Fine-Grained Adaptive Problem Sequencing for Exam Preparation in Knowledge Maximizer

Submitted in fulfillment of the requirement of the
ISSP 2990 Independent Study course

By:

Roya Hosseini

Advised By:

Peter Brusilovsky

Intelligent Systems Program
University of Pittsburgh
Pittsburgh, USA

Fall 2013

Acknowledgments

I would like to thank my advisor, Dr. Peter Brusilovsky for the valuable advice and support he has given me during my independent study for fall 2013.

Abstract

To support introductory Java programming students in their exam preparation, 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 fastest possible way. This report presents the sequencing approach behind the Knowledge Maximizer and its classroom evaluation.

Contents

1 Introduction

Exam preparation is a challenging task for college students. In many courses, in a few days of exam preparation, students need to review the content that was studied over the whole semester, identify possible knowledge gaps and misconceptions, and fill 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, a student model can distinguish concepts that were well learned and need not to be practiced again, concept with unstable knowledge that need a refresh, and concepts that were missed and may need a thorough review. Using this model, an adaptive problem-sequencing tool can individually guide each student through the exam study process.

While concept-level adaptive sequencing is a relatively old and well-explored approach [1;3;4], there are still no examples of its use in the context of exam preparation. This context, however, is different from traditional sequencing that carries a student through the course. Exam-time sequencing implies that a student has a relatively complete level of course knowledge and very 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 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. Knowledge Maximizer was integrated within a system called Knowledge Zoom. This paper briefly presents both of the system and will more focus on sequencing approach behind the Knowledge Maximizer and the results of its classroom study.

2 Knowledge Zoom

To demonstrate the importance of fine-grained indexing, we look to a system called Knowledge Zoom (KZ). The goal of KZ is to help the students identify their course knowledge gaps and provide tools to bridge these gaps in an effective way. The first part of this dual goal is supported by the KE component, a concept-based hierarchical zoomable open student model.

The second goal is supported by the KM, a concept-based adaptive problem sequencing tool. The interface of KZ (Fig.1) provides direct access to the Knowledge Explorer (KE) model and a button to launch the Knowledge Maximizer (KM). KZ is based on a concept-level model of knowledge about Java and OOP. This model is formed by a subset of concepts from the Java ontology-http://www.sis.pitt.edu/~paws/ont/java.owl built by the PAWs lab. The Java ontology includes 344 concepts organized into an 8-level tree. The learning content in KZ is formed by 103 parameterized self-assessment questions that were developed in our team as a part of an earlier project [2]. Each question is indexed with ontology concepts. The indexing classifies the prerequisite concepts that should be known before approaching the question and the outcome concepts to be mastered by working with

the question. The number of concepts associated with a single question ranges from 5 to 52 (0 to 41 prerequisites, 1 to 12 outcomes). These questions cover the 188 most important concepts of Java which form the KZ domain model.

Knowledge Explorer is a multi-level open student model visualized with a zoomable Treemap. The information presented by KE is an overlay model of Java Knowledge based on the KZ ontological domain model. The overlay student model in KZ is maintained by a user modeling service, PERSEUS [5], which updates the model after every attempt to answer a question and changes the knowledge level of concepts re-lated to the ques-tion.

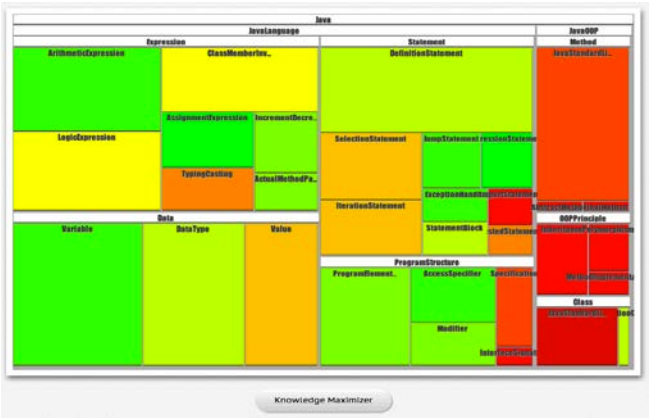

Fig. 1. The KnowledgeZoom interface showing the top level of the Knowledge Explorer map and a button to launch Knowledge Maximizer.

Fig. 2. Zooming on the node Expressions (top left corner in Fig. 1) reveals next level of the concept hierarchy. Now the user can see that the node LogicExpression that has intermediate knowledge as a whole (shown as yellow) consists of several well learned and several unknown concepts.

A zoomable Treemap was selected to present the student model due to its relatively large size and hierarchical nature. The Treemap layout shows only four levels of concept hierarchy starting from the current top node and hiding lower-level nodes behind its ancestor node. The user, however, can zoom in any node. After zooming in, the node expands becoming the top node and occupying the whole view. Zooming-in immediately exposes previously hidden levels of hierarchy. For example, Fig. 2 shows the results of zooming into a second level concept, Expression shown in the top left quadrant of Fig. 1.

In the Treemap layout, each node (a concept in the Java ontology) is shown as a colored rectangle. A leaf concept of the ontology corresponds to a terminal node of the Treemap. The size of a node represents the importance of a concept in the context of Java language and its chance to be checked as part of the exam. We measure it by counting how many questions are related to the leaf concept corresponding to this leaf node in the Treemap. Since the number of exercises related to nodes can be quite different, which leads to a large difference in the node sizes, we use the log2(size) to moderate the differences. The color of a node represents the level of concept knowledge demonstrated by a student. We use 10 colors from red to green to represent the progression from weaker to stronger knowledge.

