Classroom Orchestration with Problem Solving Markov Models

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

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

***Abstract*—This Innovative Practice paper presents a new approach to monitoring the performance of students working in web-based Problem Solving Learning Environments (PSLEs) which can be used to introduce active learning in large classes. This approach allows the instructor to pose a problem that all students will immediately attempt to solve. Classroom orchestration tools allow the instructor to monitor, in detail, the progress of the class on that problem. The Problem Solving Markov Model (PSMM) is one such orchestration tool. It provides a real-time graphical view of all equivalence classes of attempts that students have made on a particular problem and it shows the probabilities of going from one incorrect attempt to another. In this paper, we describe the PSMM for a particular PSLE for coding, called Spinoza, and we explore the connection between PSMM equivalence classes and student misconceptions. We also introduce an extension of the PSMM which allows the instructor to zoom in and out of the time dimension and we discuss the pedagogical applications of this extension for orchestration in large classes.**

I. Introduction

The number of students taking programming classes in US colleges and universities has been increasing rapidly in the last few years and is at record levels. The number of faculty teaching these courses however, has not increased at the same rate and hence the average class size has also been increasing rapidly. One approach to providing high quality education in large classes is to use active learning approaches. Online Problem Solving Learning Environments (PSLEs) [Reimann et al., 2013] are one way to introduce active learning into the classroom. These tools present students with a problem to be solved and an online suite of tools to solve that problem. Students make attempts and the system will automatically judge the accuracy of those attempts.

Fig. 1 shows a typical example of a programming problem for such an active learning exercise and Fig. 2 shows the view of one student's attempt on that problem in Spinoza [Deeb and Hickey, 2017], a PSLE for learning to code in Python. The student writes code to solve a particular problem given by the instructor (in the Description tab) and when they press the Run button a suite of randomly generated as well as instructor supplied unit tests are run and their results presented in the "Unit Test" tab. The figure shows a case where the student was trying to write the "is_Leap_Year" function but only considered the case of divisibility of the year by 4 and hence got the wrong answer for 1900 and 2500.

For effective use of a PSLE for in-class activities, the instructor needs to be able to monitor the progress of the students on the problem so as to decide when to stop the activity to either clear up misconceptions or to start debriefing their experience. One approach to this kind of classroom orchestration is to use the Problem Solving Markov Model (PSMM) [Deeb et al., 2016].

The PSMM in Spinoza provides a real-time graphical view of all equivalence classes of attempted solutions to a specified problem that students have made during class and it shows the probabilities of going from one equivalence class of attempts to another in one step. Two attempted solutions to a problem are in the same PSMM equivalence class if they take exactly the same values on an instructor-specified set of unit tests. The PSMM is generated in real-time and the instructor can use it to skim through all attempts in that equivalence class and to run/edit any of those attempts. We will describe the PSMM for Spinoza in detail in Sec. IV.

One problem with the PSMM is that it represents all of the attempts made by the students from the beginning of the activity to the current moment and it doesn't allow the instructor to have a clear view of what the students are doing now, or what kinds of mistakes they are currently making. The instructor can determine how many students have completed the problem by looking at the size of the PSMM node corresponding to equivalence class of correct solutions, but the PSMM doesn't give any additional information about current student performance as it summarizes all activity on that problem. Our new Time-Zooming features addresses this problem and enables and new more responsive pedagogy for this kind of active learning exercise.

II. Overview of New Pedagogy

In this paper, we discuss an extension of the PSMM, and its implementation in Spinoza, which allows the instructor to zoom in and out of the time dimension, and how it can be used to more effectively orchestrate large Python programming classes.

The Spinoza application itself is easy for both students and instructors to use. Instructors have access to a large library of

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

problems which they can copy and modify (and in the process learn how to create from scratch).

The intended pedagogical practice is for the instructor to create (or copy) several problems for the students to work on in class after they have been introduced to a concept (e.g. conditional execution and boolean expressions). These problems will appear in a Problem Set tab and students can click on each problem to see a description of what is to be done, a textarea with syntax highlighting for writing their code, a run button, and several tabs for seeing the unit test results, and if the instructor allows it, hints from the TAs or other students associated to their individual kinds of errors.

The instructor can select several views but this paper allows the instructor to get by with only one view, the Time-Zoomed Markov Model view. Initially the instructor should adjust the time-zooming settings so that the time-zoomed PSMM will continuously display a graphical representation of the last 3-5 minutes of student programming activity.

This allows the instructor to see how many students have completed the problem and, for those who have not, it shows the graphical Markov Model view of the equivalence classes of incorrect programs with size indicating the number of students making that mistake and color indicating the percentage of unit tests that mistake passes.

The instructor can rapidly detect any common errors that are arising among a large percentage of the class and immediately intervene by using the PSMM interface to show one or more of the actual student errors in that equivalence class, discuss the error with the students, and finally let them continue.

