Generation of Demand Feedback in Intelligent Tutors for Programming

Amruth N. Kumar

Ramapo College of New Jersey, Mahwah NJ 07430, USA
amruth@ramapo.edu

Abstract. In this paper, we will examine how a model-based tutor can automatically generate demand feedback. We will propose a two-stage feedback generation algorithm that maintains the principle of modularity characteristic of model-based representation, while automatically generating detailed demand feedback. We will evaluate model-based programming tutors to demonstrate that the feedback generated using this algorithm is coherent and effective at improving learning.

1 Introduction

We have been developing intelligent tutors to help students learn specific constructs in programming languages, such as loops, pointers, and expression evaluation. The tutors present a code segment involving a construct, ask the learner to solve problems based on the code, and provide demand feedback to facilitate learning. The tutors generate the problems as parameterized instances of problem templates (as in [6]).

Various technologies have been used to model the domain in an intelligent tutor, including rule-based reasoning (e.g., production rules used in ACT-R theory [1]), constraint-based reasoning (e.g., [10]), and case-based reasoning (e.g., [13]). Earlier, we had proposed using model-based reasoning for programming tutors [7] because of its relative advantages over the other technologies [5]. In this paper, we will briefly examine how demand feedback can be generated in a tutor that uses model-based reasoning for domain modeling.

One of the advantages of using model-based reasoning for modeling the domain in an intelligent tutor is that the domain model doubles as the runnable expert module [7]. Therefore, the tutor is capable of solving problems on its own. Can the tutor also **automatically generate feedback** on its own?

The representation used in model-based reasoning is modular - each component is responsible for only its behavior. A device derives its behavior by composing the behaviors of its components. Such a modular design promotes scalability. Can such a modular scheme be used to generate feedback? If so, would the resulting feedback be **coherent**? Would it be **effective** enough for the user to learn from? In this paper, we will briefly examine these questions.

2 Feedback Generation

Currently, our tutors provide demand feedback [2], i.e., they provide feedback only when the learner demands. Research shows that students who practice with demand feedback are significantly better at far-transfer tasks than those who receive immediate feedback [4][9]. Therefore, demand feedback is appropriate for our tutors on programming.

Our tutors generate demand feedback using simulation and reflection. Reflection is the process of providing explanation by reflecting on the state and the knowledge of the tutor [11]. During simulation, each tutor executes the model of the program. By way of reflection, it examines the state of the components, and generates explanation based on its observations. The resulting feedback can not only explain the behavior of the program, but also justify the correct answer.

We use a two-stage algorithm to generate feedback about a program, by composing the feedback generated by its components:

- **Process Explanation:** The interpreter that executes the code generates this explanation in a fashion similar to program tracing. For each line of code, the interpreter identifies the components participating in the line of code, and any state transitions that they undergo as a result of executing the line of code. This includes identifying any side-effect that results from executing the line, i.e., any input, output, or change in the values of variables. Since the lines of code are executed in the order specified by the program, the resulting feedback narrative is coherent.
- **Component Explanation:** This explanation is generated by the components participating in the line of code that is currently being executed. If an action is applied to a component that is not supported by the current state of the component, the component explains why the attempt to apply the action is an error. Furthermore, the component steps through and generates explanation for each of the previous states of the component. It presents the generated lines of explanation in one of two forms:
  - Detailed form: It explains all the prior states in order, and how they culminate in an error when the current action is applied.
  - Abbreviated form: It explains only the current state, and how applying the current action results in an error.

Component explanation is coordinated with process explanation as follows: during the generation of process explanation, at each line of code, component explanation is obtained from all the components participating in that line of code. Only the explanations from the components that are in an error state are inserted into the process explanation. The abbreviated form of component explanation is used during the generation of demand feedback. The algorithm may be summarized as follows:

```
Execute the code line by line:
For each line of code
   Generate process explanation
```

```
For each component involved in the line
   If the component reached an error state on this line
      Generate abbreviated component explanation for the component
      Insert it after the process explanation for the line
```

The tutor post-processes the feedback generated by process and component explanations to address the needs of the user:

- Detailed Feedback: Since the feedback includes a complete explanation of the behavior of the program, it is used for novices, for the first few problems in a tutoring session, and in instructional (as opposed to problem-solving) mode.
- Diagnostic Feedback: The feedback is post-processed to include only the following:
  - Process explanation generated by input and output statements;
  - The abbreviated version of component explanation generated by components with observed errors.

  Diagnostic feedback is used once the student starts making progress towards the educational objectives of the tutor.

The algorithm can be applied to any domain wherein the composition of behavior reflects the composition of structure, i.e., those and only those components that constitute a device contribute to the behavior of the device.

