# Plan Recognition for Exploratory Learning Environments Using Interleaved Temporal Search

*Oriel Uzan, Reuth Dekel, Or Seri, Ya'akov (Kobi) Gal*

■ *This article presents new algorithms for inferring users' activities in a class of flexible and open-ended educational software called exploratory learning environments (ELEs). Such settings provide a rich educational environment for students, but challenge teachers to keep track of students' progress and to assess their performance. This article presents techniques for recognizing students' activities in ELEs and visualizing these activities to students. It describes a new plan-recognition algorithm that takes into account repetition and interleaving of activities. This algorithm was evaluated empirically using two ELEs for teaching chemistry and statistics used by thousands of students in several countries. It was able to outperform the state-of-the-art plan-recognition algorithms when compared to a gold standard that was obtained by a domain expert. We also show that visualizing students' plans improves their performance on new problems when compared to an alternative visualization that consists of a step-by-step list of actions.*

Modern educational software is open ended and flexible, allowing students to build scientific models and examine properties of the models by running them and analyzing the results (Amershi and Conati 2006; Cocea, Gutierrez-Santos, and Magoulas 2008). Such exploratory learning environments (ELEs) are distinguished from more traditional e-learning systems in that students can build models from scratch by choosing objects from a depository, modifying the objects, and using these modified objects to construct new objects. They are also becoming increasingly prevalent in developing countries where access to teachers is limited. Such ELEs provide a rich educational experience for students and are generally used in classes too large for teachers to monitor all students and provide assistance when needed. Thus, there is a need to develop techniques that recognize and visualize students' activities in a way that supports students in their work and contributes to

  ISSN 0738-4602

their learning by using the software. Such tools can provide support for teachers and education researchers in analyzing and assessing students' use of the software.

Students' interactions with ELEs are complex, as we illustrate using concepts from an ELE for teaching the basics of chemistry. 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. They can repeat activities indefinitely in pursuit of a goal or subgoal, such as adding varying amounts of an active compound to a solution until a desired molarity is achieved. Finally, students can interleave between activities, such as preparing a solution for a new experiment while waiting for the results of a current experiment. These aspects make plan recognition a challenging task in ELEs.

This article presents a plan-recognition algorithm that works from the bottom up, matching students' actions to a predefined grammar according to heuristics that are informed by the way students use ELEs. The algorithm works offline, decomposing a student's entire interaction with the software into hierarchies of interdependent plans that best describe the student's work. It was evaluated in an extensive empirical study that involved seven different types of problems and 68 instances of students' interactions in two different ELEs for chemistry and statistics education. These ELEs varied widely in the type of interaction they entailed from students. The hierarchy of activities that was outputted by the algorithm was compared to a gold standard that was generated by a domain expert. The algorithm was able to achieve comparable or better recognition performance than the different state-of-the-art algorithms that were designed separately for each of the ELEs. It executed in reasonable time on real-world logs of students' sessions, despite the exponential worst-case complexity of the algorithm.

The second part of this article describes a study that demonstrates the benefit of visualizing plans to students. Students were shown a visualization generated by the ELE of the plan for an expert's solution to a statistics problem. Another group of students was presented with an ordered list of the actions composing the solution to the problem. Such a list represents the sole option currently available for extracting log activities from the software post hoc. Both groups of students were subsequently asked to use the statistics ELE to solve new problems that were gradually more difficult than the example problem and required students to generalize mathematical concepts. Students' performance for the new problems was analyzed using several measures, including the length of interaction time with the ELE, the number of actions performed on the ELE, and the ratio of extraneous actions representing mistakes. The results showed that students who were presented with the plan visualization outperformed those students who were presented with the list of activities for all of these measures.

These contributions demonstrate the benefit of applying novel plan-recognition technologies toward intelligent analysis of students' interactions in open-ended and flexible software. Such technologies can potentially support teachers in their understanding of student behavior as well as students in their problem solving and lead to advances in automatic recognition in other exploratory domains.

## Related Work

Early approaches to plan recognition have assumed a goal-oriented agent whose activities were consistent with its knowledge base and that formed a single encompassing plan (Kautz 1987, Lochbaum 1998). We refer the reader to Carberry (2001) for a detailed account of these approaches and focus this section on recent works that capture some of the qualities of exploratory domains, such as extraneous actions or mistakes and interleaving of activities. Avrahami-Zilberbrand and Kaminka (2005) handled temporal and free-order constraints among actions by using tree structures. They provided methods for plan recognition that traverse the tree in a manner that is temporally consistent with the observations while making minimal commitments about matching actions to the grammar. Pynadath and Wellman (2000) developed a probabilistic grammar for modeling agents' plans that also included their beliefs about the environment. These techniques were evaluated using synthetic data and did not allow for interleaving plans (all reordering among plan constituents was restricted to local permutation among constituent actions). Geib and Goldman (2009) presented a probabilistic online plan-recognition algorithm that builds all possible plans incrementally with each new observation. This work maintains all possible explanations for matching future unseen actions by the agent. As we show in the article, naively applying this approach to exploratory domains is unfeasible.

Gal et al. (2012) proposed two algorithms for inferring students' plans using an ELE for statistics education. One of these algorithms used heuristics to match students' actions to the grammar. Another reduced the plan-recognition task to a constraint satisfaction process. This approach assumes a nonrecursive grammar and cannot recognize students' plans in cases where students engage in indefinite repetition, as in physics or chemistry ELEs. Other works (Amir and Gal 2013) allow for recursive grammars in their plan-recognition algorithm for an ELE for chemistry education but make greedy choices about how to match students' actions to the grammar. In contrast, our algorithm makes more informed choices about matching actions to the grammar that are inferred by students' sequential and interleaving

