Learning Programming by Solving Problems

Amruth N. Kumar
Ramapo College of New Jersey; 505, Ramapo Valley Road; Mahwah, NJ 07430-1680
amruth@ramapo.edu

Abstract: We have been developing tutors to help students learn programming concepts by solving problems. In this paper, we will discuss the use of problem-solving in Computer Science, the effectiveness of using problem-solving tutors to learn programming concepts, and the pedagogical relationship between solving problems and learning to write programs. We will also present the design and results from the evaluation of one of our tutors.

1. PROBLEM-SOLVING AND COMPUTER SCIENCE

Problem-based learning improves long-term retention [12], and is better than traditional instruction for improving the ability of students to solve real-life problems. In Computer Science, various researchers have advocated the use of self-paced exercises [24], practice to build problem-solving skills [35], and the use of frequent, graded assignments in a course [9]. It is reported that "students universally want to see more examples both in class and in their textbooks" [38]. Solving problems facilitates active learning, whose place in Computer Science education has been established [26].

Textbooks are generally inadequate as sources of problems because of their limited, non-interactive nature. Even in disciplines such as Physics and Mathematics, where textbooks generally tend to include many more practice

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35619-8_15 problems than in Computer Science, faculty are increasingly turning to the use of technology to address this issue. Typically, programs are written to generate problems, and such programs are made available to students for practice, e.g., CAPA [16] for Physics, and CHARLIE [4] for electronics and control systems. Kashy [17] reports that the use of such programs has increased student performance by 10%, largely due to increased time spent on the task.

Different types of problem-solving tutors have been developed for Computer Science topics that provide students with frequent, self-paced exercises:

- Numerous systems demonstrate the solving of *problems entered by the student* - JFLAP [33] for Automata Theory and PSVE/GAIGS [27] for parameter passing mechanisms are two representative examples.
- Numerous algorithm animation and visualization systems (e.g., JHAVE [28], JAWAA [32]) *generate data* to animate algorithms.
- Some systems *administer problems* generated by the instructor. These include WebToTeach [2], APMS [18], WebCT (www.webct.com), and other such course administration systems.
- A few systems have been developed to *generate problems* for students to solve: PILOT [7] for graph algorithms, Gateway Labs for problems in mathematical foundations of Computer Science [3], and SAIL [8], which is a LaTeX-based scripting tool for problem generation.

Few, if any systems have been developed to help students learn programming by solving problems, wherein the systems themselves generate problems for the students to solve. Two reasons for this may be:

- Programming problems are not quantitative (vis-à-vis say, Physics or electronics). Instead, they depend on the structure of arbitrary programs, which are hard to generate automatically.
- Computer Science educators have traditionally considered the norm for student practice to be a small number of large programming exercises in a course, rather than a large number of small practice problems. However, education research indicates that focused practice (such as solving problems) is just as important for learning as contextualized and expansive projects (such as whole language approach in reading instruction) [25,14,10]. In other words, students must solve problems about specific programming constructs just as much as write comprehensive programs to build their programming and problem-solving skills.

5. PROBLETS: PROBLEM-SOLVING TUTORS FOR COMPUTER SCIENCE

We have been developing problem-solving tutors, called problets, for selected programming concepts. These tutors are capable of generating problems, grading the student's solution, providing immediate and detailed feedback about the correct answer, logging the student's performance, and determining whether the student has learned the material.

- **Problem Generation:** The problets generate an endless supply of problems on a specific topic, by randomly instantiating problem templates encoded into them in pseudo-BNF notation by either the problet designers or the instructor using the problet.
- **Problem Solving:** The problets are capable of solving the problems they generate. The answers to the problems need not be encoded into the problets.
- **Providing feedback on User's Answers:** The problets provide feedback at two levels: at the minimal level, they correct the user's answer; at the detailed level, they also explain the correct answer.
- **Grading User's Answers:** The problets report whether the user's answer is correct, incorrect or partially correct. They keep score for the user, and are capable of terminating a session when the user has reached a preset level of proficiency in the topic.

The effectiveness of using problem-solving tutors has been well documented in literature:

- The field of *Intelligent Tutoring Systems*, which is the basis of our solution, has documented an improvement of one standard deviation through the use of tutors [1].
- The use of similar tutors has been shown to increase student performance by 10% in Physics [17].
- Our own work in building and testing tutors for Computer Science has shown that the *average* performance in a class improves by 100% after using the tutor [19], [20], and that the improvement is systemic [37].

