Implicit Versus Explicit Learning of Strategies in a Non-procedural Cognitive Skill

Kurt VanLehn¹, Dumiszewe Bhembe¹, Min Chi¹, Collin Lynch¹, Kay Schulze², Robert Shelby³, Linwood Taylor¹, Don Treacy³, Anders Weinstein¹, and Mary Wintersgill³

¹ Learning Research & Development Center, University of Pittsburgh, Pittsburgh, PA, USA
{VanLehn, Bhembe, mic31, collinl, lht3, andersw}@pitt.edu
² Computer Science Dept., US Naval Academy, Annapolis, MD, USA
schulze@artic.nadn.navy.mil
³ Physics Department, US Naval Academy, Annapolis, MD, USA
{treacy, mwinter}@artic.nadn.navy.mil

Abstract. University physics is typical of many cognitive skills in that there is no standard procedure for solving problems, and yet a few students still master the skill. This suggests that their learning of problem solving strategies is implicit, and that an effective tutoring system need not teach problem solving strategies as explicitly as model-tracing tutors do. In order to compare implicit vs. explicit learning of problem solving strategies, we developed two physics tutoring systems, Andes and Pyrenees. Pyrenees is a model-tracing tutor that teaches a problem solving strategy explicitly, whereas Andes uses a novel pedagogy, developed over many years of use in the field, that provides virtually no explicit strategic instruction. Preliminary results from an experiment comparing the two systems are reported.

1 The Research Problem

This paper compares methods for tutoring non-procedural cognitive skills. A cognitive skill is a task domain where solving a problem requires taking many actions, but the challenge is not in the physical demands of the actions, which are quite simple ones such as drawing or typing, but in deciding which actions to take. If the skill is such that at any given moment, the set of acceptable actions is fairly small, then it is called a procedural cognitive skill. Otherwise, let us call it a non-procedural cognitive skill. For instance, programming a VCR is a procedural cognitive skill, whereas developing a Java program is a non-procedural skill because the acceptable actions at most points include editing code, executing it, turning tracing on and off, reading the manual, inventing some test cases and so forth. Roughly speaking, the sequence of actions matters for procedural skills, but for non-procedural skills, only the final state matters. However, skills exists at all points along the continuum between procedural and non-procedure. Moreover, even in highly non-procedural skills, some sequences
of actions may be unacceptable, such as compiling an error-free Java program twice in a row without changing the code or the compiler settings.

Tutoring systems for procedural cognitive skills can be quite simple. At every point in time, because there are only a few actions that students should take, the tutor can give positive feedback when the student’s action matches an acceptable one, and negative feedback otherwise. When the student gets stuck, the tutor can pick an acceptable next action and hint it. Of course, in order to give feedback and hints, the tutor must be able to calculate at any point the set of acceptable next actions. This calculation is often called “the ideal student model,” the “expert model.” Such tutors are often called model tracing tutors.

It is much harder to build a tutoring system for non-procedural cognitive skills. Several techniques have been explored. The next few paragraphs review three of them.

One approach to tutoring a non-procedural skill is to teach a specific problem-solving procedure, method or strategy. The strategy may be well-known but not ordinarily taught, or the strategy may be one that has been invented for this purpose. For instance, the CMU Lisp tutor (Corbett & Bhatnagar, 1997) teaches a specific strategy for programming Lisp functions that consists of first inferring an algorithm from examples, then translating this algorithm into Lisp code working top-down and left-to-right. The basic idea of this approach is to convert a non-procedural cognitive skill into a procedural one. This allows one to use a model tracing tutor. Several model tracing tutors have been developed for non-procedural cognitive skills (e.g., Reiser, Kimber, Lovett, & Ranney, 1992; Scheines & Sieg, 1994).

A second approach is to simply ignore the students’ actions and look only at the product of those actions. Such tutoring systems act like a grader in a course, who can only examine the work submitted by a student, and has no access to the actions taken while creating it. Such tutors are usually driven by a knowledge base of condition-advice pairs. If the condition is true of the product, then the advice is relevant. Recent examples include tutors that critique a database query (Mitrovic & Ohlsson, 1999) or a qualitative physics essay (Graesser, VanLehn, Rose, Jordan, & Harter, 2001). Let us call this approach product critiquing.

Instead of critiquing the product, a tutoring system can critique the process even if it doesn’t understand the process completely. Like product critiquing tutors, such a tutor has a knowledge base of condition-advice pairs. However, the conditions are applied as the student solves the problem. In particular, after each student action, the conditions are matched against the student’s action and the state that preceded it. For instance, in the first tutoring system to use this technique (Burton & Brown, 1982), students played a board game. If they made a move that was significantly worse than the best available move, the tutor would consider giving some advice about the best available move. Let us call this approach process critiquing.

The distinctions between a process critiquing tutor and a model tracing tutor are both technical and pedagogical. The technical distinction is that a model tracing tutor has rules that recognize correct actions, whereas the process critiquing tutor has rules that recognize incorrect actions. Depending on the task domain, it may be much easier to author one kind of rule than the other. The pedagogical distinction is that model