Plan Recognition in Virtual Laboratories

Ofra Amir and **Ya'akov (Kobi) Gal**

Department of Information Systems Engineering

Faculty of Engineering Sciences

Ben-Gurion University of the Negev, Israel

{ofraam,kobig@bgu.ac.il}

Abstract

This paper presents a plan recognition algorithm for inferring student behavior using virtual science laboratories. The algorithm extends existing plan recognition technology and was integrated with an existing educational application for chemistry. Automatic recognition of students' activities in virtual laboratories can provide important information to teachers as well as serve as the basis for intelligent tutoring. Student use of virtual laboratories presents several challenges: Students may repeat activities indefinitely, interleave between activities, and engage in exploratory behavior using trial-and-error. The plan recognition algorithm uses a recursive grammar that heuristically generates plans on the fly, taking into account chemical reactions and effects to determine students' intended high-level actions. The algorithm was evaluated empirically on data obtained from college students using virtual laboratory software for teaching chemistry. Results show that the algorithm was able to (1) infer the plans used by students to construct their models; (2) recognize such key processes as titration and dilution when they occurred in students' work; (3) identify partial solutions; (4) isolate sequences of actions that were part of a single error.

1 Introduction

This paper reports on the development and evaluation of algorithms for recognizing the plans of students interacting with pedagogical systems for science education. To support the types of exploratory activity that facilitate scientific learning, these systems typically are flexible and open-ended. Examples of such learning environments include the Geometer's Sketchpad for teaching geometry [Finzer and Bennett, 1995], the TinkerPlots system for teaching statistics [Konold and Miller, 2004] and the ChemCollective Virtual Labs system for introductory chemistry [Yaron *et al.*, 2010], hence referred to as "VirtualLabs".

Educational software is generally used in classes too large for teachers to monitor all students and provide assistance exactly when a student needs it [Gal *et al.*, 2008]. Such software

is also becoming increasingly prevalent in developing countries where access to teachers and other educational resources is limited [Pawar *et al.*, 2007]. Integrating into these systems capabilities for recognizing students' plans from their actions would enable them to provide richer experiences for students and more useful information to teachers.

The focus of this paper is on plan recognition algorithms for pedagogical software that allow students to design and carry out their own experiments, simulating the types of scientific inquiry that is experienced in a physical laboratory. There are several aspects to students' interactions that make plan recognition in these "virtual laboratories" particularly challenging. First, students can engage in exploratory activities involving trial-and-error, such as searching for the right pair of chemicals to combine in order to achieve a desired reaction. Second, students can repeat similar actions indefinitely in pursuit of a single goal, such as adding varying amounts of a reagent to a solution until a desired outcome is achieved. Third, students can interleave between activities, such as preparing a solution for a new experiment while waiting for the results of a current experiment. Explicitly representing all possible combinations of these activities is computationally infeasible.

The paper presents an efficient algorithm for intelligently recognizing students' problem-solving strategies that addresses these challenges. The algorithm infers students' plans based on their complete interaction histories with pedagogical software, outputting a hierarchical plan that explains the student's problem-solving strategy. It uses a recursive grammar to generate, on the fly, plan fragments for key chemical processes in the lab, such as dilution and titration. The grammar constrains which processes may occur, and the order in which they occur, as well as accounting for the chemical reactions and effects that are the result of the processes. The recognition algorithm expands tasks from the grammar using a heuristic that chooses (possibly non contiguous) actions from students' interaction sequences.

We evaluated this algorithm using real data obtained from students using the VirtualLabs system mentioned above to solve three representative problems used in introductory chemistry courses. Despite its incompleteness, the algorithm was able to correctly infer students' plans in all of the instances. In particular, it was able to distinguish activities that were salient to the solution from those representing trial-and-

error approaches. It was also able to identify partial solutions in cases where students failed to solve the complete problem, as well as capture interleaving plans. These results demonstrate the efficacy of integrating AI techniques with existing pedagogical software. Our techniques can inform the way intelligent tutors infer students' activities in pedagogical software as well as to provide support for teachers.