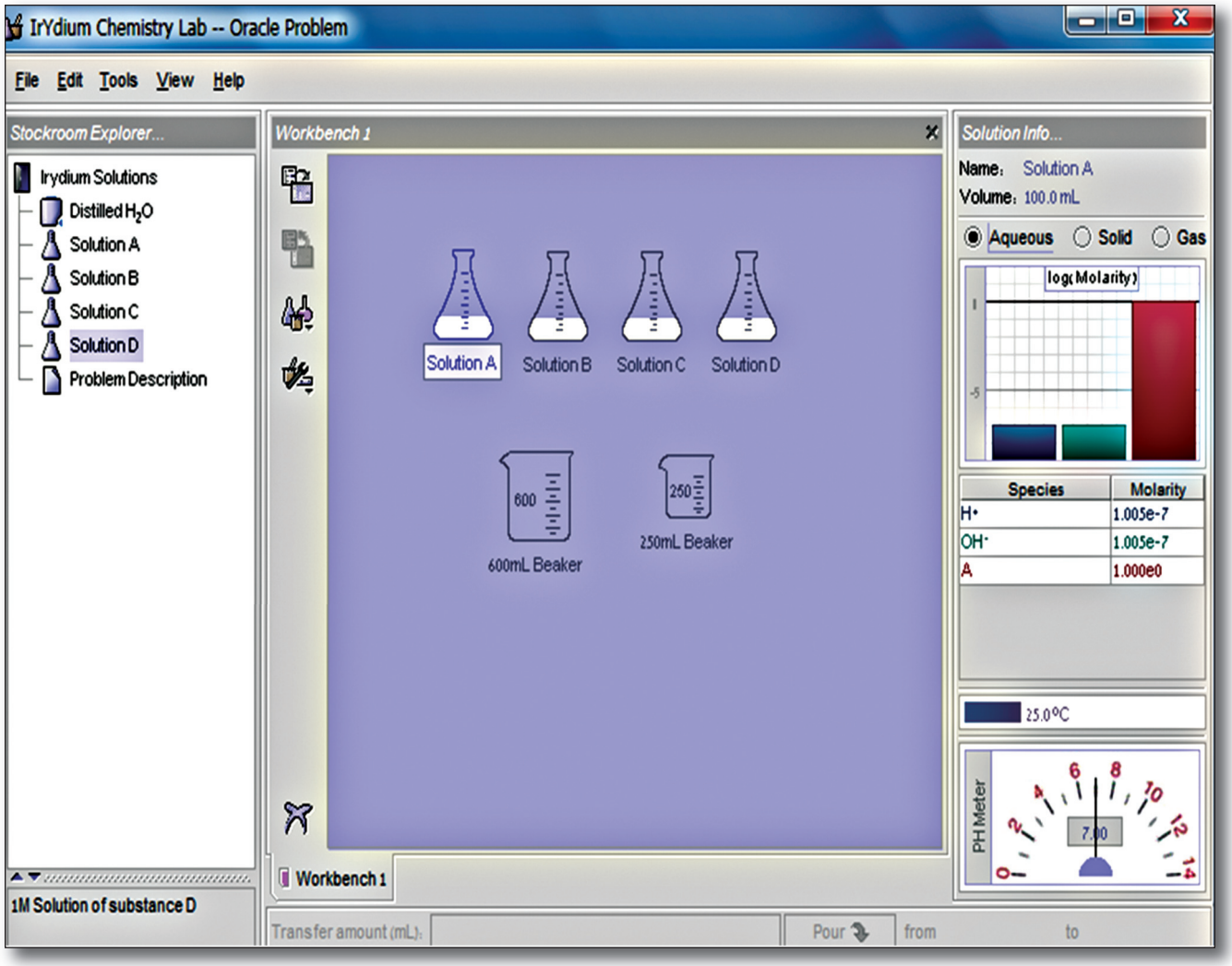


*Figure 1. Snapshot of VirtualLabs.*

interaction styles. Also, none of these past approaches have been shown to work for more than a single ELE, whereas one of our algorithms is shown to generalize to two ELEs for chemistry and statistics education that are significantly different in their design and interaction style with the student.

Finally, we mention works that use recognition techniques to model students' activities in intelligent tutoring systems (ITSs) (VanLehn et al. 2005; Conati, Gertner, and VanLehn 2002; Roll, Aleven, and Koedinger 2010; Koedinger et al. 1997). Such systems coach students during their problem solving, providing support with proven learning gains. Conati, Gertner, and VanLehn (2002) used online plan-recognition algorithms to infer students' plans to solve a problem in an educational software for teaching physics by comparing their actions to a set of predefined possible plans. The number of possible plans grow exponentially in the types of domains we consider, making it unfeasible to apply this approach.

## Actions and Plans

In this section we provide the basic definitions that are required for formalizing the plan-recognition problems in ELEs. Throughout the article we will use an ELE called VirtualLabs to demonstrate our approach. VirtualLabs allows students to design and carry out their own experiments for investigating chemical processes (Yaron et al. 2010) by simulating the conditions and effects that characterize scientific inquiry in the physical laboratory. We use the following problem called Oracle as a running example:

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

The flexibility of VirtualLabs affords two classes of solution strategies to this problem (and many variations within each). The first strategy mixes all four solutions together and infers the reactants by inspect-

ing the resulting solution. The second strategy mixes pairs of solutions until a reaction is obtained. A snapshot of a student's interaction with VirtualLabs when solving the Oracle problem is shown in figure 1.

We make the following definitions: We use the term *basic actions* to define rudimentary operations that cannot be decomposed. For example, the basic "Mix Solution" action ($MS_1[s = 1, d = 3]$) describes a pour from flask ID 1 to flask ID 3. The output of a student's interaction with an ELE (and the input to the plan-recognition algorithm described in the next section) is a sequence of basic-level actions representing students' activities, also referred to as a log. Complex actions describe higher-level, more abstract activities that can be decomposed into subactions, which can be basic actions or complex actions themselves. For example, the complex action $\underline{MSD}[s = 6 + 8, d = 2]$ represents separate pours from flask ID 6 and 8 to flask ID 2.

A *recipe* for a complex action specifies the sequence of actions required for fulfilling the complex action. Figure 2 presents a set of basic recipes for VirtualLabs. In our notation, complex actions are underlined, while basic actions are not.

Recipe a in the figure, called mix to same destination (MSD), represents the activity of pouring from two source flasks ($s_1$ and $s_2$) to the same destination flask d. Recipe b, called mix through intermediate flask (MIF), represents the activity of pouring from one source flask ($s_1$) to a destination flask ($d_2$) through an intermediate flask ($d_1$). Recipes can be recursive, capturing activities that students can repeat indefinitely. For example, the constituent actions of the complex action MSD in recipe a decompose into two separate MSD actions. In turn each of these actions can itself represent a mix to same-destination action, an intermediate-flask pour (by applying recipe c or a basic action mix, which is the base-case recipe for the recursion (recipe d). Recipe parameters also specify the type and volume of the chemicals in the mix, as well as temporal constraints between constituents, which we omit for brevity. More generally, the four basic recipes in the figure can be permuted to create new recipes, by replacing MSD on the right side of the first two recipes with MIF or MS. An example of a derivation is the following recipe for creating an intermediate flask out of a complex mix to same destination action and basic mix solution action.

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

Planning is defined as the process by which students use recipes to compose basic and complex actions toward completing tasks using TinkerPlots. Formally, a plan is a tree of basic and complex actions, such that each complex action is decomposed into subactions that fulfill a recipe for some task. A set of nodes $N$ in a plan is said to fulfill a recipe $R_C$ if there exists a one-to-one matching between the constituent actions in $R_C$ and their parameters to nodes in $N$. For example, the nodes $\underline{MSD}[s = 6 + 8, d = 2]$ and $\underline{MS_7}[s = 2, d = 7]$ fulfill the mixing through an intermediate flask recipe shown in equation 1. Each time a recipe for a complex action is fulfilled in a plan, there is an edge from the complex action to its subactions, representing the recipe constituents.

