Measuring and Visualizing Learning with Markov Models

Fatima Abu Deeb
Computer Science Department
Brandeis University
Waltham, MA 02453, USA
Email: abudeebf@brandeis.edu

Kristian Kime
Computer Science Department
Brandeis University
Waltham, MA 02453, USA
Email: kristian@brandeis.edu

Rebecca Torrey
Mathematics Department
Brandeis University
Waltham, MA 02453, USA
Email: rtorrey@brandeis.edu

Timothy Hickey
Computer Science Department
Brandeis University
Waltham, MA 02453, USA
Email: tjhickey@brandeis.edu

***Abstract*—In this paper we introduce new tools for measuring and visualizing the learning process of students in large classes where students are required to use a computer-supported problem solving learning environment (PSLE) either in class or as homework. These learning environments give students unlimited opportunities to propose a solution to the problem and give them immediate feedback. We show how to create a graphical representation, the Problem Solving Markov Model (PSMM), of the set of all attempted solutions proposed by the class. Each node in the PSMM consists of an equivalence class of attempted solutions and each edge corresponds to a transition from one attempted solution to another in one step. The edges are typically labeled with the number of times a students went directly from one attempted solution to the next, or with the probability of going between the two nodes. We give examples from two tools: CalcTutor for Calculus problems, and Spinoza for Java or Python programming, and we provide several pedagogical applications of PSMMs which give the instructor a simple way to obtain a highly nuanced understanding of mastery of the problem solving process for individual students and for the class as a whole.**

I. Introduction

In the past few years with greater governmental emphasis on increasing the STEM workforce, the enrollments in many engineering courses have expanded rapidly. In Computer Science in the US, the number of majors has nearly doubled in the past 10 years [1]. Growth in engineering and other STEM majors has also expanded recently creating a need for more effective strategies for teaching large classes in computer science and related fields. Active Learning is one pedagogical approach that has been shown to increase the effectiveness of classroom instruction, especially for large classes [2], [3] and most of these techniques involve having students engage in problem solving activities during the class. One of the challenges in teaching such a large interactive class is assessing the level of understanding of the students during the interactive activities.

In this paper, we describe a computer-assisted approach to assessment which provides the instructor with a deep and nuanced view of the problem solving behavior of all students in the class. This view is called the Problem Solving Markov Model (PSMM). It is created by having the students solve problems in a computer-mediated Problem Solving Learning Environment (PSLE) in which they make multiple attempts at solving a problem by submitting their proposed solutions to a computer application which gives feedback and records their response. Students are encouraged to repeat the process until they find the solution, but some will give up before succeeding. By recording the data and introducing equivalence relations on attempted solutions, the system can build a stochastic model of the student problem solving behavior for this problem.

The examples in this paper will come from two Problem Solving Learning Environments: an online Java Programming IDE Spinoza [4], [5] and an online gamified Calculus tutor CalcTutor [6]. This approach will work for any Problem Solving Learning Environment where there are relatively few possible solutions to the problem and a potentially large numbers of students using the system. Introductory courses in any area are good examples of this type of domain as the students are learning to solve simple problems and many students are likely to propose identical incorrect solutions, with respect to some efficiently computable equivalence relation.

There are many pedagogical interventions that are made possible by having real-time access to PSMMs generated by a Problem Solving Learning Environment. The simplest application is to rapidly identify the most common incorrect attempts and to use that information to help clear up the underlying misconceptions. The PSMM approach also provides a view of all sequences of incorrect attempts that students made on the way to the solution.

II. Related Work

This work builds on the research in Problem Solving Learning Environments (PSLEs) which are computer applications which allow students to practice problem solving skills and which simultaneously collect information about their problem solving strategies and use this information to assess their mastery of certain skills and concepts. Riemann, Kickmeier-Rust, and Albert give an excellent overview of PSLEs and their use in the assessment of students skills using Knowledge Space Theory [7]. Their approach is to associate each problem to a set of skills and to build a Bayesian model for estimating the mastery of those underlying skills by the students' success or failure on a large number of particular problems. Our focus is to look more deeply at each attempt at solving a problem and to build tools to help the instructor of a large class understand the kinds of problems students are making as well as the strategies they are using to solve problems.

