November 2008

The Predictive Validity of Student Modeling in the ACT Programming Tutor

Albert T. Corbett
Carnegie Mellon University

John R. Anderson
Carnegie Mellon University

Alison T. O'Brien
Carnegie Mellon University

The Predictive Validity of Student Modeling in the ACT Programming Tutor

ALBERT T. CORBETT, JOHN R. ANDERSON, AND ALISON T. O'BRIEN
Department of Psychology
Carnegie Mellon University, Pittsburgh, PA, 15213, USA

Abstract: The ACT Programming Tutor is a practice environment that provides assistance to students as they write short computer programs. The tutor is constructed around a set of several hundred programming rules called the *ideal model* that allows the program to solve the exercises along with the student and provide immediate feedback on student actions. This paper evaluates the tutor's student modeling procedure, called *knowledge tracing*. As the student works, the tutor maintains an estimate of the probability that the student has learned each rule in the ideal model. This paper examines whether this assessment of the student's knowledge state predicts posttest performance. Given the set of learning probabilities computed for each student in knowledge tracing, we can try to predict students' accuracy in completing the posttest exercises. Students' actual posttest performance was well fit by these predictions at the level of both complete exercises and individual goals.

This paper describes an assessment of student modeling in the ACT Programming Tutor. One of the promises of artificial intelligence in education is increased learning rates. By modeling students' knowledge state we should be able to tailor feedback and task sequencing to the individual student, enabling the student to achieve a higher level of understanding from a given time investment. The ACT Programming Tutor maintains a student model and uses this information to guide exercise sequencing. The tutor attempts to provide an appropriate number and type of practice exercises to help each student achieve mastery of the curriculum as quickly as possible.

In this paper we examine the validity of the tutor's student modeling process in predicting performance outside the tutoring environment. The tutor provides immediate feedback on each student action and requires immediate correction of errors, so the student always remains on some correct solution path. The question is whether the student model, derived from performance in this constrained environment, predicts students' programming performance when they are on their own in a no-feedback posttest environment analogous to real-world programming.

The ACT Programming Tutor

The ACT Programming Tutor is a practice environment for students learning to program in Lisp, Prolog or Pascal (Anderson, Corbett, Fincham, Hoffman, & Pelletier, 1992). The Lisp and Prolog modules are presently employed to teach a self-paced introductory programming course at Carnegie Mellon University. The Pascal module is in use in an introductory programming course in a Pittsburgh high school. The tutor presents exercises that require students to write short programs. Figure 1 depicts the computer screen at the beginning of the first Lisp exercise. The problem description appears in the window at the upper left and the student enters the program in the code window immediately below. The coding interface is similar to a structure editor. In the figure the student has just entered the Lisp operator *car*. The symbol *<EXPR0>* is not Lisp code; rather it is a syntactic symbol which represents the argument to *car*. The student will replace this goal reminder with a Lisp expression, in this exercise the literal list *'(c d e)*. Error feedback and help messages appear in the window at the lower left. As mentioned above, the tutor provides immediate feedback on a symbol-by-symbol basis and requires immediate error correction. Thus, the student always remains on a recognized solution path.

The skill meter in the upper right corner of Figure 1 represents the focus of this study. This window displays the tutor's model of the student's knowledge state. As described immediately below, this model consists of a set of rules for writing programs. As the student works, the tutor maintains an estimate of the probability that the student has learned each rule in the set, in a process we call *knowledge tracing*. Each rule that has been introduced in the curriculum is represented by a bar graph in the skill meter. The bar graph depicts the probability that the student has learned the rule. In the figure, this probability is .50 for three of the rules, but has risen to .80 for the rule the student has just exercised. In this paper we describe an experiment that evaluates this student modeling process.