This work augments recent approaches to plan recognition that relax the traditional assumptions of a goal directed agent that is pursuing a single encompassing plan: Geib and Goldman [2009] proposed a probabilistic model of plan recognition that is able to identify interleaving plans. Conati et al. [2002] used Bayesian networks to model students' interactions with intelligent tutors that recognizes interleaving actions among sub-plans. Other works [Sidner, 1985; Geib and Steedman, 2007] proposed representations and algorithms for capturing temporal relationships among actions that are derived from the analogy between plan recognition and grammar recognition. None of these approaches consider exploratory actions and mistakes which are endemic to people's interactions with pedagogical software. Albrecht et al. [1998] suggested a probabilistic approach to infer players' goals as well as their future actions from observation sequences. They are able to capture agents' mistakes, but infer the likelihood of a single goal or action, rather than recognizing a complete plan representing the entire action sequence. Reddy et al. [2009] and Quilici et al. [1998] proposed complete algorithms for implementing plan recognition as a constraint satisfaction problem. Gal et al. [2008] proposed an algorithm for recognizing students' activities in pedagogical software for statistics education. We extend this work in both representation (to allow recursive grammars and pre- and post-action effects) and method (to allow for indefinite repetition of activities in students' problem solving).

1.1 The VirtualLabs Domain

We will use the "dilution problem", posed to students that use VirtualLabs in an introductory chemistry course, as a running example to demonstrate our approach: *Your objective is to prepare a solution containing 800 milliliters (ml) or more of HNO_3 with a desired concentration of 7 M. You are allowed a maximal deviation of 0.005 M in either direction.*

We present a possible solution for this problem that is adapted from one of the student interactions with VirtualLabs that is used in our empirical evaluation. To solve this problem the student repeatedly mixed varying quantities of HNO_3 with H_2O until achieving the required concentration. Specifically, the student began by pouring 100 ml of an HNO_3 solution with a concentration of 15.4 M to a 100 ml intermediate flask, and transferred the content of the intermediate flask to a destination flask.[1] This activity was repeated four times, resulting in 400 ml of HNO_3 in the destination flask. The student proceeded to dilute this solution by mixing it with 510 ml of H_2O. This activity was carried out in two steps, one adding 10 ml of H_2O (using an intermediate flask of 10 ml) and another adding 500 ml of H_2O (using an intermedi-

---

[1]Intermediate flasks are commonly used in VirtualLabs to help measure solutions accurately, as in a physical laboratory.
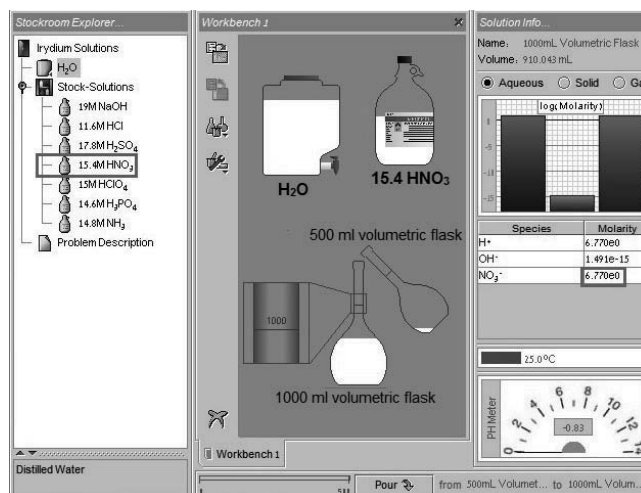

Figure 1: Snapshot of Interaction in Virtual Labs

ate flask of 500 ml). At this point the molarity of HNO_3 in the destination flask was too low (6.77 M), indicating that too much H_2O had been poured. To raise the concentration to the desired level, the student began to pour small amounts of HNO_3 to the destination flask using an intermediate 10 ml flask, while checking the concentration level of the resulting compound. The student first poured 10 ml of HNO_3, then poured another 10 ml of HNO_3, and finally added 5 ml of HNO_3 to the destination flask, which achieved the desired concentration of 7 M.