2.1 Example

The following is a typical program that the learner is asked to debug when using a tutor on pointers:

```
void main()
{
   long *indirectPointer;
   {
      long count = 1313;
      indirectPointer = &count;
   }
   cout << *indirectPointer;
}
```

The demand feedback generated by the tutor for the above code is:

```
When the program is executed:
The program always starts executing from main().
When the function main() is executed:
Pointer indirectPointer is declared on line 4.
But, it has not been assigned a value yet
When the nested block starting at line 5 is executed:
```

```
Variable count is declared on line 6.
It is initialized during declaration to 1313
Pointer indirectPointer is assigned to point to count on line 7
The nested block is exited on line 8
Variable count goes out of scope and is deallocated on line 8
An attempt is made to print the value of the variable pointed to by
indirectPointer on line 9
```
But, variable count has already gone out of the scope of its declaration.
Therefore, indirectPointer is a dangling pointer.
```
The function main() is exited on line 10
Pointer indirectPointer goes out of scope and is deallocated on line
10
```

All except the italicized lines of feedback are generated during process explanation. The italicized lines are an abbreviated version of component explanation. Note that it is important for component explanation to include the complete context so that its feedback can be understood independent of any other feedback generated by process explanation.

3 Evaluation of the Tutor

We have evaluated several of our tutors to check if demand feedback generated automatically by the tutor is effective enough to promote learning. In each case, we used a pre-test, practice, post-test protocol. The pretest and post-test were of comparable hardness, with similar problems, listed in the same order. We used controlled tests - during the practice, the control group was provided minimal feedback and the experimental group was provided detailed demand feedback. Minimal feedback listed whether the learner's answer was correct or wrong, but not why. Detailed demand feedback also explained why, and was automatically generated by the tutor.

Where possible, in order to eliminate practice effect, we considered student score per attempted question or percentage of attempted questions correctly answered, instead of raw scores. We calculated effect size as (post-test score - pretest-score) / standard-deviation on the pre-test. Note that an effect size of two is the holy grail for tutors - one-on-one human tutoring is known to improve learning by two standard deviations over traditional classroom instruction [3]. We used 2-tailed p-value to verify the statistical significance of our results.

When we evaluated our tutor on expression evaluation tutor in Fall 2002, we found that the effect size was 1.35 for the test group (N=24) that received detailed feedback versus 0.84 for the control group (N=24) that received minimal feedback. Similarly, when we evaluated our tutor on parameter passing in Spring and Fall 2002, we found that the effect size was 0.99 for the test group (N=14) that received detailed feedback versus 0.07 for the control group (N=15) that received minimal feedback. These and other similar results (e.g., [8]) indicate that the demand feedback automatically generated by our tutors using the two-stage algorithm described in Section 2 is coherent and effective in promoting

learning. We plan to continue to evaluate our tutors for the quality and quantity of feedback they provide.

4 Acknowledgments

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

References

1. Anderson, J.R.: Production Systems and the ACT-R Theory. In Rules of the Mind. Hillsdale, NJ: Lawrence Erlbaum & Associates, Inc. (1993) 1–10.
2. Anderson J.R., Corbett A.T., Koedinger K.R. and Pelletier R.: Cognitive Tutors: Lessons Learned. The Journal of the Learning Sciences, Lawrence Erlbaum Associates, Inc. Vol No 4(2) (1995) 167–207.
3. 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.
4. Corbett, A.T. and Anderson, J.R.: Locus of feedback control in computer-based tutoring: impact on learning rate, achievement and attitudes. Proceedings of CHI 2001. ACM 2001. (2001) 245–252.
5. Davis, R.: Diagnostic Reasoning Based on Structure and Behavior. Artificial Intelligence, 24 (1984) 347–410.
6. Koffman, E.B. and Perry, J.M.: A Model for Generative CAI and Concept Selection. International Journal of Man Machine Studies. 8 (1976) 397-410.
7. Kumar, A.N.: Model-Based Reasoning for Domain Modeling in a Web-Based Intelligent Tutoring System to Help Students Learn to Debug C++ Programs. Proceedings of Intelligent Tutoring Systems (ITS 2002), LNCS 2363, Biarritz, France, June 5-8, 2002, 792–801.
8. Kumar, A.N.: Learning Programming by Solving Problems, Informatics Curricula and Teaching Methods, L. Cassel and R.A. Reis ed., Kluwer Academic Publishers, Norwell, MA, 2003, 29–39.
9. Lee, A.Y.: Using tutoring systems to study learning. Behavior Research Methods, Instruments and Computers, 24(2), (1992) 205–212.
10. Martin, B. and Mitrovic, A.: Tailoring Feedback by Correcting Student Answers. Proceedings of Intelligent Tutoring Systems (ITS) 2000. G. Gauthier, C. Frasson and K. VanLehn (eds.). Springer (2000) 383-392.
11. Neuper, W.A.: A 'calculemus-approach' to high-school math? In S. Linto and R. Sebastiani (eds.), Proceedings of the 9th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, Siena, Italy, (June 2001).
12. Reiser, B., Anderson, J. and Farrell, R.: Dynamic student modeling in an intelligent tutor for LISP programming, in Proceedings of the Ninth International Joint Conference on Artificial Intelligence, A. Joshi (Ed.), Los Altos CA (1985).
13. Reyes, R.L. and Sison, R.: A Case-Based Reasoning Approach to an Internet Agent-Based Tutoring System. In: Moore, J.D. Redfield, C.L. and Johnson, W.L. (eds.): Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future, IOS Press, Amsterdam (2001) 122–129.