When the instructor calls an end to this online programming activity, they can first review the most common errors of the students who were not able to complete the problem, using the time-zooming view focused on the last moment of the exercise, thereby bringing them up to speed with the rest of the class rapidly. They can then expand the time-zooming settings to the entire activity from start to finish and discuss the most common errors of the whole class for this problem (again by finding the largest node and using the interface to see the actual student code in that class). They can also look at some (or all) of the correct solutions and discuss the pros and cons of each of the different solutions.

Finally, if the instructor wants, they can look (after class) at the kinds of mistakes students were making in the first few minutes to discover the most common initial errors and hopefully the underlying misconceptions.

Our previous version had the Markov Model view, but because there was no time-zooming, the kind of highly responsive classroom orchestration described in this section, was not possible. In the remainder of this paper we go into detail on the implementation of Problem Solving Markov Model and the Time Zooming option, as well as describing the actual user interface and we will give examples of the kinds of mistakes that form a Markov Model equivalence class.

```
Write a function, is_leap_year(y),
which returns True if y is a leap year.
Remember that y is a leap year if
    y is divisible by 4
and
    y is not divisible by 100
or
    y is divisible by 400
So
  is_leap_year(2017)--> False
  is_leap_year(2016) --> True
  is_leap_year(2000) --> True
  is_leap_year(1900) --> False
```

Fig. 1. The description of the isLeapYear problem.

III. Related Work

Problem Solving Learning Environments have been proliferating in recent years. Webwork [Roth et al., 2008] and CalcTutor [Kime et al., 2015], [Kime et al., 2017] are two examples of PSLEs for Mathematics. For coding, there are many online Integrated Development Environments (IDEs) which provide this kind of an active learning PSLE, such as

- Codingbat/Javabat [Parlante, 2007],
- Codehunt [Nikolai Tillmann and Tao Xie, 2014],
- CloudCoder [Andrei Papancea, 2013],
- AMOBEA [Berland et al., 2015],
- PCRS [Zingaro et al., 2013],
- Spinoza [Deeb and Hickey, 2017],

The first three were designed primarily for auto-grading of homework or for self-study, and do not have any orchestration features to support real-time use in a large classroom.

The AMOBEA application [Berland et al., 2015] provides a PSLE with an orchestration view built around the Ipro visual programming environment. In this scenario, the instructor asks students to solve a problem individually. After a while the AMOBEA system will create links between students based on the similarity of their code. The instructor can then ask students with similar code to work together. The students in a such a pair are solving the problem with their own devices individually but they can ask each other for help. These pairs are dynamic, and one student might work as part of several different pairs in one activity. The instructor uses the AMOBEA tool to see which students are writing similar code and to ask them to move and work together.

Another PSLE which is closer to Spinoza, is the Python Classroom Response system [Zingaro et al., 2013], which is a web based clicker system that allows the instructor to assign multiple choice and code writing questions to the students in class time. It was developed mainly to extend code writing to the Peer instruction (PI) pedagogy approach in Computer Science [Simon et al., 2010]. In real-time, the instructor can see a histogram of the percentage of final correct and incorrect solutions submitted by students, and also a histogram for each unit test showing the percentage of submitted solutions

