Rule-Based Adaptive Problem Generation in Programming Tutors and its Evaluation

Amruth KUMAR
Ramapo College of New Jersey,
Mahwah, NJ 07430, USA

Abstract. We will propose a rule-based mechanism for adaptive generation of problems in intelligent tutors. We will present the domain model, student model, and the algorithms for rule-based adaptation in the context of web-based programming tutors. Finally, we will present the web-based protocol we used to evaluate rule-based adaptation and discuss the results. Our evaluation shows that rule-based adaptation helps students learn with fewer practice problems. Rule-based adaptation has several advantages – it is domain-independent, flexible and scalable.

Keywords: Programming tutor, Problem generation, Rule-based adaptation, Evaluation

1. Introduction

We have been developing web-based tutors to help students learn programming language concepts by solving problems. To date, we have developed tutors on expression evaluation, pointers in C++, counter-controlled loops, parameter passing mechanisms, scope concepts and their implementation, and classes. The tutors present programming problems to the learner, grade the learner's answer, provide a detailed explanation of the correct answer, log the student's performance, and determine whether the student has learned the material. Our tutors address application (predicting the behaviour of a program) and analysis (debugging a program) in Bloom's taxonomy [6], as opposed to program synthesis (writing a program), which has been the focus of many earlier works (e.g., LISP Tutor [19], PROUST [10], BRIDGE [7], ELM-ART [21] and Assert [3]). Our tutors are designed to be used as supplements to traditional programming projects, as recommended by the whole language approach [15].

In this paper, we will propose a rule-based mechanism for adaptive generation of problems in intelligent tutors. It applies the traditional rule-based reasoning to adaptively generate problems in web-based tutors. We will first describe our domain and student models, followed by a description of the rule-based adaptation algorithm in the context of our programming tutors. We evaluated the rule-based adaptation in fall 2004. We will present the web-based evaluation protocol and discuss the results of our evaluation.

2. The Domain Model

We have identified a set of learning objectives for each programming topic. Learning objectives for a topic are concepts that must be understood in order to learn the topic. Preferably, these concepts are at a fine level of granularity so that problems can be designed to teach or assess them individually. For instance, the learning objectives for arithmetic expressions are:

- Correct evaluation, precedence, associativity and coercion of addition, subtraction and multiplication operators;
- Correct evaluation of integer and real division, precedence, associativity and coercion of division operator and divide by zero error;
- Correct evaluation, precedence and associativity of the remainder operator, divide by zero error and the inapplicability of remainder operator to real operands.

Typically, we identify 20-30 learning objectives per topic. Note that we also include typical errors associated with a topic as learning objectives for the topic. These are not errors in the student's application of procedures, as described in the theory of bugs [8], but rather, syntax, semantic and run-time errors that are an inherent part of the programming domain – understanding of the programming domain would not be complete without knowledge of these bugs.

We use a single unified domain model for all our programming tutors. This domain model is the concept map of the programming domain, enhanced with learning objectives. The concept map is a taxonomic map of the domain, with domain topics as nodes, and is-a and part-of relationships as arcs. The learning objectives for a topic serve as the children of the node for that topic. The domain model is a hierarchical tree, with domain topics as intermediate nodes and learning objectives as leaf nodes.

For each topic, we list the learning objectives in increasing order of complexity. Often, learning objectives are independent of each other, and can be listed in any order. E.g., precedence and associativity are two independent learning objectives for an arithmetic operator – the student can learn about one independently of the other. However, when a learning objective is dependent on another learning objective, it is listed after that learning objective. E.g., it is necessary for a student to learn nested independent loops before nested dependent loops. So, we list dependent loops after independent loops.

In the domain model, for each learning objective, we identify the level of expected proficiency. We represent the level of proficiency in terms of two measures:

- M_1 - The minimum number of problems the learner must solve on that learning objective. Some considerations for setting the value of M_1 are:
    - M_1 should be set high enough for novices to be able to learn the concept necessary to satisfy a learning objective. For instance, $M_1 = 1$ does not provide for reinforcement of learning.
    - M_1 should be set low enough that advanced students who have learned the concepts corresponding to a learning objective are not encumbered with unnecessary problems. $M_1 \geq 4$ could result in students solving redundant problems on a learning objective well after they have learned the corresponding concepts.

