On-Line Plan Recognition in Exploratory Learning Environments

Reuth Dekel and Ya'akov (Kobi) Gal
Dept. of Information Systems Engineering
Ben-Gurion University
Beer-Sheva 84105, Israel

ABSTRACT

Exploratory Learning Environments (ELE) are open-ended and flexible software, supporting interaction styles by students that include exogenous actions and trial-and-error. ELEs provide a rich educational environment for students and are becoming increasingly prevalent in schools and colleges, but challenge conventional plan recognition algorithms for inferring students' activities with the software. This paper presents a new algorithm for recognizing students' activities in ELEs that works on-line during the student's interaction with the software. Our approach, called CRADLE, reduces the amount of explanations that is maintained by the plan recognition in a way that is informed by how people execute plans. We provide an extensive empirical analysis of our approach using an ELE for chemistry education that is used in hundreds of colleges worldwide. Our empirical results show that CRADLE was able to output plans exponentially more quickly than the state-of-the-art without compromising correctness. This result was confirmed in a user study that included a domain expert who preferred the plans outputted by CRADLE to those outputted by the state-of-the-art approach for the majority of the logs presented.

1. INTRODUCTION

This paper focuses on inferring students' activities in educational environments in which students engage widely in exploratory behavior, and present new approaches for plan recognition in such settings that can outperform the state-of-the-art.

Our empirical analysis is based on students' interactions with an Exploratory Learning Environment (ELE) in which students build scientific models and examine properties of the models by running them and analyzing the results[1, 6]. Such software is open-ended and flexible and is generally used in classes too large for teachers to monitor all students and provide assistance when needed. The open-ended nature of ELEs affords a rich spectrum of interaction for students: they can solve problems in many different ways, engage in exploratory activities involving trial-and-error, they can repeat activities indefinitely, and they can interleave between activities.

These aspects significantly hinder the possibilities of making sense of students' activities without some sort of support. This paper presents a new algorithm for recognizing students' interactions with ELEs in real time, which can support both teachers and students. For teachers, this support takes the form of visualizing students' activities during their interaction in a way that faciliteates their understanding of students' learning. For students, this support can take the form of machine generated intervention that guides their learning and adapts to individual students' needs based on their inferred behavior.

The focus of this paper is on-line recognition that occurs during the students' actual interaction with the ELE, and outputs a hierarchy of interdependent activities that best describe the student's work at a given point in time. Recognizing students' activities this way is challenging because the algorithm needs to reason about and maintain possible explanations for future (yet unseen) student activities. The number of possible explanations grows exponentially with the number of observations. As we show in the empirical section of this paper, this significantly hinders the performance of the state-of-the-art, even for very short interaction sequences.

Our algorithm, called CRADLE (Cumulative Recognition of Activities and Decreasing Load of Explanations) builds on an existing approach for on-line plan recognition, but filters the space of possible explanations in a way that reflects the style of students' interactions in ELEs. The filtering aim is to produce complete, parsimonious and coherent explanations of students' interactions that can be easily understood by teachers and education researchers.

Our empirical evaluations were based on comparing CRADLE to the state-of-the-art approach for recognizing logs of students' interactions with a widely used ELE for chemistry education. We evaluated both of the approahes in terms of computation speed and correctness of the outputted explanation, as determined by a domain expert. Succeeding in both of these measures is criticial for an on-line plan recognition approach to work successfully.

Our empirical results show that CRADLE was able to outperform the state-of-the-art without compromising correctness. Specifically, although the state of the art approach is (in theory) complete, it was not able to terminate within an allocated time frame on many logs. In contrast, CRADLE was able to produce correct explanations for such logs. In addition, CRADLE significantly outperformed the state-of-the-art both in terms of correctness and speed of recognition.

These results demonstrate the benefit of applying novel plan recog-

nition technologies towards intelligent analysis of students' interactions in open-ended and flexible software. Such technologies can potentially support teachers in their understanding of student behavior as well as students in their problem solving, and lead to advances in automatic recognition in other exploratory domains.

2. RELATED WORK

Our work relates to two strands of research, inferring students' activities in educational software, and on-line planning algorithms in artificial intelligence. We relate to each of these in turn.