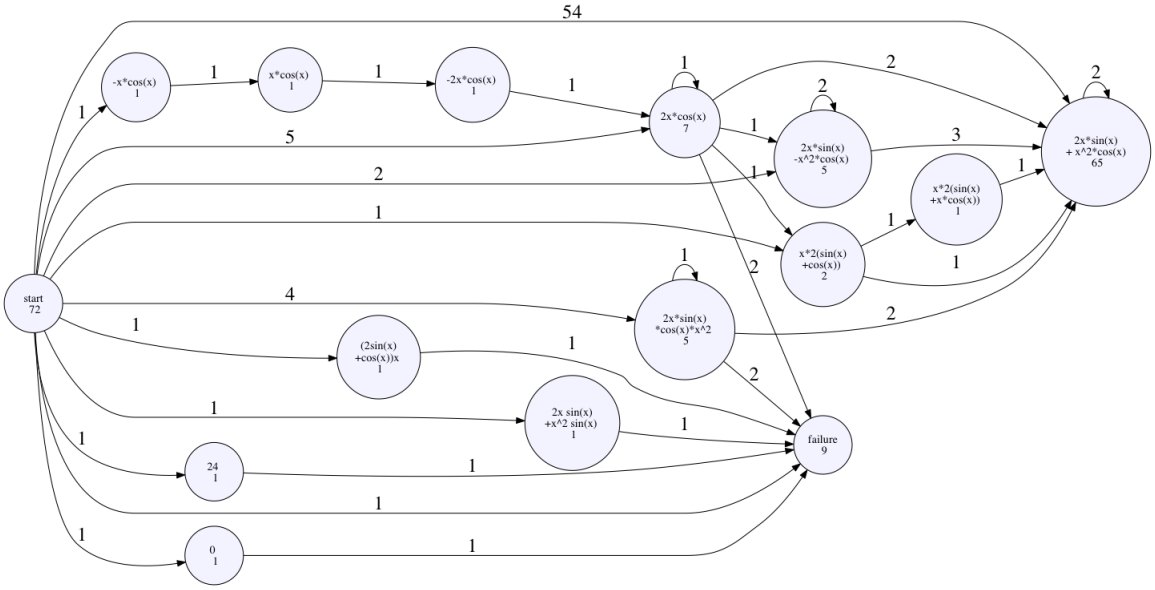

Fig. 1. The plain PSMM for finding the derivative of $x^2 \sin(x)$

There is a long history of studying student errors to better understand the learning process in a particular domain. Soloway and Spohrer were among the first to analyze the errors made by novice programmers [8]. They analyzed all attempts that students in a Pascal programming class made on 10 programming assignments. Their main conclusions were that each problem had a small number of very high frequency bugs illustrating common errors and that these errors were generally not about the programming language semantics but instead were due to poor problem solving skills and particular kinds of incorrect problem solving strategies. More recently, Pillay and Jugoo have also shown that the common errors in Java programming are primarily due to poor problem solving skills and not due to misunderstandings of the semantics of the programming language [9]. Ahmadzadeh, Elliman and Higgins used a similar approach to study debugging skills of novice students and found a strong correlation between debugging skills and course grade [10]. They also observed that programming and debugging are separate cognitive skills in the sense that some students were strong debuggers but not strong programmers and vice versa. Both programming and debugging are examples of problem solving skills needed by novice programmers.

Our work is most similar to recent work by Piech and colleagues at Stanford who developed a graphical model illustrating the problem solving process of a class [11]. Their approach was to analyze students attempts to solve Karel the Robot problems. They clustered the students using various similarity measures on their programs (e.g. word frequency, or abstract syntax trees of their code) and developed Hidden Markov Models approximating the problem solving strategies for different clusters. They found that problem solving ability as measured by classifying the students problem solving path, was strongly correlated with course grade.

Our work is similar to that of Piech in that we also build graphical representations of the problem solving strategies of students in a large class, but it differs in that our representation is simpler and fully captures all problem solving strategies of all students. We are also able to automatically discover the most common programming and debugging errors of students in a class, but we do this in the context of their problem solving ability which provides greater insights into how errors manifest during the problem solving process.

III. Problem Solving Markov Models

Education, especially in engineering, consists to a large extent of teaching students how to solve problems in particular domains, often using computer applications to help them find solutions, or submit them for grading. By observing the sequence of actions students take to solve particular problems the instructor can infer more about their understanding and facility with the problem solving techniques than she can by simply observing the finished product. Problem Solving Markov Models (PSMMs) are a visually informative way of viewing the problem solving behavior of a large group of students at a glance. They also provide the basis for an intuitive interface for accessing and analyzing the student problem solving behavior.

A. The Problem Solving Markov Model for a particular exercise

For this section, we assume that a group of students are all solving a problem in a particular domain (e.g. Calculus or Java Programming) using a program that will accept their attempted solution, give them feedback and then allow them to make another attempt, repeatedly until they give up or get a correct answer. The sequence of all attempted solutions to a problem is called the **Problem Solving Log**. For a large class

(50-500 students), it is challenging enough to look over their final solutions let alone analyze all of the incorrect submissions for all students. The Problem Solving Markov Model is a graphical representation of the problem solving log data that allows the instructor to rapidly acquire a large variety of useful information from the log files using appropriate visualization tools, which we discuss in this paper.

We let $\mathcal{D}$ be the set of all attempted solutions produced by the students and then define an equivalence relation $\approx$ on $\mathcal{D}$. For example, in the CalcTutor application [6] described in the next section, one kind of problem involves taking the derivative of a function of a single variable x constructed from arithmetic operators, constants, trig and exponential functions, etc. So $\mathcal{D}$ in this case is a set of functions entered by the students. The simplest equivalence relation is simply string equality - did they type in exactly the same characters? A more sophisticated one is mathematical equivalence – are they the same function? We approximate the mathematical equivalence relation by evaluating the two functions on 80 randomly selected points and verifying that the results are sufficiently close. This isn't foolproof, but in practice it is very effective. For the domain of Java programming, the Spinoza system [5], described in a later section, asks students to write a Java program that will pass a set of 10-20 unit tests, some of which are randomly generated. So $\mathcal{D}$ in this case is a set of Java programs entered by the students. Again we could use textual equivalence for the programs, but this is too strong. We could also test for the equivalence of their Java class files, which allows programs to have different variable names and comments, but otherwise requires the programs to be identical. Finally, we can say two programs are equivalent if they get the "same" values on a selected set of unit tests. This is the method we have found to be most illuminating, and therefore we tend to use this equivalence relation the most often.