Figure 1 shows a snapshot taken right after the student added 510 ml of H_2O to the destination flask. The panel on the left shows a "stockroom" of chemicals which can be customized for different activities. One of the flasks, labeled "15.4M HNO_3" (outlined in the figure) contains an HNO_3 solution with a concentration of 15.4 M. The middle panel shows the "workbench" of the student, used to carry out activities in the laboratory. This panel shows the flask containing HNO_3 with a concentration of 15.4 M, the H_2O flask, and the destination flask (a 1,000 ml volumetric flask). It also shows one of the intermediate flasks used by the student (a 500 ml volumetric flask). The "Solution Information" panel on the right shows the volume and concentration of selected compounds. It shows that the concentration level of HNO_3 in the destination flask is 6.77 M (outlined in the figure).

This interaction highlights several aspects endemic to scientific inquiry in physical laboratories which are supported by the VirtualLabs software. First, the concept of titration, that of repeatedly adding a measured compound to a solution until a desired result is achieved. This is apparent in the student repeatedly adding small quantities of HNO_3 to the destination flask. Second, the interleaving of actions that relate to different activities. This is apparent in the student beginning to pour HNO_3 to the destination flask, then switching to pour H_2O, and then returning to pour more HNO_3. Lastly, performing exploratory actions and mistakes. This is apparent in the student adding too much H_2O to the destination flask, and proceeding to increase the concentration of the compound by

adding more HNO_3.

2 Actions, Recipes and Plans

In this section we present the building blocks for our plan recognition algorithm, which is based on a generative grammar that captures the experimental nature of students' activities in virtual laboratories. We use the term *basic actions* [Pollack, 1990] to define rudimentary operations that cannot be decomposed. These serve as the input to our plan recognition algorithm. *Complex actions* describe higher-level, more abstract activities that can be decomposed into sub-actions, which can be basic actions or complex actions themselves. In our example, basic actions may consist of taking out a solution from the stockroom or pouring 10 ml of H_2O to an intermediate flask, while complex actions may consist of solving the dilution problem, or mixing together H_2O and HNO_3 several times. A *recipe* for a complex action specifies the sequence of operations required for fulfilling the complex action. Formally, a recipe is a set of sub-actions and constraints such that performing those sub-actions under those constraints constitutes completing the action. The set of constraints is used to (1) specify required values for action parameters; (2) enforce relationships among parameters of (sub-)actions, such as chronological order; and (3) bind the parameter values of a complex action to the value of the parameters in its constituent sub-actions. Our choice of nomenclature was based on philosophical declaratives of the foundational planning literature [Bratman *et al.*, 1988] and their use in plan recognition [Lochbaum, 1998].

Figure 2(a) presents a recipe for the complex action of Solving the Dilution Problem (SDP) composed of two complex sub-actions for Mixing Solution Components (MSC), namely H_2O and HNO_3. In our notation, complex actions are underlined, while basic actions are not. Actions in VirtualLabs are associated with identifiers that bind to recipe parameters. For example, the parameters of the action $MSC[s\_id_1, d\_id_1, sc_1 = H_2O, vol_1]$ of pouring H_2O in Figure 2(a) identify the source flask (s_id) from which a source chemical (sc) is poured, the destination flask (d_id), and the volume of the solution that was poured (vol). The constraints for this recipe require that the destination flask identifier for both MSC actions is the same ($d\_id_1 = d\_id_2$) in addition to specifying the type of chemicals in the mix ($sc_1 = H_2O$ and $sc_2 = HNO_3$).

Recipes may be recursive, capturing activities that can repeat indefinitely, as in titration. This is exemplified in the recipe shown in Figure 2(b) for the complex action (MSC) of adding a solution component of volume vol from flask s_id_1 to flask d_id_1. The constituent actions of this recipe decompose the MSC action into two separate MSC actions for adding vol_1 and vol_2 of the solution using the same source and destination flask. This recipe effectively clusters together repetitive activities. Also shown is the "base-case" recipe for MSC that includes a Mix Solution (MS) basic action.