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

*Figure 2. Recipes in VirtualLabs for Mix to Same Destination (MSD) and Mix to Intermediate Flask (MIF) Actions*

Figure 3 shows part of a plan describing part of a student's interaction when solving the Oracle problem. The leaves of the trees are the actions from the student's log and are labeled by their order of appearance in the log. A node representing a complex action is labeled by a pair of indices indicating its earliest and latest constituent actions in the plan. For example, the node labeled with the complex action $\underline{MSD}[s = 1 + 5, d = 3]$ includes the activities for pouring two solutions from flask ID 1 and ID 5 to flask ID 3. The pour from flask ID 5 to 3 is an intermediate flask pour ($\underline{MIF}[s = 5, d = 3]$) from flask ID 5 to ID 3 through flask ID 4.

In a plan, the constituent subactions of complex actions may interleave with other actions. In this way, the plan combines with the exploratory nature of students' learning strategies. Formally, we say that two ordered complex actions interleave if at least one of the subactions of the first action occurs after some subaction of the second action. For example, the nodes $\underline{MSD}[s = 6 + 8, d = 2]$ and $MS_7[s = 2, d = 7]$ fulfill the mixing through an intermediate flask recipe shown in equation 1.

## Plan Recognition

The plan-recognition problem in ELEs is defined as follows: Given a set of temporally ordered actions representing a student's complete interaction sequence with the software, and a set of recipes, output the plan that correctly describes the student's interaction with the software. By "correct," we mean that the plan outputted by the algorithm completely agrees to the plan outputted by a domain expert who was given the same set of inputs. We expand on

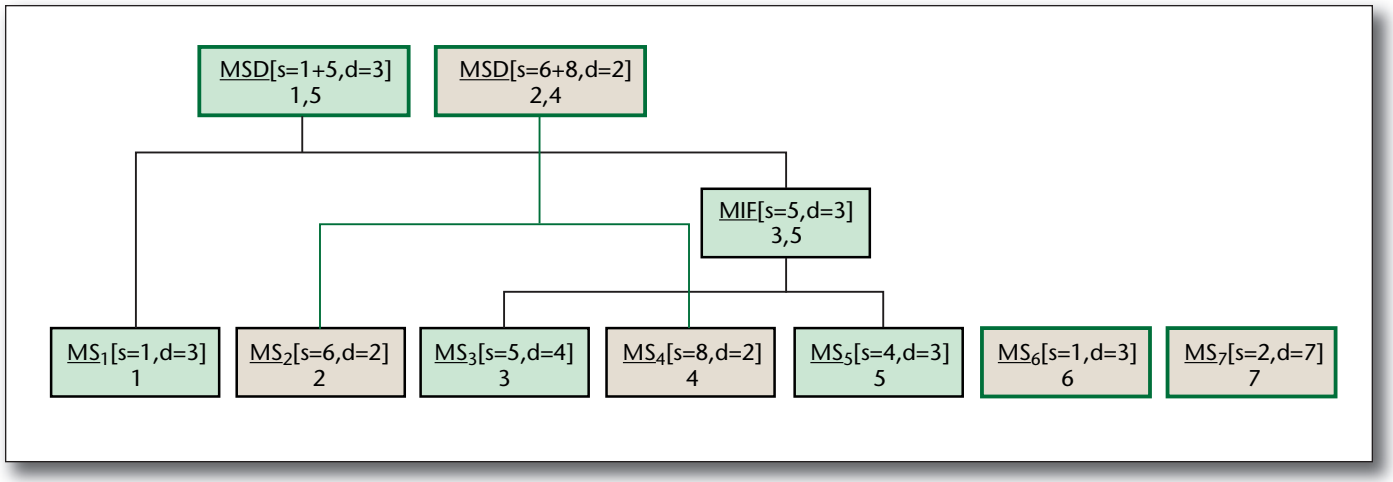


*Figure 3. A Partial Plan for a Student's Log.*

The leaves of the plan refer to the basic actions in the student's plan (numbered in order of appearance in the logs); other nodes in the plan refer to complex activities performed by the student.

this further in the Empirical Methodology section. This problem has been shown to be NP-hard (Gal et al. 2012).

The algorithm we designed to solve the off-line plan-recognition problem, called plan recognition through interleaved sequential matching (PRISM), provides a trade-off between the following two complementary aspects of students' interactions with ELEs.

First, students generally solve problems in a sequential fashion, by which we mean that actions that are (temporally) closer to each other are more likely to relate to the same subgoal. For example, in figure 3 the basic actions $\{MS_1, MS_5\}$ are more likely to fulfill the recipe $\underline{MSD}[s_1 + s_2, d] \rightarrow MS[s_1, d], MS[s_2, d]$ than the basic actions $\{MS_1, MS_6\}$ because they appear closer together in the log. Second, students may interleave between activities relating to different subgoals. For example, in figure 3, the node $MS_4[s = 8, d = 2]$ (which is a constituent action of the complex action $\underline{MSD}[s = 6 + 8, d = 2]$) occurs in between the nodes representing the constituent actions of the complex action $\underline{MIF}[s = 5, d = 3]$.

Before presenting the algorithm we first make the following definitions. Let $P$ denote the current plan at an intermediate step of the algorithm. The frontier of $P$ is all nodes in the plan that do not have parents. For example, the frontier of the plan shown in figure 3 includes the nodes $MS_6$ and $MS_7$.

Let $R_C$ denote a recipe for a complex action $C$. We say that a set of nodes $N$ is frontier compatible with a recipe $R_C$ if $N$ is a subset of the frontier of $P$ and $N$ fulfills $R_C$ in $P$. Intuitively, the nodes in a frontier compatible set of $R_C$ may be used to fulfill the recipe and add $C$ to the plan. In figure 3, the set of nodes $F = \{\underline{MSD}[s = 6 + 8, d = 2], MS_7[s = 2, d = 7]\}$ is frontier compatible with the mix intermediate flask recipe of equation 1.

A match for recipe $R_C$ in $P$ is a triple $M_C = (C, N, \langle i, j \rangle)$ such that $R_C$ is a recipe for completing action $C$, $N$ is a set of nodes that is frontier compatible with $R_C$ in $P$, and $\langle i, j \rangle$ are delimiters specifying the indices corresponding to the earliest and latest actions in $N$. For example, the triple $(\underline{MIF}[s = 6 + 8, d = 7], F, \langle 2, 7 \rangle)$ is a match for the recipe of equation 1. For brevity, we omit the frontier compatible set when referring to matches and write $M_C = (\underline{MIF}[s = 6 + 8, d = 2], \langle 2, 7 \rangle)$.