  Typically, we set $M_1 = 2$. For harder learning objectives, we set $M_1 = 3$.
- M_2 - The percentage of problems that the learner must solve correctly on the learning objective to be considered proficient in it. Some considerations for setting the value of M_2 are:
    - M_2 should not be set so low that students meet it without learning the concepts corresponding to the learning objective. M_2 should be greater than the greatest probability of guessing the correct answer to *any* problem for that learning objective.
    - M_2 should not be set so high that students are forced to solve additional problems on a learning objective even after they have learned the associated

concepts. Given a student who already knows the concepts associated with a learning objective, if n is the maximum number of problems the tutor might tolerate having the student solve, $M_2 = n / M_1$.

Typically, when $M_1 \geq 2$, we set $M_2 = 60\%$ - the learner must solve at least 2 problems correctly in order to satisfy a learning objective. For harder learning objectives, we set it lower (e.g., 50%).

If $M_1 = 0$, the tutor does not generate any problems for the learning objective. If $M_1 \neq 0$, but $M_2 = 0$, the tutor generates exactly M_1 problem(s) on the learning objective. Our tutors use these proficiency measures to determine whether the learner has "satisfied" each learning objective.

3. The Student Model

We use an overlay of the above domain model as our cognitive student model. But, instead of saving M_1 and M_2 with each learning objective, we save five terms that record the student's progress - the number of problems generated (G), attempted (A), correctly solved (C), incorrectly solved (W) and missed (M) by the student on that learning objective. Maintaining student progress in this raw form enables us to be flexible about how we interpret it. Currently, our tutors use the following two inequalities to interpret this data and determine whether a student has "satisfied" a learning objective:

- $A \geq M_1$ - Ensures that the student has attempted a minimum number of problems for the learning objective;
- $C / A \geq M_2$ - Ensures that the student has solved a minimum number of problems correctly for the learning objective.

Several researchers have proposed using a pre-test to initialize the student model in adaptive tutors (e.g., [1, 9]). Recently, researchers have proposed various improvements to the pre-test:

- Some researchers have proposed using adaptive pre-tests to minimize the number of problems the learner must solve (e.g., [2, 18]).
- Other researchers have proposed using stereotypes, using a shortened pre-test to stereotype the learner and initializing the student model according to the selected stereotype [1, 11].
- Another recent proposal is to use schema-based assessment of learner's knowledge to quickly initialize the student model [12]. If acquisition of solution schemas is a characteristic of expertise in a domain, express tests can be devised for the domain wherein the learner fills in incomplete intermediate stages in a solution rather than come up with the entire solution.

In our web-based tutors, we use a pre-test to initialize the student model. We chose not to use adaptive pre-tests because we wanted to compare the pre-test score with the score on a similarly constructed post-test to evaluate the effectiveness of the adaptive tutor.

4. Problem Templates

Limited problem set has been recognized as a potential drawback of encoding a finite number of problems into a tutor [16]. In our web-based tutors, we generate problems as instances of parameterized templates, a scheme similar to that found in [4, 13]. Every instance of a template is a new problem and no two problems are identical. This enables our tutors to present different instances of a template to different users at a given time (to prevent plagiarism), or to the same user at different times (for test-re-test). Whereas Belmont et al [4] have proposed templates to automatically generate problems such as true/false and fill-blanks, we focus on the generation of debugging problems, problems on predicting the output of programs and problems on evaluating expressions.