2.1 Inferring Students' Activities in ELEs and ITS systems

We first describe works that infer students' plans from their interactions with pedagogical software that assume the complete interaction sequence is known in advance. Gal et al. [11] and Reddy et al. [10] used plan recognition to infer students' plans from their interactions with TinkerPlots, an exploratory learning environment for statistics. Both of these approaches take as input a complete interaction sequence of a student as well as recipes for ideal solutions to TinkerPlots problems, and infer the plan used by the student retrospectively. Reddy et al. [10] proposed a complete algorithm which modeled the plan recognition task as a Constraint Satisfaction Problem (CSP). The complexity of the CSP algorithm is exponential in the size of both the interaction sequence and the data set containing the recipes. This approach requires that all possible plans can be explicitly represented, and therefore does not support recursive grammars which are needed to understand students' activities in VirtualLabs.

Other works have implemented plan recognition techniques to model students' activities in Intelligent Tutoring Systems (ITS) during their interactions. In contrast to exploratory learning environments, in intelligent tutoring systems the system takes an active role in students' interactions, as it tutors the student by providing feedback and hints. As an example, in the Andes physics tutor wrong steps are marked by the tutor and the students may ask for a "what's wrong here?" hint from the tutor. In addition, students can ask for a "what next?" hint to receive instruction when uncertain about how to proceed [20]. These systems are typically more closed-ended and less exploratory than ELEs. In the Andes physics tutor a probabilistic algorithm was used to infer the solutions plan followed by the student. For each Andes problem, a solution graph representing the possible correct solutions to the problem was automatically generated and were modeled using a dynamic Bayesian network. The algorithm observes students' actions and updates the probabilities of the different possible plans. The inferred plans were used to generate hints and to update students' cognitive models.

The tutors developed by the ACT-R group for teaching LISP, geometry and algebra, performed plan recognition using a model-tracing algorithm that tracked students' solution plans [2, 9]. These tutors maintained a list of production rules that can be triggered to accomplish the goal and sub-goals for solving a problem. The algorithm infers students' plans by identifying the production rules that were triggered according to the actions students had taken. After each observed action, the algorithm commits to a production rule that it infers the student triggered to perform the action. The system constrained students to remain on "correct paths" throughout their session by providing feedback after each action taken by the student. Moreover, ambiguities regarding the production rules being used by students were resolved by querying the student. By committing to one production rule at a time and enforcing students to remain on correct solution paths, the complexity of the plan recognition task in intelligent tutoring systems is substantially reduced.

Lastly, we mention works that use recognition techniques to model students' activities in Intelligent Tutoring Systems [20, 7, 21]. Our work is distinct from works on plan recognition in intelligent tutoring systems in several ways. First, ITS are more closed-ended from ELEs. Thus, students' activities with such software more constrained and less exploratory, and are easier to model and recognize. In addition, the tutoring systems described above provided constant feedback to students which helped them remain on correct solution paths that are recognizable by the model used. Second, the tutoring systems described above explicitly modeled all possible solution plans for solving a specific problem. This is not possible in the VirtualLabs domain, as there may be an infinite number of possible plans for solving a problem.

2.2 On-line Plan Recognition in Artificial Intelligence

We now discuss general work from Artificial Intelligence that is concerned with plan recognition in general, rather than recognizing students' activities in pedagogical software. On-line plan recognition is a significantly more difficult task than its off-line variant. The fact that the interaction sequence is not observed ahead of time raises additional challenges to on-line plan recognition. Blaylock et al. [4] developed an algorithm to infer the goal of users from their actions in a Linux shell environment. Pynadath [19] proposes a probabilistic inference of plan, but requires the observations to be fully ordered. The approach by Bui [5] used particle filtering to provide approximate solutions to on-line plan recognition problems. Avrahami and Kaminka [3] presented a symbolic on-line plan recognition algorithm which keeps history of observations and commits to the set of possible plans only when it is explicitly needed for querying. Geib and Goldman presented PHATT [14], a probabilistic on-line plan recognition algorithm that builds all possible plans incrementally with each new observation. This algorithm was applied to recognizing users' strategies in real-time video games [17].

All of these works have been evaluated on simulated, synthesized problems [19, 3, 14] or on toy problems [4, 17]. These approaches do not scale to the complexities of real-world domains. An exception is the work of Conati et al. [8, 18] who used on-line plan recognition algorithms to infer students' plans to solve a problem in an educational software for teaching physics, by comparing their actions to a set of predefined possible plans. Unfortunately, the number of possible plans grow exponentially in the types of domains we consider, making it unfeasible to apply this approach.