We define a function FINDMATCH$(R_C, P, D)$ that returns the set of all matches for $R_C$ in $P$, where $D$ is in the frontier compatible set of $R_C$. For example, consider the call

FINDMATCH $(R_C, P, \underline{MSD}[s = 6 + 8, d = 2])$

where $R_C$ is the intermediate flask recipe of equation 1, and $P$ is the plan from figure 3. This call will result in a set containing the single match that was presented earlier: $(\underline{MIF}[s = 6 + 8, d = 7], F, \langle 2, 7 \rangle)$.

The main functions composing the algorithm are shown in figure 4. The algorithm uses a global queue for storing potential matches for updating the student's plan. The queue is sorted lexicographically by the first and second indices in the delimiters of the matches. The core of the algorithm is the CONSIDER-MATCH$(M_C)$ function. This function begins by popping from the queue the first match with indices between $\langle i, j \rangle$ (line 3). The function then recursively updates the plan with all of the matches in $P$ whose delimiters lie between $i$ and $j$ (line 5).

After all inner matches are exhausted, the algorithm checks whether $M_C$ itself can be added to the plan (line 8), which is possible if the set of nodes $N$

is still frontier compatible with $R_C$. If $M_C$ can be added to the plan, the function AddMatchToPlan($M_C$) is called, which adds (1) a node to the plan with label $C$ and delimiters $i$ and $j$, and (2) edges from $C$ to all of the nodes in $M$ as they appear in the plan. At this point, all of the matches in the queue involving nodes represented in $N$ are obsolete because $C$ was added to the plan, so they are no longer in the frontier. Therefore we remove them from the queue (line 18). The final step is a call to the function ExtendMatch($M_C$) to update the queue with new matches in which $C$ is a constituent.

To demonstrate the algorithm, suppose that the plan is initialized to include only the leaves shown in figure 3, corresponding to the student's log. The initial queue will contain the following matches, sorted by their delimiters from left to right as described earlier:

(MSD[$s$ = 1 + 4, $d$ = 3], ⟨1, 5⟩), (MSD[$s$ = 1 + 1, $d$ = 3], ⟨1, 6⟩),
(MSD[$s$ = 6 + 8, $d$ = 2], ⟨2, 4⟩), (MIF[$s$ = 6, $d$ = 7], ⟨2, 7⟩),
(MIF[$s$ = 5, $d$ = 3], ⟨3, 5⟩), (MIF[$s$ = 8, $d$ = 7], ⟨4, 7⟩),
(MSD[$s$ = 1 + 4, $d$ = 3], ⟨5, 6⟩)

The first match to be popped out from the queue in ConsiderMatch(*, (1, 7)) is (MSD[$s$ = 1 + 4, $d$ = 3], ⟨1, 5⟩).

The function performs a recursive call (line 5) to update the plan with matches involving actions occurring within the delimiters of 1 and 5. The next match to be popped off the queue is (MSD[$s$ = 6 + 8, $d$ = 2], ⟨2, 4⟩). The frontier compatible set of this match is {$MS_2$[$s$ = 6, $d$ = 2], $MS_4$[$s$ = 8, $d$ = 2]}.

At this point, the queue subset is empty because there are no matches between the delimiters ⟨2, 4⟩, so the function skips to line 8. Because the nodes ($MS_2$, $MS_4$) in the frontier compatible set of the match have no parents, we can add the MSD[$s$ = 6 + 8, $d$ = 2] complex action to the plan, by calling AddMatchToPlan(MSD[$s$ = 6 + 8, $d$ = 2], ⟨2, 4⟩). This function also removes the following matches, which involve $MS_2$ or $MS_4$ in their frontier compatible set from the queue (line 18):

(MIF[$s$ = 6, $d$ = 7], ⟨2, 7⟩), (MIF[$s$ = 8, $d$ = 7], ⟨4, 7⟩),

(MSD[$s$ = 6 + 8, $d$ = 2], ⟨2, 4⟩)

Finally, in line 10, the function

ExtendMatch(MSD[$s$ = 6 + 8, $d$ = 2], ⟨2, 4⟩)

is called to find all matches that involve the action MSD[$s$ = 6 + 8, $d$ = 2] in their frontier compatible set and add them to the queue (line 27). The only one such is the action (MSD[$s$ = 6 + 8, $d$ = 7], ⟨2, 7⟩).

```
 1: function ConsiderMatch(M_C)        ▷ M_C: a match
 2:     ⟨i, j⟩ ← Limits of the match M_C
 3:     M'_C ←PopFromGlobalQueue(i,j)
 4:     while M'_C not null do
 5:         ConsiderMatch(M'_C)
 6:         M'_C ← PopFromGlobalQueue(i,j)
 7:     end while
 8:     if M_C can be added to P then
 9:         AddMatchToPlan(M_C)
10:         ExtendMatch(M_C)
11:     end if
12: end function

13: function AddMatchToPlan(M_C)
14:     add node (C, i) to P   ▷ i is the index of the earliest
    frontier compatible constituent of the match M_C
15:     for all C ∈ childs(M_C) do
16:         for all M'_C ∈ GlobalQueue do
17:             if C ∈ childrenOf(M'_C) then in P
18:                 remove M'_C from GlobalQueue
19:             end if
20:         end for
21:     end for
22: end function

23: function ExtendMatch(M_C)
24:     for all R_Q ∈ Recipes do
25:         M ← FindMatch(R_Q, P, C)
26:         for all m ∈ M do
27:             add m to GlobalQueue
28:         end for
29:     end for
30: end function
```

*Figure 4. Main Functions of the PRISM Algorithm.*

## Empirical Methodology

We evaluated the algorithm on real data consisting of students' interactions. To demonstrate the scalability of the PRISM algorithm we evaluated it on two different ELEs: the VirtualLabs system as well as an ELE for teaching statistics and probability called TinkerPlots (Konold and Miller 2004) that is used worldwide in elementary school and colleges. In TinkerPlots, students build models of stochastic events, run the models to generate data, and analyze the results. It is an extremely flexible application, allowing for data to be modeled, generated, and analyzed in many ways using an open-ended interface. There are two key differences between TinkerPlots and VirtualLabs. In TinkerPlots, recipes are question dependent and include ideal solutions to specific problems. Second, the TinkerPlots grammar is not recursive.