Figure 2(c) presents another recipe for an MSC complex action which decomposes into a constituent sub-action for Mixing the Solution using an Intermediate flask (MSI).[2]

_____
[2]For brevity, we omit the recipes for the MSI action.

$$\underline{SDP}[s\_id_1, vol_1, s\_id_2, vol_2, d\_id_1] \rightarrow$$
$$\underline{MSC}[s\_id_1, d\_id_1, sc_1 = H_2O, vol_1],$$
$$\underline{MSC}[s\_id_2, d\_id_2, sc_2 = HNO_3, vol_2]$$

$$d\_id_1 = d\_id_2$$
(a)

$$\underline{MSC}[s\_id_1, d\_id_1, sc_1, vol = vol_1 + vol_2] \rightarrow$$
$$\underline{MSC}[s\_id_1, d\_id_1, sc_1, vol_1],$$
$$\underline{MSC}[s\_id_2, d\_id_2, sc_2, vol_2]$$

$$s\_id_1 = s\_id_2, d\_id_1 = d\_id_2, sc_1 = sc_2$$

$$\underline{MSC}[s\_id, d\_id, sc, vol] \rightarrow MS[s\_id, d\_id, sc, vol]$$

(b)

$$\underline{MSC}[s\_id, d\_id, sc, vol] \rightarrow \underline{MSI}[s\_id, d\_id, i\_id, sc, vol]$$

(c)

Figure 2: Recipes for (a) solving the dilution problem; (b) repetition of activities; (c) using intermediate flasks.

We say that a recipe for a complex action is *fulfilled* by a set of sub-actions if there is a one-to-one correspondence from each of the sub-actions to one of the recipe's constituents that meets the recipe constraints. For example, in the student's interaction described in Section 2, the complex sub-actions for mixing H_2O with HNO_3 fulfill the recipe for the complex action SDP of solving the dilution problem. These actions are labeled "1, 2" and "14" in Figure 3(a).

A *plan* is a set of complex and basic actions such that each complex action is decomposed into sub-actions that fulfill a recipe for the complex action. A hierarchical presentation of a (partial) plan used by the student to solve the dilution problem is shown in Figure 3(a). The hierarchy emanating from the root node SDP (the action labeled "1") shows that the student was able to solve the dilution problem by mixing together 425 ml of HNO_3 from flask ID 1 (the action labeled "2") with 510 ml of H_2O from flask ID 4 (the action labeled "14") in destination flask ID 2. These actions further decompose to their respective constituent actions. For example, the path in bold, from left to right, shows part of the plan for the complex action of pouring 425 ml of HNO_3 from flask ID 1 to flask ID 2 (the action labeled "2"). Here, the student poured 25 ml of HNO_3 from flask ID 1 to flask ID 2 (the action labeled "3") using intermediate flask ID 3 (the action labeled "4"). The action labeled "4" is decomposed to the two sub-actions for pouring the solution from flask ID 1 to intermediate flask ID 3, and pouring from flask ID 3 to the destination flask ID 2 (actions labeled "5" and "6"). For brevity, we do not expand the complex actions in Figure 3(a) down to the leaves.

Figure 3(b) describes the student's use of titration. This plan expands the action of pouring 25 ml from flask ID 1 to flask ID 3 (action labeled "5") down to the basic-level actions corresponding to the student's interaction with the software (the three MS actions at the leaves). The constituents of this action consisted of two separate pours from flask ID 1 to flask ID 3, one pouring 20 ml (action labeled "7") and the other

MSC(sid=1, did=2, sc=HNO$_3$, vol=425ml)
2

SDP(sid$_1$=1,vol$_1$=425m, sid$_2$=4,vol$_2$=510,did=2)
1

MSC(sid=4, did=2, sc=H$_2$O, vol=510ml)
14

MSC(sid=1, did=2, sc=HNO$_3$, vol=400ml)

MSC(sid=4, did=2, sc=H$_2$O, vol=10ml)

MSC(sid=4, did=2, sc=H$_2$O, vol=500ml)

MSC(sid=1, did=2, sc=HNO$_3$, vol=25ml)
3

MSI(sid=1, did=2, iid=6, sc=HNO$_3$,vol=400ml)

MSI(sid=4, did=2, iid=3, sc=H$_2$O, vol=10ml)

MSI(sid=4, did=2, iid=5, sc=H$_2$O, vol=500ml)

MSI(sid=1, did=2, iid=3, sc=HNO$_3$, vol=25ml)
4

MSC(sid=1, did=6, sc=HNO$_3$, vol=400ml)