The tutors are designed to promote active learning. They target **application** in Bloom's taxonomy of educational objectives [5], and are expected to <u>supplement</u> the traditional programming projects assigned in the course, which emphasize **synthesis**. Since research indicates that focused practice such as that provided by the tutors is just as important for learning as large-scale projects [25], so we expect the use of tutors to <u>complement</u> the traditional programming projects.

To date, we have developed, deployed and assessed tutors on several programming topics, including expression evaluation in C++ [19], pointers for indirect addressing in C++ [20], nested selection statements in C++ [36], static scope in Pascal [21,22] and parameter passing in programming languages [37]. We also plan to build a series of tutors for most of the imperative programming constructs covered in a typical *Computer Science I* course, and evaluate the effectiveness of using them to improve retention in the course. Our tutors may be used not only for practice solving problems, but also for assignments and tests. The tutors are delivered over the Web, so they can be accessed any time, anywhere.

2. FROM SOLVING PROBLEMS TO WRITING PROGRAMS

Learning to program involves both learning to design and write programs, and learning to read and understand programs. Whereas the focus of introductory Computer Science courses is in general the former, i.e., learning to design and write programs, the importance of the latter cannot be over-emphasized. Computer Science students must learn to read and understand programs because professional programmers must often cooperate with others to write programs, and may have to maintain software written by someone else.

Learning to read and understand programs may contribute to a student's ability to, in turn, design and write programs:

- **Active Effect:** Students generate mental models when reading programs, which may in turn help them visualize solutions when writing programs;
- **Passive Effect:** Students have to read their own programs in order to debug/test them. Since debugging/testing is a part of the write-compile-debug-test cycle of program development, any improvement in the ability to read and understand programs helps students write programs more efficiently.

Solving problems to learn programming could involve either writing programs or analysing given programs. Since students already write programs as part of class assignments, the focus of our tutors is on analyzing given programs. The problems generated by our tutors engage the learners in one of two analytical activities: debugging the presented program, or predicting the output of the program. This helps students learn by examples, both good and bad. These activities promote the students' ability to read and analyze programs written by them as well as others.

The program comprehension model developed by Pennington [29,30] investigated the detailed mental representations formed by programmers studying programs written in the imperative style. It was derived from models of text comprehension developed and refined over the years [15,34,39]. Pennington's model of program comprehension includes the following two layers, which are progressively more abstract:

- **Program Model**, which consists of knowledge of operations carried out in source code, and control flow - low level details that are localized and explicitly available in the program text.
- **Domain Model**, which consists of knowledge of data flow and the goals that a program accomplishes – abstract details that are distributed and implicit in a program. It is difficult to understand programs when related data transformations are carried out in non-contiguous segments of code [23].

Pennington found that novice programmers built a strong program model, but a weak domain model after studying imperative programs written in FORTRAN, COBOL and Pascal [11], whereas expert programmers built a stronger domain model than novice programmers. Pennington argues that the performance of comprehension-demanding tasks is likely to play an important role in the formation of domain model, which is built slowly in the context of meaningful programming tasks. Program debugging and prediction of output are comprehension-demanding tasks presented by our tutors. Hence, our tutors may be said to promote the development of a stronger domain model among novice programmers. In general, our tutors address both problem domain (e.g., expression evaluation, code with syntax errors) and domain model (dangling pointers, scope issues, lost objects, memory-out-of-bounds, semantic errors, etc.).

Theorists have identified three levels of learning [14,10,40], through which novice learners progress: emergent stage, when they are first exposed to the task; developing stage, when they recognize patterns and begin using appropriate tools; and transitional stage, when they carry out the tasks increasingly correctly, despite incomplete understanding and an initial lack of confidence. Our tutors can be used in all the above stages: with detailed feedback during the emergent stage, with minimal feedback during the developing stage, and without any feedback during the transitional stage.

3. EVALUATION OF PROBLETS

Our tutor on C++ pointers presents C++ programs and asks the user to indicate whether the program contains any dangling pointers, lost objects, semantic errors (printing values of un-initialized variables), syntax errors, etc. We have evaluated this tutor in several sections of our *Computer Science II* course. In this section, we will present the results of these evaluations, addressing both cognitive and affective aspects of learning with the tutor.

3.1 Cognitive Learning with the Problet

Tutor in Isolation: In Fall 2000, we tested the tutor in two sections (N=19 combined), by administering a pretest, followed by practice using the tutor, and a post-test. These were not controlled tests. The author was the instructor in both the sections. The pre-test and post-test scores were out of 40. The results are presented in the Table below.

