Chapter 11
Plan Recognition and Visualization in Exploratory Learning Environments

Ofra Amir, Kobi Gal, David Yaron, Michael Karabinos and Robert Belford

Abstract Exploratory Learning Environments (ELEs) are open-ended software in which students build scientific models and examine properties of the models by running them and analyzing the results (Amershi and Conati, Intelligent tutoring systems. LNCS. Springer, Heidelberg, 463–472, 2006); Chen (Instr Sci, 23(1–3):183–220, 1995); (Cocea et al., 2008). ELEs are generally used in classes too large for teachers to monitor all students and provide assistance when needed (Gal et al., 2008). They are also becoming increasingly prevalent in developing countries where access to teachers and other educational resources is limited (Pawar et al., 2007). Thus, there is a need to develop tools of support for teachers' understanding of students' activities. This chapter presents methods for addressing these needs. It presents an efficient algorithm for intelligently recognizing students' activities, and novel visualization methods for presenting these activities to teachers. Our empirical analysis is based on an ELE for teaching chemistry that is

O. Amir (✉)
Harvard University, School of Engineering and Applied Sciences, 29 Oxford Street, Cambridge, MA 02138, USA
e-mail: oamir@seas.harvard.edu

K. Gal
Department of Information Systems Engineering Ben-Gurion Boulevard, Ben-Gurion University, Building 93, Beer Sheva 84105, Israel
e-mail: kobig@bgu.ac.il

D. Yaron · M. Karabinos
Department of Chemistry, Carnegie-Melon University, 4400 Fifth Avenue, Pittsburgh, PA 15213, USA
e-mail: yaron@cmu.edu

M. Karabinos
e-mail: mk7@cmu.edu

R. Belford
Department of Chemistry and Biochemistry, University of Arkansas, A 119 Chemistry Building, Fayetteville, AR 72701, USA
e-mail: rebelford@ualr.edu

used by thousands of students in colleges and high schools in several countries (Yaron et al., Science, 328(5978), 584–585, 2010).

Keywords Plan recognition · Visualization · Exploratory learning environments · Recognition algorithm · Virtual labs

Abbreviations

AI	Artificial intelligence
CCD	Create correct device action
ELEs	Exploratory learning environments
ITS	Intelligent tutoring systems
MS	Mix solution
MSC	Mixing solution components
MSI	Mixing the solution using an intermediate flask
SDP	Solving the dilution problem

11.1 Introduction

There are several aspects to students' interactions that make plan recognition in ELEs 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 activities indefinitely in pursuit of a goal or sub-goal, such as adding varying amounts of an active compound to a solution until a desired molarity 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 recognition algorithm presented in this paper addresses these challenges by using a recursive grammar to generate plan fragments for describing key chemical processes in the lab. The algorithm receives as input students' complete interaction sequence with the software, as well as a grammar describing possible activities. It expands activities from the grammar using a heuristic that chooses (possibly non-contiguous) actions from students' interaction and outputs a hierarchical plan that explains how the software was used by the student. The algorithm was evaluated using real data obtained from students using the ELE to solve six representative problems from introductory chemistry courses. Despite its incompleteness, the algorithm was able to correctly infer students' plans in all of the instances given that appropriate grammar rules were available. It was able to identify partial solutions in cases where students failed to solve the complete problem, as well as capture interleaving plans.

We used two novel visualization methods to present students' activities to teachers. One of the methods visualized the plans that were inferred by the recognition algorithm. The second method visualized students' actions over a timeline. A user study with chemistry teachers was conducted that compared these visualization methods with a baseline technique consisting of movies showing the students' application window during their work. The results showed that teachers preferred the temporal- and plan-based methods over the movie visualization, despite the fact that the movie was easier to learn. Both the plan- and temporal-based visualization methods were found useful by teachers, and improved teachers' understanding of student performance. These visualization methods will be incorporated into a separate application that will be available for use by teacher and student users of Virtual Labs.

These results demonstrate the efficacy of combining computational methods for recognizing users' interactions with intelligent interfaces that visualize how they use flexible, open-ended software. It is a first step in creating systems that provide the right machine-generated support for their users. For teachers, this support consists of presenting students' performance both after and during class. For students, this support will guide their problem-solving in a way that maximizes their learning experience while minimizing interruption.

This chapter integrates and extends a past study for recognizing students' activities in ELEs Amir and Gal [7] in several aspects. First, it introduces novel visualization methods of students' work with exploratory learning environments, one of which is informed by the recognition algorithm. Second, it demonstrates the efficacy of these visualization methods in the real world by showing they support teachers in the analysis of student performance in ELEs. Lastly, it evaluates the recognition algorithm on a significantly larger scale.

The rest of this chatper is organized as follows. Section 11.2 presents related work in two different areas: plan recognition and student assessment. Section 11.3 presents the ELE domain which is the focus of our empirical methodology. Section 11.4 presents the plan recognition algorithm and demonstrates its performance on student data.

Section 11.5 describes a user study for comparing different visualization methods of students' activities to teachers. Section 11.6 concludes this work and discusses its significance for the goal of creating collaborative systems in exploratory domains.

11.2 Related Work

The work reported in this book chapter relates to two different areas of prior work and a range of approaches within each: plan recognition and assessment of students' activities with software. The subsections below discuss related work in these two areas respectively.

11.2.1 Plan Recognition

Plan recognition is a cornerstone problem in artificial intelligence (AI) which aims to infer an agent's goals and plans given observations of its actions. Applications of plan recognition can be found in a wide range of fields, such as natural language dialog Carberry [8]; Grosz and Sidner [9], software help systems Baueret al. [10]; Mayfield [11], story understanding Wilensky [12]; Charniak and Goldman [13] and human–computer collaboration Lesh et al. [14].

Past works have used plan recognition to infer students' plans from their interactions an ELE for teaching statistics Gal et al. [4]; Reddy et al. [15]; Gal et al. [16]. Specifically, Reddy et al. [15] proposed a complete algorithm which modeled the plan recognition task as a Constraint Satisfaction Problem (CSP). Gal et al. [4] devised a heuristic algorithm that matched actions from students' logs with the recipes for the given problem. These approaches do not support recursive grammars, which are essential for capturing the type of exploratory activities that characterize the ELE in our setting, such as indefinite repetition. We further extend these works by visualizing students' activities to teachers.

Other works have implemented plan recognition techniques to model students' activities in Intelligent Tutoring Systems (ITS) VanLehn et al. [17]; Conati et al. [18, 19]; Anderson et al. [20]; Corbett et al. [21]; Vee et al. [22]. In these systems, the tutor takes an active role in students' interactions, providing feedback and hints. Plan recognition has also been used to recognize users' activities when programming in UNIX Blaylock and Allen [23], or interacting with medical diagnosis and email notification systems Bauer [24]; Horvitz [25]; Lesh [26]. All of the above settings are significantly more constrained than ELEs, severely limiting the amount of exploration that students can perform. Thus these approaches are not suitable for recognizing students' activities in ELEs. Our work also extends the plan recognition literature more generally. Traditional approaches to plan recognition Kautz [27]; Lochbaum [28] did not consider incomplete information of the agent, mistakes, extraneous actions, interleaving and multiple plans, which are endemic feature of ELEs.

More recently, Geib and Goldman [29] proposed a probabilistic model of plan recognition that recognized interleaving actions and output a disjunction of plans—rather than a single hierarchy—to explain an action sequence.

It also accounted for missing observations (e.g., not seeing an expected action in a candidate plan makes another candidate plan more likely). Our work is distinct from this approach in several ways. First, the settings studied by Geib and Goldman do not account for agents' extraneous actions, which are common to students' interactions in ELEs. Second, we show the efficacy of our approach on real-world data obtained from students using pedagogical software, whilst Geib and Goldman use synthetic data.

11.2.2 Assessment of Students' Activities

The visualization methods in this paper relate to several strands of research for analyzing and assessing students' interactions with pedagogical software. Some systems work on-line, visualizing predefined features of students' interactions to teachers. The following describe notable examples. The student tracking tool Pearce-Lazard et al. [30]; Gutierrez-Santos et al. [31] is part of the MiGen project for improving 11–14 year-old students' learning of algebraic generalization. This tool monitors students' activities during their sessions with an ELE for teaching algebra. The tracking tool visualizes "land-marks" which occur when the system detects specific actions or repetitive patterns carried out by the student.

The FORMID-Observer Gueraud et al. [32] monitors students' activities in simulation-based work sessions with the FORMID-Learner. A teacher can specify specific situations to be monitored representing certain system states, possible student mistakes, and tests that can be triggered by the student. These activities are visualized in the teacher interface which shows the situations and results of validation requests of each student, using a coloring scheme of green for correct activities and red for incorrect activities.

Other systems work post hoc, and generate reports to teachers based on students' complete interaction histories. These systems do not display the students' activities but rather summarize performance measures such as the number of hints requested and success rates in problems. Relevant examples include the ASSISTment system Feng and Heffernan [33] and Student Inspector Scheuer and Zinn [34].

Lastly, data mining techniques have been used to analyze students' performance with pedagogical software. The DataShop Koedinger et al. [35] system generates learning curves reports for students, error reports and general performance reports of students. The Tool for Advanced Data Analysis in Education (TADA-Ed) Merceron and Yacef [36] system discovers correlations between students' mistakes in different problems. Sao Pedro et al. [37] and Montalvo et al. [38] trained decision tree detectors to identify two types of students' planning approaches in microworlds, a simulation based educational software. Their modelling is based on features such as action frequencies and latency between actions. Amershi and Conati [39] have used data mining techniques to cluster and classify students' interaction behaviors in ELEs as either effective or ineffective for learning. Kardan and Conati [40] extended this work to extract association rules of each cluster and use these rules for both online classification of new learners as well as for post analysis of the behaviors that were effective for learning.

Our work differs from these data mining approaches in that it provides an individual analysis of students' problem-solving behavior. For example, while the approach described in Kardan and Conati [40] will classify a student as belonging to either a high-learning gain group or a low-learning gain group, our approach provides a temporal and hierarchical visualization of the student's interaction.

11.3 The Virtual Labs Domain

In this section we describe the ELE that provides the empirical setting for this paper. Virtual Labs simulates a real chemistry lab and used in the instruction of college and high school chemistry courses worldwide. It allows students to design and carry out experiments which connect theoretical chemistry concepts to real world applications Yaron et al. [6]. 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[1] You are allowed a maximal deviation of 0.005 M in either direction.

To solve this problem in VirtualLabs, students are required to pour the correct amounts of HNO_3 and H_2O to an empty flask which will contain the diluted solution. Despite the simplicity of this problem, students solve it in different ways. A possible solution for this problem is to repeatedly mix varying quantities of HNO_3 with H_2O until achieving the required concentration. We describe a student’s interaction adapted from our empirical analysis which follows this paradigm. 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 an empty destination flask.[2]

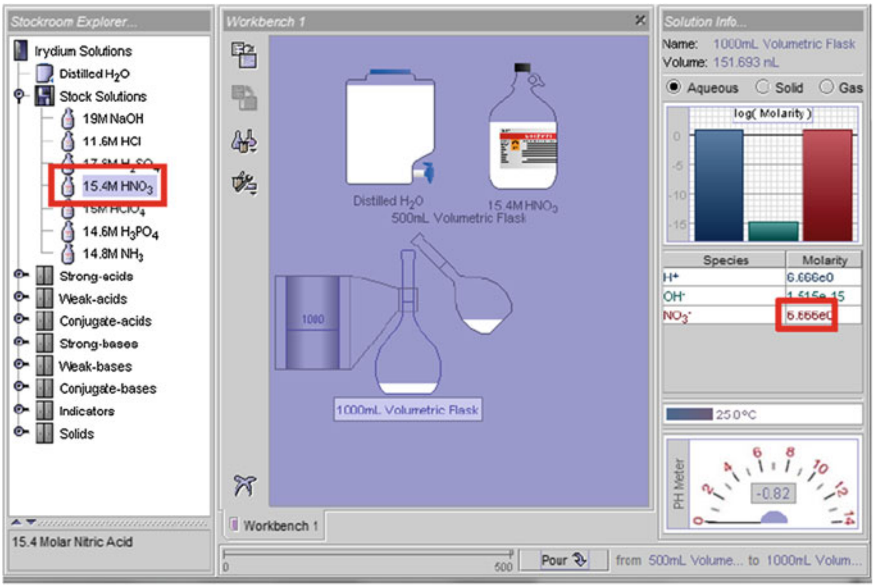
This activity was repeated four times, resulting in 400 ml of HNO_3 in the destination flask. The student proceeded to dilute this solution by mixing it with 510 ml of H_2O. This activity was carried out in two steps, one adding 10 ml of H_2O (using an intermediate flask of 10 ml) and another adding 500 ml of H_2O (using an intermediate flask of 500 ml). At this point the molarity of HNO_3 in the destination flask was too low (6.666 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 11.1 shows a snapshot of Virtual Labs 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 “15.4 M HNO_3” (outlined in red 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

[1] In chemistry, ‘M’ denotes the measure of Molar concentration of a substance.

