBINEINDEPENDENTLY(H_{n-1}, f)
  Return H_n

function COMBINEDIRECTLY(H_{n-1}, σ_n)
  for all $h \in H_{n-1}$ **do**
    for all open frontier node $o \in h$ **do**
      $h' \leftarrow$ FUSE(h, o, σ_n)
      if h' is a valid hypothesis **then**
        $H_n \leftarrow H_n \cup h'$

function COMBINEASSIBLING(H_{n-1}, f)
  $H_n \leftarrow \emptyset$
  for all $h \in H_{n-1}$ **do**
    for all plan $p \in h$ **do**
      $F' \leftarrow$ CREATEFRAGMENTS($f.root()$)
      for all $f' \in F'$ describing p and f as nodes α_i, α_j **do**
        $p' \leftarrow$ FUSE(f', α_i, p)
        $p' \leftarrow$ FUSE(f', α_j, f)
        $h' \leftarrow h \backslash p \cup p'$
        if h' is a valid hypothesis **then**
          $H_n \leftarrow H_n \cup h'$

  Return H_n

Figure 2: SLIM's Main Functions

tries to combine each of the fragments to each of the hypotheses $h \in H_{n-1}$ using each of the following approaches:

1. As an immediate child – for each $p \in h$, a plan in the hypothesis, and for each o, open frontier item in p, if $o \in \Sigma$ it will try to directly replace o with σ_n. This is performed by the function COMBINEDIRECTLY.
2. As a fragmented child – for each $p \in h$, a plan in the hypothesis, and for each f, a fragment for σ_n, it will try to fuse p with f. This is performed by the function COMBINEASCHILD.
3. As a sibling – for each $t \in h$, a plan in the hypothesis, and for each f, a fragment for σ_n, it will try to find a rule $\alpha_0 \rightarrow \alpha_1, \ldots, \alpha_k : \phi$ in which both the root of f and the root of p are in $\alpha_1, \ldots, \alpha_k$. It will then create a higher level in the plan such that the new root will be α_0, all the leaves but two will be open frontiers and the other two will be replaced with p and f. This is performed by the function COMBINEASSIBLING. The fragments must obey the ordering constraints in ϕ. The timestamp of the new fused fragment is the smallest timestamp of the original fragments.
4. Independently – $h' = h \cup \{f | f$ is a fragment for $\sigma_n\}$. We must always consider the case that a new fragment is part of a new plan, or that it will be connected to another fragment in a later stage. This is performed by the function COMBINEDIRECTLY.

Figure 3 shows the process of constructing some of the

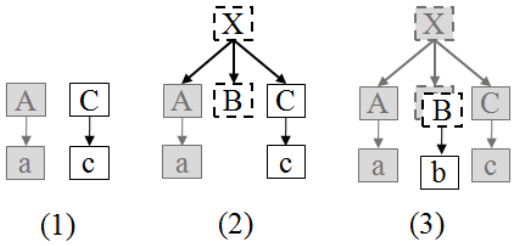

Figure 3: Example Hypotheses and Fusion Process

possible hypotheses, using the previously defined plan library and the observation sequence a, c, b. For the first observation, a, SLIM will output a single hypothesis with the fragment for $A \rightarrow a$. Upon this hypothesis and the new observation c, two new hypotheses can be constructed: hypothesis (1) will be built using the COMBINEINDEPENDENTLY method and hypothesis (2) will be built using the COMBINEASSIBLING method. Next, when trying to combine the third observation, b, it can be fused to hypotheses (2) with the COMBINEASCHILD method, resulting with hypothesis (3). Hypothesis (3) shows the shadowed hypothesis after a, c and the fusion with the new fragment for b. This is not the only valid hypothesis for these observations , both hypotheses (1) and (2) can be extended using the COMBINEINDEPENDENTLY method and the fragment for $B \rightarrow b$, thus adding two more possible hypotheses.

Complexity Analysis The bottleneck of both PHATT and SLIM is constructed from checking all combinations to add the new observation to the hypothesis set, which depends on the hypothesis set size from previous steps.