Table 3. Tutor in Isolation

(N=19)	Pre-Test	Post-Test	Effect Size
Average	12.21	26.74	2.16
Std-Dev	6.70	8.73	

The Effect Size is calculated as (post-test score - pretest-score) / standard deviation on the pre-test. An effect size of 2.16 sigma indicates that the tutor facilitated learning among the students. The improvement is statistically significant (2-tailed $p < 0.05$). It compares favourably with the result that individual human tutors can bring students 2 sigma above normal classroom instruction [6].

Tutor Versus Printed Workbook: In Spring 2001, we again tested the tutor in two sections (N=33 combined), using the pretest-practice-posttest protocol. We conducted a controlled test – between the tests, the control group practiced with printed workbooks, whereas the test group practiced with the tutor. The author was not the instructor in the sections. The pre-test and post-test scores were out of 40. The results are presented in the Table below.

Table 4. Tutor vs Printed Workbook

(N=33)	Pre-Test	Post-Test	Effect Size
Tutor Users			
Average	13.00	23.06	1.52
Std-Dev	6.61	10.12	
Workbook Users			
Average	15.24	24.71	1.33
Std-Dev	7.10	10.54	

Practicing with the tutor appeared to be slightly better than practicing with the printed workbook. Both the improvements were statistically significant (2-tailed $p < 0.05$).

Minimal Versus Detailed Feedback in the Tutor: In Fall 2001, we conducted a controlled test of the tutor in two sections (N=22). This time, we tested two versions of feedback for the tutor: minimal versus detailed. In minimal feedback, the tutor corrects the user's answer, but does not explain the correct answer. In detailed feedback, in addition, the tutor explains the correct answer. We used the same pre-test-practice-post-test protocol as before, with fixed times for each step. Incorrect answers were penalized. The author was not the instructor in either class. The pre-test and post-test scores were out of 80. The results are presented in the Table below.

Table 5. Minimal vs detailed feedback in the tutor

	Pre-Test	Post-Test	Effect Size
Detailed Feedback (N=14)			
Average	11.57	27.29	1.99
Std-Deviation	7.91	21.01	
Minimal Feedback (N=8)			
Average	8.25	17.38	1.63
Std-Deviation	5.60	14.62	

The results seem to indicate that detailed feedback may be better than minimal feedback. For detailed feedback, the 2-tailed $p < 0.05$, indicating that the improvement is statistically significant, whereas $p = 0.28$ for minimal feedback, indicating that the improvement is not statistically significant.

3.2 Affective Learning with the Problet

Students filled out a feedback form after the controlled tests in Spring 2001, in which they provided feedback about the instrument they had used for practice between pre-test and post-test (workbook for control group and tutor for test group, N=33). These feedback forms clearly indicate that the tutor facilitates affective learning. On a Likert scale of 1 (Strongly Agree) to

5 (Strongly Disagree), the average scores of the test and control groups on the questions of the feedback form are as shown in the table below.

Table 6. Student feedback

Feedback Question (Test Group: Instrument = Tutor; Control Group: Instrument = Printed Workbook)	Tutor Users	WorkBook Users
1. It was easy to (learn to) use this instrument.	2.13	2.29
2. The problems posed by the instrument were clear.	1.94	1.94
3. The instrument listed interesting problems.	2.13	2.35
4. The problems were repetitive and boring.	3.44	3.06
5. The instrument provided useful feedback.	2.20	3.06
6. The instrument helped me learn the material.	2.31	2.88
7. Using this instrument was time-consuming.	3.88	3.12
8. The instrument should be made available to all students	1.56	2.65
9. If this instrument is made available, I will use it	1.93	2.65
10. I would like to see such instruments on other topics.	1.44	2.59

Question 1 indicates that the tutor was easy to learn if we use the control group's score as the basis, since presumably, students do not need to "learn" how to use a printed workbook designed like a typical textbook. The problems in the printed workbook were themselves generated by the tutor, and the results for Question 2 validate this. Questions 3 and 4 seem to indicate a slight Hawthorne effect in that students using the online tutor felt the problems were more interesting and less repetitive and boring, although the types of problems were the same for both the tutor and the printed workbook. Question 5 clearly indicates the superiority of the tutor, which provided detailed problem-specific feedback whereas the printed workbook just listed the correct answer for each problem. Questions 6 and 7 indicate that the tutor facilitated better affective learning than the printed workbook, which is encouraging. Questions 7 through 10 clearly indicate the students' preference for the tutor over the traditional printed workbook.