[2] Intermediate flasks are commonly used in Virtual Labs to help measure solutions accurately, as in a physical laboratory.



Name: 1000mL Volumetric Flask
Volume: 151.693 mL

[✓ Aqueous] Solid Gas

Species	Molarity
H^+	6.666e0
OH^-	1.515e-15
NO_3^-	6.666e0

Fig. 11.1 Snapshot of interaction in virtual labs

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 red in the figure).

The student's interaction described above highlights several aspects endemic to scientific inquiry that are supported by the Virtual Labs software. First, the concept of titration, which 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.

Whereas the student in the example interaction described above used a trial-and-error approach to solve the dilution problem, there are other possible solution strategies. For example, students can pre-calculate the exact amounts of H_2O and HNO_3 that should be mixed to achieve the desired molarity. After calculating these quantities students can proceed to immediately mix them in Virtual Labs and achieve the desired concentration.

11.4 Plan Recognition in Virtual Laboratories

This section describes the devised grammar and plan recognition algorithm. We define the plan recognition problem in Virtual Labs, the formalisms used by our approach, and the proposed recognition algorithm. Finally, we present the results of an empirical evaluation performed on real data taken from students' interactions with VirtualLabs.

11.4.1 Actions, Recipes, and Plans

Our plan recognition algorithm is based on a generative grammar that captures the experimental nature of students' activities in ELEs. We use the term basic actions Pollack [41] to define rudimentary operations that cannot be decomposed. 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 be "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 four times".

A recipe for a complex action specifies the sequence of operations required for fulfilling the complex action, called sub-actions. 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

Fig. 11.2 Recipes for **a** solving the dilution problem; **b** repetition of activities; **c** using intermediate flasks

(a)

$$\begin{aligned}&\underline{\mathrm{SDP}}[s\_id_1, vol_1, s\_id_2, vol_2, d\_id_1] \rightarrow\\&\quad \underline{\mathrm{MSC}}[s\_id_1, d\_id_1, sc_1 = H_2O, vol_1],\\&\quad \underline{\mathrm{MSC}}[s\_id_2, d\_id_2, sc_2 = HNO_3, vol_2]\end{aligned}$$

$$d\_id_1 = d\_id_2$$

(b)

$$\begin{aligned}&\underline{\mathrm{MSC}}[s\_id_1, d\_id_1, sc_1, vol = vol_1 + vol_2] \rightarrow\\&\quad \underline{\mathrm{MSC}}[s\_id_1, d\_id_1, sc_1, vol_1],\\&\quad \underline{\mathrm{MSC}}[s\_id_2, d\_id_2, sc_2, vol_2]\end{aligned}$$

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

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

(c)

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

parameter values of a complex action to the value of the parameters in its constituent sub-actions.

Figure 11.2a presents a recipe for the complex action of Solving the Dilution Problem ($\underline{\text{SDP}}$) composed of two complex sub-actions for Mixing Solution Components ($\underline{\text{MSC}}$), 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 $\underline{\text{MSC}}[s\_id_1; d\_id_1; sc_1 = H_2O; vol_1]$ of pouring $= H_2O$ in Fig. 11.2a 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 $\underline{\text{MSC}}$ 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 $\text{sc}_2 = HNO_3$).

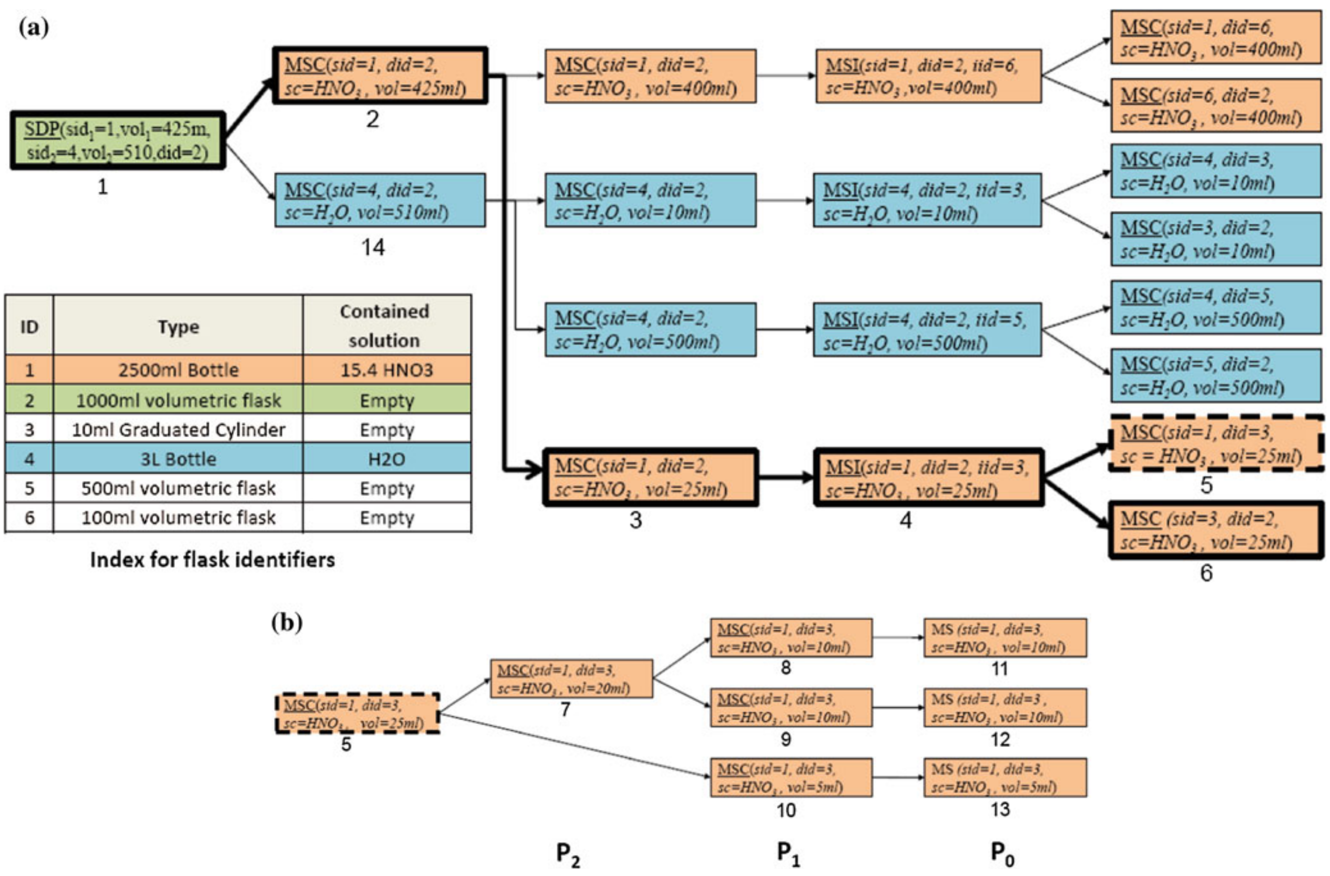
Recipes may be recursive, capturing activities that can repeat indefinitely, as in titration. This is exemplified in the recipe shown in Fig. 11.2b for the complex action ($\underline{\text{MSC}}$) of adding a solution component of volume *vol* from flask s_id_1 to flask d_id_1. The constituent actions of this recipe decompose the $\underline{\text{MSC}}$ action into two separate $\underline{\text{MSC}}$ 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 $\underline{\text{MSC}}$ that includes a Mix Solution (MS) basic action.

Figure 11.2c presents another recipe for an $\underline{\text{MSC}}$ complex action which decomposes into a constituent sub-action for Mixing the Solution using an Intermediate flask ($\underline{\text{MSI}}$).[3] 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 Sect. 11.3, the complex sub-actions for mixing H_2O with HNO_3 fulfill the recipe for the complex action $\underline{\text{SDP}}$ of solving the dilution problem. These actions are labeled "1, 2" and "14" in Fig. 11.3a.

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 Fig. 11.3a. Time is represented in the Figure from top to bottom, thus crossing edges signify interleaving between actions.

The hierarchy emanating from the root node $\underline{\text{SDP}}$ (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

[3] For brevity, we omit the recipes for the $\underline{\text{MSI}}$ action. The complete recipe library for the dilution problem can be found in Sect. 11.8.


(a)
SDP(sid₁=1,vol₁=425m, sid₂=4,vol₂=510,did=2)
1
MSC(sid=1, did=2, sc=HNO₃, vol=425ml)
2
MSC(sid=4, did=2, sc=H₂O, vol=510ml)
14
MSC(sid=1, did=2, sc=HNO₃, vol=400ml)
MSI(sid=1, did=2, iid=6, sc=HNO₃, vol=400ml)
MSC(sid=1, did=6, sc=HNO₃, vol=400ml)
MSC(sid=6, did=2, sc=HNO₃, vol=400ml)
MSC(sid=4, did=2, sc=H₂O, vol=10ml)
MSI(sid=4, did=2, iid=3, sc=H₂O, vol=10ml)
MSC(sid=4, did=3, sc=H₂O, vol=10ml)
MSC(sid=3, did=2, sc=H₂O, vol=10ml)
MSC(sid=4, did=2, sc=H₂O, vol=500ml)
MSI(sid=4, did=2, iid=5, sc=H₂O, vol=500ml)
MSC(sid=4, did=5, sc=H₂O, vol=500ml)
MSC(sid=5, did=2, sc=H₂O, vol=500ml)
MSC(sid=1, did=2, sc=HNO₃, vol=25ml)
3
MSI(sid=1, did=2, iid=3, sc=HNO₃, vol=25ml)
4
MSC(sid=1, did=3, sc = HNO₃, vol=25ml)
5
MSC (sid=3, did=2, sc=HNO₃, vol=25ml)
6
Index for flask identifiers
(b)
MSC(sid=1, did=3, sc=HNO₃, vol=25ml)
5
MSC(sid=1, did=3, sc=HNO₃, vol=20ml)
7
MSC(sid=1, did=3, sc=HNO₃, vol=10ml)
8
MSC(sid=1, did=3, sc=HNO₃, vol=10ml)
9
MSC(sid=1, did=3, sc=HNO₃, vol=5ml)
10
MS (sid=1, did=3, sc=HNO₃, vol=10ml)
11
MS (sid=1, did=3, sc=HNO₃, vol=10ml)
12
MS (sid=1, did=3, sc=HNO₃, vol=5ml)
13
P₂
P₁
P₀

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

Fig. 11.3 **a** A partial plan for the dilution problem corresponding to the student's interaction described in Sect. 3; **b** a plan for the MSC complex action (labeled "5", *dashed outline*)

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 subactions 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 Fig. 11.3a down to the leaves.

Figure 11.3b 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 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").

11.4.2 The Plan Recognition Algorithm

As described in Sect. 11.3, students take diverse approaches to solving the dilution problem. They perform an indefinite number of mixing actions, choose whether to use intermediate flasks and interleave activities. For example, Fig. 11.3a shows 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. Such an approach was used by Reddy et al. [15] in an ELE for teaching statistic. Due to the exploratory and repetitive nature of students' actions in Virtual Labs, naively considering each of these possibilities is not possible.

The proposed algorithm shown in the program code for Bottom-up plan recognition method, incrementally builds a plan which describes students' activities with Virtual Labs. The algorithm 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 action sequence P_0 is initialized with the original action sequence, *X*. During each step, the algorithm attempts to replace subsets of actions from P_t with the complex actions they represent. Each of the complex actions in P_t is a partial plan that explains some activity in the user's interaction. The algorithm iterates over the recipes in *R* (step 3) according to the following (partial) ordering criteria: if the complex action $\underline{C_2}$ is a constituent sub-action for a recipe for a complex action $\underline{C_1}$, then recipes for action $\underline{C_2}$ are considered before the recipes for action $\underline{C_1}$.

Note that the recipe language allows for cycles, but in practice recipes cannot be applied indefinitely in Virtual Labs, because interaction sequences are finite. An ordering over recipes can always be created (possibly by duplicating or renaming actions), such that it meets the sorting constraint. 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), which is described later in this section. FindMatch(R_C,OL) returns a set of actions $M_C \in$ OL such that M_C *fulfills* R_C.

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 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). 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. 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 parent of all of the actions in R_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 Ø), the BUILDPLAN algorithm proceeds to consider a new recipe, and terminates once all recipes have been considered.