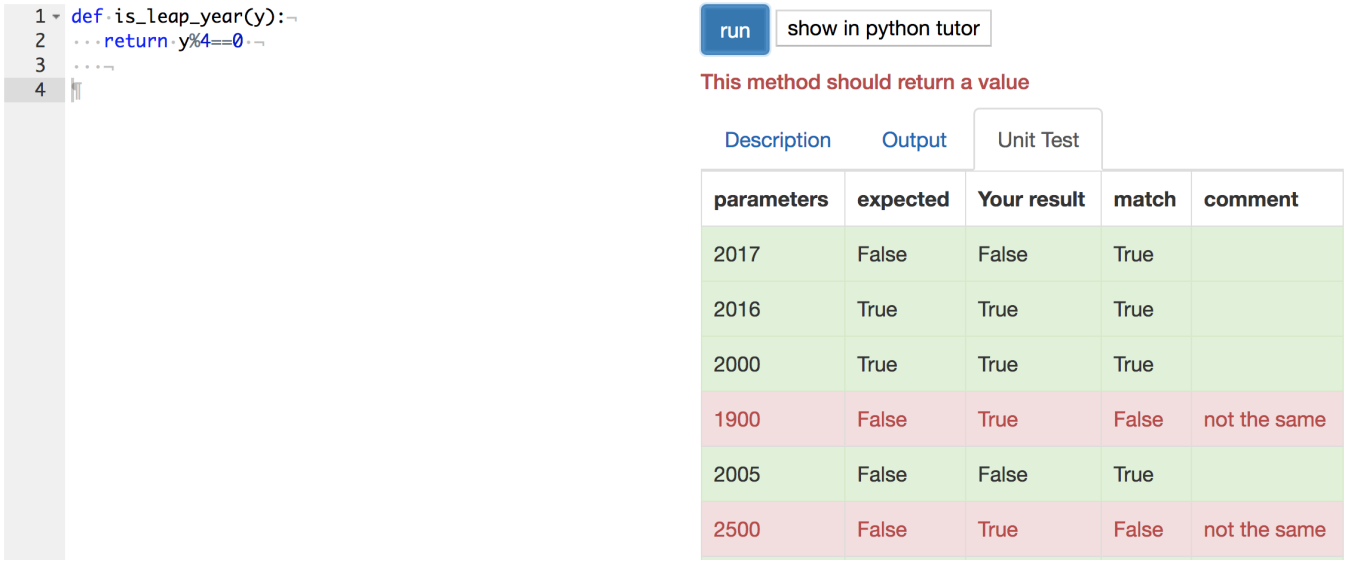

Fig. 2. The student view of is Leap programming problem in Spinoza. Description tab shows the following: Write a function, is_leap_year(y), which returns True if y is a leap year. Remember that y is a leap year if y is divisible by 4 and y is not divisible by 100 or y is divisible by 400

that failed that test. The instructor can choose solutions from each category to modify and discuss as part of his classroom orchestration activity. The PSMM we introduce in this paper provides more detailed information about the errors in student programs over any time interval, not just the current moment. This makes it a more versatile debriefing tool since the instructor can discuss all of the kinds of errors that students made while trying to solve the problem.

IV. PROBLEM SOLVING MARKOV MODELS

Problem Solving Markov Models (PSMM) are annotated graphs which represent the set of all attempted solutions by a group of students to a problem [Deeb et al., 2016]. They can be automatically created for any PSLE in which there is a meaningful (and computable) equivalence relation on attempted solutions. One can always just use string equality on the attempts, but this provides too fine of a distinction. For a mathematics PSLE where the attempts are functions of one variable (e.g. differentiation problems), one can say two attempts are equivalent if they represent the same function, or more practically if they take approximately the same values at a (randomly generated) set of points. For coding PSLEs the attempts are typically functions or methods, and the system typically provides a set of unit tests. Two attempts can be said to be equivalent if they produce the same values on all of the unit tests.

Assuming one has an equivalence relation on attempts, then the PSMM for a problem is a graph whose nodes are all equivalence classes of attempts to that problem, and the edges correspond to sequential attempts by an individual. The edges are labeled with the number of times students went from an attempt in the source node to an attempt in the destination node in one step. The nodes are labeled with the number of attempts in the equivalence class.

The PSMM also has three special nodes. The **Start** node corresponding to the initial scaffolding code. The **Success** node, corresponding to the equivalence class of successful attempts, and the **Failure** node which is pointed to by all unsuccessful final attempts by the students. The PSMM is created from the problem solving log files for all of the students. It can be thought of as adding an edge from the start node to the first attempt for each student, and then adding an edge from each student's last attempt to either the Success or the Failure node. The PSMM is then the quotient graph with respect to the equivalence relation on attempts. To provide more visual information, the size of a node is rendered proportional to the number of attempts in that node and the color is rendered proportional to its "correctness" if there is such a natural measure. For coding, the percentage of unit tests that the attempt satisfies is a good measure of correctness (from red=0% to green=100%).

One problem with PSMMs for large classes is that the generated graph can have hundreds or thousands of nodes. For example, there are typically only a relatively few very common mistakes, but there are many ways of making uncommon mistakes (e.g. a variety of syntax errors, and logic error based misunderstandings). Fig. 3 shows the full PSMM for a class of 157 students solving the isLeapYear problem, which we discuss in more detail below. The Reduced Problem Solving Markov Model provides a solution to this problem by filtering out all attempts that were not shared by at least k students for some instructor-specified k. More precisely, the attempts that were made by fewer than k students are edited out of all students logs before forming the PSMM. Fig. 4 shows the Reduced PSMM for the same isLeapYear problem where we only look at nodes with at least k attempts.

