Playing to Program: An Intelligent Programming Tutor for RUR-PLE

Marie desJardins, Amy Ciavolino, Robert Deloatch, Eliana Feasley and David Walser
University of Maryland, Baltimore County
1000 Hilltop Circle
Baltimore MD 21250
mariedj@cs.umbc.edu

Abstract

Intelligent tutoring systems (ITSs) have proven their effectiveness in contributing to student learning. ITSs are automated programs that provide students with a one-on-one tutor, allowing them to work at their own pace, so they can spend more time on their weaker areas of the subject matter. The RUR–Python Learning Environment (RUR-PLE), a virtual environment to help students learn to program in Python, provides an interface for students to write their own Python code and then be presented with a visualization of that same code [1]. The RUR-PLE system provides a sequence of learning lessons for students to explore. We are extending RUR-PLE to provide an intelligent tutoring system interface that consists of three components:
(1) a Bayesian student model that tracks student understanding,
(2) a diagnosis module that provides tailored feedback to students, and
(3) a problem selection module that guides the student's learning process.
In this paper, we describe the basis RUR-PLE system and our extensions, and present the results of a user study in which we evaluated the effectiveness of our student modeling approach.

1. Introduction and Related Work

Intelligent tutoring systems rarely live up to their name, all too often failing to provide the intellectual support and customizability we demand of human instructors. For complex problems with multiple concepts, this lack of support for different levels of understanding can be harmful and render a tutoring system useless. In the past, some have attempted to solve this problem using Bayesian analysis [1]. We build upon this work by building a complex framework of a students current level of understanding. In this experiment, we will evaluate the accuracy of our Bayesian model by evaluating how well predictions we make about a students level of understanding model their actual performance on real, challenging problems. The experiment was conducted using a modified version of the Python programming learning environment, RUR-PLE. In RUR-PLE, students write small programs to perform actions that control a robots interaction with a virtual environment and solve a problem. The original system provided a visualization of student code execution, but lacked the ability to diagnosis student problems, tailor feedback, problem select, or a model of student understanding. The enhancements we made are to change RUR-PLE into a user friendly, intelligent system.

[1] rur-ple.sourceforge.net/

2. System Design

The original RUR-PLE design presented a student with a tabular interface that contained an information page, the RUR-PLE specific programming environment, a python interpreter, and a text editor. The information page acts as a panel that permits the student to load instructions for specific problems or view built-in RUR-PLE specific functions. The programming environment allows the student to manipulate the world, load and save the world, and load, save, and run source code. Our system builds upon this original framework to minimize the actions the student must perform and create a more user friendly, question and answer system.

We programmed the system to perform tasks that the original framework required students to execute, such as problem selection, saving of code, and displaying problem instructions. In addition, we added an overlaying interface for student specific access to the system. Students must create a unique student name and complete demographic information to access the system. Once created, they can logout or login at any point with their user name. A password is not required by the system because keeping each students code private is not necessary, only the uniqueness of students is needed.

The enhanced system contains the same tabular interface with changes only made in the information page and the RUR-PLE specific programming environment. The information page contains a static reference page that displays built-in functions. This acts as a quick reference for RUR-PLE specific functions that can be used in the creation of student programs. The RUR-PLE programming environment tab behaves similarly to the original version. Differences arise when loading, selecting, and saving problems. In the enhanced system, each problem is loaded automatically after submission of the previous problem and the instructions for that problem are presented in a separate panel. Student source code and the world environment are saved in the background upon testing or submitting the problem.

3. Student Model and Problem Design

We designed a pretest to administer to students that will evaluate their understanding of various concepts. This pretest consists of twenty multiple-choice questions each of which assesses student understanding across a variety of concepts. The concepts we include in our evaluation are loops, function calls, conditionals and variables. As the students answer various questions correctly or incorrectly, we are able to predict their level of understanding of the various concepts from the accuracy of their responses. We designed pretest problems at a variety of difficulty levels in order to detect fine grained levels of comprehension.

We included a variety of question formats, most of which are multiple choice. These included predictive questions that ask what error will occur when the program attempts to execute a snipped of code, filling in code to complete a simple algorithm, and selecting from several short programs those which successfully accomplish a given task. These problems are designed to test each concept in isolation and the concepts in combination. Likewise, the complex excercises we assigned the students

The problems we presented to the students were more complex, as they involved writing programs to solve problems. These tasks also incorporated different concepts at different levels of difficulty.

4. Mathematical Basis

We model each problem as including a number of concepts, each of which may be present at a different level of difficulty. The concepts represented in the pretest are the ones we use to build our model. As the student answers each question on the pretest correctly or incorrectly , we update our cognitive model. For each problem, there is a prior probability that the student understood the concept and answered it incorrectly, that she understood it and answered it correctly, that she did not understand and guessed the correct answer, or that she did not understand and answered incorrectly. From the evidence we have in the form of the pretest we generate the most likely student model using the formula $P(Model|Evidence) = \frac{P(Evidence|Model)P(Model)}{P(Evidence)}$.

Using this student model, we make predictions about the performance of each student on the more involved problems. Because the students which understand some concepts quite well are more likely to have a high level of understanding of other concepts than those who have no understanding of these concepts, our model will have prior probabilities corresponding to this conceptual fact. We will evaluate the success of our model by comparing our predictions by those made with a more conventional non-Bayesian model such as a decision tree.

5. Experimental Design

Twenty-four participants will be recruited for the study. Participants will be UMBC students who have completed at least one computer science major class but have not completed any computer science classes designed for juniors or seniors. First, the participants will be asked to complete a pretest (described above). After the student has completed the pretest they will be asked to fill out a short form in PIFFL. The form includes the students name, UMBC user name, age, sex, number of credits taken, whether they are a transfer student, the computer science class the student has taken and the grade they have received in each class. The participants will be presented with 8 programming problems to solve in PIFFL. While solving each problem, the student is able to edit the Python code and test the program repeatedly. When the student is done solving the problem, they can submit their solution to move to the next problem. The participants will be able to skip a problem if they have trouble solving it. The pretest and the problems solved are available at our repository [2].

6. Conclusions and Future Work

To create a useful intelligent tutor, it will be necessary to not only predict which problems students are expected to answer correctly, but to use these predictions to provide students with a higher quality learning experience. We plan to integrate our system with an interactive process to continually provide students with more challenging but still appropriate problem sets. This problem selection will have to implement a similar conceptual map to the one we have used to generate our Bayesian network.

Additional future work includes creating a diagnostic module so that instead of merely evaluating whether a problem was answered correctly or not, we can attempt to evaluate where a student went wrong - whether it is their mathematics, their syntax, or their boundary conditions. This can aid us by providing more detailed information with which to update our student model and by assisting us in providing higher quality advice to the student.

Once these additions are implemented, discovering whether this concept map based form of analyzing student understanding helps students learn will be a matter of empirical fact that we can verify through further experiments on student computer scientists.

References

1. Mitrovic, Antonija and Michael Mayo. Optimising ITS Behaviour with Bayesian Networks and Decision Theory, Page 124-153 AI-ED 12(2), 2001.

[2] code.google.com/p/play-to-program/