FINDMATCH, shown in the program code for the algorithm for finding a match using depth-first search, iterates over the actions in the open list OL performing a complete depth-first search for actions that together fulfill the complex action $\underline{C}$, as defined by the recipe. The algorithm maintains an action set denoted M_C, which at each step of the algorithm contains a subset of actions from the open list that match the sub-actions in the recipe. At each step, the algorithm removes the next action a_P from the open list (step 8), and attempts to add it to the current match M_C. The procedure makes use of the EXTENDS function, a Boolean function that takes as input an action a_P, a partial match M_C, and recipe R_C (step 9). The function EXTENDS returns true if a_P can be added to M_C, such that (1) a_P corresponds to one of the constituent sub-actions of R_C and is not already in M_C and (2) the addition of a_P to M_C will not violate any of the recipe constraints in R_C. For example, given $M_C = \emptyset$, the action MSC[*sid* : 1; *did* : 3, *sc* : H2O; *vol*_1 : 100] extends the recipe for $\underline{\text{SDP}}$ shown in Fig. 11.2a. If the action a_P extends the recipe, it is added to the match M_C, and a recursive call to FINDMATCH is performed, with the updated open list and match.

Each time FINDMATCH is called, it performs a call to the Boolean function FULFILLS(M_C, R_C) (step 12), which returns true if M_C is a complete match for the recipe R_C. We then say that M_C *fulfills* R_C. For example, the actions MSC[*sid* : 1, *did* : 3, *sc* : H_2O, *vol*_1 : 100] and MSC[*sid* : 2, *did* : 3, *sc* : HNO_3, *vol*_1 : 200] fulfill the recipe for SDP shown in Fig. 11.2a. Note that M_C can include both basic and complex actions.

```
1: procedure BuildPlan (R,X).
2:   $P_0 \leftarrow X$
3:   for $R_C \in$ SortRecipes (R) do
4:     $P_{t+1}$, $OL \leftarrow P_t$
5:     $M_C$ = FindMatch ($R_C$,$OL$)
6:     while $M_C \neq \emptyset$ do
7:       BindParams ($C$, $M_C$, $R_C$)
8:       Add $C$ to $OL$ and $P_{t+1}$ positioned after last $a \in M_C$
9:       for all $a \in M_C$ 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 ($R_C$, $OL$)
```

[Bottom-up plan recognition method]

The algorithm backtracks when it does not succeed in finding a match, by removing a_P from M_C and searching for another action to add to the match. It is therefore complete and guaranties to find a match for R_C, given that there is a subset of actions in the open list which fulfill the given recipe. Note that a match can contain non-continuous actions, as long as the constraints defined in the recipe hold, thus allowing for interleaving plans to be found.

We demonstrate this process using the plan in Fig. 11.3b 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 Fig. 11.2(b). At step P_2, the MSC actions labeled "8, 9" were chosen to match the recipe for the MSC action labeled "7".

```
1: procedure FindMatch($R_C$,$OL$) ▷$R_C$: a recipe, $OL$: open list
2:   return FindMatch ($R_C$, $OL$, null)
3: procedure FindMatch ($R_C$, $OL$, $M_C$) ▷$M_C$: a partial match
4:   if FulFills ($M_C$, $R_C$) then
5:     return ($M_C$, $OL$)
6:   $OL' \leftarrow OL$
7:   for $a_P \in OL$ do ▷ $a_P$: an action
8:     remove $a_P$ from $OL'$
9:     if Extends ($a_P$, $M_C$, $R_C$) then
10:      Add $a_P$ to $M_C$
11:      ($M_C$,$OL$) = FindMatch ($R_C$, $OL'$, $M_C$)
12:      if FulFills ($M_C$; $R_C$) then
13:        return ($M_C$, $OL$)
14:      remove $a_P$ from $M_C$
15:  return (null, $OL$)
```

[Algorithm for finding a match using depth-first search]

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.

Although FINDMATCH is complete, 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. Thus, 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, which dominates the complexity of the overall approach.

11.4.3 Empirical Methodology

We evaluated the algorithm on real data consisting of 20 students' interactions with VirtualLabs. These interactions were sampled from a depository of log files describing homework assignments of over 100 students from an R1 private university in a second semester general chemistry course (the sessions with VirtualLabs were not controlled in any way). The sampled interactions included students' solutions to six problems intended to teach different types of experimental and analytical techniques in chemistry, taken from the curriculum of introductory chemistry course using VirtualLabs (students were not repeated across problems). One of these was the dilution problem that was described in Sect. 11.3. A detailed description of all of the problems is given in Sect. 11.7. For diversity, the chosen students' logs varied greatly in size, ranging from 20 actions to 187 actions.

The recipes were created by transforming written descriptions of students' possible solution processes for each problem. These written descriptions were obtained from a domain expert who is a chemistry researcher and one of the developers of VirtualLabs. In addition, we also randomly sampled 5–6 of the students' logs for each problem from the depository of homework assignments described above and added recipes if they were not already given by the domain expert. The log files used in process of creating recipes were not used in the evaluation of the algorithm.

We ran the algorithm on each of the 20 log files using the recipe library of the corresponding VirtualLabs problem. The outputted plans ranged in depth from 3 to 21 levels. The algorithm was evaluated by the domain expert. 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

Table 11.1 Performance measures for the recognition algorithm

	N	Log size	Plan size	Plan depth	Run-time (s)
Coffee	4	33.25	41.75	12	0.28
Oracle	4	92.75	57.75	6.25	1.06
Dilution	4	63	39.75	8	0.54
Unknown acid	4	54.25	56.25	12	0.8
Camping	2	76	31.5	5	0.4
Coffee 2	2	67.5	62	12	1.0
Overall	20	63	48.45	9.35	0.68

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.

The results revealed that BUILDPLAN correctly inferred students' plans for 19 out of the 20 problem instances. Specifically, 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. We demonstrate this capability in the user study described in Sect. 11.5.

The reason for the sole incorrect plan was revealed to be a recipe that was lacking a temporal constraint for enforcing an ordering between its constituent actions. It is important to note that this incorrect inference was not caused by the greediness of the BUILDPLAN algorithm, but by an incomplete recipe data base. This does not impede on the algorithms correctness, as it was always able to infer students' plans given that recipes were available.

Table 11.1 summarizes the performance of the algorithm according to several measures: *N*, representing the number of instances for each problem; log size, representing 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; plan depth, representing the length of the longest path in the inferred plan(s); run time of the algorithm (in seconds) on a commodity quad-core computer. All of the reported results were averaged over the different instances in each problem. As shown in the table, the overall average time for inferring students' plans was 0.68 s, with a relatively high variance (std. 0.79), due to the diversity of the students' interactions and the experimental processes required to solve each of the problems. The longest time to infer students' plans occurred for interactions relating to the "oracle" problem (1.06 s.) and "coffee 2" problem (1.0), which also resulted in the largest plans (57.75 and 62 nodes respectively). The key determinant of the algorithm's runtime was the size of the log that described the student's interaction. These results show the feasibility of using the proposed algorithm in practice, as students' interactions are finite and limited.

11.4.4 Complete Algorithms

In this section we present two plan recognition algorithms that are complete. Both algorithms work by converting the plan recognition problem into one or more constraint satisfaction problems and using standard techniques for their solution. A limitation of this approach is that it is constrained to non-recursive grammars, in which actions cannot be repeated indefinitely. To this end we employed a different exploratory learning environment called TinkerPlots, used world-wide to teach students in grades 4–8 about statistics and probability Konold and Miller [42].

TinkerPlots is an educational software system used world-wide to teach students in grades 4 through 8 about statistics and mathematics [42]. It provides students with a toolkit to actively model stochastic events, and to create and investigate a large number of statistical models [43]. As such, it is an extremely flexible application, allowing for data to be modeled, generated, and analyzed in many ways using an open-ended interface.

To demonstrate our approach towards recognizing activities in TinkerPlots we will use the following running example, called: The probability of rain on any given day is 75 %. Use TinkerPlots to compute the probability that it will rain on each of the next four consecutive days. This problem is a simple example drawn from a set of problems posed to students using TinkerPlots in schools and to subjects during our data collection procedure.

One of the possible approaches towards modeling this problem in Tinker Plots are shown in Fig. 11.4. The top part of the figure shows a sampler object containing "spinner" devices used to model distributions. The spinner device in the left-hand model contains two possible events, "rain" and "sun". The likelihood of "rain" is three times that of "sun", as determined by the surface area of these events within the spinner. Each draw of this sampler will sample the weather for a given day. The number of draws is set to four, making the sampler a stochastic model of the weather on four consecutive days.

The basis of the complete approaches make use of a structure called a plan tree for representing and reasoning about recipes in the database, essentially a search tree for capturing the set of possible plans consistent with the recipe database. A plan tree has two types of nodes: AND nodes, whose children represent actions that must be carried out to complete a recipe, and or nodes, whose children represent a choice of recipes for completing an action. The root, action C, is an OR node. For each recipe for C, a child AND node is added to the root and labeled with the sub-actions of that recipe. The children of this AND node are the plan trees of each sub-action. A branch terminates when a basic action is reached, as a basic action has no recipe by definition.

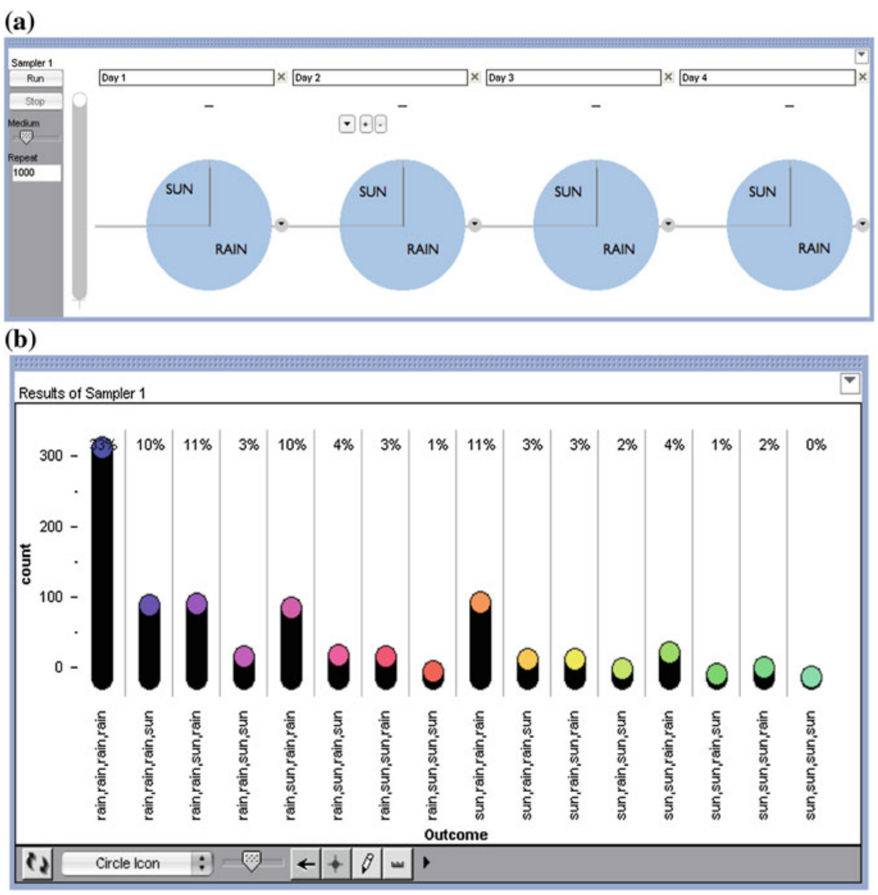


Fig. 11.4 Snapshots of tinkerplots interaction when solving the problem. **a** Using four spinners. **b** Plotted results

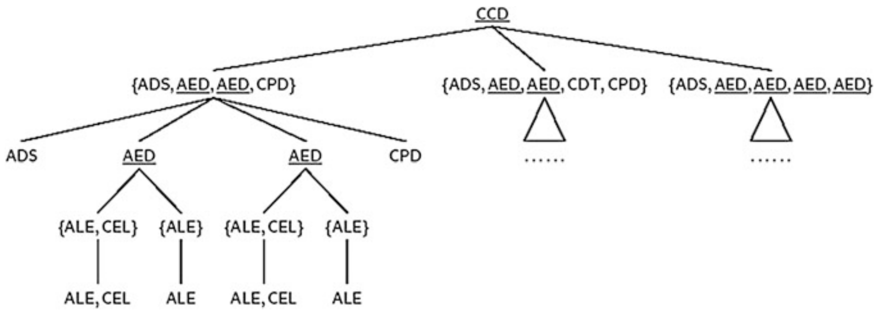


Fig. 11.5 A partial plan tree for the CCD complex action

An example of a plan tree for an activity in TinkerPlots called Create Correct Device Action (CCD) is shown in Fig. 11.5. The AND nodes contain set brackets, while OR nodes do not. Triangles denote unfinished subtrees which were omitted for expository convenience.