MSC(sid=6, did=2, sc=HNO$_3$, vol=400ml)

MSC(sid=4, did=3, sc=H$_2$O, vol=10ml)

MSC(sid=3, did=2, sc=H$_2$O, vol=10ml)

MSC(sid=4, did=5, sc=H$_2$O, vol=500ml)

MSC(sid=5, did=2, sc=H$_2$O, vol=500ml)

MSC(sid=1, did=3, sc = HNO$_3$, vol=25ml)
5

MSC (sid=3, did=2, sc=HNO$_3$, vol=25ml)
6

ID	Type	Contained solution
1	2500ml Bottle	15.4 HNO3
2	1000ml volumetric flask	Empty
3	10ml Graduated Cylinder	Empty
4	3L Bottle	H2O
5	500ml volumetric flask	Empty
6	100ml volumetric flask	Empty

Index for flask identifiers

(a)

MSC(sid=1, did=3, sc=HNO$_3$, vol=25ml)
5

MSC(sid=1, did=3, sc=HNO$_3$, vol=20ml)
7

MSC(sid=1, did=3, sc=HNO$_3$, vol=10ml)
8

MSC(sid=1, did=3, sc=HNO$_3$, vol=10ml)
9

MSC(sid=1, did=3, sc=HNO$_3$, vol=5ml)
10

MS (sid=1, did=3, sc=HNO$_3$, vol=10ml)
11

MS (sid=1, did=3 , sc=HNO$_3$, vol=10ml)
12

MS (sid=1, did=3, sc=HNO$_3$, vol=5ml)
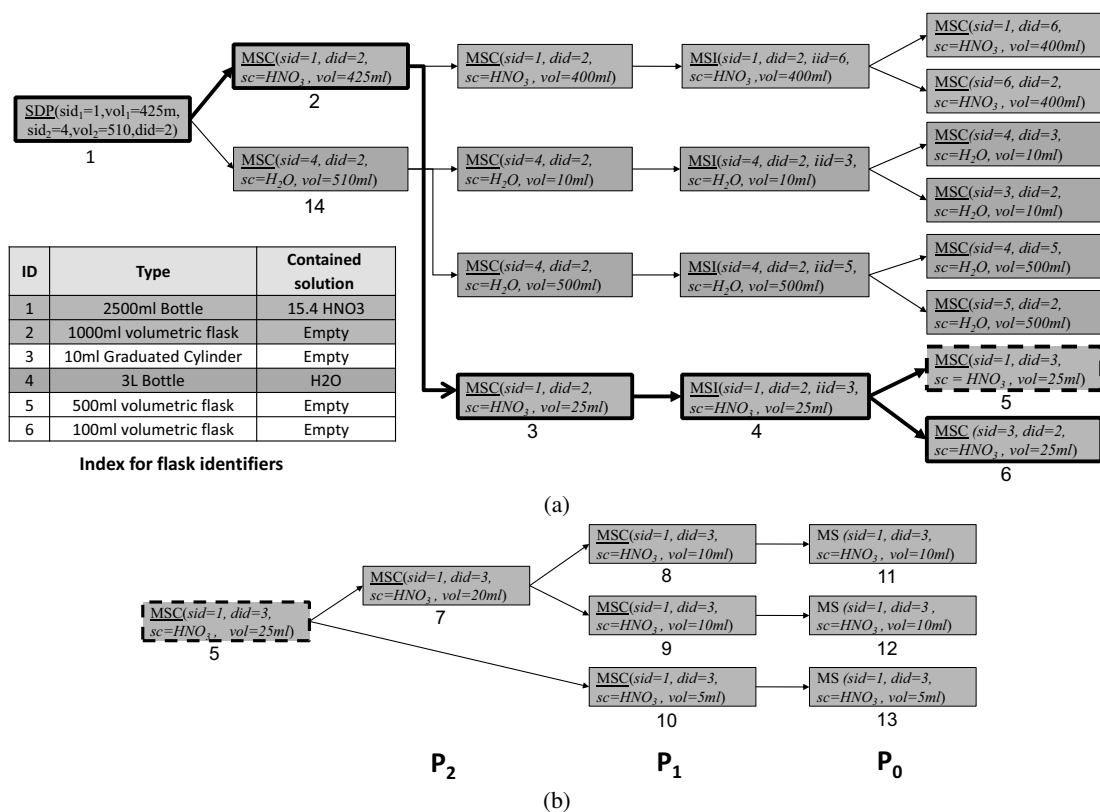13