4. FUTURE WORK

It is clear from the improvement from pre-test to post-test scores, that students learn how to solve problems using our tutors. We would like to test whether this improvement in problem-solving ability translates to better ability to write programs.

Pennington [29] found that a cross-referenced mental representation, containing a balanced mix of program and domain model is associated with better program comprehension. She also found that modification tasks promoted the development of a cross-referenced mental representation. Our tutors currently do not ask the users to modify the programs, only to debug or predict their output. We may include program modification as another activity in our tutors in the future.

Self-generated elaborations are better than text-supplied elaborations for learning [31]. In other words, if the user is provided with an environment in which the user can construct his/her own explanation for a program, the user will benefit more than if the tutor generates all the explanations. It would be an interesting exercise to incorporate this meta-cognitive reasoning into our tutors. Since the problem (code segment), solution and feedback are all textual in our problets, they favor verbal learners over visual learners in the Felder-Silverman Learning Style model [13]. We would like to address the needs of visual learners by incorporating program animation into our problets.

5. ACKNOWLEDGMENT

Partial support for this work was provided by the National Science Foundation's Course, Curriculum and Laboratory Improvement Program under grant DUE-0088864.

6. REFERENCES

[1] Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R. "Cognitive Tutors: Lessons Learned". *The Journal of the Learning Sciences.* Vol 4(2), 167-207, 1995.

[2] Arnow D. and Barshay, O., WebToTeach: An Interactive Focused Programming Exercise System, In proceedings of FIE 1999, San Juan, Puerto Rico (Nov. 1999), Session 12a9.

[3] Baldwin, D., Three years experience with Gateway Labs, Proceedings of ITiCSE 96, Barcelona, Spain, June 1996, 6-7.

[4] Barker, D.S., CHARLIE: A Computer-Managed Homework, Assignment and Response, Learning and Instruction Environment, *Proc. of FIE 1997*, Pittsburgh, PA (Nov. 1997).

[5] Bloom, B.S. and Krathwohl, D.R. Taxonomy of Educational Objectives: The Classification of Educational Goals, by a committee of college and university examiners. Handbook I: Cognitive Domain, NewYork, Longmans, Green, 1956.

[6] Bloom, B.S.: The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring. *Educational Researcher*, Vol 13 (1984) 3-16.

[7] Bridgeman, S., Goodrich, M.T., Kobourov, S.G., and Tamassia, R., PILOT: An Interactive Tool for Learning and Grading, *Proceedings of the 31st SIGCSE Technical Symposium*, Austin, TX, (March 2000), 139-143.

[8] Bridgeman, S., Goodrich, M.T., Kobourov, S.G., and Tamassia, R., SAIL: A System for Generating, Archiving, and Retrieving Specialized Assignments Using LaTeX, *Proc. of the 31st SIGCSE Technical Symposium,* Austin, TX, (March 2000), 300-304.

[9] Campbell, J.O., Evaluating Costs and Benefits of Distributed Learning, Proceedings of FIE 1997, Pittsburgh, PA (November 1997).

[10] Calkins, L., The Art of teaching writing, Heinemann, 1986.

[11] Corritore, C.L. and Widenbeck, S. What do Novices Learn During Program Comprehension? *Intl. Journal of Human-Computer Interaction*, 1991, 3(2), 199-222.

[12] Farnsworth, C. C., Using computer simulations in problem-based learning. In *Proc. of Thirty Fifth ADCIS conference*, Omni Press, Nashville, TN, (1994), 137-140.

[13] Felder, R., Reaching the Second Tier: Learning and Teaching Styles in College Science Education. *Journal of College Science Teaching*. 23(5): 286-190, 1993.

[14] Holdaway, D., The Foundations of Literacy, Heinemann, 1980.

[15] Johnson-Laird, P.N. Mental Models: Towards Cognitive Science of Language, Inference and Consciousness. Cambridge University Press, Cambridge, 1983.

[16] Kashy, E., Sherrill, B.M., Tsai, Y.., Thaler, D., Weinshank, D., Engelmann, M., and Morrissey, D.J., CAPA, An Integrated Computer Assisted Personalized Assignment System, American Journal of Physics, Vol 61(12), 1993, 1124-1130.

[17] Kashy E., Thoennessen, M., Tsai, Y., Davis, N.E., and Wolfe, S.L. Using Networked Tools to Enhance Student Success Rates in Large Classes. In *Proceedings of FIE '97* (Pittsburgh, PA, November 1997), IEEE Press, Session T3A.