The basis of the complete approach is the EXPAND function, shown in the program code for the algorithm for generating expanded recipes, to convert plans to flat representations containing solely basic actions, called *expanded recipes*. An expanded recipe is a series of basic actions (with associated restrictions) that the user may perform to realize a potential plan. To create an expanded recipe, a path is traversed through the plan tree, beginning at the root and ending with basic actions at the leaves. This path provides a trace of the plan corresponding to the expanded recipe. For example, one expanded recipe can be achieved by traversing the plan tree in Fig. 11.5 and choosing the left-most recipe at each OR node. Notice that the path taken matches the plan in Fig. 11.3. In this expanded recipe, each complex AED action and its restrictions are replaced with two basic actions, ALE and CEL, and corresponding restrictions.

The method EXPAND(T_A) takes as input a plan tree T_A for complex action A and returns a set of expanded recipes for A. Each AND node represents a possible recipe for its parent node, a complex action. For each AND node, The EXPAND recursively generates all expanded recipes for each sub-action of the recipe. This algorithm alternates between two sub-procedures, DIRECTSUM and UNION. Given a recipe, the DIRECTSUM procedure computes all possible replacements of complex sub-actions with basic actions. Each time a complex action is replaced, DIRECTSUM ensures that all restrictions involving the complex action are propagated to its sub-actions. Lastly, the UNION sub-procedure takes the union over the expanded recipes generated for each recipe of A.

The complexity of EXPAND is costly in the worst case. Let S be the maximum number of *complex* sub-actions for each recipe, N be the maximum number of recipes for a *single* complex action, and C be the number of distinct complex actions. A plan tree has depth of at most $C + 1$, as we do not allow for recursive recipes. At the lowest depth of the plan tree, all actions are basic and do not have recipes. At the second lowest depth, complex actions have at most N expanded recipes, as none of the N recipes contain any complex sub-actions. At the third lowest depth, each recipe for a complex action may contain at most S complex sub-actions, and each sub-action may have at most N recipes. The DIRECTSUM procedure then creates at most N^S expanded recipes per recipe.

```
1: procedure Expand ($T_C$) ▷ $T_C$: the plan tree for action $C$
2: ERs[$C$] ← ∅ ▷ ERs [C]: the expanded recipes for $C$
3: for all $r_j$, a child of $C$ do ▷ $r_j$: a recipe
4: ERs [$r_j$] ← ∅
5: for all $a_i$, a child of $r_j$ do ▷$a_i$: an action
6: ERs [$r_j$] ← DirectSum (Expand ($T_{ai}$), ERs [$r_j$])
7: ERs [$a$] ← Union(ERs [$a$], ERs [$r_j$])
8: if ERs [$a$] = ∅ then
```

```
 9: ERs [a] ← {a}
10: return ERs [a]
```

[Algorithm for generating expanded recipes]

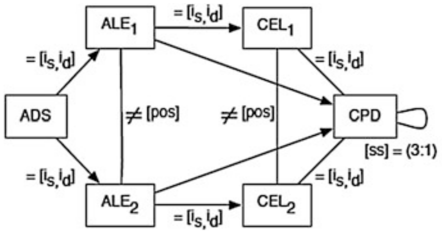
The UNION procedure collects the expanded recipes resulting from each recipe for that action, resulting in a maximum of $N(N)^S$, or N^{S+1}, recipes. At the fourth lowest depth, each complex action can again have at most N recipes with at most S complex sub-actions in each. Each of these S sub-actions can contain at most $N(N)^S$ expanded recipes. So, the DIRECTSUM and UNION procedures create at most $N(N(N)^S)^S$, or N^{S^2+S+1}, expanded recipes per recipe. Continuing this reasoning, the top level action can have at most (11.1) recipes, yielding an overall complexity of $N^{0(S^C)}$.

$$N^{\sum_{i=0}^{C-1} S^i} \tag{11.1}$$

Constraint Satisfaction Algorithm. In this subsection we explain how to combine an expanded recipe and action sequence to create a constraint satisfaction problem (CSP). A solution to the resulting CSP is the plan representing the users' activities. Formally, a CSP is a triple (X, Dom, C), where $X = \{x_1,\ldots, x_n\}$ is a finite set of variables with respective domains Dom $= \{D_1,.., D_n\}$, each a set of possible values for the corresponding variable, $D_i = \left\{v_1^i, v_k^i\right\}$, and a set of constraints $C = \{c_1,\ldots,c_m\}$ that limit the values that can be assigned to any set of variables.

The algorithm CONVERTTOCSP, shown in the program code for converting an expanded recipe and action sequence to a CSP, receives as input an expanded recipe E_A and an action sequence **X** and returns a CSP. If a solution exists for this CSP, a subset of the actions in **X** realize the expanded recipe E_A. We first show how to create variables in the CSP, and we use as a reference Fig. 11.6, which provides a graphical representation of the CSP resulting from some action sequence and expanded recipe. We used a graphical layout suggested by Dechter [44]. Note that parameters belonging to actions are not pictured unless they participate in some constraint.

Let $S = \{s_1,\ldots, s_n\}$ and R be the set of sub-actions and restrictions in the expanded recipe, respectively. Each action in S becomes a unique variable in the CSP by calling the subroutine ADDVARIABLEANDDOMAIN(s, **X**). Based on



Variable:	Domain
ADS:	ADS[$i_s = 11, i_d = 2, t_d =$ spinner, pos = 6], ADS[$i_s = 11, i_d = 3, t_d =$ mixer, pos = 13]
ALE$_1$,ALE$_2$:	ALE[$i_s = 11, i_d = 2, i_e = 1, l_e =$ a, pos = 7], ALE[$i_s = 11, i_d = 2, i_e = 2, l_e =$ b, pos = 8]
CEL$_1$,CEL$_2$:	CEL[$i_s = 11, i_d = 2, i_e = 1, l_e =$ rain, pos = 10], CEL[$i_s = 11, i_d = 2, i_e = 2, l_e =$ sun, pos = 11]
CPD:	CPD[$i_s = 11, i_d = 2$, ss = 1 : 3, pos = 12]

Fig. 11.6 CSP resulting from an action sequence and an expanded recipe

the expanded recipe, six variables are added at this time: ADS, ALE_1, CEL_1, ALE_2, CEL_2, and CPD. These variables appear, outlined, in the graph of Fig. 11.6.

```
1: procedure ConvertTOCSP($E_A$ = ($S$,$R$), X) ▷ EA: an expanded
   recipe $S$ and restrictions $R$ for complex action $A$, $X$: an
   action sequence
2: for all $s \in S$ do ▷ S: a set of sub-actions
3: AddVariableAndDomain ($s$, X)
4: for all $r \in R$ do ▷ R: a set of restrictions
5: AddRestrictionConstraint ($r$)
6: for all $s \in S$ do
7: AddRedundancyConstraint ($S$)
```

[Converting an expanded recipe and action sequence to a CSP]

Each variable's domain is then derived from the actions in the action sequence. For each occurrence of action s in the action sequence, a value is added to the domain of s in the CSP. The right-hand box of Fig. 11.6 gives the resulting domain for each variable based on the action sequence.

Lastly, we add restrictions to our CSP. For each restriction r in R over actions $\{s_1,\ldots, s_m\}$ in S, a constraint over the corresponding CSP variables is added to the CSP using the ADDRESTRICTIONCONSTRAINT(r) subroutine. Directed edges in the Fig. 11.6 represent temporal constraints between two variables. Undirected edges represent other parametric constraints. The edge from ADS to ALE_1 expresses the constraint ADS $\prec$ ALE_1 as well as the constraint ADS$[i_s, i_d]$ = $ALE_1[i_s, i_d]$.

For variables corresponding to the same action, additional redundancy constraints are added using the ADDREDUNDANCYCONSTRAINT subroutine. These constraints ensure that such variables are assigned distinct values, as these variables share the same domain. An example is the constraint connecting the ALE_1 and ALE_2 variables, which requires that these variable assignments have distinct pos parameters.

A solution for a CSP provides a match between an expanded recipe and an action sequence. In this section we present two algorithms that use CSPs to output a plan from an action sequined X for a desired complex action C given a set of recipes R.

The algorithm shown in the program code for brute force algorithm takes a brute force approach, calling EXPAND to generate each expanded recipe for C, converting it to a CSP and solving the CSP. This algorithm returns the first solution found to the CSP or Ø if no solution is found.

```
1: procedure CSPBRUTE ($T_C$,X) ▷ TC: the plan tree for action
   C, X: an action sequence
2: $E$ ← EXPAND($T_C$) ▷ E: a set of expanded recipes
3: for all $e \in$ E do
4: C ← CONVERTTOCSP ($e$, X) ▷ C: a CSP
5: solution ← SOLVE(C)
```

```
6: if $solution \neq \emptyset$ then
7: return solution
8: return $\emptyset$
```

[Brute force algorithm]

The complexity of CSPBRUTE can be analyzed in terms of the FINDMATCH2 and EXPAND procedures. Recall that calling EXPAND results in at most $N^{O(S^C)}$ expanded recipes, where N is the maximum number of recipes for a single complex action. In the worst case, all expanded recipes are considered, and for each expanded recipe a CSP solver must be run. The complexity of this CSP solver can be bounded by the complexity of a complete backtracking search, which we have seen to be $|X|!/S!$. So, an overall worst-case complexity of CSPBRUTE is (11.2).

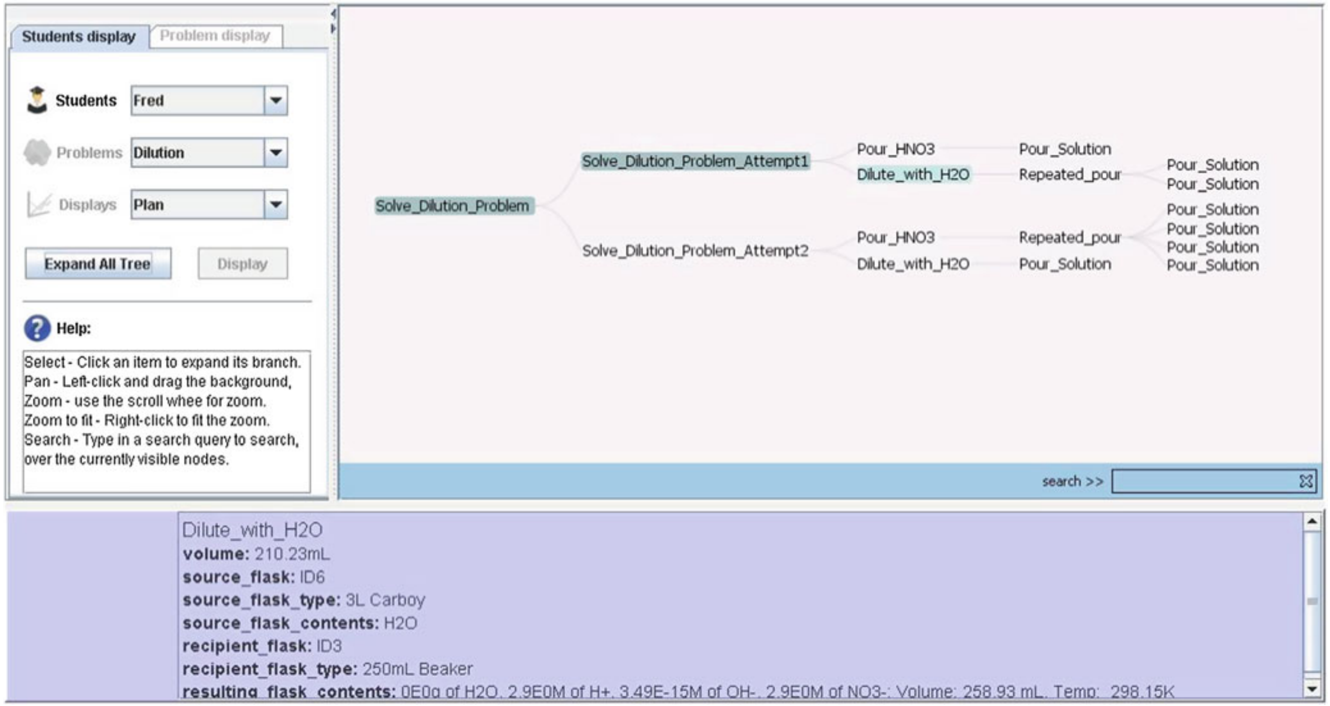
$$N^{O(S^C)}O\left(\frac{|X|!}{S!}\right) \tag{11.2}$$

To evaluate the complete approach, we collected interaction sequences of people's interaction with TinkerPlots. Each subject received an identical 30 min tutorial about TinkerPlots and was then asked to complete four problems in succession; these problems are detailed in Sect. 11.7. TinkerPlots is equipped with a logging facility that records the basic actions that make up users' action sequences. As in the VirtualLabs domain, we noted whether each problem was solved, and we constructed the (possibly multiple) plans used to solve the problem. The analyzed user logs range in length from 14 to 80 actions. The average length of an interaction sequence for problems collected from adult subjects was 35 actions. Adults solved the assigned problems 70 % of the time. In contrast, the average length of an interaction sequence for problems collected from students was 68 actions. Students solved the assigned problems 60 % of the time. Also, people engaged in exploratory behavior using the software. For example, there were on average 15 exogenous actions in each problem that was obtained from adults. As expected, the complete approaches were able to achieve perfect performance on all of the logs. They also took reasonable time, measuring from 2 to 4 s on the logs.