We construct the **Problem Solving Markov Model** from the Problem Solving Log by appending a start node to the beginning of each student's log and either a success or a fail node to the end of their log, and we replace each attempted solution with its equivalence class. The union of all of the Problem Solving Logs for all students working on a particular problem forms a directed graph where the number of edges from one node n to another n' is the number of times a student first submitted n as an attempted solution and then submitted n' immediately after. If $\mathcal{D}$ is the set of all attempted solutions, then let $\bar{\mathcal{D}}$ denote the equivalence classes in $\mathcal{D}$. The nodes of the PSMM are the elements of $\bar{\mathcal{D}}$. Also, rather than actually drawing N edges between two nodes, we usually just draw a single edge and label it with the number of times a user went directly between the two nodes in that direction.

Figure 1 shows the Problem Solving Markov Model for a group of students trying to find the derivative of $x^2 \sin(x)$ soon after first learning the necessary differentiation rules. This example is not the most interesting pedagogically, but is small enough to allow the key ideas to be clearly explained.

The nodes in the Figure are labeled by just one of the attempted solutions in the equivalence class as well as the actual number of attempts in that class. The edges are labeled by the number of students who went from one attempted solution to the next in exactly one step. The equivalence relation used to group solutions is mathematical equivalence as described above.

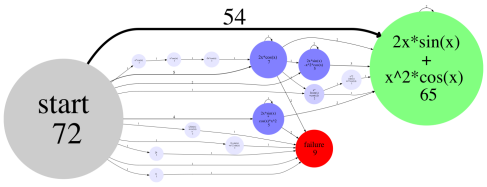

Fig. 2. The visually enhanced PSMM for finding the derivative of $x^2 \sin(x)$. The interface allows the instructor to pan and zoom into the image to read the labels on the smaller nodes.

In this simple example we see that 54 of the 72 students in the class arrived at the correct solution $2x \sin(x) + x^2 \cos(x)$ in one step and these students are represented by the edge that goes from the start node over the top to the correct solution node. Of the remaining 18 students, half eventually got the right solution and half eventually gave up. There were 12 (mathematically) different attempts and four of those were attempts tried at least twice and represent the common misunderstandings. Self-loops on the nodes indicate cases where a student made a purely syntactic change which didn't change the underlying mathematical function (e.g. replacing $x * x$ with x^2). Observe that the correct solution node has 65 attempts but corresponds to only 63 students as there is a self-loop where two students modified their correct answer, perhaps to make it "cleaner".

We have found that paths in the PSMM which consist of a long sequence of unshared attempts usually indicate students who have little to no understanding of how to solve the problem and are thrashing around. On the other hand, for students who generally understand the problem solving process they may share common misunderstandings and these result in attempted solutions that several students independently submit. For our simple example, there are four shared incorrect attempted solutions:

1) $2x \cos(x)$ – using the "rule" $(f * g)' = f' * g'$
2) $2x \sin(x) - x^2 \cos(x)$ – using the "rule" $\sin' = -\cos$
3) $2x * \sin(x) * x^2 * \cos(x)$ – using the "rule" $(f * g)' = f' * g \ * \ f * g'$
4) $2 * x(\sin(x) + \cos(x))$ – using the "rule" $(f * g)' = f' * (g + g')$

B. Using Color and Size to Convey Information

Although the graph in Figure 1 is much easier to understand than reading through 72 log files, too much of the information is textual and not visually apparent. We have developed another view of PSMMs which uses the size of the node to indicate the number of attempts in that equivalence class and the color to indicate the type of node (red for failure, green for success, gray for start, blue for incorrect attempt). Figure 2 shows the corresponding graph for the $x^2 \sin(x)$ example.

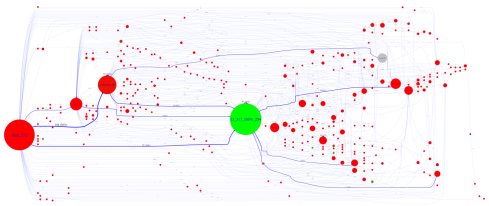

Fig. 3. The PSMM for 272 students writing a method to convert strings representing European-style "DD.MM.YYYY" dates to US format "MM.DD.YYYY". Each node is labeled in the form id_N_P%_U where id is a unique id for the node, N is the total number of programs in that equivalence class, P is the percentage of unit tests that were correct for those programs, and U is the number of students who submitted programs in that equivalence class.

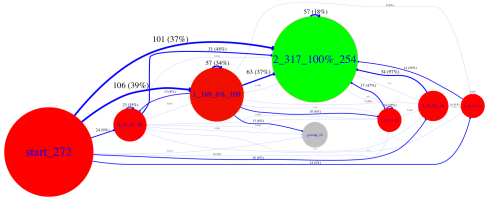

Fig. 4. The Reduced PSMM for the Eurodate example ignoring nodes of size under 20. Filtering out those attempted solutions shared by fewer than 10% of a class typically results in a handful of common errors, often with a clear graph structure.