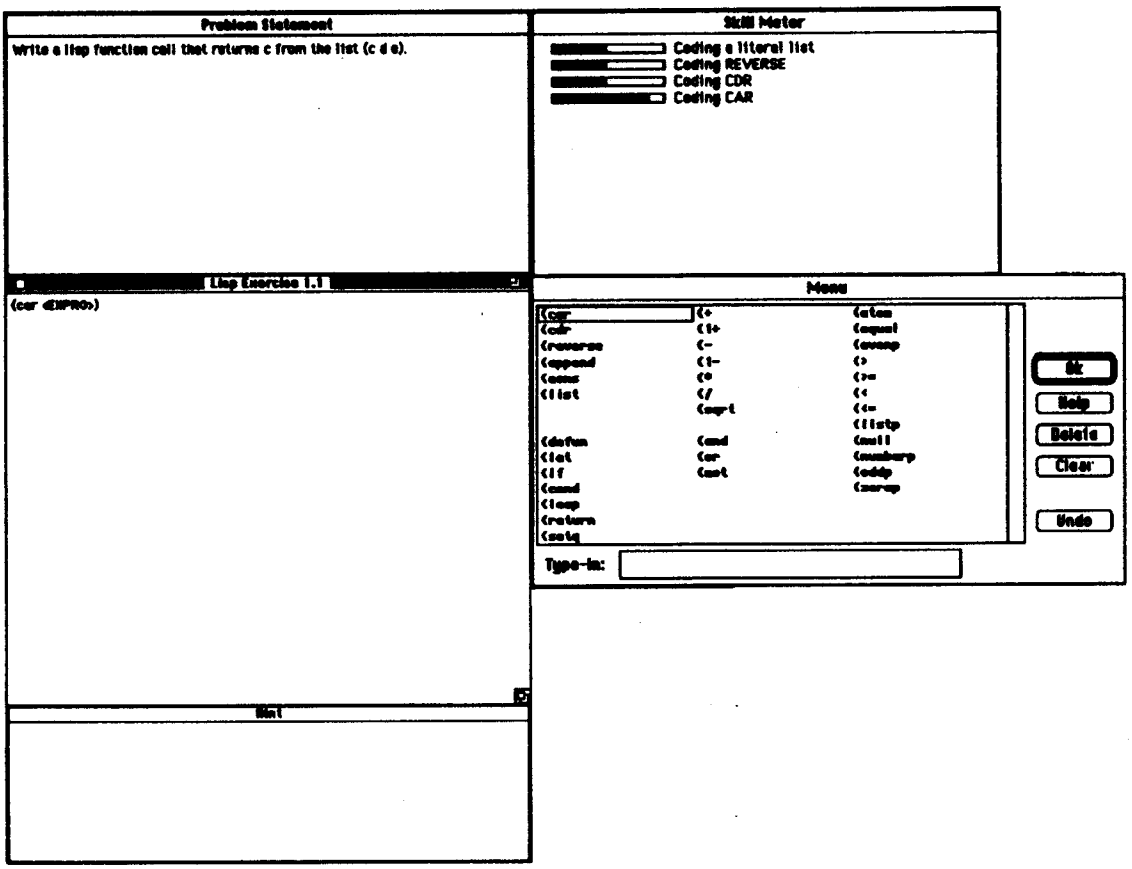

Figure 1. The APT Programming Tutor interface at the beginning of a Lisp exercise.

The Cognitive Model

A central assumption underlying the tutor is that a cognitive skill such as programming can be modeled as a set of independent production rules (Anderson, Conrad, & Corbett, 1989). The Lisp, Pascal and Prolog modules in the programming tutor are each constructed around a set of several hundred language-specific rules for writing programs called the *ideal student model*. These rules essentially model performance at the grainsize of individual code symbols. For example, the following two rules (in English form) are introduced in the first Lisp lesson:

IF the goal is to extract the first element of a list,
THEN code a function call to *car*
and set a goal to code the list.

IF the goal is to extract the nth element of a list,
THEN code a function call to *car*
and set a goal to move the element to the front of the list.

The tutor attempts to match the student's action at each step in solving an exercise to an applicable rule in the ideal student model in a process we call *model tracing*. If a match is found, the assumption is made that the student has applied an analogous cognitive rule and the tutor's internal representation of the problem state is updated accordingly. If not, the tutor notifies the student of the error.

The Learning Model

The ideal model also serves as an overlay model of the individual student's knowledge state (Goldstein, 1982). In addition to a cognitive model, knowledge tracing requires learning and performance assumptions. For this purpose the tutor assumes a simple two-state learning model. Each coding rule is either in the learned state or the unlearned state. In knowledge tracing, the tutor maintains a probability estimate that the student has learned each rule in the ideal model. At each opportunity to apply a rule, the probability that the student knows the rule is updated, contingent on the accuracy of the student's action. The Bayesian computational procedure is a variation of one described by Atkinson (1972) that employs two learning parameters and two performance parameters, displayed in Table 1. The value of each parameter is allowed to vary across production rules in the model and is estimated empirically from earlier tutor data. See Corbett and Anderson (1992b) for complete details on the model and computations.