11.5 Visualizing Students' Activities

This Section presents visualization methods that were designed for the purpose of presenting students' activities to teachers. It then describes a user study that evaluated these different methods with chemistry teachers.

[4] We used the Prefuse package to implement this application Heer et al. [45].



Select - Click an item to expand its branch.
Pan - Left-click and drag the background,
Zoom - use the scroll whee for zoom.
Zoom to fit - Right-click to fit the zoom.
Search - Type in a search query to search,
over the currently visible nodes.

Dilute_with_H2O
volume: 210.23mL
source_flask: ID6
source_flask_type: 3L Carboy
source_flask_contents: H2O
recipient_flask: ID3
recipient_flask_type: 250mL Beaker
resulting_flask_contents: 0E0g of H2O, 2.9E0M of H+, 3.49E-15M of OH-, 2.9E0M of NO3-; Volume: 258.93 mL, Temp: 298.15K

Fig. 11.7 A temporal visualization of a student’s solution to the dilution problem

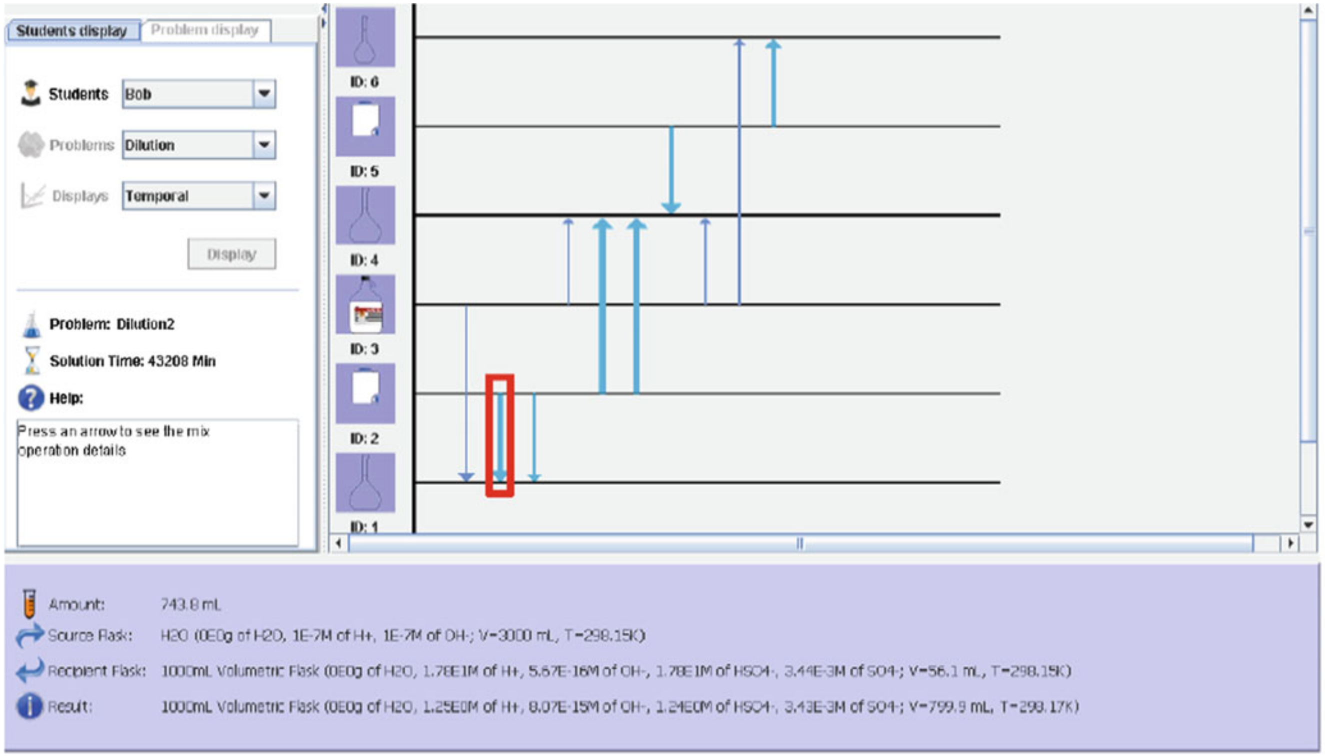
11.5.1 Visualization Methods

We hypothesized that showing students' plans to teachers would facilitate their understanding of students' work. In addition, we wished to evaluate an alternative visualization method that emphasizes the temporal aspects of students' interactions, which is lacking in the plan visualization. We therefore used the following three visualization methods that differ in the type of data they present as well as the way in which this data is presented.

The *plan visualization method* presents students' plans as they are inferred by the recognition algorithm. The plan is presented using an interactive interface that enables to explore the plan tree.[4] An example of this visualization on a student's plan for solving the dilution problem is shown in Fig. 11.7. The plan is presented as a tree. Each of the nodes in the tree represents a student's activity. The leaves of the plan represent the basic actions of the student that constitute students' interactions with VirtualLabs. The other nodes represent higher level activities that were inferred by the algorithm. As shown by the nodes "solve dilution problem attempt 1" and "solve dilution problem attempt 2", the student made two attempts at solving the dilution problem. The descendants of these nodes decompose the activities that constitute each of the attempts. When clicking on a node in the plan, the parameters of the action that corresponds to this node are displayed in the information panel shown at the bottom of the Fig. 11.7. As can be seen, the complex action "dilute with H_2O" consisted of pouring a total of 210.23 ml of H_2O to dilute the acid. The "resulting flask contents" shows the solution consistency in the flask after this dilution activity. The child node of the "dilute with H_2O" action is "repeated pour". Clicking on this node will show the two separate pours from the H_2O bottle that comprises the dilution activity.

The *Temporal visualization* presents students' interactions over a time line. The vertical axis displays the objects used by the student, while the horizontal axis displays students' actions in the order in which they were created. An example of a temporal visualization of a student's interaction with VirtualLabs when solving the dilution problem is shown in Fig. 11.8. This student's interaction consisted of mixing solutions in flasks, and each arrow in the figure represents one of these mixing actions. The base of the arrow represents the source flask, while the head of the arrow indicates the recipient flask. Thicker arrows correspond to larger volumes of solution being mixed. The information panel at the bottom of the figure describes the parameters of the mixing action represented by the boxed arrow in Fig. 11.8. It shows that the student poured 743.8 mL of H_2O, to a 1,000 ml Volumetric Flask. Also shown is the resulting consistency of the solution in the recipient flask.

The *Movie visualization* describes students' actions exactly as they occurred during their interactions with VirtualLabs, and is analogous to a teacher that is looking over the shoulder of a student. This is the only type of support that is currently available to teachers. This visualization replays the actions from the log in the order they were created by the student, but does not reflect the actual



Problem: Dilution2

Solution Time: 43208 Min

Help:

Press an arrow to see the mix operation details

```
Amount:          743.8 mL
Source Flask:    H2O (0E0g of H2O, 1E-7M of H+, 1E-7M of OH-; V=3000 mL, T=298.15K)
Recipient Flask: 1000mL Volumetric Flask (0E0g of H2O, 1.78E1M of H+, 5.67E-16M of OH-, 1.78E1M of HSO4-, 3.44E-3M of SO4-; V=56.1 mL, T=298.15K)
Result:          1000mL Volumetric Flask (0E0g of H2O, 1.25E0M of H+, 8.07E-15M of OH-, 1.24E0M of HSO4-, 3.43E-3M of SO4-; V=799.9 mL, T=298.17K)
```

Fig. 11.8 A plan visualization of a student's solution to the dilution problem

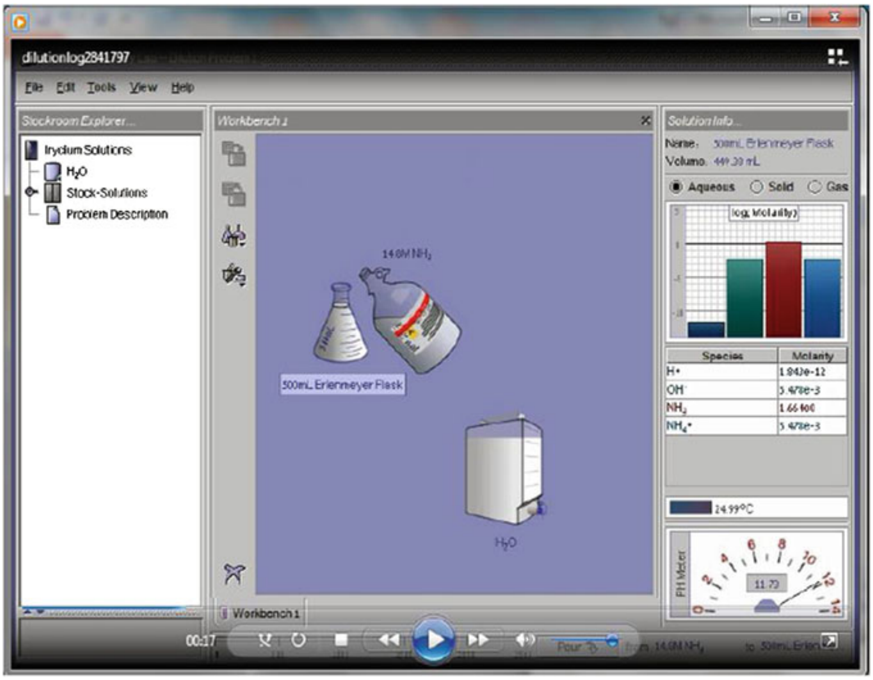
passage of time between students' actions. The movie can be stopped, rewound and fast-forwarded to focus on the students' display at particular points in their interaction. A snapshot of this visualization for one of the students solving the dilution problem is shown in Fig. 11.9. In the snapshot the student is pouring NH_3 to a 500 ml Erlenmeyer flask. On the right side of the figure, the current contents of the selected flask are shown (in the "Solution Info" panel).

These three visualizations differ widely in the way they present information to teachers. First, both the movie and the temporal visualization methods render students' activities directly from the log. The plan visualization supersedes these visualizations in that it also visualizes higher level activities as inferred by the recognition algorithm. Second, the movie presents snapshots of the user's application window, while the temporal and the plan visualizations present a more expansive account of the student's work-flow. In particular, the temporal and plan visualization specify the amount of solution being poured from flask to flask, while this information is not directly shown in the movie.

To illustrate these differences, we describe how teachers and researchers may use each visualization method to identify that a student made several attempts to solve the dilution problem. Using the movie visualization, teachers need to keep track of which flasks the student used to mix acid with H_2O, and pause the movie after each mixing action to observe the resulting concentration of the solution in the flask in the "Solution Info" panel. Because the movie visualization presents a single action at each time-frame, it can be difficult to distinguish whether a mixing action using a new flask represents the commencement of a new attempt to solve the problem or an exploratory action (or a mistake). Using the temporal visualization, teachers can observe the set of flasks used by the student to dilute the acid, and the pouring actions that are associated with each flask.

To characterize the activities making up each of the student's attempts, teachers need to identify the relevant actions over the time line, starting from the action that poured acid to a new flask and terminating in the pouring action that resulted in the diluted solution. The temporal aspect of this presentation makes it easy to identify such sets of pouring actions when they occur close together in time. This is illustrated in Fig. 11.8, in which the three contiguous actions pouring solutions into Flask ID 1 and the 3 contiguous actions pouring solutions into Flask ID 3 represent two distinct attempts (and the next 4 pours represent additional two distinct attempts). However, this procedure may be difficult to do when students' interactions are long, or when students interleave activities, as any two adjacent actions may belong to different attempts.