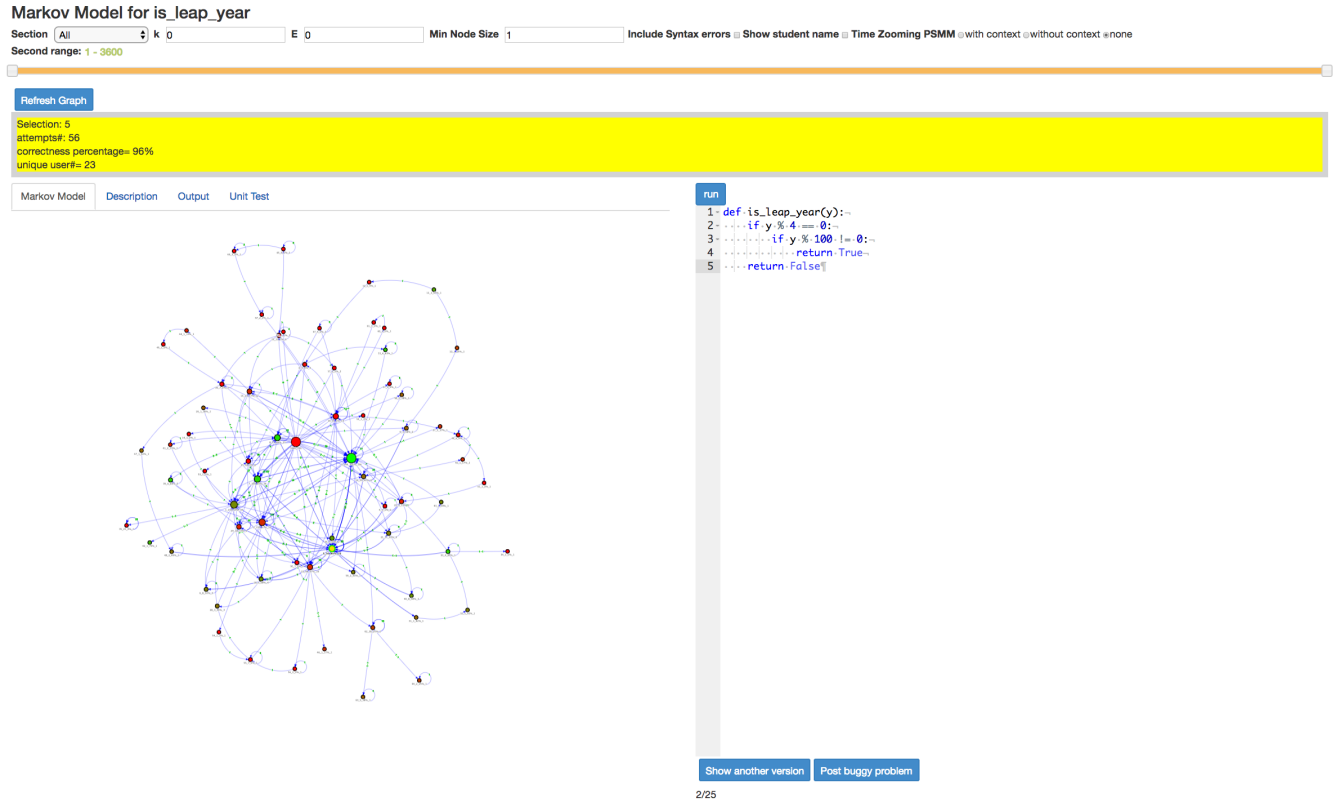

Fig. 3. User Interface for the Time Zooming Problem Solving Markov Model feature. This shows the student performance on the isLeapYear problem in Fig. 1 after most students had completed the problem. The instructor has selected one node, and the code for the node appears in the right panel. The instructor can edit this code (without changing the underlying database) to explain it to the class, and can run the code which will populate the "unit test" tab in the left pane. The instructor can also look at all other student programs in that PSMM node by clicking the "Show Another Version" button at the bottom of the right pane, and can post a problem as a debugging problem with the "Post Buggy Problem" button. At the top are the controls for specifying the time zooming and other settings for the Markov Model, the time zoom widget is the horizontal bar with handles on either side. The suggested practice is to select the settings when the problem is assigned and set the time bar to a short region on the right corresponding to the most recent activity, then to refresh the page occasionally to see what the students are doing.

V. PSMM Nodes and Students' Misconceptions

In this section, we look in detail at a particular Reduced Spinoza Markov Model to get a better idea of the student misconceptions represented by a PSMM node, both in terms of the abstract function represented by the equivalence class, and the particular incorrect attempts that lie in that equivalence class.

For this section, we look at the 6 non-start nodes in the Reduced Spinoza Markov Model for the isLeapYear problem shown in Fig. 4. That problem asks the student to write a Python function `is_leap_year(year)` which returns true if a given year is a leap year. The six nodes correspond to very simple Python programs which we discuss below, for example, node 5 is the function which always returns the boolean value **False**.

Figure 4 represents the common errors that were made by at least 20/150 students and the common order of these errors. Most of the edges in the Figure are unidirectional meaning that most students go from the source node to the destination node, indicating a commonly observed order for that pair of nodes. The settings for the Markov Model display can be tuned so that only edges with a sufficiently large percentage of the outgoing traffic are shown, this removes the edges which were rarely traversed and helps make it easier to see common or shared orderings of nodes.

This particular problem was solved by all the students so no one had given up by the end of the class. The graph shows 5 different classes of solutions that some students commonly produce before solving the problems (nodes 1,2,4,5,6) in addition to node number 3 that represents the correct solution, and the node labeled "start" which represents the Start node.

