Playing to Program: Towards an Intelligent Programming Tutor for RUR-PLE

Marie desJardins, Amy Ciavolino, Robert Deloatch, and Eliana Feasley
University of Maryland Baltimore County
1000 Hilltop Circle
Baltimore MD 21250
{mariedj,aci1,rob10,elf4}@umbc.edu

Abstract

Intelligent tutoring systems (ITSs) provide students with a one-on-one tutor, allowing them to work at their own pace, and helping them to focus on their weaker areas. The RUR[1]–Python Learning Environment (RUR-PLE), a game-like virtual environment to help students learn to program, provides an interface for students to write their own Python code and visualize the code execution (Roberge 2005). RUR-PLE provides a fixed sequence of learning lessons for students to explore. We are extending RUR-PLE to develop the Playing to Program (PtP) ITS, which consists of three components: (1) a Bayesian student model that tracks student competence, (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 summarize RUR-PLE and the PtP design, and describe an ongoing user study to evaluate the predictive accuracy of our student modeling approach.

1. Introduction

The challenge of intelligent tutoring systems is to provide the personalized instruction that we expect of human instructors. For complex problems that require a student to understand and apply multiple concepts, the ability to model a student's different levels of competence for these concepts is essential to providing effective instruction. Mitrovic and Mayo (2001) attempted to solve this problem using Bayesian analysis. We apply these ideas in the context of introductory computer programming. Our PtP ITS is being developed within the RUR-PLE learning environment, in which students write small programs to control a robot's interaction with a virtual world. RUR-PLE provides a visualization of student code execution, but does not diagnosis student problems, tailor feedback, select appropriate problems, or model a student's level of competence. Our goal is to extend RUR-PLE with AI techniques to build a user-friendly, intelligent programming tutor.

2. System Design

RUR-PLE presents students with a tab-based interface that contains an information page, the RUR-PLE programming environment, a Python interpreter, and a text editor. The information page permits the student to load instructions for specific problems and to view built-in RUR-PLE functions. The programming environment allows the student to manipulate the world, load and save the world, and load, save, and run source code. Our PtP ITS will automate tasks that the original framework required students to execute manually, including selecting appropriate problems, saving code, and displaying problem instructions. The PtP system contains the same tab-based interface with changes to the information page and the RUR-PLE programming environment. The information page in PtP adds a static reference page that displays built-in functions that can be used in the creation of student programs. The programming environment tab provides more automated support for the loading, selecting, and saving of problems and solutions. In PtP, each problem is selected and loaded automatically after the previous problem's solution is submitted and analyzed; the instructions for each problem are presented in a separate panel. Student source code and the world environment are saved in the background whenever the student tests or submits a problem solution. Ultimately, the student modeling module will be used to allow the problem selection module to identify and provide the student with problems that are neither too difficult nor too easy, to improve the rate at which the student learns new material.

3. Student Modeling Approach

Learning how to program involves developing deep understanding of a number of interrelated concepts. In PtP, we model these concepts using a Bayesian network, with levels of competence inferred from student performance on tasks. Each task is associated with a set of related concepts and associated levels of difficulty. As the student attempts to solve problems, correct solutions provide evidence that a student has a strong command of the associated areas, whereas incorrect solutions provide evidence that a student has failed to develop competence in one or more associated concepts. Our long-term goal is to develop a *concept map* that can be represented as a structured Bayesian network to construct a student model that represents the dependencies among concepts. The structure of the concept map models the fact that some concepts (e.g., function calls) are easier to learn after others (e.g., variables) have been mastered. For the prelimi-

[1]RUR: Rossum's Universal Robots, after Capek's famous play.