[18] Kohne, G.S., An Autograding (Student) Problem Management System for the Compeuwtir Illittur8, Proceedings of ASEE Annual Conference, June 1996 (CD ROM).

[19] Krishna A. and Kumar A. A Problem Generator to Learn Expression Evaluation in CS I and Its Effectiveness. *Journal of Computing in Small Colleges*, Vol 16(4), 2001, 34-43.

[20] Kumar A. Learning the Interaction between Pointers and Scope in C++, *Proceedings of The Sixth Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2001)*, Canterbury, UK, (June 2001), 45-48.

[21] Kumar A.: Dynamically Generating Problems on Static Scope, Proceedings of The Fifth Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2000), Helsinki, Finland, (July 2000), 9-12.

[22] Kumar, A, Schottenfeld, O. and Obringer, S.R. Problem Based Learning of 'Static Referencing Environment in Pascal, *Proc. of the 16th Annual Eastern Small College Computing Conference (ESCCC 2000)*, University of Scranton, PA, (Oct 2000), 97-102.

[23] Littman, D.C., Pinto, J., Letovsky, S., and Soloway, E. Mental Models and Software Maintenance. In E. Soloway and S. Iyengar (Eds.), *Empirical Studies of Programmers*, 1986, Ablex Publishers, Norwood, NJ, 80-98.

[24] Liu, M.L., and Blanc, L., On the retention of female Computer Science students, *Proc. of the 27th SIGCSE Technical Symposium*, Philadelphia, PA, March 1996, 32-36.

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

[26] McConnell, J., Active Learning and its use in Computer Science, Proceedings of ITiCSE 96, Barcelona, Spain, June 1996, 52-54.

[27] Naps, T.L., and Stenglein, J., Tools for Visual Exploration of Scope and Parameter Passing in a Programming Languages Course, The Proceedings of 27th SIGCSE Technical Symposium on Computer Science Education, February 1996, 305- 309.

[28] Naps, T.L., Eagan, J.R.. and Norton, L.L. JHAVE – An Environment to Actively Enhage Students in Web-Based Algorithm Visualizations. *Proceedings of the 31st SIGCSE Technical Symposium*, Austin, TX, March 2000, 109-113.

[29] Pennington, N. Comprehension Strategies in Programming. G.M. Olson, S. Sheppard and E. Soloway (Eds.), *Empirical Studies of Programmers: Second Workshop*, Ablex Publishers, Norwood, NJ, 100-113, 1987.

[30] Pennington, N. Stimulus Structures and mental Representations in Expert Comprehension of Computer Programs. *Cognitive Psychology*, 19, 295-341, 1987.

[31] Reder, L., Charney, D., and Morgan, K. The Role of Elaborations in Learning a Skill from Instructional Text. *Memory and Cognition*. 14: 64-78, 1986.

[32] Rodger, S., JAWAA, 1997, http://www.cs.duke.edu/~rodger/tools/tools.html

[33] Rodger, S., and Gramond, E., JFLAP: An Aid to Study Theorems in Automata Theory, Proceedings of ITiCSE 98, Dublin, Ireland, August 1998, 302.

[34] Schmalhofer, F. and Glavonov, D. Three Components of Understanding a Programmer's Manual: Verbatim, Propositional and Situtational Representations. *Journal of Memory and Language*, 1986, 25, 295-313.

[35]`Schollmeyer, M., Computer Programming in Highschool versus College, Proceedings of the 26th SIGCSE Technical Symposium, Philadelphia, PA, February 1996, 378-382.

[36] Singhal N., and Kumar A., "Facilitating Problem-Solving on Nested Selection Statements in C/C++", *Proc. of FIE '00*, Kansas City, MO, October 2000, IEEE Press.

[37] Shah, H. and Kumar, A., "A Tutoring System for Parameter Passing in Programming Languages", *Proc. of The 7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2002)*, Aarhus, Denmark, (June 2002), 170-174.

[38] Tilbury, D., and Messner, W., Development and Integration of Web-based Software Tutorials for an Undergraduate Curriculum: Control Tutorials for MATLAB, Proceedings of FIE 97, Pittsburgh, PA, November 1997.

[39] Van Dijk, T.A. and Kintsch, W. Strategies of Discourse Comprehension. Academic Publishers, New York, 1983.

[40] Vygotsky, L., Mind in Society: Development of Higher Psychological Functions, Harvard University Press, 1978.