- node 1 represents a class of solution where the function would return true for any year that is divisible by 4 and false otherwise. This class of solution was tried by 34 students out of 150.

```
def is_leap_year(y):
    return y % 4 == 0
```

- node 2 represents a class of solution where the function would return true all the time for any year. This class of solution was tried by 31 students out of 150.

```
def is_leap_year(y):
```

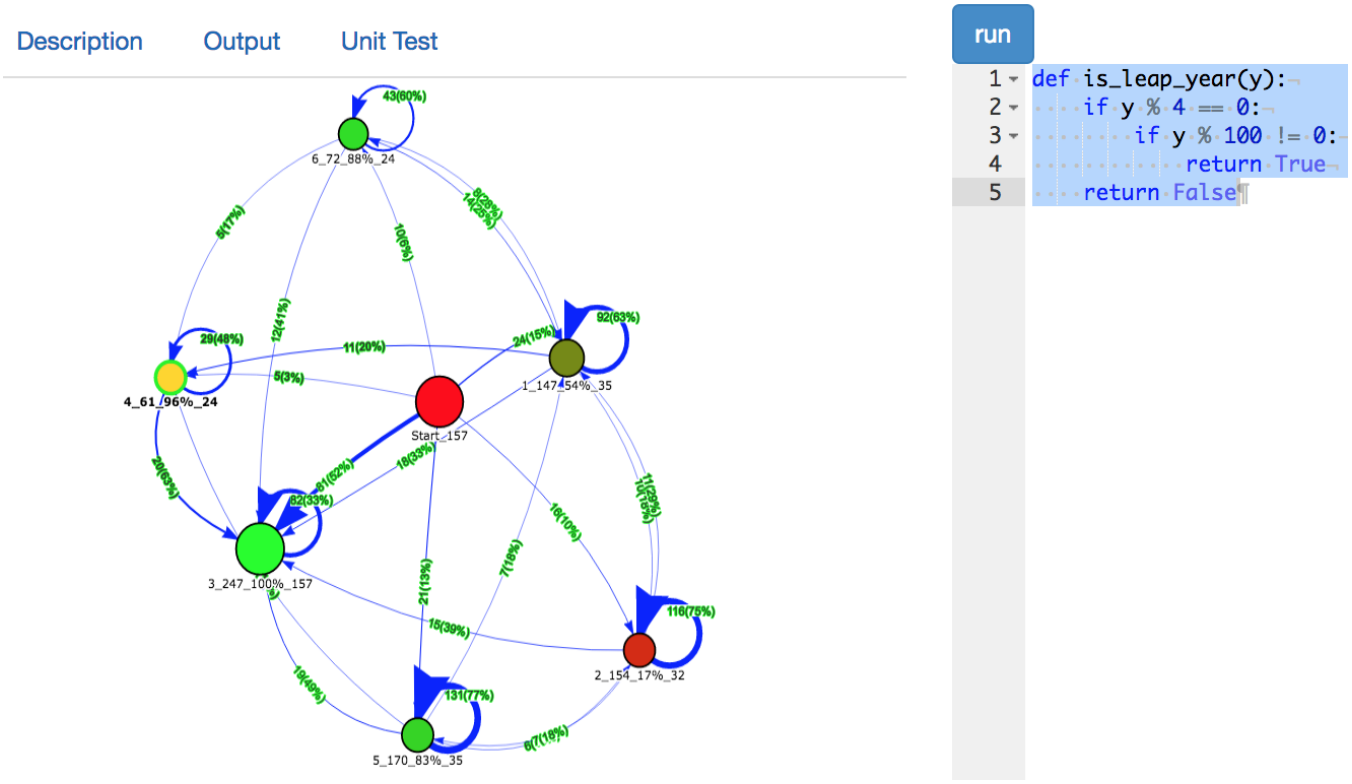

Fig. 4. This figure shows the real-time Spinoza view of the Reduced PSMM for the isLeapYear problem. The nodes represent equivalence classes of student attempts on this problem and only those nodes with at least 20 common attempts are shown. The color represents the correctness of the attempts in that node, with green = 100% and red = 0% (Spinoza also has an option to switch the palette for color-blind students or instructors). The size of each node is proportional to the number of attempts in that node and the width of each edge is proportional to the number of students who made that transition. The nodes (except for the start and give-up nodes) are labeled with four numbers: the node id, the number of attempts in that node, the correctness percentage for attempts in that node, and the number of unique students who contributed to that node. The Start node shows the number of students who have made at least one attempt. The Give-up node (not pictured here as everyone succeeded eventually) shows the number of students who have not yet found the correct solution.

```
    return True
```

- node 3 represents the correct solution

```
def is_leap_year(y):
    return y % 4 == 0 and y % 100 != 0
    or y % 400 == 0
```

- node 4 represents a class of solution where the function returns true if the year is divisible by 4 and not divisible by 100. It is missing the condition allowing years divisible by 400. This class of solution was tried by 23 students.

```
def is_leap_year(y):
    return y % 4 == 0 and y % 100 != 0
```

- node 5 represent a class of solution where the function returns false all the time. This class of solution was tried by 34 students.

```
def is_leap_year(y):
    return False
```

- node 6 represent a class of solution where the function would only return true if the year is divisible by 400. This class of solution was tried by 23 students.

```
def is_leap_year(y):
    return y%400==0
```

Even though these six solutions all look very simple, the actual student code that produced these equivalence classes is often much more complex. It just happens to reduce to one of these five cases.