On each topic, i.e., for each tutor, we have coded a repository of problem templates. These templates are indexed by learning objectives. Each template T_j and its associated learning objective(s) L_i constitute a rule of the following form in our template knowledge base:
If tutoring is desired for the learning objective L_i, then use template T_j.
E.g., the following is a template on arithmetic expressions:

> **Template No.** 120
> **Learning Objective:** /.Real.Correct
> **Template:** 24 / <R1#integer;2<=R1<=8;#>
> **Type:** expression

The learning objective associated with the above template is the correct evaluation of real division. The template contains a meta-variable R1, which is instantiated during problem generation to an integer value between 2 and 8, inclusive. So, the tutor may generate any of the following problems from the above template: 24 / 2, 24 / 3, 24 / 4, 24 / 5, 24 / 6, 24 / 7 or 24 / 8. Typically, we have encoded 20-25 templates per learning objective in our template knowledge base.

5. Rule-Based Adaptation of Problem Generation

We will now present the algorithm for rule-based adaptation of problem generation. This algorithm assumes that the problem templates are indexed by learning objectives and the student model is represented in terms of learning objectives.

The Algorithm:

1. Let the set of all the learning objectives on the topic be AL = $\{L_1, L_2, \ldots, L_m\}$, where $L_1, L_2, \ldots, L_m$ are individual learning objectives.
2. For each learning objective L_i, extract from the template knowledge base, all the templates that match the objective. Let the resulting set of templates be $T_i = \{T_{i1}, T_{i2}, \ldots, T_{ip}\}$, where $T_{i1}, T_{i2}, \ldots, T_{ip}$ are individual templates that match L_i.
3. Identify the list of learning objectives that the learner has not yet satisfied. Let this set be $L = \{L_1, L_2, \ldots, L_n\}$, $n \leq m$. If the set L is empty, the student has mastered this topic, exit.
4. Select the next learning objective L_j from the set L.
5. Select the next template T_{jk} from the set of templates T_j corresponding to the learning objective L_j and generate the next problem as an instance of the template.
6. After the learner has attempted the problem, update G,A,C,W and M for the learning objective L_j in the student model, as well as any other learning objective affected by the template T_{jk}. Repeat from Step 3.

Sub-algorithm for Step 4: First, we define **persistence** p as the maximum number of problems a tutor generates back to back on a learning objective before moving on to the next learning objective. Given the last learning objective was L_i, the algorithm to select the next learning objective is as follows:

1. If L_i has been satisfied, return the next learning objective in the list L_{i+1}. If $i + 1 > n$, the number of learning objectives not yet satisfied, set $i = 1$, and return L_1
2. If p problems have been generated back to back on the learning objective L_i, return L_{i+1}. If $i + 1 > n$, set $i = 1$, and return L_1
3. Else, return L_i.

Since the learning objectives are listed in increasing order of complexity in the domain model (of which the student model is an overlay), the tutor generates problems on a learning objective only after generating problems for all of its pre-requisite learning objectives. As to the value of persistence p, the limit that we introduced on the number of problems the tutor would present back to back on a learning objective:

- $p = 1$ means that the learning objective is changed from one problem to the next. This may not reinforce learning due to rapid switching of the learning objective.
- $p = 2$ or 3 helps reinforce learning since the tutor presents 2-3 problems back to back on a learning objective. However, if the student satisfies the learning objective with fewer than p problems, the above algorithm moves the student to the next learning objective.
- $p > 3$ may make the tutor predictable and boring. The student may begin guessing the correct answer to problems, which would negatively affect learning.

Sub-algorithm for Step 5: We use the round-robin algorithm for selecting the next template for a learning objective. If the last template used by the tutor for a learning objective is T_{ij}, the next time it revisits the learning objective, it uses the template T_{ij+1}.

This rule-based algorithm is independent of the domain: it can be used for any domain wherein 1) appropriate learning objectives can be identified; 2) the student model is maintained in terms of learning objectives; and 3) problem templates are indexed by learning objectives. This rule-based adaptation algorithm has several advantages over vector spaces [20] and learning spaces [14] that have been popularly used to implement adaptation:

- The rule-based system is easier to build - there is no need to place all the problem templates in an exhaustive vector or learning space.
- The rule-based system is easily scalable - in order to add a new learning objective, we simply insert it in the domain model of which the student model is an overlay, and add additional problem templates to the template knowledge base, indexed by the new learning objective. This will not affect any existing learning objectives or their templates.