3. PLANS AND EXPLANATIONS

In this section we provide the basic definitions that are required for formalizing the on-line plan recognition problems in ELEs. Throughout the paper we will use an existing ELE for chemical education called VirtualLabs to demonstrate our approach which is actively used by students worldwide as part of their introductory chemistry courses. VirtualLabs allows students to design and carry out their own experiments for investigating chemical processes by simulating the conditions and effects that characterize scientific inquiry in the physical laboratory [22]. We use the following problem called "Oracle", which is given to students:

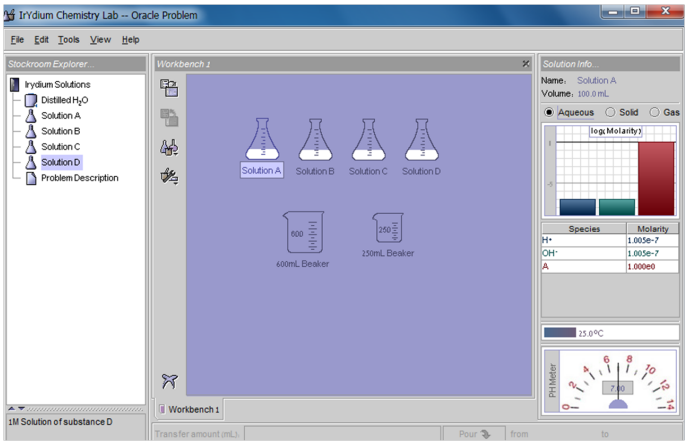

Figure 1: Snapshot of VirtualLabs

(a) $\underline{\mathsf{MSD}}[s_1 + s_2, d] \rightarrow \underline{\mathsf{MSD}}[s_1, d], \underline{\mathsf{MSD}}[s_2, d]$
(b) $\underline{\mathsf{MIF}}[s_1, d_2] \rightarrow \underline{\mathsf{MSD}}[s_1, d_1], \underline{\mathsf{MSD}}[d_1, d_2]$
(c) $\underline{\mathsf{MSD}}[s, d] \rightarrow \underline{\mathsf{MIF}}[s, d]$
(d) $\underline{\mathsf{MSD}}[s, d] \rightarrow \mathsf{MS}[s, d]$

Figure 2: Recipes for VirtualLabs

> Given four substances A, B, C, and D that react in a way that is unknown, design and perform virtual lab experiments to determine the correct reaction between these substances.

The flexibility of VirtualLabs affords two classes of solution strategies to this problem (and many variations within each). In the first strategy, a student mixes all four solutions together, and infers the reactants by inspecting the resulting solution. In the second strategy, a student mixes pairs of solutions until a reaction is obtained. A snapshot of a student's interaction with VirtualLabs when solving the Oracle problem is shown in Figure 1.

3.1 Definitions

We make the following definitions taken from the classical planning literature [16]. We use the term *basic actions* to define rudimentary operations that cannot be decomposed. These serve as the input to our plan recognition algorithm. For example, the basic "Mix Solution" action ($\mathsf{MS}_1[s = 1, d = 3]$) describes a pour from flask ID 1 to flask ID 3. A *log* is the output of a student's interaction. It is a sequence of basic level actions representing students' activities'. This is also the input to the plan recognition algorithm described in the next section.

Complex actions describe higher-level, more abstract activities that can be decomposed into sub-actions, which can be basic actions or complex actions themselves. For example, the complex action $\underline{\mathsf{MSD}}[s = 1 + 5, d = 3]$ (as shown in Figure 3) represents separate pours from flask ID 1 and 5 to flask ID 3.

A *recipe* for a complex action specifies the sequence of actions required for fulfilling the complex action. Figure 2 presents a set of basic recipes for VirtualLabs . In our notation, complex actions are underlined, while basic actions are not. Actions are associated with parameters that bind to recipe parameters. Recipe (a) in the figure, called Mix to Same Destination ($\underline{\mathsf{MSD}}$), represents the activity of pouring from two source flasks (s_1 and s_2) to the same destination flask (d). Recipe (b), called Mix via Intermediate Flask ($\underline{\mathsf{MIF}}$), represents the activity of pouring from one source flask (s_1) to a destination flask (d_2) via an intermediate flask (d_1).

Recipes can be recursive, capturing activities that students can repeat indefinitely. Indeed, this is a main characteristic of students' use of ELEs. For example, the constituent actions of the complex action $\underline{\mathsf{MSD}}$ in recipe (a) decompose into two separate $\underline{\mathsf{MSD}}$ actions. In turn each of these actions can itself represent a Mix to Same-Destination action, an intermediate-flask pour (by applying recipe (c)) or a basic action mix which is the base-case recipe for the recursion (recipe (d)). Recipe parameters also specify the type and volume of the chemicals in the mix, as well as temporal constraints between constituents, which we omit for brevity.