The width of the edges is also proportional to the number of corresponding edges in the problem solving log files.

Spinoza and CalcTutor both use the GraphViz application to generate and view these graphs and it allows one to easily zoom and pan so that the text in the small nodes can be viewed normally if desired, but the main focus is on the most common errors and the fraction of students who made a particular attempt. Spinoza also has a web-based interface which allows one to click on a node and then see the corresponding student code in a separate pane.

C. Reduced Markov Models

For very large classes, these PSMMs can have thousands of nodes. We have found that a reduced version of the PSMM which focuses only on the most common errors provides a more effective way to understand the most common errors in the problem solving process itself. These reduced Markov Models are constructed by creating a modified Problem Solving Log file that omits any attempts that were not shared by at least k other students. This reduced log file is then used to create a Markov model. Students whose attempts were not shared by anyone will be represented by an edge that goes directly from the start node to the success or failure node, depending on whether they eventually found a correct solution. Fig. 3 shows an example of a PSMM for a class with 272 students working on a program to switch between different date formats. The sizes of the nodes represent the number of students that tried that solution. Close examination of this data allows one to easily identify students who repeatedly make small changes using a trial and error approach. There are also clear pathways to success that multiple students follow. The reduced PSMM in Fig. 4 highlights only the most common misunderstandings and shows how students moved from one erroneous program to another in the problem solving process.

IV. CalcTutor Markov Models

CalcTutor [6] is an application that was originally designed to study a particular Game Theory-based approach to collaborative learning called the Teacher's Dilemma [12]. It is similar to the WebWork application [13] that has been widely used and studied as an online Math homework tool with automatic grading [14]. There is evidence that allowing students to get immediate feedback on their attempted solutions and to make multiple attempts is a more effective strategy for learning in that students spend less time studying the material and achieve the same results [15].

One of the ways in which CalcTutor differs from WebWork is that it allows students to easily create quizzes for each other without having to know (or specify) the correct answer. To achieve this goal, CalcTutor restricts the classes of problems that students and instructors are allowed to create and provides a simple and intuitive interface for creating and solving problems. This enables instructors to easily create online quizzes and pre/post tests.

A. The CalcTutor Application

CalcTutor is an online system primarily based around creating and answering questions appropriate for an introductory level college calculus class. Teachers use the system to create an online class which students can sign up for (the classes can be open enrollment or a code can be required to register). Teachers then create quizzes for students or students can challenge each other to games in which they create a custom set of questions specifically for the other student they are playing.

The system has a number of question types: finding the derivative of a function, finding the tangent line at a point, identifying the graph of a function, etc. Each problem type has an easy-to-use interface which allows quick creation and answering. These interfaces all have a number of useful features:

- Building and answering questions is fast. Once a user understands the interface it only takes a matter of seconds to create or answer a question.
- Answers are automatically checked by the system, which means teachers do not need to do any grading and students get immediate feedback.
- Improper questions and answers are automatically rejected. The interfaces are designed so a user cannot enter a nonsensical (or impossible to solve) question or answer.
- Students can attempt to answer a question as many times as they like and they receive immediate feedback about whether their answer was correct or not.

An example of one of the question types is the "Derivative" question. Here the question creator enters a function (of x) and the student answering the question must enter its derivative.

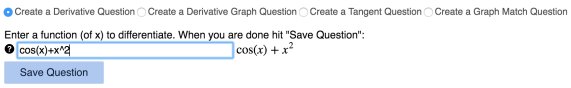

Fig. 5. CalcTutor interface for creating a "Derivative" question

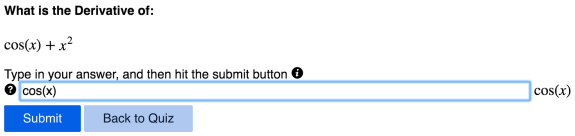

Fig. 6. CalcTutor interface for answering a "Derivative" question

The question and answer are specified using "ASCII" math but they are also shown a rendered version of the formula to minimize confusion. For example "cos(x) + x^2" would be interpreted as the obvious $cos(x) + x^2$ as seen in Fig. 5. Similarly, students answering the question are shown the function for which they need to take the derivative and they have a text box to input the answer (see Fig. 6). The system will automatically reject a question or answer if it cannot interpret the text as a function of x (or if the question function does not have a derivative). The system has the capability to do symbolic differentiation and function equivalence testing to check whether the answers are correct.

B. CalcTutor Pedagogy

We have used CalcTutor in three ways in the introductory Calculus class. The first is to have students work on a set of questions in class while the instructor examines their progress in real-time from her laptop. This allows the instructor to discover which students are struggling and provide support in real-time.

The second approach is to require students to take a CalcTutor pre-test soon after being introduced to the differentiation rules and then taking a very similar CalcTutor post-test after they have completed the unit on differentiation rules. By comparing the pre- and post-test PSMMs for each of the questions we can rapidly identify students who are still struggling even though they eventually get the correct answer, and we can diagnose their conceptual difficulties.

The third approach is to require students to play a certain number of games with their classmates or to gain a certain number of student and teacher points by playing these games. Each game requires them to create short quizzes for each other and they get scored based on whether their questions are "good" and whether their answers are correct.