For VirtualLabs, we used four problems intended to teach different types of experimental and analytic techniques in chemistry, taken from the curriculum of introductory chemistry courses using VirtualLabs in the United States. One of these was the

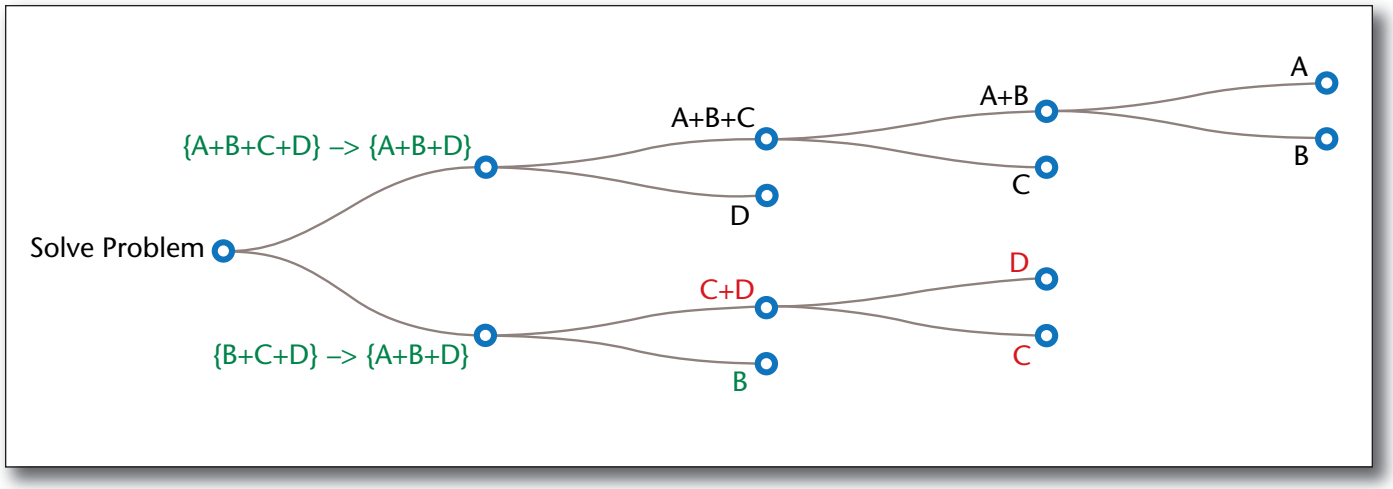


*Figure 5. Visualization of Oracle Plan.*

Oracle problem that was described earlier. Another, called coffee, required students to add the right amount of milk to cool a cup of coffee down to a desired temperature. The third problem, called unknown acid, required students to determine the concentration level and Ka level of an unknown solution. The fourth problem, called dilution, required students to create a solution of a base compound with a specific desired volume and concentration. For TinkerPlots, we used two problems for teaching probability to students in grades 8 through 12. The first problem, called ROSA, required students to build a model that samples the letters A, O, R, and S and to compute the probability of generating the name ROSA using the model. The second problem, called rain, required students to build a model for the weather on a given day and compute the probability that it will rain on each of the next four consecutive days. In contrast to VirtualLabs, recipes in TinkerPlots are question dependent, and describe possible solution strategies for solving each problem.

We compared PRISM to the best algorithms from the literature for each ELE: (1) the algorithm of Gal et al. (2012) for TinkerPlots, which reduces the plan-recognition problem to a CSP, and is complete, and (2) the algorithm of Amir and Gal (2013) for VirtualLabs, which uses heuristics to match recipes to actions in the log, and is incomplete. For each problem instance, a domain expert was given the plans outputted by PRISM and the other algorithm, as well as the student's log. We consider the inferred plan to be "correct" if the domain expert agrees with the complex and basic actions at each level of the plan hierarchy that is outputted by the algorithm. The outputted plan represents the student's solution process using the software.

To illustrate how plans were presented to domain experts, figure 5 shows the visualization of the final plan for the log actions in the leaves of figure 3. The visualization groups all trees in the student's plans as children to a single root node "Solve Oracle problem." Complex nodes are labeled with information about the chemical reactions that occurred during the activities described by the nodes.

For example, the node labeled $A + B + C + D \rightarrow A + B + D$ represents an activity of mixing four solutions together, which resulted in a chemical reaction that consumed substance $C$. The coloring of the labels indicates the type of chemical reaction that has occurred.

Table 1 summarizes the performance of the PRISM algorithm according to accuracy and run time of the algorithm (in seconds on a commodity core i-7 computer). The column SoA (state of the art) refers to the appropriate algorithm from the literature for each problem. For VirtualLabs, this algorithm was suggested by Amir and Gal (2013); for TinkerPlots, this algorithm was suggested by Gal et al. (2012). All of the reported results were averaged over the different instances in each problem. As shown in the table, for VirtualLabs, the PRISM algorithm was able to recognize significantly more plans than did the state-of-the-art ($p = 0.001$) using a proportion based Z test). The state-of-the-art algorithm was not able to recognize 10 plan instances (3 for Oracle; 3 for unknown acid; 3 for coffee; and 1 for dilution). In addition, the PRISM algorithm was able to recognize all of the plans in VirtualLabs that were correctly recognized by the state of the art. In TinkerPlots, PRISM failed to recognize 2 plans (1 for ROSA; 1 for seatbelts). The instances that the algorithms failed to recognize are false negatives that represent bad matches in the plan-recognition process. However, the fact that PRISM achieved comparable or better performance than both state-of-the-art algorithms speaks well for its performance.

We conclude this section with discussing the limitations of PRISM. First, PRISM is not a complete plan-

| | | Number of Instances | PRISM Accuracy | PRISM Run Time | SoA Accuracy | SoA Run Time |
|---|---|---|---|---|---|---|
| Virtual Labs | Camping | 2 | 100% | 1.107 | 100% | 0.548 |
| | Coffee | 9 | 100% | 21.075 | 67% | 2.781 |
| | Dilution | 4 | 100% | 1.720 | 75% | 0.437 |
| | Oracle | 6 | 100% | 9.675 | 50% | 2.689 |
| | Unknown Acid | 7 | 100% | 53.561 | 57% | 4.441 |
| TinkerPlots | Earrings | 10 | 90% | 27.514 | 100% | 1.004 |
| | Rosa | 25 | 100% | 4.430 | 100% | 0.049 |
| | Rain | 18 | 100% | 6.334 | 100% | 38.576 |
| | Seatbelts | 11 | 90.9% | 6.064 | 100% | 1.121 |

*Table 1. Results of PRISM Algorithm.*

recognition algorithm, as can be attested by the fact that it failed to recognize 2 out of 64 instances in TinkerPlots. Second, it was significantly slower than the state-of-the-art approaches. This is not surprising given its worst case complexity. However, PRISM is designed to run offline after the completion of the student's interaction. Therefore an average run time of 30 seconds is a low price to pay given the significant increase in performance and its ability to generalize across different ELEs.