P$_2$ P$_1$ P$_0$

(b)

Figure 3: (a) a partial plan for the dilution problem; (b) a plan for the MSC complex action (labeled "5", dashed outline).

pouring 5 ml (action labeled "10"). The action labeled "10" was further decomposed to the basic action of adding 5 ml of HNO_3 to flask ID 3 (action labeled "13").

3   Plan Recognition

Students take diverse approaches to solving the dilution problem. They can perform an indefinite number of mixing actions, choose whether to use intermediate flasks and interleave activities. For example, Figure 3 shows that the constituent sub-actions of the action labeled "14" occurred in between the constituent sub-actions of the action labeled "2". This reflects that the student interleaved the actions for adding HNO_3 and H_2O. A brute-force approach involves non-deterministically finding all ways in which a complex action may be implemented in students' interaction sequences. Due to the exploratory and repetitive nature of students' activities in VirtualLabs , naively considering each of these possibilities is impossible.

The proposed algorithm shown in Figure 4 incrementally builds a plan which describes students' activities with VirtualLabs . BUILDPLAN(R, X) receives as input a finite action sequence representing a student's interaction, denoted X, and the set of recipes for the given problem, denoted R. At each step t, the algorithm maintains an ordered sequence of actions, denoted P_t and an open list OL. The algorithm iterates over the recipes in R (step 3) according to the following (partial) ordering criteria: if the complex action C$_2$ is a constituent sub-action for a recipe for a complex action C$_1$, then

recipes for action C$_2$ are considered before the recipes for action C$_1$.[3] The algorithm repeatedly searches for a match for each recipe R_c for action C in the open list by calling the function FINDMATCH(R_C, OL) (step 5). This function returns a set of actions $M_C \in OL$ such that M_C *fulfills* R_C. Actions are chosen from OL in any order such that they agree with the restrictions in R_C (allowing us to capture interleaving plans). If there exists no action set M_c that fulfills R_c, then the function returns $\emptyset$.

For each match M_C that fulfills R_C, BUILDPLAN performs the following: First, the values of the parameters in C are set based on the values of the parameters of the actions in M_C and the restrictions specified in the recipe R_C (step 7). This incorporates into C the reactions and effects arising from carrying out the constituent actions in R_C. Second, the action C is added to the action sequences in P_{t+1} and OL, in the position held by the latest action in M_C (step 8).[4] Adding the action to OL supports recursive recipes, in that it allows the action C itself to be part of the action set that fulfills R_C in the next iteration. Third, the action C in P_{t+1} is made a

---

[3]The recipe language allows for cycles, but in practice recipes cannot be applied indefinitely in VirtualLabs. An ordering over recipes can always be created (possibly by duplicating or renaming actions) that meets the constraint.

[4]This is done to preserve the temporal ordering of the actions in the open list, which facilitates checking temporal constraints when matching recipes to actions in the open list.

```
 1: procedure BUILDPLAN(R, X)
 2:     P_0 ← X
 3:     for R_C ∈ SORTRECIPES(R) do
 4:         P_{t+1}, OL ← P_t
 5:         M_C = FINDMATCH(R_C, OL)
 6:         while M_C ≠ ∅ do
 7:             BINDPARAMS(C, M_C, R_C)
 8:             Add C to OL and P_{t+1} positioned after last
      a ∈ M_C
 9:             for all a ∈ M_C do
10:                 Create a branch from C in P_{t+1} to a in P_t
11:             Remove M_C from OL and P_{t+1}
12:             M_C = FINDMATCH(R_C, OL)
```

Figure 4: Bottom-up plan recognition method

	Coffee	Oracle	Dilution
Run-Time (sec)	0.15	**1.31**	0.54
Log size	24.33	**110.33**	63
Plan size	34	**66**	39.75

Table 1: Performance measures for the recognition algorithm

parent of all of the actions in M_C in P_t (step 10). This creates the hierarchy between a complex action in P_{t+1} and its constituent actions in P_t. Finally, the actions in M_C are removed from both the open list OL and P_{t+1} (step 11). Removing the actions in M_C from the open list prevents the same actions from fulfilling more than one recipe. Once no more matches for R_C can be found, (i.e, FINDMATCH(R_C, OL) returns ∅), the BUILDPLAN algorithm proceeds to consider a new recipe, and terminates once all recipes have been considered.