Intuitively, although SLIM will create more hypotheses than PHATT (due to inability to enforce ordering constraints between fragments), the size of each hypothesis is smaller and the number of possible combinations will be smaller. To analyze the complexity, we define C_i to be the size of options to combine the i-th observation. For PHATT, C_i was analyzed to be $O((w_i \times b)^{h_i})$ where h_i is the maximum depth of a plan (for recursive grammars, h_i is bounded artificially), w_i is number of open frontiers in all hypotheses and b is the maximal OR branching factor of the plan library. Unlike PHATT, SLIM works bottom-up and develops a fragment instead of a complete path to the root and each fragment can be joined in a way that will increase either w_i (by joining as a sibling or independently) or h_i (by joining as a child) but not both. This means that for SLIM, the size of the combinations at each step will be $C_i = O((w_i \times b)^{log(h_i+w_i)})$.

4.2 Top-Down Inference

The generated hypotheses from SLIM's bottom-up parsing are local hypotheses that do not commit to the agent's underlying goal. However, we might wish to know the hierarchy of the outputted plans up until the goal. Usually, we don't require the complete set of hypotheses, just the k-most probable ones. In the tutoring scenario, for example, the complete plans might be sent to a teacher for review, or used to aggregate the paths used by different students.

A similar difference between the goal-rooted and local hypotheses was defined in the SBR's paper as querying about current state vs. querying about history of states, where the "current state" query is similar to SLIM's bottom-up output and the "history of states" query is similar to PHATT's output. We wish to be able to perform a "history of states" query in SLIM, meaning to query about plans that cover complete paths from the observed actions to goals. This is done by calling the PHATT algorithm with minor modifications.

As discussed in previous sections, PHATT takes an observation and generates a complete path to a goal root. Here, we feed the PHATT algorithm with the fused fragments of a hypothesis instead of with observations. The rest of the process remains the same – PHATT tries to combine the local plans to goal-rooted plans. We perform this for the k-most probable hypotheses. In order to keep consistent with the ordering constraints, the fragments must be given to the modified PHATT by the order they were constructed, which is the timestamp that each fragment receives upon creation. This process can be exemplified using hypothesis (a) from Figure 3: when requesting for goal-rooted plans, the modified PHATT will first receive the fragment for A, as it was created earlier, and will produce a plan like plan (1) in Figure 1. Next, it will receive the fragment for C, which it will be able to combine with plan (1). Like in the example described above, there might be an overlap between the hypotheses constructed in the bottom-up step and the hypotheses constructed in the top-down step. In order to avoid such duplication, we must force the top-down inference to discard the leftmost trees which we already constructed. This is done by a straightforward bookkeeping, where we check all leftmost trees which were already produced by the bottom-up inference. The complexity of this step is same as the complexity of PHATT.

Completeness The local hypotheses can be ranked by their probabilities, which is calculated as the multiplication of the constituting fragments' probabilities. Since the underlying probability model is the same, performing top-down inference for the k-most probable local hypotheses will result with the k-most probable hypotheses in PHATT, iff the prior probability of the goals distribute uniformly.

5 Empirical Evaluation

We evaluated the SLIM algorithm on a simulated domain, based on AND/OR trees used by Kabanza et al. [2013]. We used their same configuration which includes 100 instances with a fixed number (100) of basic actions, five identified goals, a branching factor of 3 for AND rules in the grammar and a branching factor of 2 for OR rules. Using these settings, each instance contains a sequence of 9 basic actions.

After compiling the rules to match the grammar representation in this paper, we got a plan library with 140 complex level actions and 244 rules. We ran both PHATT and SLIM's bottom-up inference on the 100 instances and compared runtimes, number of created hypotheses and C_i for each algorithm. We did not run SBR on this domain due to complexity issues – SBR requires the rules to be modeled using only OR graphs. The transfer to this formalization requires 2^{24} different rules, which makes it unfeasible for this domain.