## Visualizing Plans to Students

In this section we demonstrate the benefits of plan-recognition technology in education, in that visualizing expert solutions to students improves their performance on new problems. Presenting such "worked examples" to students has already been shown to provide effective instructional strategies for teaching complex problem-solving skills (Van Merriënboer 1997) and is widely accepted as an effective learning technique (Catrambone 1994). Our hypothesis was that showing plans of sample problems in TinkerPlots will improve students' performance on new problems when compared to a default presentation method that consisted of an ordered list of their activities. Such a list is the sole visualization technique currently available for TinkerPlots. Specifically, we expected students who were shown plans to be able to solve new problems that required generalization of mathematical concepts more quickly, with fewer mistakes and with committing less redundant actions when compared to students who were shown the list.

We used the following problem, called COIN, to visualize worked examples to students:

A fair coin with a side of "0" and a side of "1" is tossed three times. What is the average expected sum of the tosses?

The all-purpose sampling object in TinkerPlots is called a *sampler.* A sampler is an object into which the user can place random devices, including spinners and mixers, in order to create a stochastic model of the world. Examples of basic actions in TinkerPlots can be adding or running a sampler. Actions in TinkerPlots represent higher-level activities such as flipping a coin or solving the coin problem.

Figure 6 shows a snapshot of the TinkerPlots desktop when solving the COIN problem. The sampler mechanism shown in the left contains a coin with the elements 0 and 1. The parameter number of draws in the sampler is set to 10 to represent 10 questions. The parameter number of repetitions in the sampler is set to 2,000 so that the number of samples will produce a representative sample. When a sampler is run, it generates data according to the distribution defined by the parameters of its model. The results of this sample are shown on the right. Also shown is the formula window, which is used to compute the sum of the three tosses for each instance.

Using the PRISM algorithm, we created a visualization of a worked example of a solution to the COIN problem. Figure 7 shows part of this visualization as a tree of basic actions (leaf nodes) and complex actions (parent nodes). The root of the plan (Correct_Solution) decomposes into three constituent complex actions for creating the coin model (Create_Coin_Model), running the model (Generate Results) and computing the average sum of tosses (Compute Average Sum). In turn, the complex action Create_Coin_Model decomposes into the basic action of adding a sampler (add sampler), the complex action of creating a coin (Create_Coin), and the basic actions of setting the number of times the coin should be tossed (set_draws_in_sampler) and the number of times the experiment should be repeated (set_repetitions). The leaves of the tree correspond to the students' action sequences recorded by TinkerPlots. Actions appear in left-to-right order in a