We demonstrate this process using the plan in Figure 3(b) describing the student's use of titration. At step P_1, the MS basic action (labeled "11") was chosen to match the recipe for the complex MSC action (labeled "8") using the second recipe in Figure 2b. At step P_2, the MSC actions labeled "8, 9" were chosen to match the recipe for the MSC action labeled "7". We note that BUILDPLAN is capable of inferring multiple hierarchies, representing students' failed attempts to solve a problem, or exploratory activities that are exogenous to the actual solution path. Such behavior occurred in our empirical evaluation that is described in the next section.

The function FINDMATCH can be implemented in many ways. For this study, we used a standard depth-first search on the open list OL to find a match for R_C. This function is complete, in that for a given recipe R_C and OL, if there exists a match for R_C in OL, then FINDMATCH will find it. However, BUILDPLAN is a greedy algorithm. Once an action set M_C matches a recipe R_C, it does not backtrack and consider any of the actions in M_C for alternative recipes. Therefore it may fail to recognize a student's plan.

The complexity of BUILDPLAN is dominated by the complexity of the FINDMATCH algorithm, denoted C_{FM}. Let $|R|$ and $|X|$ be the number of recipes in R and the number of actions in the action sequence X, respectively. Then, BUILDPLAN calls FINDMATCH at most $|X|$ times per recipe, yielding an overall complexity of $O(|R| \cdot |X| \cdot C_{FM})$. Since FINDMATCH was implemented as a depth first search, its complexity is exponential in the size of the action sequence X.

4 Empirical Methodology

We evaluated the algorithm on real data consisting of students' interactions with the software. Our hypothesis was that the BUILDPLAN algorithm will in practice be able to recognize students' plans with VirtualLabs despite its incompleteness. We used three problems intended to teach different types of experimental and analytical techniques in chemistry, taken from the curriculum of introductory chemistry courses using VirtualLabs in the U.S. One of these was the dilution problem that was described in Section 2; the second, called the "oracle problem", required students to determine unknown reactions between four substances; the third, called the "coffee problem", required students to add the right amount of milk to cool a cup of coffee down to a desired temperature. We constructed a set of recipes for each of the problems. VirtualLabs automatically logs the interactions of its users. These logs constituted the basic actions that served as the input for the algorithm, together with the set of recipes for each problem.

The algorithm was evaluated by a domain expert who is a chemistry researcher and one of the developers of VirtualLabs. For each problem instance, the domain expert was given the plan(s) outputted by BUILDPLAN, as well as the student's log. We consider the inferred plan(s) to be "correct" if the domain expert agrees with the complex and basic actions at each level of the plan hierarchy that is outputted by the algorithm. If the student was able to complete the problem, the outputted plan(s) represent the student's solution process. Otherwise, the outputted plan(s) represent the students' failed attempts to solve the problem.

We ran the algorithm on ten problem instances (four instances of the dilution problem, and three instances of each coffee and oracle problem). Students' logs varied greatly in size, ranging from 20 actions to 187 actions, while the outputted plans ranged in depth from 3 to 14 levels.

The results revealed that all plans outputted by BUILDPLAN were deemed correct by the domain expert. Because of the burden required to manually verify the performance of the algorithm on each instance, only ten instances were evaluated by the domain expert. However, the perfect record of the algorithm on these instances speaks well for its overall performance. In particular, the algorithm was able to capture trial-and-error approaches as well as explorations and mistakes. For instance, one of the students performed three separate attempts to solve the dilution problem. The first two attempts resulted in a wrong molarity of the solution, and after each of these unsuccessful attempts the student started over using different flasks. The algorithm represented each of these three attempts in a separate plan hierarchy. This is an important capability, as it allows teachers to gain important insights regarding students' problem solving processes by reviewing their plans.