Table 1
The learning and performance parameters employed in knowledge tracing.

pL_0	The probability a rule is in the learned state prior to the first opportunity to apply the rule (i.e., from reading the text).
pT	The probability a rule will make the transition from the unlearned to the learned state following an opportunity to apply the rule
pG	The probability a student will guess correctly if the applicable rule is in the unlearned state
pS	The probability the student will slip and make an error when the applicable rule is in the learned state

Mastery Learning

The tutor uses the knowledge tracing mechanism in an attempt to implement mastery learning. Each lesson in the tutor is divided into sections in which a small set of programming rules is introduced. Students complete a set of required exercises in each section that cover the rules being introduced, then continue working on remedial exercises in the section until the learning probability of every rule in the section has reached a criterion value, 0.95.

Student Modeling Validity

The remediation mechanism passed a minimal validity test when first introduced. Posttest scores were higher when knowledge tracing was in operation (Anderson, Conrad & Corbett, 1989). Recently we have begun examining the validity of knowledge tracing in more detail. While the goal of knowledge tracing is to infer a student's knowledge state, the assumptions of the model can also be used to predict student accuracy at each goal (step) in completing the tutor exercises. When a set of best fitting parameter estimates is used, the correlation of actual and expected error rates across the goals in a tutor lesson is quite high, on the order of 0.85 - 0.90 (Corbett & Anderson, 1992a, 1992b).

The tutor's knowledge tracing mechanism has also been shown to have some posttest predictive validity, although we have been limited to a fairly coarse analysis previously. For each student we have computed the

mean of the final learning probabilities (the probabilities that the student has learned the rules) and correlated this mean with posttest score across subjects. This correlation was reliable in one study, r = 0.47 (Corbett & Anderson, 1992b). However, if students practice with the tutor until reaching the mastery criterion, the final learning probabilities are tightly distributed between 0.95 and 1.0 and there is little potential for a predictive relationship.

The purpose of this study is to perform a more rigorous assessment. The analyses to date have been derived from course data. While we could examine the correlation of learning probabilities and posttest scores as described above, we were not able to predict absolute performance in the course's posttest environment for a variety of reasons. In the current study, students take posttests with a structure editor that is virtually identical to the tutor, except that no help is provided. This editor has three useful qualities: (1) symbols (atoms) are the basic unit of code entry, as in the tutor, (2) the editor ensures that the student's code is syntactically well-formed and (3) the editor generates a complete trace of the students' actions.

This posttest environment should enable us to make absolute performance predictions at the level of whole exercises and at the level of individual goals in completing the posttest. An analysis at the level of individual posttest goals is of interest for two reasons. First, it represents a fine-grained analysis of the predictive validity of student modeling in the tutor. Second, it affords us an opportunity to begin applying the knowledge tracing model to environments in which students are not channeled onto a correct solution path through immediate feedback and error correction. We have implemented a number of non-immediate feedback tutors in the past that have a variety of attractive features (Corbett & Anderson, 1989, 1990, 1991). While the analysis becomes more complicated, it should be possible to generalize knowledge tracing to such environments.

The Design of the Study

Students in this experiment worked through the early sections of the ACT Programming Tutor Lisp curriculum and completed three posttests. The study is designed to assess the posttest predictive validity of the tutor's knowledge tracing procedure.

Subjects

Twenty college students were recruited for pay. The students' mean Quantitative and Verbal SAT scores were 615 and 568 respectively. These students had taken an average of 1.8 programming courses previously in high school and/or college, but were not familiar with Lisp.