Lastly, the plan visualization separates each of the students' dilution attempts into a separate branch, and the nodes in each branch comprise those pouring actions that characterize each attempt. This is illustrated in Fig. 11.7, in which each attempt aimed at solving the problem is a sub-plan that emanates from the "Solve_Dilution_Problem_Attempt1" and "Solve_Dilution_Problem_Attempt2" nodes. However, the plan does not order students' actions along a time line, and thus it is difficult to recognize the order in which actions were performed.



Name: 500mL Erlenmeyer Flask
Volume: 449.30 mL

Species	Molarity
H^+	1.940e-12
OH^-	5.478e-3
NH_3	1.66+00
NH_4^+	5.478e-3

Fig. 11.9 A movie visualization of a student's solution to the dilution problem

11.5.2 Empirical Methodology

A user study with chemistry educators was conducted to evaluate these three visualization methods. The goals of this study were to determine how each of these visualizations contributes to teachers' analysis of students' work in VirtualLabs and which visualization methods teachers found helpful.

The interactions in the study were taken from the log files of students solving two problems (out of the six problems for which we collected data). These log files were also used in the evaluation of the plan recognition algorithm, such that the plans were validated as correct by a domain expert prior to the user study.

One of the two problems was the dilution problem described in Sect. 11.3. The other problem (called "coffee") required students to add the right amount of milk to cool a cup of coffee down to a desired temperature. These problems differed in the type of reasoning they demanded from students. The dilution problem was characterized with longer, more complex student solutions. For example, students solving the dilution problem used more intermediate flasks and more attempts to solve the problem. To illustrate, the average log size of solutions to the dilution problem was 51 actions, whereas the average log size of solutions to the coffee problem was 29.67 actions. Thus, we were able to evaluate the visualization methods on two problems with significantly different solution processes.

Seventeen participants took part in the study. Fifteen of the participants were graduate students of chemistry and chemistry engineering serving as teaching assistants (14 students from Ben-Gurion University, and one student from the Weizmann Institute). Two of the participants were a professor of chemistry from the University of British Columbia who uses VirtualLabs in the classroom and a professor of education and technology at Haifa University with a master degree in chemistry.

All participants received an online survey that included all of the materials used in the study (tutorials of VirtualLabs and the visualization methods, and questionnaires for the evaluation of the visualizations). The participants first watched an identical video tutorial of VirtualLabs and were asked to perform several tasks using the software to demonstrate their understanding. Participants were also provided with an identical tutorial about each of the three visualization methods. Each subject was presented with three student interactions solving the same problem. Each of these interactions was shown using one of the three visualization methods, and the order in which the visualizations were presented varied across participants. To avoid biasing the participants, each interaction that was visualized was chosen from a different student. For each problem, participants were presented with interactions that were similar in length and complexity of the student's solution.

Each participant was asked to comment on the visualization methods by answering the following questions: (1) Based on the presentation can you tell whether the student solved the problem?; (2) Based on the presentation can you tell how the student solved the problem?; (3) Assuming you were using VirtualLabs in your class, would you be likely to use this presentation style to understand students' work after a classroom session?[5] After seeing all of the visualizations, participants were asked to quantitatively compare between the different methods according to the same set of criteria, and were also asked to compare how easy it was to learn how to use the different methods. For this comparison participants were requested to rate each visualization method using a Likert scale of 1–7 (where 1 stands for "strongly disagree", 7 stands for "strongly agree", and 4 being a neutral answer of "neither"). Finally, participants were asked whether they preferred one visualization to the others, and how they would combine some or all of the visualizations. One of the researchers was present throughout each of the sessions, answered any questions participants had about the visualization methods, and validated that teachers' conclusions about students' work was correct not.[6] That is, when participants reported that they could infer whether or how a student solved the problem, the researcher validated that their

[5] We also asked participants to explain each of their answers. The full questionnaire can be found in Sect. 11.9.

[6] The researcher was physically present in the laboratory with all of the graduate students from Ben-Gurion University, and used VoIP technology (Skype) to connect with the other three participants.

inference was correct. In all cases participants that reported to have understood students' solutions, did so correctly.

11.5.3 Results

In this section we present the analysis of the responses we received from the participants in the user study. First, we describe a qualitative analysis of participants' responses with regards to the visualizations, followed by a quantitative analysis of their comparisons of the different visualization methods.

Qualitative Analysis of the Visualization Methods. We first describe participants' responses with regards to the *movie visualization*. When asked if the movie visualization demonstrated *whether* the student had solved the problem, participants' responses depended on the type of problem they were shown. Most of the participants (7 out of 9) who viewed solutions to the coffee problem claimed that they were able to tell whether the student had solved the problem, while the other two participants reported that this information was not apparent to them. Those participants that inferred how the students solved the problem did so by observing the final contents of the flasks used by the student. A typical response was "Yes, by following the temperature and volume meters on the right side and watching the actions the student took."

Half of the participants (4 out of 8) who viewed solutions to the dilution problem could not infer whether the student had solved the problem. These participants reported that the movie was too fast and difficult to follow. A typical response was "No, I can not. The added amounts are not clear and the appearance and disappearance of elements on the screen are confusing." Only 2 of 8 participants reported that the movie clearly demonstrated whether the problem was solved by the student, in contrast to 7 out of 9 participants who could infer this information from the coffee problem. A possible explanation for this discrepancy is the length of students' interactions. The average length of students' interactions for the dilution problem was significantly longer than the average length of students' interactions for the coffee problem. This made it more difficult for participants to keep track of students' actions using the movie visualization.

The participants expressed a more homogeneous opinion when asked whether they understood *how* the student had solved the problem, and these responses were not dependent on the problem shown. Ten of the participants reported that the movie enabled them to determine how the student solved the problem. A typical response explained, "It is easy to see the steps the student used to solve the problem." Three of the participants stated that they were not able to determine how the student solved the problem. One of them stated "The presentation [visualization] created a confusion. Irrelevant steps, such as moving flasks were shown, which created a confusion and made it difficult for me to distinguish important actions." Four participants claimed they could determine how the problem was solved in general terms, but were missing the exact quantities mixed by the students. Lastly, only four of the participants reported they would be likely to use the movie visualization in their class. The other participants found it to be too slow to be useful. A typical response was "This method seems to be much slower that could be a problem when checking 30 students or so…".

When evaluating the *temporal visualization*, all participants but one answered that this visualization method clearly demonstrated *whether* and *how* the student had solved the problem, and would be likely to use this method in their class. However, two participants were concerned that if a problem solution would require many steps, the method may not be useful. One of them explained: "[…] If an exercise requires moving many solutions from beaker to beaker, and lots of mixing, this method might not be as clear and become a bit messy."

In their evaluation of the *plan visualization*, all but one of the participants reported that the visualization demonstrated *whether* the student had solved the problem, and 14 out of the participants found that the plan visualization demonstrated *how* the students solved problems. Several of the participants specifically commented on the higher level activities that were represented in this visualization: "The presentation [visualization] focuses on the important actions and summarizes the student's activities.", or "I can read the final concentration in an easy way at each stage and the [different] attempts of the student are very clear."

Three of the participants commented that they have found the plan visualization difficult to understand. One of them explained: "This presentation [visualization] was more complicated. I had to click the nodes and observe the volumes and students actions at each step." In all, 14 out of 17 participants reported they would be likely to use the plan visualization in their classroom.

After observing and evaluating all of the visualization methods, we asked participants which visualization method they preferred. Six participants expressed a strict preference for the plan visualization and six participants preferred the temporal visualization. Only one participant strictly preferred the movie visualization over the other two proposed visualizations. Another participant stated an equal preference for the movie and temporal visualizations, while one participant claimed that he would prefer using the temporal visualization for simple problems, and the plan visualization for complex problems. Table 11.2 summarizes the qualitative responses described in this section. The table only includes the number of participants who expressed a non-ambiguous positive answer to each of the questions discussed above.

Finally, we were interested to see whether teachers would want to use a combination of some or all of the visualizations, given their distinct differences. Seven participants suggested combining the temporal visualization with the plan visualization. Three participants suggested combining the temporal visualization with the movie visualizations, while two participants suggested combining the plan visualization with the movie, and one participant said he would want to combine all visualizations. Several participants indicated in their response that they believe teachers may have different preferences, and therefore suggested to provide all visualizations and let the teacher choose which of them to use. They further envisioned using different visualizations for different purposes, for example

Table 11.2 Summary of qualitative responses

	Demonstrates whether the student solved the problem	Demonstrates how the student solved the problem	Likely to be used	Strictly preferred
Movie	9/17	10/17	4/17	1/17
Temporal	16/17	16/17	16/17	6/17
Plan	16/17	14/17	14/17	6/17

For each visualization the table shows the number of participants who expressed a clear positive response to each of the questions

using the plan visualization to get a one-image quick view of the solution structure, and then use the temporal visualization for a more in depth exploration of solutions they found more interesting.

Analysis of Quantitative Responses. After observing and separately evaluating each of the visualizations, participants were asked to make a quantitative comparison of the different methods. Fig. 11.10 shows the average score given by participants to each of the visualization methods when using a Likert scale of 1–7. $N = 17$ for each of the questions as all participants responded to all questions. As shown in the Fig. 11.10, the movie had a higher average score than the other methods with respect to ease of learning (Mean = 6.23, STD. = 1.35). The plan visualization was the hardest to learn (Mean = 4.17, STD. = 2.19), and also exhibited the highest variance in scores.

The plan visualization scored highest with respect to demonstrating whether the student solved the problem (Mean = 5.94, STD. = 1.89), closely followed by the temporal visualization (Mean = 5.53, STD. = 1.94). The movie was ranked last (Mean = 4.18, STD. = 2.27). The temporal and plan visualization methods scored highest with respect to demonstrating how the student solved the problem (Mean = 5.88, STD. = 1.65), followed by the plan visualization (Mean = 5.53, STD. = 1.56). Again, the movie visualization was ranked last (Mean = 4.65, STD. = 2.06). The plan (Mean = 5.88, STD. = 1.54) and temporal (Mean = 5.88, STD. = 1.54) visualizations scored highest with respect to being used in the classroom. The movie was ranked last (Mean = 3.41, STD. = 2.2).

We used the Friedman non-parametric test for analysis of variance to distinguish between participants' quantitative responses and found a significant effect of visualization type in all of the questions ($X^2 > 6.3$, $P < 0.05$). Post-hoc analysis with Wilcoxon Signed-Rank Test was conducted with a Bonferroni correction applied.

Median scores for ease of learning were 7 (3 to 7), 6 (1 to 7) and 4 (1 to 7) for the movie, temporal and plan visualizations respectfully. Both the movie and temporal visualizations were significantly easier to learn than the plan visualization ($Z < -2.55$, $P < 0.011$). The plan, with median score 7 (1 to 7), was significantly more helpful than the movie, with median score 5 (1 to 7), when inferring whether the student solved the problems ($Z = -2.61$, $P = 0.003$).

No significant difference was found between the temporal visualization with median 6 (1 to 7) and the other visualizations ($Z > -2.1$, $P > 0.031$). The temporal

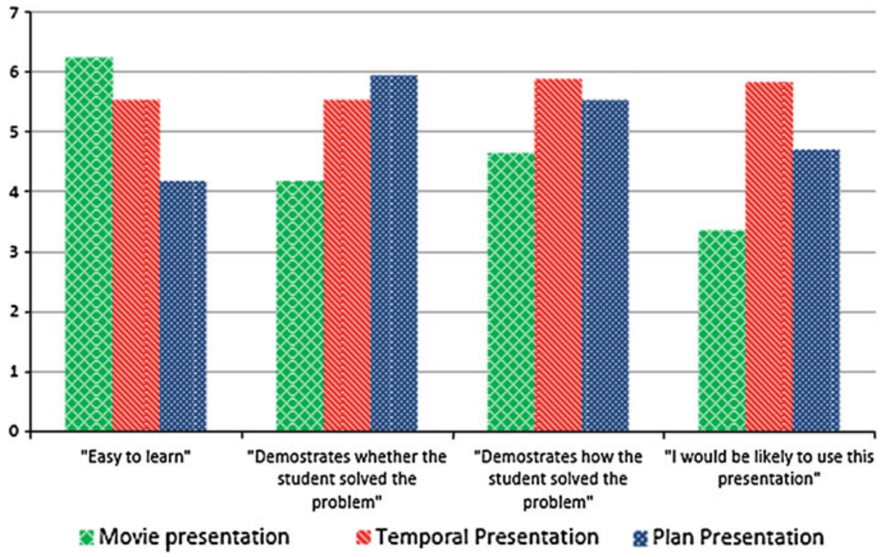


Fig. 11.10 Average scores for the quantitative questions

visualization was found to be significantly more helpful than the movie when inferring how the student solved the problem ($Z = -2.23$, $P = 0.011$) with medians 6 (1 to 7) and 5 (1 to 7) respectfully. There was no significant difference between the plan with median 6 (1 to 7) and the other visualizations ($Z > -1.51$, $-P > 0.075$). Lastly, the temporal visualization with median 6 (1 to 7) was found significantly more likely to be used than the movie visualization with median 3 (1 to 7) ($Z = -3.05$, $P = 0.001$). The differences between the plan visualization with median 5 (1 to 7) and both temporal and movie visualizations were not significant ($Z > -1.96$, $P > 0.027$).