In a hierarchical zoomable layout, a leaf node directly represents the importance and knowledge level of a concept with its size and color respectively, while each intermediate node accumulatively aggregates importance and concept knowledge from its child nodes. As a result of the aggregation, the upper-level views show overviews of students' state of knowledge on higher levels (Fig. 1), while being able to explore detailed knowledge of every concept as zooming into lower levels of the ontology (Fig. 2). The calculation of the aggregated size and color is important to bridge the gaps between lower and higher levels of views. In KE, the size aggregation is provided by Treemap. For the color aggregation, the color of an intermediate node is the average color of its direct child nodes weighted with their sizes in order to reflect the importance of the associated concepts.

3 Knowledge Maximizer

Knowledge Maximizer is the component inside KZ which provides fine-grained adaptive problem sequencing. The goal of the Knowledge Maximizer (KM) is to provide the learner with a sequence of questions that help her to complete gaps in her Java knowledge as fast as possible. To this end, KM uses an overlay student model built over a concept-level model of Java knowledge represented in the form of Java ontology. The learning content in KM is formed by 103 parameterized self-assessment questions (activity) indexed with ontology concepts. The indexing distinguishes *prerequisite* and *outcome* concepts of each activity. To rank and select top 10 activities, KM uses the following factors:

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

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

where K is the level of the learner's knowledge in 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 in the i^{th} concept and M_r is the set of prerequisite concepts for the activity. Higher knowledge of prerequisite concepts of an activity (larger K) makes it a better candidate to be selected by the optimizer. Note that due to the short duration of the course and complex level of Java concepts we do not take knowledge decay into account in (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 of the activity (i.e., concepts that are mastered by the student while working with the activity). Impact I of selecting a certain activity measures how much it addresses the current lack of knowledge. An activity with a higher impact is a better candidate to be selected 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 tried the activity.

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

Having calculated the above factors, we can simply rank the activities using Eq. 4 where R is the rank of the activity; and α, β, and γ are the weights assigned to each of the above-mentioned factors respectively. Fig. 3 shows KM interface. The question with the highest rank is shown first. User can navigate the ranked list of questions using navigation buttons at the top. 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 to green).

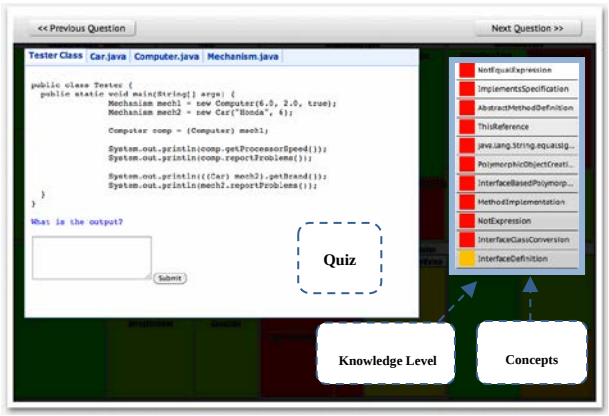

Fig. 3. The Knowledge Maximizer interface

4 The Evaluation

To assess the value of Knowledge Maximizer (KM) 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 for the final exam preparation. The study started about a week before the final exam. Over the course duration, students used two other adaptive tools, QuizGuide, and Progressor+ to work with Java problems. Both tools reported student knowledge update to the central student model server also used by KM. As a result, many students learned a considerable number of Java concepts by the time they started with KM and were in a good position to benefit from its "gap filling" nature.

In our analysis we separately counted question accesses from KM and questions accessed from either QuizGuide/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" student steps 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 easy, moderate, and complex questions from KM and from QuizGuide/Progressor+. The data reveals that the number of attempts to complex questions which help students reach their goal faster by covering many concepts at once was about 2.5 times greater. Despite a remarkable increase in complex questions, the success rates across all systems were comparable.

To compare the effect of KM and other systems on the improvement of students' performance we compared quiz grades obtained by the students in the second part of the course and post-test results. Since in-class quizzes and post-test have different number of questions we used relative score as a percentage of total. We found out that the average increase in performance percentage among the students who used QuizGuide/Progressor+ was 12% (0.68% to 0.8%) while KZ users had the average increase of 19% (0.53% to 0.72%). Moreover, students who used KZ "for real" (i.e., made at least 10 problem attempts using KZ) achieved 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 the systems) that KZ acted as a good exam preparation tool surpassing more traditional adaptive systems QuizGuide/Progressor+ that were not designed for exam preparation time.

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%

5 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 showed capability 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 affects the selections of questions such as timing factor.

References

1. Brusilovsky, P. A framework for intelligent knowledge sequencing and task sequencing. In: Frasson, C., Gauthier, G. and McCalla, G. (eds.) Proc. of Second International Conference on Intelligent Tutoring Systems, ITS'92, (Montreal, Canada, June 10-12, 1992), Springer-Verlag, 499-506.
2. Hsiao, I.-H., Sosnovsky, S., Brusilovsky, P.: Guiding students to the right questions: adaptive navigation support in an E-Learning system for Java programming. Journal of Computer Assisted Learning 26, 4 (2010) 270-283
3. Kumar, A.N. A Scalable Solution for Adaptive Problem Sequencing and its Evaluation. In: Wade, V., Ashman, H. and Smyth, B. (eds.) Proc. of 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'2006), (Dublin, Ireland, June 21-23, 2006), Springer Verlag, 161-171.
4. McArthur, D., Stasz, C., Hotta, J., Peter, O., and Burdorf, C. Skill-oriented task sequencing in an intelligent tutor for basic algebra. Instructional Science, 17, 4 (1988), 281-307.
5. Yudelson, M., Providing service-based personalization in an adaptive hypermedia system. PhD Thesis. U. of Pittsburgh, 201