V. Spinoza Markov Models

Spinoza is a web-based IDE designed to be used to help teach Java and/or Python programming as an in-class activity for large CS1 classes. The major design goal was to allow the instructor to assign simple programming problems as an in-class activity and to monitor the progress of the students as they try to solve the problem in real-time. The interface also needed to allow the instructor and the teaching assistants to

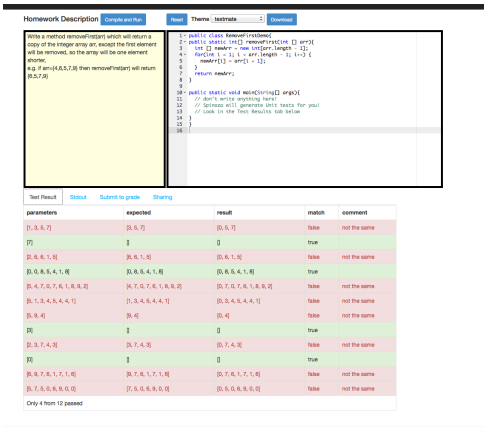

Fig. 7. Student View of a Programming Problem

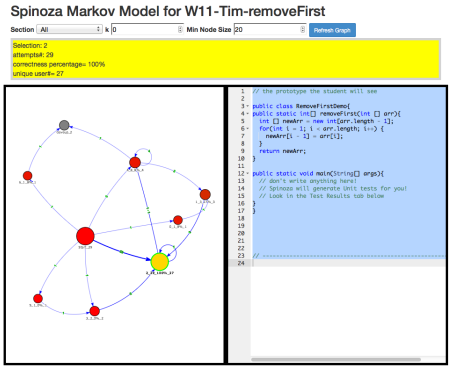

Fig. 8. Spinoza Markov Model View

view and run any selected student's code, and to rapidly identify the most common solutions and errors that students were making, so they could be discussed and analyzed. Moreover, it was designed to store all intermediate versions of a program so that the instructor could study the process of how students solved a problem, including how many times they pressed the run button, what kinds of mistakes they made, how long it took them to make a run, debug, edit, re-run cycle, etc.

A. Spinoza application

Spinoza has a simple web-interface that allows the instructor to create programming problems. The instructor must provide a description of a problem, some scaffolding code, a solution for the problem, and code for a unit test generator. Currently Spinoza supports programs in Java and in Python.

The student interface allows users to select a problem, which takes them to a problem-solving page shown in Figure 7. This view provides the following information to the student:

- the text describing what they should do;
- a text-editor pane with syntax highlighting that may (or may not) contain initial scaffolding code;
- a "run" button that compiles and runs the code on all of the unit tests specified by the instructor;
- a stdout pane showing the output when the main program is run; and

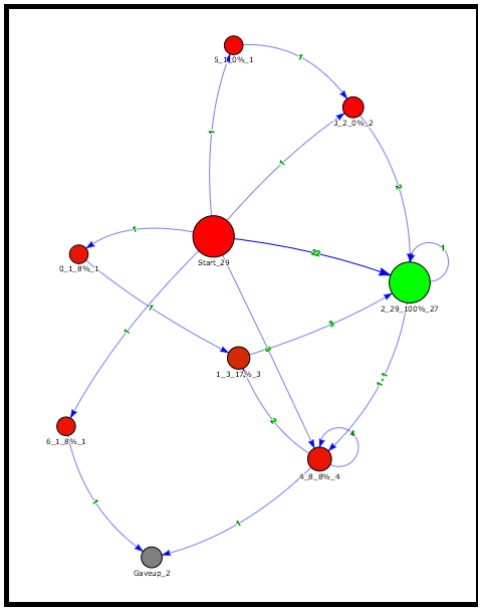

Fig. 9. ”Remove first element” programming exercise Markov model view

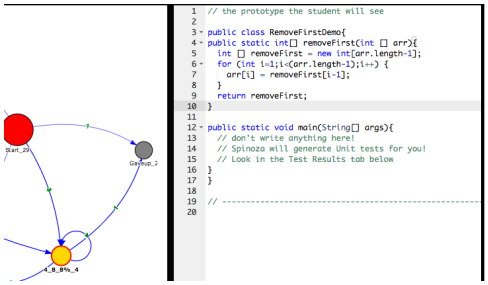

Fig. 10. An example of a common mistake node in the ”Remove first element” Spinoza Markov Model

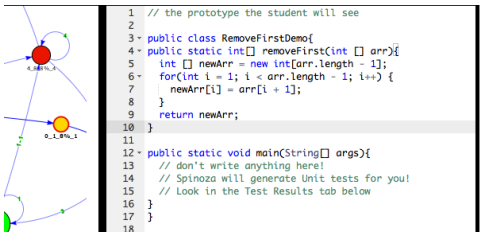

Fig. 11. An uncommon mistake attempted by only 1 student in the ”Remove first element” problem

- a unit test result pane showing the results of the unit tests with the expected and the actual values computed by the student's program, color coded by correctness.