The learning path of individual learners is determined by the matching of the templates in the template knowledge base with the unsatisfied learning objectives in the student model. A rule-based system automatically supports all the learning paths - even those that may not have been explicitly modelled in a vector or learning space. Therefore, the resulting adaptation is more flexible. Our rule-based adaptation is similar to the adaptation mechanism used in ActiveMath [17] to determine the information, exercises, and examples presented to the learner, and the order in which they are presented.

5.1 An Example

Consider the tutor on arithmetic expressions. For this example, we will consider only the following learning objectives: correct evaluation and precedence of +, * and / operators. Let the following table represent the initial student model, where m / n denotes that the student has correctly solved m out of the n problems (s)he has attempted on the learning objective:

Student Model	+	*	/
Correct Evaluation	2/2	½	0/2
Precedence	0/2	2/2	1/2

Assuming $M_1 = 2$ and $M_2 = 60\%$, the student has not yet satisfied the following learning objectives: correct evaluation of * and /, and precedence of + and /. Assume that the next template for the correct evaluation of * yields the expression 3 + 4 * 5, and the student correctly solves the entire expression. Since the expression includes the correct evaluation and precedence of + and * operators, the student gets credit for all four learning objectives:

Student Model	+	*	/
Correct Evaluation	3/3	2/3	0/2
Precedence	1/3	3/3	1/2

Since the student just satisfied the learning objective of the correct evaluation of *, the tutor considers the next unsatisfied learning objective, viz., correct evaluation of /. Assume that the next template for the correct evaluation of / yields the expression 5 + 10 / 4, and the student correctly solves the entire expression. Since the expression includes the correct evaluation and precedence of + and / operators, the student gets credit for all four learning objectives:

Student Model	+	*	/
Correct Evaluation	4/4	2/3	1/3
Precedence	2/4	3/3	2/3

If persistence p = 2, the tutor generates a second problem on the correct evaluation of /. Note that even if the student solves the second problem correctly, the learning objective of correct evaluation of / will remain unsatisfied (2/4 < 60%). All the same, since persistence p = 2, the tutor will pick the next learning objective for the subsequent problem.

Following are highlights of our adaptive algorithm:

- A student may satisfy a learning objective without attempting any problem on it. Note that the student satisfied the precedence of / operator while attempting problems on the other learning objectives. However, in order for this to occur, the tutor must be capable of automatically allocating (partial) credit. Our tutors on expression evaluation are capable of doing so.
- It is possible for a student who has already satisfied a learning objective to revert to the unsatisfied state. For instance, if the student had incorrectly solved the last two problems, correct evaluation of + would have reverted from satisfied (2/2) to unsatisfied state (2/4).

6. Evaluation of the Adaptive Tutor

In spring 2005, we conducted a web-based evaluation [5] of the rule-based adaptation in our tutor on arithmetic expressions. We used a between-subjects design: students were randomly assigned to either the control or the experimental group by the tutor. The control group used the non-adaptive version of the tutor and the experimental group used the adaptive version. Students used the tutor asynchronously, as part of a mandatory non-credit course assignment.

Protocol: We used the pre-test-practice-post-test protocol for evaluation of both the versions of the tutor:

- **Pre-test** – We used this stage to assess the prior knowledge of the students. The tutors used the pre-test to initialize the student model. The pre-test consisted of 21 problems covering over 20 different learning objectives for arithmetic expressions. Students were allowed 7 minutes for the pre-test. The tutor did not provide any feedback during the test.

- **Practice** – This stage was designed to help students learn from the tutor. The tutor provided detailed feedback for each problem.
  - **Non-Adaptive** tutor: This tutor presented 3 practice problems per learning objective, in the same order of learning objectives as on the pre-test. All the students were presented the same sequence of problems, regardless of how well they did on the pre-test. In other words, the tutor did not adapt to the learner's needs. The practice session lasted 15 minutes.
  - **Adaptive** tutor: This tutor adapted to the student's needs in two ways:
    - It presented problems on only those learning objectives that the student did not satisfy on the pre-test.
    - For each learning objective that the student did not satisfy, it presented 3 problems at a time or until the student satisfied the learning objective, whichever came first, before continuing with the next learning objective not yet satisfied by the student.