Table 1 summarizes the performance of the algorithm according to several measures: run time of the algorithm (in seconds) on a commodity dual-core computer; log size, rep-

resenting the size of the interaction history that serves as input to the algorithm; plan size, representing the number of nodes in the plan(s) outputted by the algorithm. All of the reported results were averaged over the different instances in each problem. As shown in the table, the longest time to infer students' plans occurred for interactions relating to the oracle problem (1.31 sec.), which also resulted in the largest plans (66 nodes). This problem is more exploratory in nature than the dilution and coffee problems, and students' solutions were characterized by longer interaction histories and a high degree of experimentation. Another factor contributing to the complexity of the oracle problem is the number of recursive recipes. There were 6 recursive recipes for this problem, more than the number of recursive recipes for the dilution problem (4) and the coffee problem (2).

5   Conclusion and Future Work

This paper proposed a computationally efficient plan recognition algorithm for integrating with pedagogical software in which students use virtual laboratories to design and conduct experiments. Students' problem-solving in virtual laboratories is characterized by exploration and trial-and-error, interleaving between activities and mistakes. We showed that the algorithm was successfully able to recognize students' plans when solving three separate problems using existing software for teaching chemistry concepts. It was able to recognize such key processes as titration and dilution when they occurred in students' work. This work is a necessary step towards a pedagogical agent that is truly collaborative, in the sense that it provides the right machine-generated support for its users. For teachers, this support consists of notification of students' performance both after and during class. Although our empirical evaluation uses one type of software, the techniques are general and can be used to support the analysis of students interactions for other types of virtual laboratories.

We are currently applying these results in several directions. First, we are developing methods for presenting plan recognition output to teachers in order to provide them with a broad and organized view of students' activities. Second, we are constructing recipes for recognizing the common types of mistakes students make when using pedagogical software, as well as using the recipe-based approach in different pedagogical software systems. In future work, we will use our algorithms as a basis for building intelligent tutors that will augment existing software tools for mathematics education.

Acknowledgments

References

[Albrecht *et al.*, 1998] D.W. Albrecht, I. Zukerman, and A.E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User modeling and user-adapted interaction*, 8(1):5–47, 1998.

[Bratman *et al.*, 1988] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational intelligence*, 4(3):349–355, 1988.

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

[Finzer and Bennett, 1995] B. Finzer and D. Bennett. From drawing to construction with the Geometer's Sketchpad. *Mathematics Teacher*, 88(5):428–431, 1995.

[Gal *et al.*, 2008] Y. Gal, E. Yamangil, A. Rubin, S. M. Shieber, and B. J. Grosz. Towards collaborative intelligent tutors: Automated recognition of users' strategies. In *Proceedings of Ninth International Conference on Intelligent Tutoring Systems (ITS), Montreal, Quebec*, 2008.

[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 and Steedman, 2007] C.W. Geib and M. Steedman. On natural language processing and plan recognition. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1612–1617, 2007.

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

[Lochbaum, 1998] K. E. Lochbaum. A collaborative planning model of intentional structure. *Computational Linguistics*, 4(525–572), 1998.

[Pawar *et al.*, 2007] U.S. Pawar, J. Pal, and K. Toyama. Multiple Mice for Computers in Education in Developing Countries. In *Conference on Information and Communication Technologies and Development*, pages 64–71, 2007.

[Pollack, 1990] M.E. Pollack. Plans as complex mental attitudes. *Intentions in communication*, 1990.

[Quilici *et al.*, 1998] A. Quilici, Q. Yang, and S. Woods. Applying plan recognition algorithms to program understanding. *Automated Software Engineering*, 5(3):347–372, 1998.

[Reddy *et al.*, 2009] S. Reddy, Y. Gal, and S. M. Shieber. Recognition of users' activities using constraint satisfaction. In *Proceedings of the First and Seventeenth International Conference on User Modeling, Adaptation and Personalization*, 2009.

[Sidner, 1985] C.L. Sidner. Plan parsing for intended response recognition in discourse. *Computational intelligence*, 1(1):1–10, 1985.

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