There are two different monitoring views for the instructor:

- the **Current Code View** that shows the students' most recently submitted solutions grouped by an equivalence relation (e.g. producing identical results on the unit tests), and ordered by the number of students who have that attempted solution (correct or not). This interface is typically used to decide when to stop the activity and to start reviewing the students' solutions. We have found that stopping when 80% of the class no longer has syntax errors is a pedagogically effective time to start sharing their attempted solutions with the rest of the class.
- the **Spinoza Markov Model View**, which is a graphical representation of the paths of attempted solutions all students have taken to arrive at their current code. See Figure 8. This view is especially effective in finding common errors which students create and resolve during the debugging process, even if they eventually get a correct solution. The instructor can click on any node in this graph view and see the corresponding code in the nearby textpane, see Figs. 10 and 11.

B. Spinoza Pedagogy

Spinoza has a range of common use cases that support various forms of active learning.

- **In-class programming exercises for recently learned concepts**: The instructor creates a fully-specified problem with a full set of unit-tests including randomly generated tests before class to check the understanding of a new concept he plans to teach. In class, after teaching the new concept, the instructor tells the students which problem to work on. Students work on the problem and the instructor monitors what the class is doing through the Current Code View. When enough students no longer have syntax errors, the instructor starts reviewing the students' code. The instructor could choose one or more of the most common solutions in current-code view to discuss, or he could jump to Markov model view to discuss the most common mistakes students made during the process. Also, the instructor could look at a program with syntax errors and engage students to help him find and correct the errors.
- **Under-specified programming exercise**: The instructor creates a problem with no description of what the program is supposed to do and with minimal scaffolding, e.g. just the method signature as initial code. The students must infer the functionality of the program by examining the expected results of the unit tests. After making a hypothesis, they need to implement it in code and test its correctness. This type of problem helps the instructor check students' computational thinking as it requires the students to infer the functionality, create an algorithm and implement it. The instructor also asks the students to add comments explaining what the ”secret” method does. The Code Hunt application provides similar kinds of problems but for a simpler domain [16].
- **Buggy programming exercise**: When the instructor wants to test his class's debugging skills or to expose misconceptions that are common with novices, the instructor creates a fully specified problem but with a buggy solution as the initial code and asks students to debug it.
- **Scaffolding level of programming exercise**: Spinoza allows instructors to create problems that vary in the level of initial code provided to students. Instructors initially

give problems with a lot of scaffolding, to familiarize students with code reading and syntax and ask students to complete the code. Then as students gain expertise, the instructors test students' synthesis level by giving less, and eventually no, scaffolding for the problems.
- **Homework programming exercise**: The instructor assigns problems as homework for students to work on before the next class. By looking at Markov model view and Current Code View, the instructor can plan his lecture to address the most common misconceptions among students, or to shift to more advanced topics.
- **Pair programming**: In class the instructor asks students to work in pairs to solve a programming exercise by sharing a single computer, one student types and the other would monitor what the other student is writing. Pair programming results in higher quality code and gives the pair the opportunity to explain why they are choosing one strategy over the other. Spinoza records the two students working on such a problem and gives both access to their attempted solution.

Figure 9 shows the Markov model instructor view of the problem of writing a method to remove the first element of an array in Java. This was generated as an in-class activity in a recitation section with 29 students. Each node, except the **start** and **gave up** nodes, represents a solution that produced certain values for the unit tests. Node labels encode information about that attempt, for example the label 4_8_8%_4 means that this node has id 4, the unit test values were produced by the students 8 times, the solution passed 8% of the unit tests and this solution was submitted by 4 unique students. These nodes are connected by a directed edge labeled with the number of students who took this path from one node to the other. If we divide the number on the edge by the number of attempts in the started node, we can get the probability of the students advancing from one node to another. For this figure, 29 students start working on the exercise, 22 students found the correct solution in one attempt (which is almost 75% of the students), 2 gave up, and the rest were able to find the correct solution after at most 2 attempts. Figure 10 shows a solution attempted 8 times by 4 different students. The mistake in this attempted solution was to return an array of the proper size, but initialized to zero. On the other hand figure 11 shows an attempted solution that was unique for this class. The mistake here is to return a copy of the original array but with a zero in the initial position. Apparently, this student had a misconception about what it means to remove the first element of an array.

VI. Pedagogical Applications of PSMMs

In this section, we give an overivew of the main pedagogical applications enabled by having a Problem Solving Learning Environment like Spinoza or CalcTutor which can automatically generate PSMMs.

A. Diagnosing common problems and Just-in-time or Proactive Teaching

The simplest application of PSMMs is to use them to rapidly find common mistakes that students make while attempting to solve problems. These mistakes can be analyzed to determine the underlying misconceptions which can then be discussed either in class or in a tutorial. When the PSMMs are available in real-time this can be done as part of a classroom activity. We have used the Spinoza Markov Models during in-class problem solving sessions to identify and discuss the most common errors. The current version of Spinoza allows the instructor to turn a particular incorrect solution attempt into a new problem for the entire class, where everyone attempts to debug that particular program!

If the Markov Models are generated from homework assignments, then the instructor can quickly analyze the errors and modify the lecture to cover those misconceptions. This is an example of Just-In-Time-Teaching [17]. If the course has been taught for multiple years, then the instructor can be proactive and review the PSMMs for problems from the previous year. This information can be used to help the current students avoid making that type of mistake. Looking at the Markov Models of their homework, she can then confirm the degree to which her intervention was successful.