A. Each PSMM Node Might Contain a Variety of Different Kinds of Errors

Students quickly discovered that the challenge for the isLeapYear problem is to combine a few boolean expressions using boolean operators and/or conditionals to generate the correct result. There are three main expressions needed for this problem:

a) `year%4==0`
b) `year%100==0`
c) `year%400==0`

where `c` implies `b` which implies `a`. The correct solution is to return `(a and not b) or c`, but students have difficulty finding this expression and produced instead one of the 5 incorrect functions listed above. For some of these nodes, the students all generally are making the same mistake. For example, in node 4, most of those attempts are simply

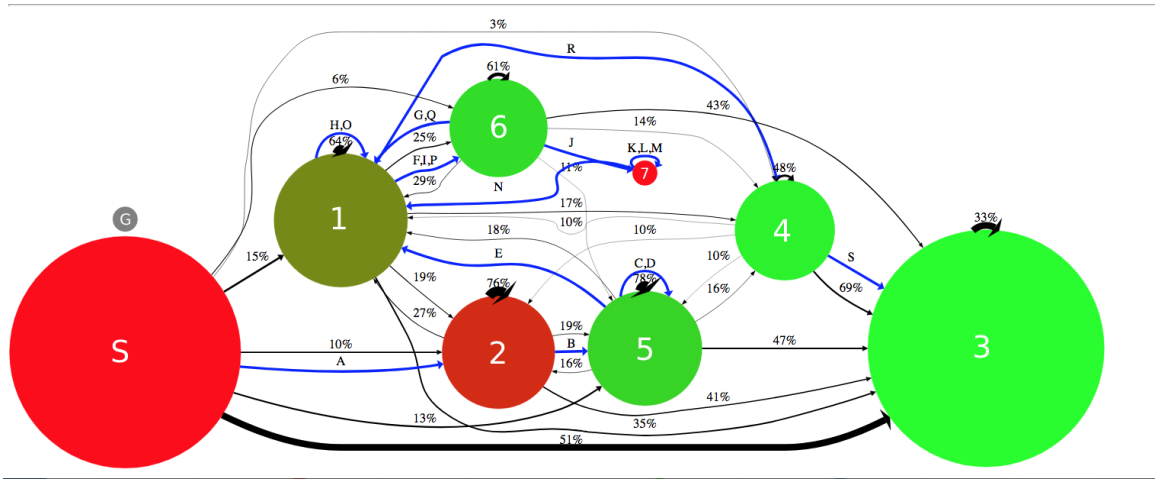

Fig. 5. One student's path (in blue) through the `isLeapYear` PSMM. Node Number 7 is the only attempt of that student which wasn't one of the most common errors captured in the 6 node Reduced PSMM. This view of an individual student's path is not currently available in Spinoza, it was constructed just for this paper.

missing condition c (divisibility by 400). They may have been intentionally not including it as a way of using step-wise development to solve the problem. For other nodes, such as node 5 where the function always returns false, they are making a variety of different mistakes usually involving incorrect boolean operators e.g. `a and not b and c`. So, even though students aren't making exactly the same error, they are making the same kind of error, i.e. trying to use an incorrect boolean expression. We also find that some students for this problem make all of the most common errors as they try to solve the problem.

Fig. 5 shows the path of one particular student through the Problem Solving Markov Model in 19 steps. All of their 19 attempts, except for one (node 7), were in the 6 node Reduced Spinoza Markov Model. The sequence of steps is shown with blue edges labeled with capital letters (A,B,C,...,S). This was a typical student in the sense that the average number of steps taken by students for this problem was also 19.

This student visited several nodes multiple times each. Sometimes because they would revert their code after trying an unsuccessful change, and other times because they would make a change that had no real impact on the functionality of their code.

VI. Debriefing with Problem Solving Markov Models

The main advantage of PSMMs is that they allow the instructor to see the most common mistakes that students made while solving the problem, even if everyone eventually gets the right answer. They also highlight common sequences of incorrect attempts as students get closer to the correct solution in similar ways. The instructor can review the problem solving activity by looking at the most common errors, and also by reviewing the variety of solutions in the equivalence class of correct solutions. For coding, this is a good opportunity to talk about coding style, variable naming, indentation, etc.

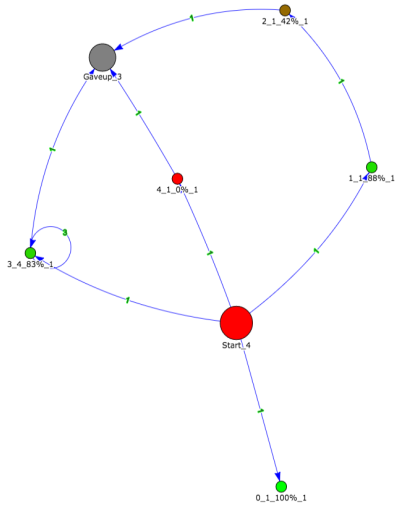

Fig. 6. Time Zooming PSMM first 2 minutes. Four students had attempted the problem. One found the correct solution in one step, two made two incorrect attempts, that were nearly correct as indicated by the greenish color of the node, and the fourth student made one incorrect attempt. The labels on the nodes become visible when the image is zoomed and have the form `Id_N_P_U` where `Id` is a unique identifier for the node, `N` is the total number of attempts in that node, `P` is the percentage of unit tests that pass for the attempts in that node (and this is shared among all the nodes as, by definition of the equivalence relation they all take exactly the same values on all of the unit tests), and finally `U` is the number of unique users who contributed to this node. Some users contribute multiple examples by making changes to their code which doesn't change the values on the unit tests. These are represented by self loops on the nodes.