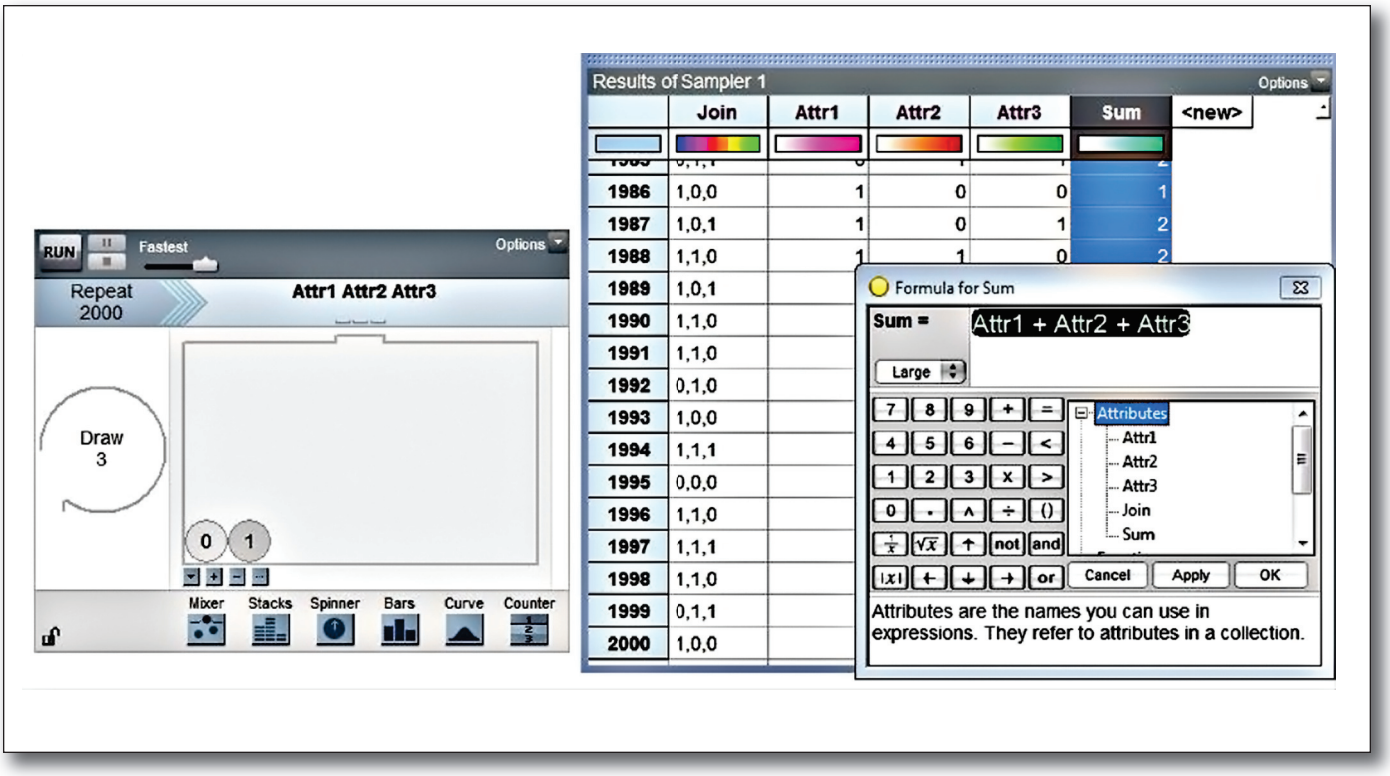


*Figure 6. Solving the COIN problem Using TinkerPlots.*

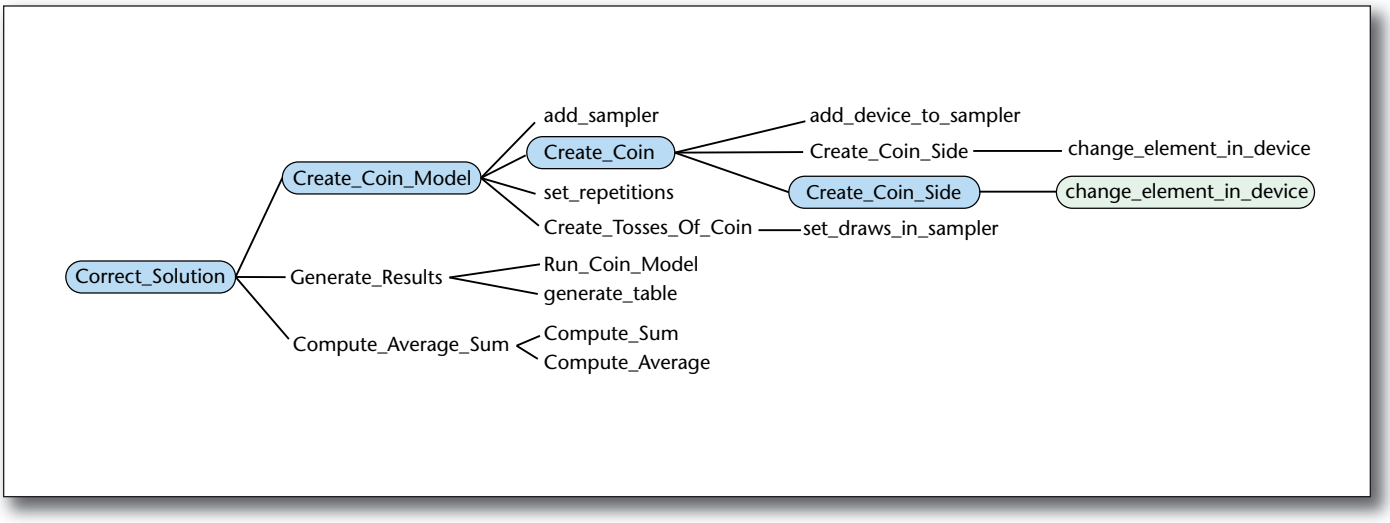


*Figure 7. Tree Visualization of the Expert Solution to the COIN Problem.*

way that is consistent with the temporal order of students' activities. For expository convenience, only part of the plan is shown in the figure (for example, the Compute_Sum complex action is not presented). The plan visualization was introduced to the students using an interactive tool that allowed them to drill down to reveal the constituents actions of each node in the tree.

The list visualization presented students with a bulleted list of the basic actions performed to solve the example problem. This visualization is obtained from a linear sequence of temporally ordered actions. It represents the default support that is currently available to students using the software. A partial list of these actions is shown in figure 8.

## Participants

The study involved 61 first-year undergraduates students enrolled in a statistics and probability course for engineering majors at Ben-Gurion University. The study was carried out during the middle of the semester after the students had acquired basic knowledge of probability theory and undergone a midterm. All students were given a home exercise to familiarize with the TinkerPlots software. In total 32 students were presented with a plan solution, and 29 were presented with a list solution.

The study was conducted in a designated lab in which each student was situated in front of a computer. In the first part of the study, all students were provided with the COIN problem description in writing, and the expert solution to the problem was subsequently presented to them using the list or plan visualization, depending on their assigned condition. In addition, all students were shown an (identical) snapshot of the TinkerPlots desktop following the solution procedure. The students were asked several comprehension questions about the solution, such as whether (and why) the interaction shown to them constitutes a correct solution to the COIN problem; to indicate the role in the solution of one of the actions in interaction; to explain the solution to a friend using free text. There were two purposes for these questions: First, to confirm that the students comprehended the solution; and second, to compare between students' self-explanations of the solution to the COIN problem in the two experimental conditions. Students were allocated up to 20 minutes to complete this part of the study.

In the second part of the study, students were asked to solve two new problems in sequence using TinkerPlots. Students were allocated up to 30 minutes to complete this part of the study. The problems were taken from the curriculum of an introductory course in probability. We wanted the problems to be nontrivial but still possible to solve by the majority of students in the allocated 30 minutes. We chose problems whose solutions exhibited similar concepts (reasoning about combinations and events in sample space). The test problem, called DICE, did not mention explicitly the notion of expectation, and it required students to reason about the disjunction of complex events that relate to the sample space.

> John and Mary compete in a dice-tossing game. They take turns tossing a die, and sum the result of each toss. The winner is the first to accumulate more than 10 points. Compute the probability that (1) John will win after two rounds of the game. (2) Either John or Mary will win after two rounds of the game.

We collected the TinkerPlots log from each student's interactions as well a snapshot of their desktop recording during the activity.

```
Add device 1 to sampler1
Change the label of element 0 in device 1 in sampler 1 to "0"
Change the label of element 1 in device 1 in sampler 1 to "1"
Set the repetitions in sampler 1 to "1500"
Set the draws in sampler 1 to "3"
Run sampler 1 and generate columns Draw1,Draw2 and Draw3
Add an attribute named "Sum" to table 2
Edit the formula of attribute "Sum" in table 2 to  Draw1+Draw2+Draw3
Add plot 3
Drag the attribute "Sum" to plot 3
Divide the values in plot 3
Average on plot 3 of attribute "Sum"
```

*Figure 8. List Visualization of the Expert Solution to the COIN Problem.*

| | Time (Minutes) | Number of Actions | Redundancy |
|---|---|---|---|
| Plan | 9.01 | 39.65 | 28% |
| List | 12.47 | 57.06 | 46% |

*Table 2. Performance Measures on DICE Problems for Students in Plan and List Conditions.*

# Results

We hypothesized that students assigned to the plan condition would exhibit better performance than students assigned to the list condition when using TinkerPlots to solve the test problems. We measured student performance using the following metrics: the length of interaction (in minutes); the total number of actions in an interaction; the ratio of redundant actions in an interaction, which represent mistakes; and exogenous actions that do not play a part in the students' solution.

We provide a description of students' performance for the DICE problem. All of the results we report below were statistically significant $p = 0.04$ using a nonparameterized two-tailed Mann-Whitney test. We found that the plan visualization significantly improved students' performance across all measures. These results are summarized in table 2.

Specifically, the average interaction length of the students in the plan condition (AVG = 9.01 minutes, SD = 3.92) was significantly shorter than the average interaction length for students in the list condition

(AVG = 12.47 minutes, SD = 6.6). Additional analysis reveals that the total number of actions in a student's interaction (AVG = 39.65, SD = 11.79) was significantly lower than the total number of actions in the list condition (AVG = 57.069, SD = 32.628). The ratio of redundant actions of the plan condition (28 percent) was significantly lower than the ratio of redundant actions for students in the list condition (46 percent). Although the number of total actions in the plan condition (AVG = 47, SD = 12.01) was slightly higher than in the list condition (AVG = 43.66, SD = 18.11), the ratio of redundant actions in the plan condition (33 percent) was lower than the ratio of redundant actions in the list condition (43 percent). This indicates that visualizing the hierarchical aspect of the example facilitates students' ability to generalize mathematical and structural concepts across new problems (in our case, the use of expectation to reason about events in the sample space).