B. Identifying at-risk students

Even if most students do eventually get all of the answers correct, those students with fundamental knowledge gaps or misunderstandings often have a characteristic behavior where they use a trial-and-error approach. This is easily detected from the PSMM and one can, at a glance, identify the students in most need of help. Moreover, by examining their attempted solutions one can often uncover the gaps in their understanding and address these issues in office hours.

There are other features of the PSMM that can be used to help identify at-risk students. For example, the numbers of steps they require and the amount of time required to solve the problem, and the type of errors they make [18]. This can be somewhat challenging however when students are allowed to pair program.

C. Assessing the Effectiveness of a Pedagogical Unit

It is common for an instructor to begin and end a unit with a pair of tests, often called the pre-test and post-test, to measure learning. Figure 12 shows the Markov Models for a pair of similar questions on a pre- and post-test. In this example, the students were asked to take the derivative of e^{3x} in the pre-test and e^{5x} in the post-test. Looking only at the final results, we don't see much difference between the pre- and post-tests. The number of students eventually getting the right answer for the pre-test was 76/86 and for the post-test was 78/86, but looking a little more closely we see that only 50 students got the answer correct on the first try in the pre-test, compared with 70/86 on the post-test. We still have about 10/78 students that just missed the problem and were probably having other challenges. For the students who were eventually successful

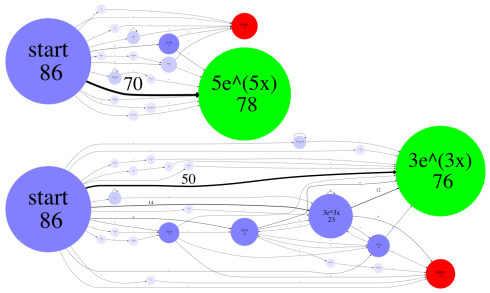

Fig. 12. PSMM for Post-test $d/dx(e^{5x})$ above and Pre-test $d/dx(e^{3x})$ below. The post test has fewer incorrect attempts and fewer students sharing any incorrect attempt.

but didn't get it on the first try, we see much more exploration and many common errors for the pre-test. For example, in the pre-test 19 students omitted the parentheses `3*e^3x` and four students tried other variations of this incorrect attempt (e.g. `e^3x*3`. This type of analysis allows a more nuanced view of the effectiveness of a particular instructor and class in promoting learning of specific problem solving skills.

VII. Future Work

We have had two semesters of experience using Problem Solving Markov Models and have found that they enable the instructor to gain a deeper, and real-time, understanding of the problem solving strategies of students in the class. There are many other potential applications of PSMMs that we have not yet put into practice, but which are promising.

A. Forming Study Groups based on common misunderstanding

One potentially fruitful applications of PSMMs is to have Spinoza or CalcTutor automatically form study groups consisting of students who have common gaps or misconceptions. Grouping students with similar pedagogical issues should lead to more effective mentored study groups. One approach to forming such groups is to focus on a collection of problems (e.g. the past weeks' assignments) and to successively form groups of students with similar problem solving behavior.

For example, one can start by forming the group of students who found the correct answer in one step. These students do not need any additional support, at least as far as this diagnostic goes. Removing these students from the PSMM gives a smaller graph consisting of students that made at least one error. One can then select one or more of the most common errors and look at students whose errors fell mostly within those selected common errors. Once a group of the desired size has been formed, those students can be removed and the process repeats. At some point, the only students remaining will be those whose mistakes are mostly idiosyncratic. These are typically the at-risk students who need one-on-one or small group support with a mentor.

B. Nuanced Assessment of Multiple Sections of a Class

One benefit of the PSMM approach is that it can be used to collect data from multiple sections of a class in one semester. For example, Calculus is typically taught with a large number of small sections that have common exams and possibly common homework assignments. By comparing the Markov Models of different sections, the lead instructor can identify particular sections that are struggling to learn particular concepts.

C. Comparing the performance of a single class to a larger population

Another intriguing possible application, is to collect data for particular formative assessments from a large number of different classes and to create PSMMs for this large population, e.g. U.S. college students taking Calculus in 2017. Any particular class could compare their PSMM with the global PSMM to see whether their class is typical. It could also be used to compare different types of pedagogy, e.g. looking at the PSMMs for active learning classes on a particular homework compared with lecture-based classes on the same problem.

VIII. Conclusions and Future Work

Computer-mediated Problem Solving Learning Environments such as Spinoza and CalcTutor are effective tools for implementing active learning activities in programming and Calculus classes, respectively; and they simultaneously generate a wealth of information about how students in the class navigate the problem solving process when provided with a simple correctness feedback interface.

In this paper we have shown how this information can be visualized using Problem Solving Markov Models in a way that provides instant feedback to the instructor on the nature of the problem solving process for all students in the class.

The simplest application of PSMMs is discovering the most common errors and using that information to guide further pedagogical interventions, ranging from directing appropriate resources to students making the same kinds of errors, to providing new exercises for the entire class based on debugging the common errors.

The PSMM formalism can also be used to assess qualitative measures of learning by comparing pre-test and post-test PSMMs. Even when the final results of the pre/post-tests show little difference, comparison of the corresponding PSMMs highlights the increased fluency students have in finding solutions as well as singling out those students who still lack proficiency.