VII. Time Zooming and Orchestration

The main disadvantage of PSMMs as an orchestration tool is that they present a summary of the entire problem solving process and give very little information about the students' current activity. The Success and Failure nodes in the PSMM show what percent of the class has solved the problem, but

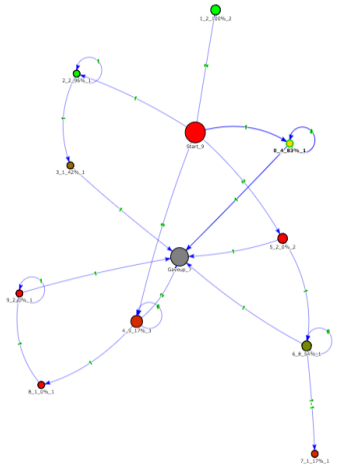

Fig. 7. Time Zooming PSMM for a 4 minute period without context. This only shows nodes corresponding to attempts which were made during that interval. It gives a true view of the actual activity in that time interval. This view lets the instructor see if students are still actively making attempts.

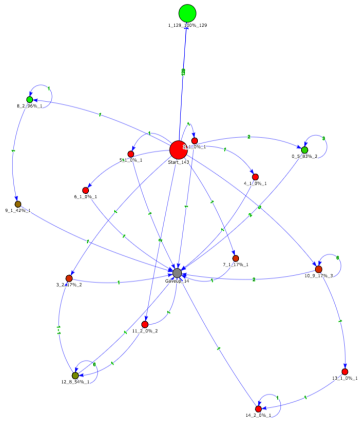

Fig. 8. Time Zooming PSMM for the same 4 minute period as in Fig 7 with context. This view includes all of the nodes from the without-context view, but it also includes the most recent of attempt of all students who did not make an attempt in that interval. This view gives a snapshot of the entire class during that interval. Students who didn't make an attempt in the interval are either represented by nodes that go directly to the "gave up" node or in the success node, if they have successfully solved the problem. This is the view we recommend for monitoring progress of the class.

they don't show what kinds of difficulties students are having at the present moment.

The idea behind time zooming is to narrow the focus to one particular time interval and display the kinds of problem solving behavior in that interval. For example, the instructor could try to examine all attempts in the last five minutes, or look at the set of all most recent attempts by students in the class. We introduce two flavors of the Time Zooming Problem Solving Markov Model and discuss their implementation and their uses in classroom orchestration.

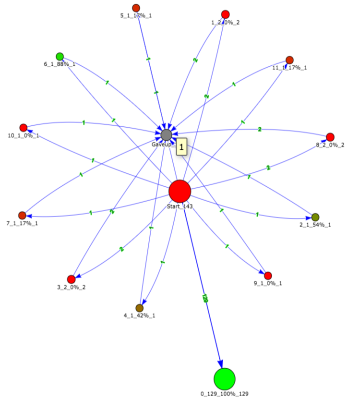

Fig. 9. Time Zooming PSMM with context and a point interval at the end of the same 4 minute period. This is an example of specifying a interval of width 0 by bringing the start and end markers of the Time Zooming slider together at some point. It shows the most recent behavior of all students who have submitted at least one attempt, and it always has this flower shape as each student is represented exactly once, either in the success node or in an incorrect solution node which then maps to the "gave up" node. We recommend this view for the debriefing when the instructor ends the programming exercise, as it allows the instructor go look (anonymously) at the code for the students who didn't complete the problem and to explain and debug those attempts, thereby helping those students get caught up with the rest of the class.

A. *Time Zooming without Context*

This is the simpler version of time zooming. The idea is to allow the instructor to specify a time interval and then to filter out all student attempts except those generated during that time interval. The PSMM is then formed as usual and it illustrates the most common attempts during that interval as well as the probability of going from one to another. The Start, Success, and Failure nodes are present as usual. Moreover, only those students who made an attempt during that interval are included in this PSMM.

B. *Time Zooming with Context*

In this approach, we filter out all attempts that were not generated in the specified interval except that, for students who didn't make any attempt during that interval, we add the last attempt they made before the beginning of that interval. In this way, every student that has made at least one attempt at any time before the end of the Time Zooming interval will be represented in this PSMM. We say that the last attempt before the interval provides the "context" of their attempts in the interval, hence the name.

C. *Orchestration examples*

In this section we present some examples of the Time Zooming Problem Solving Markov Model feature in Spinoza. The user can select Time Zooming with context or without context using a radio button on the far right of the control bar at the top, as can be seen in Fig. 3.