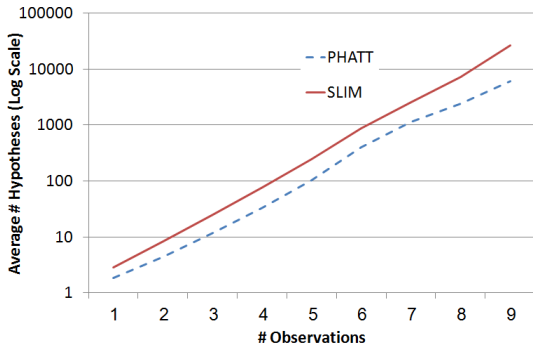

Figure 4: Average Number of Hypotheses per Length of Observations Sequence

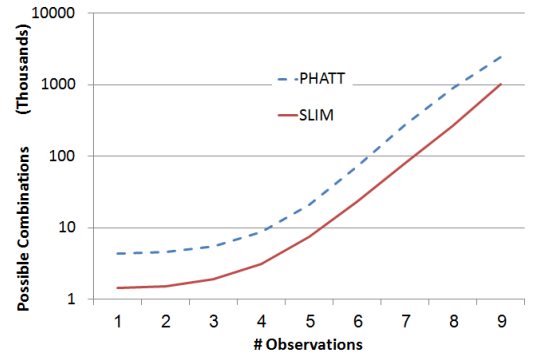

Figure 5: Average Value of C_i (Possible Combinations) per Length of Observations Sequence

As seen in Figure 4, the number of local hypotheses that the algorithm requires to keep is still exponential as in PHATT. SLIM even contains a larger number of hypotheses than PHATT, though not statistically significant. However, since in PHATT a hypothesis contains goal-rooted plans right from the first observation, the depth of the plans is larger and there are more open frontiers to consider. SLIM, on the other hand, contains plans as shallow as possible. The cost of PHATT's long hypotheses can be seen in Figure 5, where it is shown that the number of combinations SLIM requires to consider is an order of magnitude less than PHATT ($p \leq 0.05$).

Table 1 presents the runtime of the algorithms. These tests were conducted on the same commodity computer and were based on the same code, differing only in the step function of the different algorithms. Using these settings, SLIM was significantly faster the PHATT at every step ($p \leq 0.05$).

Next, we ran several variations of SLIM, notated using SLIM-i where i is the differing number of hypotheses calculated by the top-down inference (for example, calculating 0-most probable hypotheses, meaning only the bottom-up inference is performed, is notated SLIM-0). We used these variations to evaluate the difference in runtime for the top-down step. The results are shown in Figure 6. SLIM-* is the complete compilation of all the hypotheses (which can be more than 10,000, as seen in Figure 4) to goal-rooted ones. As seen in this table, calculating the 100-th most probable hypotheses to the root was not significantly slower than SLIM-0 and still faster than PHATT ($p \leq 0.05$). Compiling the complete set of hypotheses will induce a significant increase in the runtime. However, for the challenge of creating a robust online plan recognition algorithm that can output description of the agent's actions in real-time, the results show significant improvement for SLIM in comparison to PHATT.

Obs.	1	2	3	4	5	6	7	8	9
PHATT	0.24	0.34	0.55	0.84	1.18	1.66	2.42	4.46	9.48
SLIM	0.16	0.16	0.18	0.25	0.43	0.74	1.21	1.92	4.45

Table 1: Average Runtime (in seconds) of SLIM's bottom-up inference and PHATT

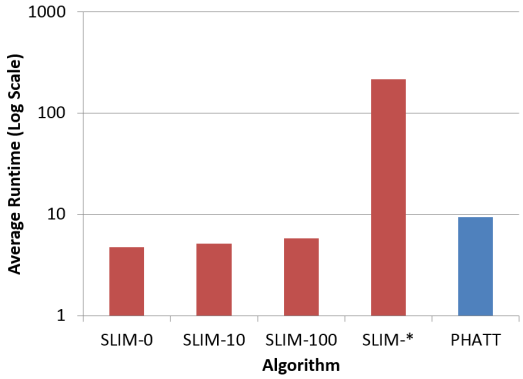

Figure 6: Average Runtime (in Seconds) per Algorithm Variant

6 Discussion and Conclusion

In this paper we presented a new plan recognition algorithm which uses the benefits of two well-known algorithms. We showed that compared to state of the art algorithms, both in terms of expressiveness and capabilities, SLIM is superior.