    The practice session lasted 15 minutes or until the student satisfied all the learning objectives, whichever came first. Therefore, students who satisfied all the learning objectives on the pre-test were presented no problems during practice. Those who did not satisfy any learning objective, and worse, solved all the problems incorrectly on the pre-test were presented problems in the same sequence as the non-adaptive version of the tutor.
- **Post-test** – We used this stage to assess the effect of practicing with the tutor, on the learning of the students. The post-test consisted of 21 problems, in the same order of learning objectives as on the pre-test. Students were allowed 7 minutes for the post-test. The tutor did not provide any feedback during the test.

The three stages: pre-test, practice and post-test were administered by the tutor back-to-back, with no break in between. The students did not have access to the tutor before the experiment.

Analysis: We calculated the percentage correctness of each answer, and calculated the average of these percentages for each student. Since this is per-problem average correctness, it eliminates practice effect that usually leads to students solving more problems on the post-test than on the pre-test. Table 1 lists the class average of these student averages on the pre-test and post-test for the non-adaptive and adaptive versions of the tutor. The improvement from the pre-test to the post-test was statistically significant (paired t-test 2-tailed *p* value < 0.05) for both the versions of the tutor. One-way ANOVA analysis showed that the difference from the pre-test to the post-test was statistically significant in both the groups. The only other statistically significant difference was between non-adaptive pre-test and adaptive post-test groups.

Table 1. Non-adaptive versus Adaptive Tutor

Average correctness of answers	Pre-Test	Post-Test	Change	Significance
Without adaptation (N = 15)				
Average	0.47	0.65	0.17	p = 0.014
Standard Deviation	0.24	0.20	0.24	
With adaptation (N = 25)				
Average	0.55	0.69	0.14	p = 0.0002
Standard Deviation	0.21	0.20	0.16	

However, the difference in the number of problems solved by the adaptive and non-adaptive groups was statistically significant (independent 2-tailed t-test p-value < 0.05). The minimum, maximum and average number of problems solved by the two groups during the practice session is listed in Table 2. Given that the improvement in learning was similar for both the groups, and that there was a statistically significant difference between the numbers of problems solved by the two groups during practice, our results are in accordance with the earlier result that adaptive problem sequencing helps students learn with fewer problems. For this evaluation, we did not consider the time spent by the students on practice since all the students on the control group were required to practice for 15 minutes.

Table 2. Problems Solved by the Control and Experimental Groups during 15-minute Practice

	Control Group (Non-Adaptive Tutor)	Experimental Group (Adaptive Tutor)	Statistical Significanc e
Minimum problems solved	28	1	
Maximum problems solved	86	60	
Average Problems solved	45.80	24.22	$p = 0.00017$
Standard Deviation	15.44	14.56	

7. Conclusions

We proposed a rule-based mechanism for adaptive generation of problems in web-based intelligent tutors. We described the domain and student models in our programming tutors, and presented an algorithm for rule-based adaptation of problem generation. We presented the protocol and results of a web-based within-subjects evaluation of the adaptation. The improvement in student learning from the pre-test to the post-test was statistically significant for both the versions of the tutor. However, there was a statistically significant difference in the number of problems solved by the two groups during practice – on the average, students using the non-adaptive tutor solved nearly twice as many problems during practice than those who used the adaptive tutor. Therefore, rule-based adaptive problem generation in web-based tutors helps students learn with fewer practice problems.

8. Acknowledgements

Partial support for this work was provided by the National Science Foundation's Course, Educational Innovation Program under grant CNS-0426021.

References

[1] Aimeur, E., Brassard, G., Dufort, H., and Gambs, S. CLARISSE: A Machine Learning Tool to Initialize Student Models. S. Cerri, G. Gouarderes, F. Paraguacu (eds.), Proc. of ITS 2002, Springer (2002). 718-728.