More generally, the four basic recipes in the figure can be permuted to create new recipes, by replacing $\underline{\mathsf{MSD}}$ on the right side of the first two recipes with $\underline{\mathsf{MIF}}$ or MS. An example of a derivation is the following recipe for creating an intermediate flask out of a complex Mix to Same Destination action and basic Mix Solution action.

$$\underline{\mathsf{MIF}}[s_1, d_2] \rightarrow \mathsf{MSD}[s_1, d_1], \mathsf{MS}[d_1, d_2] \quad (1)$$

These recipes may be combined to describe the different solution strategies by which students solve problems in VirtualLabs (e.g., capturing students mixing all possible substance pairs versus mixing all four pairs together).

A set of nodes N *fulfills* a recipe R if there exists a one-to-one matching between the constituent actions in R and their parameters to nodes in N. For example, the nodes $\mathsf{MS}_3[s = 5, d = 4]$ and $\mathsf{MS}_5[s = 4, d = 3]$ fulfill the Mixing via an Intermediate Flask recipe shown in Equation 1.

3.2 Planning

Planning is the process by which students use recipes to compose basic and complex actions towards completing tasks using VirtualLabs . Formally, a *plan* is an ordered set of basic and complex actions, such that each complex action is decomposed into sub-actions that fulfill a recipe for the complex action. Each time a recipe for a complex action is fulfilled in a plan, there is an edge from the complex action to its sub-actions, representing the recipe constituents.

Figure 3 shows part of a plan describing part of a student's interaction when solving the Oracle problem. The leaves of the trees are the actions from the student's log, and are labeled by their order of appearance in the log. For example, the node labeled with the complex action $\underline{\mathsf{MSD}}[s = 1 + 5, d = 3]$ includes the activities for pouring two solutions from flask ID 1 and ID 5 to flask ID 3. The pour from flask ID 5 to 3 is an intermediate flask pour ($\underline{\mathsf{MIF}}[s = 5, d = 3]$) from flask ID 5 to ID 3 via flask ID 4. The root of the plan represents te complex action of pouring three substances from flasks ID 1, 5 and 6 to flask ID 3.

In a plan, the constituent sub-actions of complex actions may interleave with other actions. This way, the plan combines the free-order nature of VirtualLabs recipes with the exploratory nature of students' learning strategies. Formally, we say that two ordered complex actions *interleave* if at least one of the sub-actions of the first action occurs after some sub-action of the second action. For

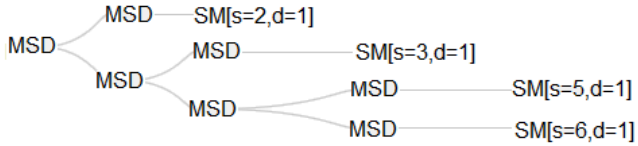

Figure 4: Example of an Explanation containing a Single Plan

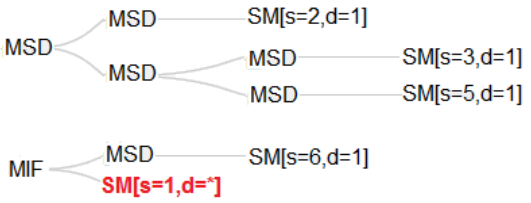

Figure 5: Example of an Explanation containing Two Plans, one of which has an open frontier

example, the nodes $\mathsf{MS}_3[s=5,d=4]$ and $\mathsf{MS}_5[s=4,d=3]$ and $\mathsf{MS}_2[s=6,d=8]$ and $\mathsf{MS}_4[s=8,d=3]$ both fulfill the Mixing via an Intermediate Flask recipe shown in Equation 1, but they are interleaved in the log. This interleaving quality makes the plan recognition task even more challenging.

4. ONLINE PLAN RECOGNITION

In this section we address the problem of on-line recognition in which agents' plans need to be inferred in real-time during execution. On-line recognition is essential for settings in which it is necessary to generate interventions to users. In ELEs, such intervention can provide feedback to students about their progress, alerting them to recurring mistakes or giving them hints about next steps during their exploration.

The fact that the interaction sequence is not known in advance requires to maintain the set of all plans that can explain the observations, including leaving place-holders for actions in the plan that relate to unseen future activities. Following Geib and Goldman [12], we define an *explanation* of actions O at time t a set of plans, such that there is an injective mapping from each action in O to a leaf in one of the plan instances. Each plan in an explanation describes a non-overlapping subset of the actions O. Some leaves in an explanation may not be included in O, and describe actions that are expected to appear in the future. These leaves are called the *open frontier* of the plan.