The price SLIM's bottom-up inference pays to gain faster performance, is that the ultimate goals of the agents are no longer available. However, at any given point in time the top-down inference can combine the set of fragments into goal-rooted hypotheses by looking at each fragment's root as an observation. Moreover, in cases where the bottom-up inference manages to reach the goal, the top-down process may no longer be necessary. To conclude, for the purpose of reasoning about the agent's actions in real-time, it is sometimes sufficient to know only the complex actions that are already being executed, as SLIM's bottom-up inference outputs. If at any point the inference of the goals becomes important, the top-down inference complements the process, either by providing small set of hypotheses in real-time, or by inferring the complete set offline.

The most immediate investigation we wish to pursue is making SLIM an anytime algorithm that provides an increasing number of goal-rooted hypotheses. We also intend to improve the top-down step of the algorithm. by to using an existing goal recognition algorithm, which is significantly faster, and check consistency of the proposed goals with the out-

putted plans from SLIM's bottom-up mechanism.

Acknowledgments

This work was supported in part by EU FP7 FET project no. 600854, and the Israeli Science Foundation Research Grant no. 1276/12. R.M. is a recipient of the Pratt fellowship at the Ben-Gurion University of the Negev.

References

[Amir and Gal, 2013] 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.

[Avrahami-Zilberbrand and Kaminka, 2005] D. Avrahami-Zilberbrand and G.A. Kaminka. Fast and complete symbolic plan recognition. In *IJCAI*, volume 14, 2005.

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

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

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

[Bui, 2003] H.H. Bui. A general model for online probabilistic plan recognition. In *IJCAI*, volume 3, pages 1309–1315, 2003.

[Charniak and Goldman, 2013] E. Charniak and R.P. Goldman. Plan recognition in stories and in life. *arXiv preprint arXiv:1304.1497*, 2013.

[Conati *et al.*, 1997] C. Conati, A.S. Gertner, K. VanLehn, and M.J. Druzdzel. On-line student modeling for coached problem solving using bayesian networks. *Proceedings of the Sixth International Conference on User Modeling*, pages 231–242, 1997.

[Fagundes *et al.*, 2014] M. S. Fagundes, F. Meneguzzi, R. H. Bordini, and R. Vieira. Dealing with ambiguity in plan recognition under time constraints. In *AAMAS*, pages 389–396. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

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

[Geib *et al.*, 2008] Christopher W Geib, John Maraist, and Robert P Goldman. A new probabilistic plan recognition algorithm based on string rewriting. In *ICAPS*, pages 91–98, 2008.

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

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated planning: theory & practice*. Elsevier, 2004.

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

[Katz *et al.*, 2007] S. Katz, J. Connelly, and C. Wilson. Out of the lab and into the classroom: An evaluation of reflective dialogue in ANDES. *Frontiers In Artificial Intelligence and Applications*, 158:425, 2007.

[Mirsky *et al.*, 2016] R. Mirsky, R. Stern, Y. Gal, and Kalech M. Sequential plan recognition. In *IJCAI*, 2016.

[Ng and Mooney, 1992] H.T. Ng and R.J. Mooney. Abductive plan recognition and diagnosis: A comprehensive empirical evaluation. *KR*, 92:499–508, 1992.

[Pynadath and Wellman, 2000] D.V. Pynadath and M.P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 507–514, 2000.

[Qin and Lee, 2004] X. Qin and W. Lee. Attack plan recognition and prediction using causal networks. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 370–379. IEEE, 2004.

[Ramırez and Geffner, 2009] M. Ramırez and H. Geffner. Plan recognition as planning. In *AAAI*, 2009.

[Shieber, 1984] S.M. Shieber. Direct parsing of ID/LP grammars. *Linguistics and Philosophy*, 7(2):135–154, 1984.

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

[Uzan *et al.*, 2015] 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.

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