[2] Arroyo, I., Conejo, R., Guzman, E., & Woolf, B.P. An Adaptive Web-Based Component for Cognitive Ability Estimation., Proc. of AI-ED 2001, IOS Press (2001). 456-466.

[3] Baffes, P. and Mooney, R. J.: A Novel Application of Theory Refinement to Student Modeling. Proc. of AAAI 1996, Portland, OR, (1996) 403-408.

[4] Belmont, M.V., Guzman, E., Mandow, L., Millan, E., and Perez-de-la-Cruz, J.I. Automatic generation of problems in web-based tutors. In Virtual Environments for Teaching & Learning, L.C. Jain, R.J. Howlett, N.S. Ichalkaranje and G. Tonfoni (ed.), World Scientific, 2002.

[5] Birnbaum, M.H. (Ed.) Psychological Experiments on the Internet. San Diego, Academic Publishers, 2000. http://psych.fullerton.edu/mbirnbaum/web/IntroWeb.htm

[6] Bloom, B.S. and Krathwohl, D.R.: Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain, New York, Longmans, Green (1956).

[7] Bonar, J. and Cunningham, R.: BRIDGE: Tutoring the programming process, in Intelligent tutoring systems: Lessons learned. J. Psotka, L. Massey, S. Mutter (Eds.), Lawrence Erlbaum Associates, Hillsdale, NJ (1988).

[8] Brown, J.S. and Burton, R.R. Diagnostic models for procedural bugs in basic mathematical skills. Cognitive Science, Vol 2 (1978). 155-191.

[9] Czarkowski, M. and Kay, J. Challenges of Scrutable Adaptivity. U. Hoppe, F. Verdejo and J. Kay (eds.), Proc. of AI-ED 2003, IOS Press (2003). 404-406.

[10] Johnson, W.L. Intention-based diagnosis of novice programming errors. Morgan Kaufman, CA 1986.

[11] Kay, J.: Stereotypes, Student Models and Scrutability. Proc. of ITS 2000. G. Gauthier, C. Frasson and K. VanLehn (eds.). Springer (2000). 19-30.

[12] Kalyuga, S. Rapid Assessment of Learner's Knowledge in Adaptive Learning Environments, Proc. of AI-ED 2003, IOS Press, (2003). 167-174.

[13] Koffman, E.B. and Perry, J.M.: A Model for Generative CAI and Concept Selection. International Journal of Man Machine Studies. Vol. 8 (1976) 397-410.

[14] Kurhila, J., Lattu, M., and Pietila, A. Using Vector Space Model in Adaptive Hypermedia for Learning. Proc. of ITS 2002, Springer (2002). 129-138.

[15] Mann, P., Suiter, P., & McClung, R.: A Guide for Educating Mainstream Students. Allyn and Bacon, 1992.

[16] Martin, B. and Mitrovic, A. Tailoring Feedback by Correcting Student Answers. Proc. of ITS 2000. Springer (2000). 383-392.

[17] Melis, E., Andres, E., Budenbender, J., Frischauf, A., Goguadze, G., Libbrecht, P., Pollet, M. and Ullrich, C. ActiveMath: A Generic and Adaptive Web-Based Learning Environment. International Journal of Artificial Intelligence in Education, Vol 12 (2001). 385-407.

[18] Millan, E., Perez-de-la-Cruz, J.L., and Svazer, E. Adaptive Bayesian Networks for Multilevel Student Modeling. Proc. of ITS 2000. Springer (2000), 534-543.

[19] Reiser, B., Anderson, J. and Farrell, R.: Dynamic student modelling in an intelligent tutor for LISP programming, Proc. of IJCAI 1985. Los Altos CA (1985).

[20] Salton, G., Wong, A. and Yang, C.S. A Vector Space Model for Automatic Indexing. Communications of the ACM, Vol. 18(11), (1975). 613-620.

[21] Weber, G. and Brusilovsky, P. ELM-ART: An Adaptive Versatile System for Web-Based Instruction. International Journal of Artificial Intelligence in Education, Vol 12 (2001). 351-384.