To illustrate, consider the recipes for VirtualLabs and the following explanations: Figure 4 shows a possible explanation for the observation sequence $SM[s=2,d=1], SM[s=3,d=1], SM[s=5,d=1], SM[s=6,d=1]$ in which all of the actions are constituents of the complex action MSD.[1] The explanation consists of a single plan.

Figure 5 shows a possible explanation for the same observation sequence, but in this case, the explanation consists of two plans. Here, the bold action $SM[s=1,d=*]$ represents a future (unseen) observation and is in the plan frontier. If the fifth observation turns out to be an SM action with $s=1$ (the parameter d does not hold any constraints), then the algorithm will incrementally combine this observation into the explanation. Otherwise, a third plan instance will be added to the explanation that matches the new observation, leaving $SM[s=1,d=*]$ in (and possibly adding new actions to) the plan frontier. We note that the plan frontier may also include complex actions, allowing to reason about future higher-level activities for which none of the constituents have been observed. The fact that the algorithm needs to maintain explanations for unseen observations is a significant computational challenge, as the possible number of explanations grows exponentially with the number of observations.

[1]For expository purposes we have omitted the parameters from nodes above the leaves.

5. CRADLE AND PHATT

The purpose of this section is to describe the state-of-the art in online plan recognition approach called PHATT, and our proposed extension to this approach for recognizing students' activities in ELEs.

We define the on-line plan recognition as follows: Given a set of observation at time t, output a set of explanations such that each explanation in the set can be used to derive the observations. PHATT is a top-down probabilistic algorithm that incrementally builds the set of possible explanations for explaining an observation sequence. PHATT works as follows: For each observation o^{t+1}, it takes the set of the possible explanations for the previous observations O^t, and tries to incorporate the new observation into each of the explanations in the set. This can be done either by integrating the new observation into one of the existing plans of the explanation, or by adding the observation as the first step of a new plan that will be added to the forest of plans in the explanation.

5.1 Using Filters

We now describe the basis for our proposed extension to PHATT, which is constraining the space of possible explanations in a way that reflect students' use of educational software. Our approach is called CRADLE (Cumulative Recognition of Activities and Decreasing Load of Explanations).[2]

Cradle extents the PHATT algorithm by constraining the space of possible explanations. We designed several "filters" that reduce the size of the explanation set in a way that reflects the intended use of plan recognition in ELEs. Specifically, the filters aim to produce complete, parsimonious and coherent explanations of students' interactions that can be easily understood by teachers and education researchers. We detail these filters below:

Explanation size This filter prefers explanations with smaller number of plans. Specifically, we discard explanations in which the number of plans is larger than a pre-computed threshold (the average number of plans per explanation).

Aging This filter prefers explanations in which successive observations extend existing sub-plans in the explanation rather than generate new plans. We discard explanations in which observations have not extended an existing plan for a given number of iterations.

[2]Also, cradle is the name of the mechanical contrivance used in placer mining, consisting of a box on rockers and moved by hand, used for washing out the gold-bearing soil, leaving only nuggets of gold.

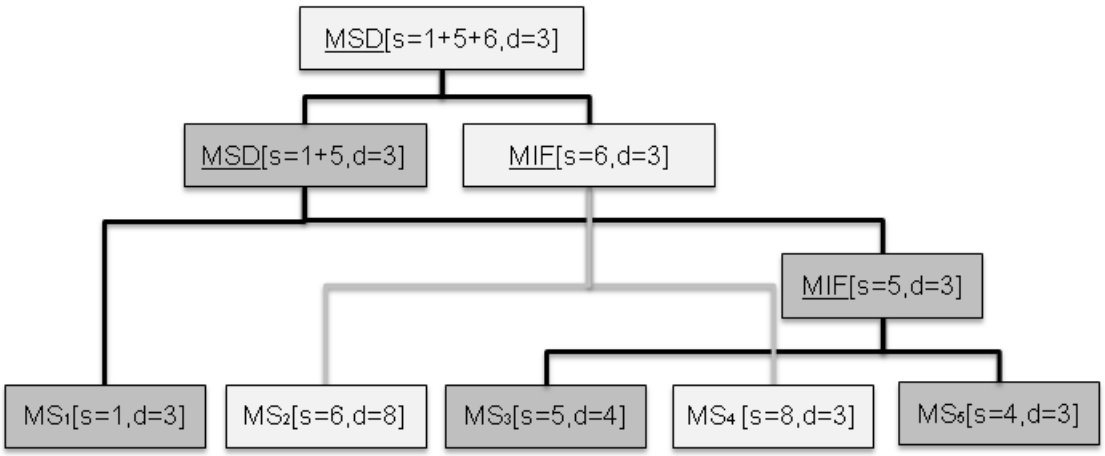

Figure 3: A partial plan for a student's log

Frontier size This filter prefers explanations which makes fewer commitments about future observations. It measure the amount of actions in the frontier that exist in each explanation, and discard explanations where this amount is above the average.