nary user study we describe below, we have implemented a simple version of the Bayesian student model, using a naive Bayes model that treats the concepts as being independent. (The design of the user study actually does not depend on the representation of the student model, so we will later be able to expand the model to a richer representation and re-analyze the data using this expanded model.) The student model is a generative model: that is, it describes the expected behavior of a student, given a known level of understanding. For each concept, a student's ability to correctly solve a problem using that concept depends probabilistically on two factors: (1) their level of understanding for the concept and (2) the degree of difficulty with which that concept is expressed in the problem. We model students as having competence in each concept at one of four possible levels of understanding: no competence, very little competence, moderate competence, or full competence. For each concept in each problem, we model the prior probability that the student will answer the the problem correctly as a function of the student's competence of the concept and the difficulty of the problem. Even a student who is highly competent in a concept might answer a problem involving that concept incorrectly for a variety of reasons, including an incorrect interpretation of the problem or a lack of competence in another concept that is required to produce a successful solution. For problems that include multiple concepts, the naive Bayes student model give the probability that the student will answer the problem correctly as the product of the probabilities that they would answer a questions containing each concept individually.

4. Experimental Design

We have designed a user study to evaluate the accuracy of our Bayesian model by measuring the accuracy of predictions about student performance on real, challenging problems, based on observed student performance on simpler "warmup", "pretest", and "posttest" problems. These problems are designed to evaluate student competence in each of various concepts. In all, there are 44 multiple-choice and short-answer questions, each of which assesses student understanding of specific individual concepts, with a difficulty level associated with each concept for each problem. In the current design, the concepts are limited to a few basic programming concepts: loops, function calls, conditionals, and variables. We will use the Bayesian student model to update the student's most likely level of competence for each concept, given which questions they answer correctly and incorrectly. The questions require the student to perform a range of tasks, including predictive questions (identifying what outcome or error will occur when the program attempts to execute a snippet of code), completion questions (filling in code to complete a simple algorithm), and recognition questions (determining which of several short programs will successfully accomplish a given task). These problems are designed to test both individual concepts in isolation and multiple concepts in combination. While the pretest questions are useful for measuring student competence in concepts, the goal of the tutoring is to ensure that students are able to write their own programs from scratch. We use the pretest to create a student model, and then test this model using a set of programming problems that are both more complex and more open-ended than the pretest questions. Like the pretest questions, they incorporate multiple concepts at different levels of difficulty. The student modeling module will use these levels of difficulty, in conjunction with the learned model of the particular student, to predict which questions are likely to be answered correctly. Two metrics of predictive accuracy will be considered: simple accuracy (reducing the probabilistic predictions to yes (greater than 50%)/no (less than 50%) and weighted accuracy (mean squared loss of the predicted probabilities, where a correct answer is treated as 1.0 and an incorrect answer is treated as 0.0). We will also compare the predictive accuracy of student performance on these open-ended problems to that of a control group who answered a set of program-prediction questions. Twenty participants (3 female, 17 male) were recruited for the study. Participants were 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. The scores on the diagnostic warm-up ranged from 14 - 86 percent ($\mu = 68.5$). The scores on the pre and post-tests ranged from 36 - 100 percent ($\mu = 77$). We gathered demographic information (age, sex, programming classes completed and grades in those classes, etc.) in order to analyze the effect of these characteristics on student performance. The pretest and the problems solved are available at our repository (UMBC MAPLE Laboratory 2011). We are currently in the data analysis phase of the user study.

5. Conclusions and Future Work

A useful intelligent tutor must not only predict which problems students are expected to answer correctly, but also 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 leverage the Bayesian student model that uses our proposed structured concept map. Additional future work includes creating a diagnostic module so that instead of merely evaluating whether a problem was answered correctly, we can attempt to evaluate where a student went wrong at a more fine-grained level (e.g., a conceptual misunderstanding, a syntactic error, or incorrect boundary condition checking). This improved diagnosis will aid in student modeling and problem selection.

References

Mitrovic, A., and Mayo, M. 2001. Optimising ITS behaviour with Bayesian networks and decision theory. *International Journal of Artificial Intelligence in Education* 12(2):124–153.

Roberge, A. 2005. RUR: A Python learning environment. http://rur-ple.sourceforge.net/.

UMBC MAPLE Laboratory. 2011. Playing to Program repository. http://code.google.com/p/play-to-program/.