Fig. 6 shows the Time Zooming PSMM for the first 2 minutes after the problem was made available to the students. We can see that four students attempted a solution. One got

the correct solution, the other three made one or more attempts but had not yet solved the problem. This new approach allows the instructor to review who were the first students to solve the problem, and see the kinds of errors they made along the way.

Fig. 7 shows the Time Zooming PSMM for this problem during a 4 minute interval after most students had completed the problem. In that time period, nine students submitted an attempt (as we can tell by looking at the label on the large red start node) and 2 found the correct solution while 7 were still working. We can see that a few students were making multiple attempts, the self-loops indicate changes to the code that didn't change the values on the unit tests. This can be a very useful view to see how many students are still actively engaged in problem solving and what kinds of issues they are having. This can also be used to direct in-class TA resources to help those students.

Fig. 8 shows the Time Zooming PSMM for the same interval but with context. In this case, the Start node is labeled with the number of students (142) who have made at least one attempt on the program since the beginning, and the Success node is labeled with the number (129) of those who have succeeded by the end of the interval. This differs from the without-context view in that the final attempts by all students are represented in this graph, not just the ones who have been active in the interval. This can be used to decide when to stop the activity and switch to another problem or activity.

Fig. 9 shows the Time Zooming PSMM with context for the end of that four minute interval. This can be thought of as a very short interval ($<$ 1 sec) in which no attempts were made, and hence all of the nodes shown are "contextual", so it shows the most recent attempts of all students and always has a "flower" shape where there is a single edge from the Start Node to the Success node representing all students who had correctly solved the problem, and one edge to each equivalence class of incorrect final attempts, the size and color of each such attempt represents the number of students making that attempt and their degree of correctness. All of these incorrect attempts have a link to the Failure node. This view can be used to get a quick idea of the progress of the class at any point during the problem solving activity. The instructor can also browse the most common incorrect attempts at that point.

VIII. Conclusions and Future Work

This paper has presented an extension of the Problem Solving Markov Model feature of the Spinoza Problem Solving Learning Environment for use in large introductory programming classes and demonstrated some examples of the views and how they can be used for classroom orchestration. In the future we plan to look at adding Time Zooming Problem Solving Markov Models to other PSLEs.

Another interesting and useful extension of the PSMM user interface would be to programmatically include the path of an individual student through the PSMM (as shown in Fig. 5 and to extend the controls so that one can either flip through all attempts in a single PSMM node as we do now, or move forward and backward in the history of the particular student whose code is currently being viewed. This would allow the instructor to talk about different debugging approaches.

References

[Andrei Papancea, 2013] Andrei Papancea, Jaime Spacco, D. H. (2013). An open platform for managing short programming exercises. *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, pages 47–52.

[Berland et al., 2015] Berland, M., Davis, D., and Smith, C. P. (2015). AMOEBA: Designing for collaboration in computer science classrooms through live learning analytics. *International Journal of Computer-Supported Collaborative Learning*, 10(4):425–447.

[Deeb and Hickey, 2017] Deeb, F. A. and Hickey, T. (2017). Flipping introductory programming classes using spinoza and agile pedagogy. In *Frontiers in Education Conference (FIE), 2017 IEEE*, pages 1–9. IEEE.

[Deeb et al., 2016] Deeb, F. A., Kime, K., Torrey, R., and Hickey, T. (2016). Measuring and visualizing learning with markov models. In *Frontiers in Education Conference (FIE), 2016 IEEE*, pages 1–9. IEEE.

[Kime et al., 2017] Kime, K., Hickey, T., and Torrey, R. (2017). The calculus dashboard - leveraging intelligent tutor techniques to provide automated fine-grained student assessment. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–8.

[Kime et al., 2015] Kime, K., Torrey, R., and Hickey, T. (2015). CalcTutor: Applying the teachers dilemma methodology to calculus pedagogy. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–8.

[Nikolai Tillmann and Tao Xie, 2014] Nikolai Tillmann, J. D. H. and Tao Xie, J. B. (2014). Code hunt: Gamifying teaching and learning of computer science at scale. *Proceedings of the first ACM Conference on Learning at Scale conference*, pages 221–222.

[Parlante, 2007] Parlante, N. (2007). Nifty reflections. *SIGCSE Bull.*, 39(2):25–26.

[Reimann et al., 2013] Reimann, P., Kickmeier-Rust, M., and Albert, D. (2013). Problem solving learning environments and assessment: A knowledge space theory approach. *Computers & Education*, 64:183–193.

[Roth et al., 2008] Roth, V., Ivanchenko, V., and Record, N. (2008). Evaluating student response to webwork, a web-based homework delivery and grading system. *Computers & Education*, 50(4):1462–1482.

[Simon et al., 2010] Simon, B., Kohanfars, M., Lee, J., Tamayo, K., and Cutts, Q. (2010). Experience report: Peer instruction in introductory computing. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 341–345, New York, NY, USA. ACM.

[Zingaro et al., 2013] Zingaro, D., Cherenkova, Y., Karpova, O., and Petersen, A. (2013). Facilitating code-writing in pi classes. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 585–590, New York, NY, USA. ACM.