Probability This filter prefers explanations with a higher likelihood. It discards explanations whose probability of generating the observation sequence is lower than the average probability of the other explanations.

5.2 Augmenting PHATT

Figure 6 describes how CRADLE extends PHATT using the following methods, which we outline in some level of abstraction.

- Expand. Given a set of explanations that derive O^t, it is given a new observation o^{t+1}, this method creates all possible subplans in which o^{t+1} is a leaf, and tries to combine each of these subplans in all possible ways to each explanation. Each such subplan can be combined in two ways: (1) combineInExistingTrees - if the root of the subplan matches one of the plan frontier items, it replaces the frontier item with the subplan (replacing the placeholder with a concrete observation) or (2) extendWithANewTree - if the root of the subplan matches a possible goal, it is adding the subplan as the top levels of a new plan in the explanation's forest of plans.
- Filter. This function takes a set of explanations, calculates the average age, frontier size and amount of trees per explanation and filtered away all explanations with values above average. This means it prefers explanations with small frontier (less future expected observations), small age (observations continue existing plans instead of creating new ones) and small amount of trees (observations related to the same plan rather than describe different plans).
- Main. This is the main function of the new recognition process. It is made out of the two previous described stages - Extend and Filter - performed alternatly for each new observation encountered.

6. EMPIRICAL METHODOLOGY

The purpose of this section is to evaluate CRADLE to PHATT algorithm for real-world data sets of students' interactions with VirtualLabs . The PHATT approach is representative of an array of algorithms in the literature for performing on-line plan recognition by maintaining sets of observations (see for example the ELEXIR and YAPPR algorithm [13, 15]) and would behave similarly on our ELE data sets.

Specifically, we sampled 16 logs of students' interactions who solved two problems. The first was the Oracle problem described earlier. The second problem was called "Unknown Acid" and required students to determine the concentration level of an unknown solution. The length of the logs were chosen to have a wide range, between 4 to 152 actions.

6.1 Completeness and Run-time

The number of explanations maintained by the PHATT approach grows exponentially in the number of observations. It can be shown that for n observations and a set g of possible extensions for an explanation, the number of possible explanations is bounded by $n * |g|^n$. To illustrate, a 4 observation log outputted 142 different explanations, and a log of 12 observations generated more than 10,000 explanations. Most of these explanations included an abundance of plan instances with extremely large frontiers, clearly not the most coherent descriptions of the students' work.

Figure 7 shows the performance obtained using PHATT, augmentation of PHATT with single filter, and CRADLE. The x-axis in the figure corresponds to ranges of different log sizes. The y-axis determines the success ratio by measuring whether the algorithm was able to terminate and produce the explanations describing the student's activities within an upper bound of two hours of CPU time. As shown by the figure, PHATT was not able to terminate on logs

```
1: function EXPAND(o, Exps)  ▷ o: a new observation, Exps is the set of all explanations until o
2:     newExps = []
3:     for all explanation e ∈ Exps do
4:         newExps += e.combineInExistingTrees(o)
5:         newExps += e.extendWithANewTree(o)
6:     end for
       return newExps
7: end function

8: function FILTER(Exps)  ▷ Exps is the set of all explanations collected so far
9:     filteredExps = []
10:    for all explanation e ∈ Exps do
11:        if e.age ≤ averageAge & e.frontierSize ≤ averageFrontierSize & e.trees ≤ averageTrees then
12:            filteredExps += e
13:        end if
14:    end for
       return filteredExps
15: end function

16: function MAIN(Obs)  ▷ Obs is the set of all observations
17:    tempExps = [⟨emptyExp⟩]  ▷ Only one explanation - the empty explanation
18:    for all observation o in Obs do
19:        allExps = Expand(o, tempExps)
20:        filteredExps = Filter(allExps)
21:        tempExps = filteredExps
22:    end forreturn tempExps
23: end function
```

Figure 6: Main functions of the CRADLE algorithm

over 4 actions within this designated time frame. In contrast, CRADLE was able to significantly increase the performance of PHATT algorithm by applying the filters. Specifically, applying the different filters independently allowed to improve the success ratio for some of the logs, with the highest improvement attributed to the CRADLE approach which applied the age, frontier size and explanation size filters. Interestingly, there was not a single filter method that outperformed all of the other methods for all log size.