Finally, there were striking differences in the way students explained the solution to the COIN problem based on their respective visualization condition. Overall, 75 percent of the students in the plan condition used and referred to subgoals when describing the solution to the COIN problem, as compared to 66 percent of the list students. In our study, subgoals represent higher-level activities such as generating and running a sampler, projecting the results to a plot, and computing the average sum of a random variable. These activities recur in all three of the problems in the study, and recognizing and internalizing these concepts may have contributed to the success of the plan visualization. The students in the list condition were far less likely to use such concepts when describing the problem.

## Conclusion and Future Work

This article proposed new algorithms for recognizing students' plans in exploratory learning environments, which are open-ended and flexible educational software. Our algorithm is shown to outperform (or perform comparably with) the state-of-the-art plan-recognition algorithms for two different ELEs for teaching chemistry and statistics. It is also the first recognition algorithm that is shown to generalize successfully to several ELEs. It demonstrates that using hierarchical visualizations of expert solutions positively affects students' problem solving in ELEs. We are currently applying these results in several directions. First, we are designing plan-recognition algorithms for ELEs that do not depend on a predefined grammar. Second, we are designing intelligent tutors in ELEs that use plan recognition to guide their interactions with students. Finally, we are designing automatic methods for aggregate analysis of students' activities that is based on students' plans.

## Acknowledgements

This research is supported in part by EU grant FP7-ICT-2011-9 no. 600854 and by Israeli Science Foundation grant no. 1276/12. Reuth Dekel was supported in part by a grant from the Israeli Chief Scientist for advancing the role of women in science.

## References

Amershi, S., and Conati, C. 2006. Automatic Recognition of Learner Groups in Exploratory Learning Environments. In *Intelligent Tutoring Systems,* Lecture Notes in Computer science, Volume 4053, 463–471. Berlin: Springer. dx.doi.org/10.1007/11774303_46

Amir, O., and Gal, Y. 2013. Plan Recognition and Visualization in Exploratory Learning Environments. *ACM Transactions on Interactive Intelligent Systems* 3(3): Article 20.

Avrahami-Zilberbrand, D., and Kaminka, G. 2005. Fast and Complete Symbolic Plan Recognition. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence* (IJCAI). Amsterdam: Morgan Kaufmann.

Carberry, S. 2001. Techniques for Plan Recognition. *User Modeling and User-Adapted Interaction* 11(1–2): 31–48. dx.doi.org/10.1023/A:1011118925938

Catrambone, R. 1994. Improving Examples to Improve Transfer to Novel Problems. *Memory and Cognition* 22(5): 606–615. dx.doi.org/10.3758/BF03198399

Cocea, M.; Gutierrez-Santos, S.; and Magoulas, G. 2008. S.: The Challenge of Intelligent Support in Exploratory Learning Environments: A Study of the Scenarios. In *Proceedings of the First International Workshop in Intelligent Support for Exploratory Environments,* CEUR Workshop Proceedings Volume 381. Aachen, Germany: RWTH Aachen University.

Conati, C.; Gertner, A.; and VanLehn, K. 2002. Using Bayesian Networks to Manage Uncertainty in Student Modeling. *User Modeling and User-Adapted Interaction* 12(4): 371–417. dx.doi.org/10.1023/A:1021258506583

Gal, Y.; Reddy, S.; Shieber, S.; Rubin, A.; and Grosz, B. 2012. Plan Recognition in Exploratory Domains. *Artificial Intelligence* 176(1): 2270–2290. dx.doi.org/10.1016/j.artint.2011.09.002

Geib, C., and Goldman, R. 2009. A Probabilistic Plan Recognition Algorithm Based on Plan Tree Grammars. *Artificial Intelligence* 173(11): 1101–1132. dx.doi.org/10.1016/j.artint.2009.01.003

Kautz, H. A. 1987. A Formal Theory of Plan Recognition. Ph.D. Dissertation, University of Rochester Department of Computer Science, Rochester, NY.

Koedinger, K. R.; Anderson, J. R.; Hadley, W. H.; Mark, M. A. 1997. Intelligent Tutoring Goes to School in the Big City. *International Journal of Artificial Intelligence in Education* 8(1): 30–43.

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

Lochbaum, K. E. 1998. A Collaborative Planning Model of Intentional Structure. *Computational Linguistics* 4(4): 525–572.

Pynadath, D., and Wellman, M. 2000. Probabilistic State-Dependent Grammars for Plan Recognition. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence,* 507–514. San Francisco: Morgan Kaufmann.

Roll, I.; Aleven, V.; and Koedinger, K. R. 2010. The Invention

Lab: Using a Hybrid of Model Tracing and Constraint-Based Modeling to Offer Intelligent Support in Inquiry Environments. In *Intelligent Tutoring Systems,* Lecture Notes in Computer Science Volume 6094, 115–124. Berlin: Springer. dx.doi.org/10.1007/978-3-642-13388-6_16

Van Merriënboer, J. J. 1997. *Training Complex Cognitive Skills: A Four-Component Instructional Design Model for Technical Training*. Englewood Cliffs, NJ: Educational Technology Publications.

VanLehn, K.; Lynch, C.; Schulze, K.; Shapiro, J. A.; Shelby, R. H.; Taylor, L.; Treacy, D. J.; Weinstein, A.; and Wintersgill, M. C. 2005. The Andes Physics Tutoring System: Lessons Learned. *International Journal of Artificial Intelligence and Education* 15(3): 147–204.

Yaron, D.; Karabinos, M.; Lange, D.; Greeno, J.; and Leinhardt, G. 2010. The ChemCollective–Virtual Labs for Introductory Chemistry Courses. *Science* 328(5978): 584. dx.doi.org/10.1126/science.1182435



**Oriel Uzan** is a master's student in the Department of Information Systems Engineering at Ben-Gurion University. He completed his BSc in engineering at Ben-Gurion University in 2011. His research interests include plan recognition and visualisation in exploratory learning environments.

**Reuth Dekel** is a master's student in the Department of Information Systems Engineering at Ben-Gurion University. She completed her BSc in computer science at Ben-Gurion University in 2012. Her thesis focuses on novel algorithms for online plan recognition.

**Or Seri** is a graduate student at the Department of Brain and Cognitive Science at Ben-Gurion University. She completed her BSc in math, computer science, and political science at Bar-Ilan University in 2004. Her thesis focuses on visualizing expert solutions to students in e-learning.

**Ya'akov (Kobi) Gal** leads the human-computer decision-making research group in the Department of Information Systems Engineering at Ben-Gurion University. He received his PhD from Harvard University in 2006. He is a recipient of the Wolf foundation's 2014 Krill prize for young Israeli scientists, a Marie Curie International fellowship for 2010, a two-time recipient of Harvard University's Derek Bok award for excellence in teaching, as well as the School of Engineering and Applied Science's outstanding teacher award.