11.5.4 Discussion

A challenge to performing this user study was the relatively large overhead involved in teaching participants about the three visualization methods, and the requirement that participants have prior teaching experience in chemistry. The fact that many of our conclusions reported above were found to be statistically significant is striking given these limitations.

The study revealed, unsurprisingly, that the movie was the most intuitive visualization style and the easiest to learn. However, it was also ranked the least useful for understanding students' work, as can be attested by one of the participants: "Even though the movie style is the easiest to learn it is the hardest to use." The movie was a "playback" of students' work in the lab, and presented both significant actions and irrelevant steps without distinction. All subjects used the added functionality provided by the movie visualization (pausing, rewinding, fast-forwarding). However, fewer participants were able to infer students' solutions using the movie than the other visualizations. This was due to the inherent continuous nature of the movie, in which the system state constantly changes, making it difficult for teachers to identify those actions that are salient to the students' solution.

In contrast to the movie visualization, both the temporal and plan visualization methods provided a higher level and more comprehensive description of students' activities. They were preferred by most of the participants in all of the criteria. The results comparing between the plan and the temporal visualizations were more mixed. On the one hand, most participants preferred the plan visualization to the temporal visualization for inferring whether the student solved the problem.

On the other hand, the temporal visualization was rated higher for inferring how students used VirtualLabs to solve problems. Also, most participants consistently rated the temporal visualization highly in all criteria while exhibiting a significantly higher variance when ranking the plan visualization.

To explain this discrepancy, we note that it was easy for participants to discern whether the student solved the problem by looking at the root of the plan hierarchy, while this information was not explicitly represented in the temporal visualization. We hypothesize that the hierarchical nature of the plan visualization was harder for participants to learn than the temporal visualization. This may explain why they preferred the temporal visualization to the plan visualization when inferring how students solve problems.

We found a 0.7 correlation between the likelihood of using the plan visualization and its ease of learning, and a 0.56 correlation between determining how the student solved the problem and its ease of learning. There was limited time in our lab study for participants to practice the plan visualization method, which was harder to understand than the other methods. However, this result suggests that teachers who understand the plan visualization are likely to adopt it, and that the plan visualization may be very useful to teachers in practice.

Lastly, the diversity of participants' suggestions for combining the various visualizations methods and the possible uses of the visualizations emphasizes the need to adjust to different educators preferences. There is no "silver bullet" visualization that is most useful in all cases and to all users. This was supported by participants' responses which suggested different uses of the visualization methods, and their suggestions for combining the different methods.

11.6 Conclusion and Future Work

This chapter presented novel methods and algorithms for augmenting existing pedagogical software for science education. It addressed two main problems: automatic recognition of students' activities in open-ended pedagogical software and the visualization of these activities to teachers in a way that supports their analysis of students' interactions with such software. To address the first problem, the chapter presented a general plan recognition algorithm for exploratory learning environments. The algorithm was successfully able to recognize students' plans when solving six separate problems in VirtualLabs, as verified by a domain expert. To address the second problem, the paper presented novel methods for visualizing students' interactions with VirtualLabs. Both of these methods were preferred by participants in a user study to a movie of students' interactions with the software.

Our long term goal is the design of collaborative systems for supporting the interaction of students and teachers in a variety of pedagogical domains. These tools embody the principals of collaborative decision-making, in that the system provides the best possible support for its users while minimizing the amount of intervention.

Our future work will extend the methods and algorithms proposed in this chapter in order to build such collaborative systems. To do so we intend to extend our work on both the plan recognition and visualization methods. One limitation of the plan recognition approach is the reliance on domain experts to construct appropriate recipes in a formal way.

In future work we will design novel methods for automatically extracting recipes and allowing teachers to design recipes in a straightforward way. We also intend to design new plan recognition algorithms that recognize students' activities in real time, during their interaction with the software. We will construct computer agents that use these recognition algorithms to generate interventions with the student while minimizing the amount of intrusion.

We will extend the work on visualization methods to study how other types of state-based visualizations affect teachers' understanding of students' activities, such as showing selected snapshots of students' interactions. Also we intend to develop aggregate visualization methods for describing groups of students.

Although our techniques were demonstrated on one software system their applicability has been shown to other open-ended pedagogical software Gal et al. [16]. We also plan to apply our approach to other types of domains in which users engage in exploration, such as Integrated Development Environments (IDEs).

11.7 Experimental Problems

We detail the six VirtualLabs problems used in our empirical evaluation.

DILUTION: You are a work study for the chemistry department. Your supervisor has just asked you to prepare 500 ml of 3 M HNO_3 for tomorrow's undergraduate experiment. In the stockroom explorer, you will find a cabinet called "Stock Solutions". Open this cabinet to find a 2.5 L bottle labeled "11.6 M HNO_3". The concentration of the HNO_3 is 15.4 M. Please prepare a flask containing 500 ml of a 3 M (±0.005 M) solution and relabel it with its precise molarity. Note that you must use realistic transfer mode, a buret, and a volumetric flask for this problem. Please do any relevant calculations on the paper supplied. As a reminder, to calculate the volume needed to make a solution of a given molarity, you may use the following formula: $C_1V_1 = C_2V_2$

ORACLE: Given four substances A, B, C, and D that are known to react in some weird and mysterious way (an oracle relayed this information to you within a dream), design and perform virtual lab experiments to determine the reaction between these substances, including the stoichiometric coefficients. You will find 1.00 M solutions of each of these chemical reagents in the stockroom.

COFFEE: During the summer after your first year at Carnegie Mellon, you are lucky enough to get a job making coffee at Starbucks, but you tell your parents and friends that you have secured a lucrative position as a "Java engineer". An eccentric chemistry professor (not mentioning any names) stops in every day and orders 250 ml of house coffee at precisely 95 °C. He then adds enough milk at 10 °C to drop the temperature of the coffee to 90 °C. (a) Calculate the amount of milk (in ml) the professor must add to reach this temperature. Show all your work, and circle the answer. (b) Use the Virtual Lab to make the coffee/milk solution and verify the answer you calculated in (a). Hint: the coffee is in an insulated travel mug, so no heat escapes. To insulate a piece of glassware in Virtual Lab, Mac-users should hold down the command key while clicking on the beaker or flask; Windows users should right click on the beaker or flask. From the menu that appears choose "Thermal Properties". Check the box labeled "insulated from surroundings". The temperature of the solution in that beaker or flask will remain constant.

COFFEE 2: During the summer after your first year at Carnegie Mellon, you are lucky enough to get a job making coffee at Starbucks, but you tell your parents and friends that you have secured a lucrative position as a "java engineer." An eccentric chemistry professor (not mentioning any names) stops in every day and orders 250 ml of Sumatran coffee. The coffee, initially at 85 °C. is way to hot for the professor, who prefers his coffee served at a more reasonable 65.0 °C. You need to add enough milk at 5.00 °C, to drop the temperature of the coffee.

How much milk do you add? Calculate the amount of milk (in ml) you must add to reach this temperature. In the previous part of the problem, you solved it assuming that both coffee and milk have the same specific heat capacities and densities as water. Since milk is a mixture of water, fat and proteins, its specific heat capacity is likely to be different than the one assumed. Solve again the same problem determining the specific heat of milk and considering it in your calculations. Assume the density is 1.000 g/ml for milk and coffee and the specific heat capacity is 4.184 J/(g °C) for coffee.

CAMPING: You and a friend are hiking the Appalachian Trail when a storm comes through. You stop to eat, but find that all available firewood is too wet to start a fire. From your Chem 106 class you remember that heat is given off by some chemical reactions; if you could mix two solutions together to produce an exothermic reaction, you might be able to cook the food you brought along for the hike. Luckily, being the dedicated chemist that you are, you never go anywhere without taking along a couple chemical solutions, just for times like this. The

Virtual Lab contains aqueous solutions of compounds *X* and *Y* of various concentrations. These compounds react to produces a new compound, *Z*, according to the reaction: $x + y \rightarrow z$. The following activities will guide you in using this reaction to produce the heat needed to warm up your food. Use the virtual lab to measure the enthalpy of the reaction shown above.

UNKNOWN ACID: The "Homework Solutions" cabinet contains a solution labeled "Unknown Acid", which is a weak mono-protic acid with an unknown *Ka* and with an unknown concentration. Your job is to determine the concentration and *Ka* to two significant figures.

11.8 The Recipe Library for the Dilution Problem

This section lists the complete recipe library for the dilution problem. Table 11.3 provides a key to the action abbreviations used in the recipes.

11.8.1 Dilution Problem Recipes

1. $\underline{\mathrm{MSC}}[sc, dt; sid, did, vol, scd, dcd, rcd] \rightarrow \mathrm{MS}[sc, dt; sid, did, vol, scd, dcd, rcd]$
2. $\underline{\mathrm{MSC}}[sc, dt, sid, did, vol = vol_1 + vol_2, scd_2, dcd_2, rcd_2] \rightarrow \underline{\mathrm{MSC}}[sc, dt, sid, did, vol_1, scd_1, dcd_1, rcd_1], \underline{\mathrm{MSC}}[sc, dt, sid, did, vol_2, scd_2, dcd_2, rcd_2]$ $sid_1 = sid_2, did_1 = did_2, scd_1 = scd_2$
3. $\mathrm{MSI}[sc, dt, sid, did, vol, scd, dcd, rcd] \rightarrow \mathrm{MSC}[sc : H_2O, dt, sid; did; vol; scd, dcd, rcd]$
4. $\underline{\mathrm{MSI}}[sc, dt, sid, did, vol, scd, dcd, rcd] \rightarrow \mathrm{MSC}[sc : 15{:}4\ \mathrm{M}\ HNO_3, dt, sid, did, vol, scd, dcd, rcd]$
5. $\underline{\mathrm{MSI}}[sc_1, dt_2, sid1, did_2, vol_1] \rightarrow \underline{\mathrm{MSI}}[sc_1 : H_2O, dt_1, sid_1, did_1, vol_1, scd_1, dcd_1, rcd_1], \underline{\mathrm{MSC}}[sc_2, dt_2, sid_2, did_2, vol_2, scd_2, dcd_2, rcd_2][0]$ $did_1 = sid_2, rcd_1 = scd_2$

Table 11.3 Abbreviation key for complex actions used in recipes

Abbreviation	Action	Meaning
MS	Mixing solution	Basic solution mix operation as observed directly from log files
MSC	Mixing solution component 2	A complex action representing repeated mixing of a solution to the same destination
MSI	Mixing solution through intermediate flask	A complex action representing the use of intermediate flasks when mixing solution
SDP	Solve dilution problem	The root of the plan(s), composed of mixing H_2O and the solution to be diluted

6. $\underline{\text{MSI}}[sc_1, dt_2, sid_1, did_2, vol_1] \rightarrow \underline{\text{MSI}}[sc_1 : 15{:}4\ \text{M HNO}_3, dt_1, sid_1, did_1, vol_1, scd_1, dcd_1, rcd_1], \underline{\text{MSC}}[sc_2, dt_2, sid_2, did_2, vol_2, scd_2, dcd_2, rcd_2][0]\ did_1 = sid_2, rcd_1 = scd_2$
7. $\underline{\text{MSC}}[sc, dt, sid, did, vol, scd, dcd, rcd] \rightarrow \underline{\text{MSI}}[sc, dt, sid, did, vol, scd, dcd, rcd]$
8. $\underline{\text{MSC}}[sc, dt, sid, did, vol = vol_1 + vol_2, scd_2, dcd_2, rcd_2] \rightarrow \underline{\text{MSC}}[sc, dt, sid, did, vol_1, scd_1, dcd_1, rcd_1], \underline{\text{MSC}}[sc, dt, sid, did, vol_2, scd_2, dcd_2, rcd_2]\ sid_1 = sid_2, did_1 = did_2, scd_1 = scd_2$
9. $\underline{\text{SDP}}[sc, dt, sid, did, vol = vol_1 + vol_2, scd_2, dcd_2, rcd_2] \rightarrow \underline{\text{MSC}}[sc : \text{H}_2\text{O}, dt, vol_1, did_1], \underline{\text{MSC}}[\text{sc} : 15{:}4\ \text{M HNO}_3, dt, vol_2, did_2]$

11.8.2 Recipes Explanation

Recipes Explanation: Recipes 1 and 2 capture repeated pouring activities, where users pour the same solution from the same source flask to the same destination flask (1 is the base of the recursion). Recipes 3 and 4 capture the activity of using an intermediate flask when pouring H_2O (i.e. pouring from flask 1 to flask 2 and then from flask 2 to flask 3). Recipes 5 and 6 are the same as 3 and 4, only for HNO_3. Recipes 7 and 8 are the same as 1 and 2, only now they can capture higher level activities which served the same overall goal (for example pouring from flask 1 to flask 2 through intermediate flask 3, and pouring from flask 1 to flask 2 through intermediate flask 4, both serve the same goal of pouring from flask 1 to flask 2). Recipe 9 forms the root of a plan, as it is composed of the pouring actions that involved H_2O and those of pouring HNO_3.