Next, we compare the run-time of CRADLE and PHATT on fragments of logs for which PHATT was able to terminate. Figure 8 shows the average run-time on each size of log, measured in seconds, presented in a logarithmic scale. It can be seen that the average run-time of CRADLE is exponentially better than the average run-time of PHATT for the aforementioned logs.

6.2 Domain Expert Evaluation

In this section, we show that although the CRADLE approach reduces the number of possible explanations that is maintained by the plan recognition algorithm, it does not hinder the correctness of the algorithm. To this end, we sampled 20 logs of the Oracle problem and presented the output of the PHATT and CRADLE approach to a domain expert.[3]. We ran the cut logs on PHATT and CRADLE and collected the outputted set of explanations for each log. For

[3]Logs of length greater than 6 actions were cut arbitrarily at 6,7,9,10 and 11 actions, in order to simulate incomplete interaction sequences and to allow PHATT to terminate on these logs in reasonable time.

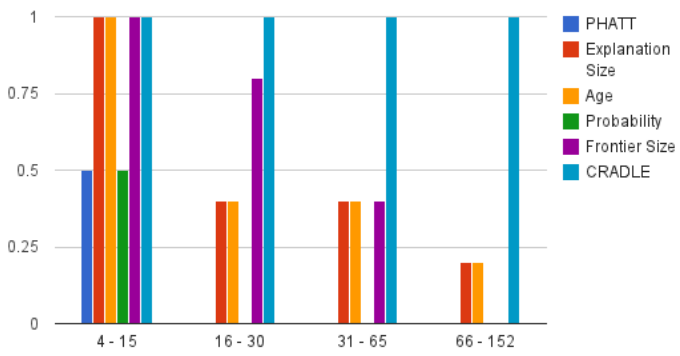

Figure 7: Performance of PHATT, CRADLE and Single Filter Variants on Various Log Sizes

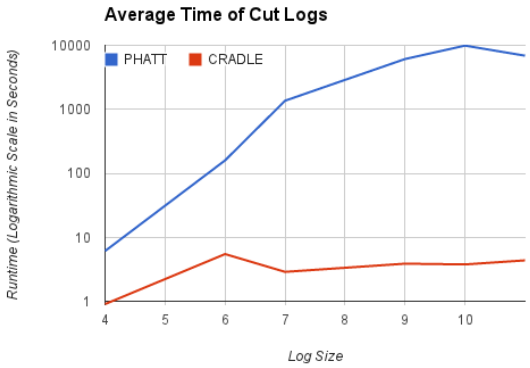

Figure 8: Runtime of PHATT and CRADLE

each of the approaches, we chose to present the domain expert with the explanation that did not include an open frontier (that is, the explanations provided a complete description of the activities of the student). If there was no explanations without an open frontier, we chose the most likely explanation as measured by its probability.

Out of the 20 examined logs, in 9 logs PHATT and CRADLE's explanations were the same (though CRADLE was able to output the solution exponentially faster). We presented the explanations for which CRADLE and PHATT different to a domain expert, who is one of the developers of the VirtualLabs software, who compared between the two explanations. We did not label the explanations with the algorithm that generated them. In 8 out of these 11 logs, the domain expert preferred explanations which were presented by CRADLE over the explanations of PHATT. In one case, the domain expert said none of the explanations describe the activities of the student correctly. To illustrate, Figure 4 shows the explanation outputted by CRADLE for a particular log which included a mix of 4 substances into a single flas. Figure 9 shows the PHATT explanation for that same log, using two plans to explains the observation sequence. In this case, the domain expert preferred the CRADLE explanation, which explained the observation sequence using a single plan.

