Knowledge Maximizer: Concept-Based Adaptive Problem Sequencing for Exam Preparation

Roya Hosseini[1], Peter Brusilovsky[1], and Julio Guerra[2]

[1] University of Pittsburgh, Pittsburgh, USA
{roh38,peterb}@pitt.edu
[2] Universidad Austral de Chile, Valdivia, Chile
jguerra@inf.uach.cl

Abstract. To support introductory Java programming students in preparing for their exams, we developed Knowledge Maximizer as a concept-based problem sequencing tool that considers a fine-grained concept-level model of student knowledge accumulated over the semester and attempts to bridge the possible knowledge gaps in the most efficient way. This paper presents the sequencing approach behind the Knowledge Maximizer and its classroom evaluation.

Keywords: problem sequencing, concept-based student model.

1 Introduction

Exam preparation is a challenging task for college students. For many courses, students need to review the content that was studied over the whole semester within a short time frame, identify possible knowledge gaps and misconceptions, and remediate these gaps. An adaptive problem-sequencing tool, based on a fine-grained concept-level student model, could be very helpful in this context. By reflecting students' progress over the whole semester, the student model can distinguish between: 1) concepts that were learned well and need not be practiced again; 2) concepts that were not mastered and need to be reviewed; 3) and concepts that were missed and may need a thorough review. Based on this model, an adaptive problem-sequencing tool can individually guide each student through the exam preparation process.

While concept-level adaptive sequencing is a relatively mature and well-known approach [1; 2], there are still no instances of its use in the context of exam preparation. This context, however, is different from the traditional sequencing that carries a student through the course. Exam-time sequencing implies that a student has a relatively complete knowledge of course materials and little time to improve it. Instead of gradual coverage of concepts, exam-time sequencing should focus on bridging knowledge gaps while trying to maximize the number of concepts that are assessed and mastered by completing each suggested problem. To explore sequencing in this interesting context, we developed Knowledge Maximizer, a concept-based problem sequencing tool for Java programming exam preparation. This paper presents the sequencing approach of Knowledge Maximizer and the results of its classroom study.

K. Yacef et al. (Eds.): AIED 2013, LNAI 7926, pp. 848–851, 2013.

2 The Knowledge Maximizer

The goal of the Knowledge Maximizer (KM) is to provide the learner with a sequence of questions to help address gaps in Java knowledge as quickly as possible. To this end, KM uses an overlay student model in conjunction with a concept-level model of Java knowledge represented in the form of Java ontology. The learning content in KM comprises 103 parameterized self-assessment questions (activity) indexed by ontology concepts. The indexing distinguishes *prerequisite* and *outcome* concepts for each activity. To select and rank the 10 most important activities, KM uses the following factors:

How prepared is the student to do the activity? The activities for which the student has less knowledge of prerequisite concepts are not appropriate suggestions. We calculate the learner knowledge for each of the prerequisite concepts in an activity to see how well the student is prepared to do it. Eq.1 shows the formula:

$$K = \frac{\sum_{i}^{M_r} k_i w_i'}{\sum_{i}^{M_r} w_i'} \qquad w_i' = \log(w_i) \qquad \text{Eq. (1)}$$

where K is the learner's knowledge level about the prerequisites of the activity; w_i' is the log-smoothed weight for the concept; k_i is the level of the learner's knowledge about the i^{th} concept and M_r is the set of prerequisite concepts for the activity. More knowledge of prerequisite concepts for an activity (higher K) makes it a better candidate for selection by the optimizer. Due to the short duration of the course and the complexity of the Java concepts, we do not take knowledge decay into account in Eq.1.

What is the impact of the activity? The formula for this impact is shown as Eq.2 where M_o is the set of outcome concepts for the activity (i.e., concepts that are mastered by the student while working with the activity). Impact I of a certain activity measures how well it addresses the current lack of knowledge. An activity with a higher impact factor is a better candidate for selection by the optimizer.

Has the user already completed the activity? We define it as Eq.3 where $\bar{S}$ is the inverse success rate of the student in the activity; s is the number of the times the student has succeeded in the activity; and t is the total number of times the student has attempted to complete the activity.

$$I = \frac{\sum_{i}^{M_o} w_i'(1-k_i)}{\sum_{i}^{M_o} w_i'} \qquad \text{Eq. (2)} \qquad \bar{S} = 1 - \frac{s}{t+1} \qquad \text{Eq. (3)} \qquad R = K + I + \bar{S} \qquad \text{Eq. (4)}$$

Having calculated these factors, we simply rank the activities using Eq. 4 where R is the rank of the activity which is obtained by summing over the values of K, I, and $\bar{S}$.