11.9 User Study Questionnaire

After observing each of the visualization methods, the participants responded to the following questions:

- Based on the presentation, can you tell WHETHER the student solved the problem? Please describe how you can tell whether the student solved the problem, or why you can't.
- Based on the presentation, can you tell HOW the student solved the problem? Please describe how the presentation helps you understand the student solution, or what information is missing.
- Assuming you were using VirtualLabs in your class, would you be likely to use this presentation style to understand a student's work after a classroom VirtualLabs session? Why?
- Additional comments. For example: What are the problems of this presentation style? How would you improve it? What information did you find helpful? What information was missing?

In the second part of the questionnaire participants stated their level of agreement (on a scale of 1–7) with the following statements with regards to each of the visualization methods:

- This presentation style was easy for me to learn.
- This presentation style demonstrates WHETHER the student solved then problem.
- This presentation style demonstrates HOW the student solved the problem.
- Assuming I would be using VirtualLabs in my class, I am likely to use this presentation style to understand a student's work after a classroom VirtualLabs session.

There was also space for additional comments after each of these statements. Finally, participants responded to the following two open questions:

- Did you prefer one style to all of the others? If so, which? Would you use one or some of the styles rather than the other/s to visualize students' work?
- Would you combine some or all of these presentation styles together? If so, can you list, for each presentation style, which aspects of a students' interaction are best visualized by that style?

References

1. Amershi, S., Conati, C.: Automatic Recognition of Learner Groups in Exploratory Learning Environments. In: Ikeda, M., Ashley, K.D., Chan, T.W. (eds.) Intelligent Tutoring Systems. LNCS, vol. 4053, pp. 463–472. Springer, Heidelberg (2006)
2. Chen, M.: A methodology for characterizing computer-based learning environments. Instr. Sci. **23**(1–3), 183–220 (1995)
3. Cocea, M., Gutierrez-Santos, S., Magoulas, G.D.: The Challenge of Intelligent Support in Exploratory Learning Environments: A Study of the Scenarios. In: Gutierrez-Santos, S., Mavrikis, M. (eds.) 1st International Workshop in Intelligent Support for Exploratory Environments (ISEE-2008), vol. 381, CEUR-WS, Maastricht (2008)
4. Gal, Y., Yamangil, E., Rubin, A., Shieber, S.M., Grosz, B.J.: Towards Collaborative Intelligent Tutors: Automated Recognition of Users' Strategies. In: Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (eds.) Ninth International Conference on Intelligent Tutoring Systems (ITS 2008), LNCS, vol. 5091, pp. 162–172. Springer, Heidelberg (2008)
5. Pawar, U. S., Pal, J., Toyama, K.: Multiple mice for computers in education in developing countries. In: Conference on Information and Communication Technologies and Development, pp. 64–71. University of California, Berkeley (2007)
6. Yaron, D., Karabinos, M., Lange, D., Greeno, J.G., Leinhardt, G.: The chemcollective-virtual labs for introductory chemistry courses. Science **328**(5978), 584–585 (2010)
7. Amir, O., Gal, Y.: Plan Recognition in Virtual Laboratories. In: Walsh, T. (ed.) 22nd International Joint Conference on Artificial Intelligence (IJCAI), pp. 2392–2397. AAAI Press, Menlo Park (2011)
8. Carberry, S.: Plan Recognition in Natural Language Dialogue. MIT Press, Cambridge (1990)
9. Grosz, B.J., Sidner, C.L.: Plans for Discourse. In: Morgan, J.L., Pollack, M.E., Cohen, P.R. (eds.) Intentions in Communication, pp. 417–444. The MIT Press, Cambridge (1990)

10. Bauer, M., Biundo, S., Dengler, D., Koehler, J., Paul, G.: PHI—Logic-based Tool for Intelligent Help Systems. In: Bajcsi, R. (ed.) 13th International Joint Conference on Artificial Intelligence (IJCAI), pp. 460–466. Morgan Kaufmann, San Francisco (1993)
11. Mayfield, J.: Controlling inference in plan recognition. User Model. User-Adap. Inter. **2**(1), 55–82 (1992)
12. Wilensky, R.: Why John married Mary: understanding stories involving recurring goals. Cogn. Sci. **2**(3), 235–266 (1978)
13. Charniak, E., Goldman, R.P.: A Bayesian model of plan recognition. Artif. Intell. **64**(1), 53–79 (1993)
14. Lesh, N., Rich, C., Sidner, C.L.: Using Plan Recognition in Human-Computer Collaboration. In: Kay, J. (ed.) Seventh International Conference on User Modeling, pp. 23–32. Springer, New York (1999)
15. Reddy, S., Gal, Y., Shieber, S.M.: Recognition of Users' Activities Using Constraint Satisfaction. In: Houben, G.J., McCalla, G., Pianesi, F., Zancanaro, M. (eds.) User Modeling, Adaptation, and Personalization, LNCS, vol. 5535, pp. 415–421. Springer, Heidelberg (2009)
16. Gal, Y., Reddy, S., Shieber, S., Rubin, A., Grosz, B.: Plan recognition in exploratory domains. Artif. Intell. **176**(1), 2270–2290 (2012)
17. VanLehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R.H., Taylor, L., Treacy, D.J.: Weinstein, A, Wintersgill, M.C.: The Andes physics tutoring system: lessons learned. Int. J. Artif. Intell. Educ. **15**(3), 147–204 (2005)
18. Conati, C., Gertner, A.S., VanLehn, K., Druzdzel, M.J.: On-line Student Modeling for Coached Problem Solving Using Bayesian Networks. In: Jameson, A., Paris, C., Tasso, C. (eds.) Sixth International Conference on User Modeling, pp. 231–242. Springer Wien, New York (1997)
19. Conati, C., Gertner, A.S., VanLehn, K.: Using Bayesian networks to manage uncertainty in student modeling. User Model. User-Adap. Inter. **12**(4), 371–417 (2002)
20. Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R.: Cognitive tutors: lessons learned. J. Learn. Sci. **4**(2), 167–207 (1995)
21. Corbett, A., McLaughlin, M., Scarpinatto, K.C.: Modeling student knowledge: cognitive tutors in high school and college. User Model. User-Adap. Inter. **10**(2–3), 81–108 (2000)
22. Vee, M.H.N.C., Meyer, B., Mannock, K.L.: Understanding novice errors and error paths in object-oriented programming through log analysis. In: Workshop on Educational Data Mining at the 8th International Conference on Intelligent Tutoring Systems (ITS 2006), pp. 13–20. Jhongli (2006)
23. Blaylock, N., Allen, J.: Recognizing Instantiated Goals Using Statistical Methods. In: Kaminka (ed.) Workshop on Modeling Others from Observations, pp. 79–86, Edinburgh (2005)
24. Bauer, M.: Acquisition of user preferences for plan recognition. In: Fifth International Conference on User Modeling, pp. 105–112. User Modeling Incorporated, Kailua-Kona (1996)
25. Horvitz, E.: Principles of mixed-initiative user interfaces. In: ACM SIGCHI Conference on Human Factors in Computing Systems, pp. 159–166. ACM, New York (1999)
26. Lesh, N.: Adaptive goal recognition. In: 15th International Joint Conference on Artificial Intelligence (IJCAI), pp. 1208–1214. Morgan Kaufmann, San Francisco (1997)
27. Kautz, H. A.: A formal theory of plan recognition. Ph. D Thesis, University of Rochester (1987)
28. Lochbaum, K.E.: A collaborative planning model of intentional structure. J. Comput. Linguist. **24**(4), 525–572 (1998)
29. Geib, C.W., Goldman, R.P.: A probabilistic plan recognition algorithm based on plan tree grammars. Artif. Intell. **173**(11), 1101–1132 (2009)
30. Pearce-Lazard, D., Poulovassilis, A., Geraniou, E.: The Design of Teacher Assistance Tools in an Exploratory Learning Environment for Mathematics Generalisation. In: Wolpers, M., Kirschner, P.A., Scheffel, M., Lindstaedt, S., Dimitrova, V. (eds.) Sustaining TEL: From

Innovation to Learning and Practice, LNCS, vol. 6383, pp. 260–275. Springer, Heidelberg (2010)

31. Gutierrez-Santos, S., Geraniou, E., Pearce-Lazard, D., Poulovassilis, A.: Design of teacher assistance tools in an exploratory learning environment for algebraic generalisation. IEEE Trans. Learn. Technol. **5**(4), 366–376 (2012)
32. Gueraud, V., Adam, J.M., Lejeune, A., Dubois, M., Mandran, N.: Teachers need support too: Formid-observer, a flexible environment for supervising simulation-based learning situations. In: 2nd International Workshop on Intelligent Support for Exploratory Environments, pp. 19–28. Brighton (2009)
33. Feng, M., Heffernan, N.T.: Towards live informing and automatic analyzing of student learning: reporting in assistment system. J. Interact. Learn. Res. **18**(2), 207–230 (2007)
34. Scheuer, O., Zinn, C.: How did the e-learning session go? The student inspector. In: 2007 Conference on Artificial Intelligence in Education, pp. 487–494. IOS Press, Amsterdam (2007)
35. Koedinger, K.R., Baker, R.S.J.D., Cunningham, K., Skogsholm, A., Leber, B., Stamper, J.: A Data Repository for the EDM community: The PSLC DataShop. In: Romero, C., Ventura, S., Pechenizkiy, M., Baker, R.S.J.D. (eds.). Handbook of Educational Data Mining, Chapman and Hall/CRC Data Mining and Knowledge Discovery Series Boca Raton, pp. 43–55. CRC Press, Boca Raton (2010)
36. Merceron, A., Yacef, K.: Tada-ed for educational data mining. Interact. Multimedia Electron. J. Comput. Enhanced Learn. **7**(1), 267–287 (2005)
37. Sao Pedro, M.A., Baker, R.S.J., Montalvo, O., Nakama, A., Gubert, J.D.: Using Text Replay Tagging to Produce Detectors of Systematic Experimentation Behavior Patterns. In: Baker, R.S.J.D., Merceron, A., Pavlik Jr., P.I. (eds.) 3rd International Conference on Educational Data Mining, pp. 181–190. International Educational Data Mining Society, Pittsburgh (2010)
38. Montalvo, O., Baker, R.S.J., Sao Pedro, M.A., Nakama, A., Gobert, J.D.: Identifying Students Inquiry Planning Using Machine Learning. In: Baker, R.S.J.D., Merceron, A., Pavlik Jr., P.I. (eds.) 3rd International Conference on Educational Data Mining, pp. 141–150. International Educational Data Mining Society, Pittsburgh (2010)
39. Amershi, S., Conati, C.: Combining unsupervised and supervised classification to build user models for exploratory learning environments. J. Educ. Data Min. **1**(1), 18–71 (2009)
40. Kardan, S., Conati, C.: A Framework for Capturing Distinguishing User Interaction Behaviours in Novel Interfaces. In: Pechenizkiy, M., Calders, T., Conati, C., Ventura, S., Romero, C., Stamper, J. (eds.) 4th International Conference on Educational Data Mining, pp. 159–168. International Educational Data Mining Society, Eindhoven (2011)
41. Pollack, M.E.: Plans as complex Mental Attitudes. In: Morgan, J.L., Pollack, M.E., Cohen, P.R. (eds.) Intentions in Communication, pp. 77–103. The MIT Press, Cambridge (1990)
42. Konold, C., Miller, C.: TinkerPlots Dynamic Data Exploration 1.0. Key Curriculum Press. URL http://www.keypress.com/x5715.xml (2004)
43. Hammerman, J.K., Rubin, A.: Strategies for managing statistical complexity with new software tools. Stat. Educ. Res. J. **3**(2), 17–41 (2004)
44. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
45. Heer, J., Card, S.K., Landay, J.A.: Prefuse: A toolkit for interactive information visualization. In: SIGCHI conference on human factors in computing systems, pp. 421–430. ACM, New York (2005)