7. DISCUSSION AND FUTURE WORK

Our results show that the CRADLE approach was able to extend the state-of-the-art (PHATT algorithm) towards successfully rec-

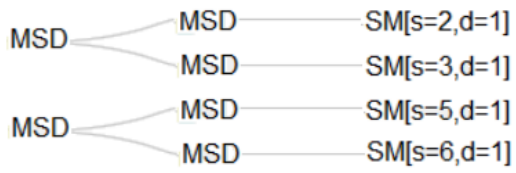

Figure 9: Example of PHATT explanation

ognizing students' activities in an ELE for chemistry education. We showed that CRADLE was able to produce better explanations than PHATT, and with exponentially faster running time. Specifically, the ouputted explanations of CRADLE were as good as or better than PHATT in 18 out of the 20 logs that we sampled, giving CRADLE a success rate of 90% at an exponentially lower runtime. The paper demonstrate that on-line plan recognition in ELEs is a challening computational problem, and show the efficacy of the CRADLE approach in addresssing these problems by reducing the number of explanations maintaind by the algorithms in an intelligent way. We are currently pursuing work with CRADLE in several directions. First, we are evaluating the scalability of the CRADLE approach by evaluating it with different ELEs for statistics education, as well as simulated data that simulates users' interactions with software. This ELE is significantly different than VirtualLabs in that student's interactions are more likely to engage in trial-and-error, which we predict will further challenge the recognition problem. Second, we are developing a formal langauge that explains students' activities with ELEs that will help us construct more accurate grammars for the recognition algorithms.

8. ACKNOWLEDGEMENTS

This work was supported in part by Israeli Science Foundation Grant no. 1276/12.

9. REFERENCES

[1] S. Amershi and C. Conati. Automatic recognition of learner groups in exploratory learning environments. In *Intelligent Tutoring Systems (ITS)*, 2006.

[2] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier. Cognitive tutors: Lessons learned. *The Journal of Learning Sciences*, 4(2):167–207, 1995.

[3] D. Avrahami-Zilberbrand, G. Kaminka, and H. Zarosim. Fast and complete symbolic plan recognition: Allowing for duration, interleaved execution, and lossy observations. In *Proc. of the AAAI Workshop on Modeling Others from Observations, MOO*, 2005.

[4] N. Blaylock and J. F. Allen. Statistical goal parameter recognition. In *ICAPS*, volume 4, pages 297–304, 2004.

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

[6] M. Cocea, S. Gutierrez-Santos, and G. Magoulas. S.: The challenge of intelligent support in exploratory learning environments: A study of the scenarios. In *Proceedings of the 1st International Workshop in Intelligent Support for Exploratory Environments on European Conference on Technology Enhanced Learning*, 2008.

[7] C. Conati, A. Gertner, and K. VanLehn. Using Bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adapted Interaction*, 12(4):371–417, 2002.

[8] C. Conati, A. Gertner, and K. Vanlehn. Using bayesian networks to manage uncertainty in student modeling. *User modeling and user-adapted interaction*, 12(4):371–417, 2002.

[9] A. Corebette, M. McLaughlin, and K. C. Scarpinatto. Modeling student knowledge: Cognitive tutors in high school and college. *User Modeling and User-Adapted Interaction*, 10:81—108, 2000.

[10] Y. Gal, S. Reddy, S. Shieber, A. Rubin, and B. Grosz. Plan recognition in exploratory domains. *Artificial Intelligence*, 176(1):2270 – 2290, 2012.

[11] Y. Gal, E. Yamangil, A. Rubin, S. M. Shieber, and B. J. Grosz. Towards collaborative intelligent tutors: Automated recognition of users' strategies. In *Intelligent Tutoring Systems (ITS)*, 2008.

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

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

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

[15] C. W. Geib, J. Maraist, and R. P. Goldman. A new probabilistic plan recognition algorithm based on string rewriting. In *ICAPS*, pages 91–98, 2008.

[16] B. Grosz and S. Kraus. The evolution of sharedplans. *Foundations and Theories of Rational Agency*, pages 227–262, 1999.

[17] F. Kabanza, P. Bellefeuille, F. Bisson, A. R. Benaskeur, and H. Irandoust. Opponent behaviour recognition for real-time strategy games. In *Plan, Activity, and Intent Recognition*, 2010.

[18] 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.

[19] D. V. Pynadath and M. P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 507–514. Morgan Kaufmann Publishers Inc., 2000.

[20] K. VanLehn, C. Lynch, K. Schulze, J. A. Shapiro, R. H. Shelby, L. Taylor, D. J. Treacy, A. Weinstein, and M. C. Wintersgill. The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence and Education*, 15(3), 2005.

[21] M. Vee, B. Meyer, and K. Mannock. Understanding novice errors and error paths in object-oriented programming through log analysis. In *Proceedings of Workshop on Educational Data Mining at ITS*, pages 13–20, 2006.

[22] D. Yaron, M. Karabinos, D. Lange, J. Greeno, and G. Leinhardt. The ChemCollective–Virtual Labs for Introductory Chemistry Courses. *Science*, 328(5978):584, 2010.