Fig. 1 shows the KM interface. The question with the highest rank is shown first. Users can navigate the ranked list of questions utilizing navigation buttons at the top. The right side of the panel shows the list of concepts covered by the question. The color next to each concept visualizes the student's current knowledge level (from red representing less knowledge to green representing more knowledge).

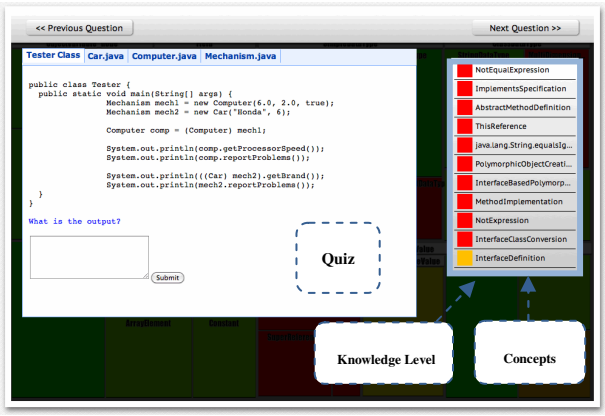

Fig. 1. The Knowledge Maximizer interface

3 The Evaluation

To assess the value of Knowledge Maximizer, we conducted a classroom study in the context of a Java-based undergraduate programming course at the School of Information Sciences, University of Pittsburgh. All students enrolled in this course were invited to use the KM during preparation for the final exam. The study began about a week prior to the final exam. Throughout the course, students used two other adaptive tools, QuizGuide, and Progressor+ to work with Java problems. Both tools reported student knowledge updates to the central student model server which was also used by KM. As a result, many students mastered a significant number of Java concepts by the time they started with KM and were ready to benefit from its "gap filling" nature.

In our analysis, we counted separately questions accessed from KM and questions accessed from QuizGuide or Progressor+. Attempts made from KM were made by 14 students while attempts made from QuizGuide/Progressor+ were made by 17 students. To assess whether KM was successful in "maximizing" students' progress towards the goal, we grouped questions into three different complexity levels based on the number of involved concepts: 1) Easy, 2) Moderate, and 3) Complex. Table 1 lists the number of attempts made to do easy, moderate, and complex questions from KM and from QuizGuide/Progressor+. The data reveals that the number of attempts to access complex questions was about 2.5 times greater in KM. Despite a remarkable increase in complex questions in KM, the success rates across all systems were comparable.

To compare the effect of KM and the other systems on the improvement of students' performance, we compared quiz grades obtained by the students in the second part of the course and their post-test results. Since in-class quizzes and post-tests have different numbers of questions, we used a percentage of the total as a relative score. We discovered that the average increase in performance percentage among the students who used QuizGuide/Progressor+ was 12% (0.68% to 0.8%) while KM users experienced an average increase of 19% (0.53% to 0.72%). Moreover, students who

used KM "for real" (i.e., made at least 10 attempts using KM) achieved a 28% increase (0.48% to 0.76%). This provides some evidence (as much as could be collected in a non-controlled classroom situation where learning can happen outside of the systems) that KM acted as a strong exam preparation tool, surpassing the more traditional adaptive systems QuizGuide/Progressor+ not designed for exam preparation.

Table 1. Number of Attempts, success rates by System and complexity level

Complexity	KM (n=14)		QG,P+ (n=17)	
	Number of Attempts	Success rate	Number of Attempts	Success rate
Easy	27 (6.2%)	93%	1123 (34.6%)	73%
Moderate	189 (43.5%)	68%	1471 (45.3%)	61%
Complex	218 (50.2%)	46%	651(20.1%)	55%
Total	434	58%	3245	64%

4 Conclusion and Future Work

We have explored adaptive problem sequencing in KM to support exam preparation in a Java programming class. Results of our study revealed the ability of KM to generate challenging questions that shortened the path to students' learning goals. KM can be applied to any other domains with ontology and questions indexed by ontology concepts. Our future work will focus on improving KM by considering more parameters that affect the selections of questions, such as the timing factor.

Acknowledgement. This research was supported in part by the National Science Foundation under Grant No. 0447083. Julio Guerra is supported by a Chilean Scholarship (Becas Chile) from the National Commission for Science Research and Technology (CONICYT, Chile) and the Universidad Austral de Chile.

References

1. Brusilovsky, P.: A framework for intelligent knowledge sequencing and task sequencing. In: Frasson, C., McCalla, G., Gauthier, G. (eds.) ITS 1992. LNCS, vol. 608, pp. 499–506. Springer, Heidelberg (1992)
2. Kumar, A.N.: A Scalable Solution for Adaptive Problem Sequencing and its Evaluation. In: Wade, V., Ashman, H., Smyth, B. (eds.) AH 2006. LNCS, vol. 4018, pp. 161–171. Springer, Heidelberg (2006)