In the near future, we are planning on using the results of PSMMs to form mentored study groups with the expectation that the more nuanced level of assessment provided by PSMMs will allow the formation of study groups who have common misconceptions, and the expectation is that this will allow us to focus the mentoring sessions more effectively.

REFERENCES

[1] S. Zweben and B. Bizot, "CRA Taulbee Survey Report 2014," May 2015. [Online]. Available: http://cra.org/crn/2015/05/2014_taulbee_survey/

[2] J. M. Fraser, A. L. Timan, K. Miller, J. E. Dowd, L. Tucker, and E. Mazur, "Teaching and physics education research: bridging the gap," *Reports on Progress in Physics*, vol. 77, no. 3, p. 032401, Mar. 2014. [Online]. Available: http://stacks.iop.org/0034-4885/77/i=3/a=032401?key=crossref.2338152e2386ee2984e1e1c8c44add75

[3] D. C. Haak, J. HilleRisLambers, E. Pitre, and S. Freeman, "Increased Structure and Active Learning Reduce the Achievement Gap in Introductory Biology," *Science*, vol. 332, no. 6034, pp. 1213–1216, Jun. 2011. [Online]. Available: http://science.sciencemag.org/content/332/6034/1213

[4] T. J. Hickey and F. Abu Deeb, "Spinoza: the Code Tutor," in *Proceedings of the International Conference on Computer and Information Science and Technology*. Ottowa, CA: Avestia, May 2015, pp. 132.1–132.8.

[5] F. Abu Deeb and T. Hickey, "The Spinoza Code Tutor: Faculty Poster Abstract," *J. Comput. Sci. Coll.*, vol. 30, no. 6, pp. 154–155, Jun. 2015. [Online]. Available: http://dl.acm.org/citation.cfm?id=2753024.2753055

[6] K. Kime, R. Torrey, and T. Hickey, "CalcTutor: Applying the teachers dilemma methodology to calculus pedagogy," in *IEEE Frontiers in Education Conference (FIE), 2015. 32614 2015*, Oct. 2015, pp. 1–8.

[7] P. Reimann, M. Kickmeier-Rust, and D. Albert, "Problem solving learning environments and assessment: A knowledge space theory approach," *Computers & Education*, vol. 64, pp. 183–193, May 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360131512002977

[8] J. Spohrer and E. Soloway, "Analyzing the high frequency bugs in novice programs." Ablex Publishing Corp., 1986, pp. 230–251. [Online]. Available: http://portal.acm.org/citation.cfm?id=28897

[9] N. Pillay and V. R. Jugoo, *An Analysis of the Errors Made by Novice Programmers in a First Course in Procedural Programming in Java*, 2006.

[10] M. Ahmadzadeh, D. Elliman, and C. Higgins, "An Analysis of Patterns of Debugging Among Novice Computer Science Students," in *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '05. New York, NY, USA: ACM, 2005, pp. 84–88. [Online]. Available: http://doi.acm.org/10.1145/1067445.1067472

[11] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein, "Modeling How Students Learn to Program," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '12. New York, NY, USA: ACM, 2012, pp. 153–160. [Online]. Available: http://doi.acm.org/10.1145/2157136.2157182

[12] A. Bader-Natal and J. B. Pollack, "Motivating Appropriate Challenges in a Reciprocal Tutoring System." in *AIED*, 2005, pp. 49–56.

[13] M. Gage, A. Pizer, and V. Roth, "Assessment: Gage, M., Pizer, A., Roth, V. (2002). WeBWorK: Generating, delivering, and checking math homework via the Internet. In ICTM2 International Congress for Teaching of," 2002. [Online]. Available: http://webwork.maa.org/moodle/mod/resource/view.php?id=37

[14] S. R. Helena Dedic, "Just Computer Aided Instruction Is Not Enough: Combining WeBWorK with In-Class Interactive Sessions Increases Achievement and Perseverance of Social Science Calculus Students," *Pedagogie Collegiale*, vol. 22, no. 5, 2009.

[15] V. Roth, V. Ivanchenko, and N. Record, "Evaluating student response to WeBWorK, a web-based homework delivery and grading system," *Computers & Education*, vol. 50, no. 4, pp. 1462–1482, May 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360131507000152

[16] N. Tillmann, J. Bishop, N. Horspool, D. Perelman, and T. Xie, "Code Hunt: Searching for Secret Code for Fun," in *Proceedings of the 7th International Workshop on Search-Based Software Testing*, ser. SBST 2014. New York, NY, USA: ACM, 2014, pp. 23–26. [Online]. Available: http://doi.acm.org/10.1145/2593833.2593838

[17] K. A. Marrs and G. Novak, "Just-in-Time Teaching in Biology: Creating an Active Learner Classroom Using the Internet," *Cell Biology Education*, vol. 3, no. 1, pp. 49–61, Mar. 2004. [Online]. Available: http://www.lifescied.org/content/3/1/49

[18] W. T. Tarimo, F. A. Deeb, and T. J. Hickey, "Early detection of at-risk students in CS1 using teachback/spinoza," *Journal of Computing Sciences in Colleges*, vol. 31, no. 6, pp. 105–111, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?id=2904471