The Curriculum

The curriculum in this study introduces two data structures, *atoms* (symbols) and *lists* (ordered hierarchical groupings of atoms), *function calls* (operations) and *function definitions*. The curriculum is divided into six sections. The first section introduces three extractor functions, *car*, *cdr* and *reverse*, that return components of or a transformation of a list. The second and third sections introduce three constructor functions, *append*, *cons* and *list* that build new lists. In the fourth section extractor algorithms are introduced - nested functions calls that apply successive extractor functions to extract components of lists. In the fifth section students learn to define new functions that employ these extractor algorithms. In the sixth section students define functions that employ both constructor and extractor functions. These six sections contain a total of sixty-four required exercises. Each of these exercises involves anywhere from two to fourteen coding steps for a total of 345 programming steps or goals, each of which corresponds to a cognitive rule firing. The cognitive model for this portion of the curriculum contains sixty-eight rules.

Procedure

Students worked through the curriculum at their own pace. In each section students read text describing Lisp, then completed a set of required exercises that cover the programming rules being introduced. In the first five sections, students then completed remedial exercises as needed to bring all production rules introduced in the section to a mastery criterion (minimum learning probability of 0.95). In the final section, students completed a fixed set of exercises with no remediation. This fixed set contained twice the minimum number of exercises that is required to bring all rules to criterion if no errors are made.

Following the first, fourth and sixth sections students completed a posttest. The posttests contained six, twelve and fifteen programming exercises respectively. These exercises are similar to the ones completed with the tutor. The quiz interface was a structure editor, identical to the tutor interface, except that (1) students can freely edit their code and (2) no help is available.

Results

Students completed an average of 18 remedial exercises in addition to the 64 required exercises. The number of remedial exercises ranged from 3 to 63. The posttest results of one student were lost because of technical difficulties.

Internal Validity

We can use the knowledge tracing model to predict accuracy at each goal (step) as students work with the tutor. Students complete 345 coding goals across the sixty-four required tutor exercises. The probability that student *s* will respond correctly at goal *i* is given by the following performance equation:

$$pC_{is} = pL_{rs} * (1 - pS_r) + (1 - pL_{rs}) * pG_r$$

That is, the probability that student *s* will successfully apply an appropriate coding rule *r* at step *i* across the exercises is the sum of two products: (1) the probability that rule *r* is currently in the learned state for student *s* times the probability of a correct response if the rule is in the learned state, and (2) the probability that rule *r* is not currently in the learned state for student *s* times the probability of a correct guess if the rule is not in the learned state.

To assess the internal validity of the knowledge tracing process we examined the tutor protocol files and traced each student's performance goal-by-goal through the curriculum. At each of the 345 goals in the required exercises, we first predicted the probability of a correct response given the applicable rule, then applied the the knowledge tracing procedure to update the learning probability for the applicable rule. We did not fit (predict accuracy for) the remedial exercises, however we did update learning probabilities at these goals, just as the tutor would.

The learning and performance parameter estimates built into the tutor, yielded a strong correlation of actual and expected accuracy rates across the 345 goals, $r = 0.75$, $p < .0001$. The mean squared error in the fit was 0.01. In an effort to maximize the internal validity of the model, we employed a curve fitting program to generate the best fitting parameter estimates for each rule. As before, we fit the goals in the required exercises and traced both the required and remedial exercises. These best fitting parameter estimates yielded somewhat greater internal validity; actual and expected accuracy values correlated 0.90 and the mean squared error of the fit was .0001. We used these best fitting parameter estimates, rather than the estimates built into the tutor, in assessing posttest predictive validity.

Posttest Exercise Predictions

In principle we can predict response accuracy at each goal in the posttest with the above performance equation just as in the tutor. We can use the final probability pL_{rs} computed for each rule and subject in knowledge tracing to predict whether subject *s* will successfully apply rule *r* when needed in the posttest. In applying the performance equation to the posttests, we assume the same performance parameter estimates pS_r and pG_r as in the tutor, and make the simplifying assumption that no learning is occurring during the posttest, ($pT_r = 0$).

If production rules are independent as the model assumes, then we can derive one straightforward accuracy prediction. The probability that a student will complete an exercise correctly is given by

$$\pi pC_i$$

the product of the probabilities that the student will respond correctly at each successive goal in solving the exercise.

For each student and each posttest we computed (1) the proportion of exercises actually completed correctly and (2) the mean expected probability of completing each exercise correctly. The average of these statistics across subjects is displayed in Table 2. (The standard deviation of these means is presented in parentheses). In addition the correlation and mean squared error of the actual and expected measures across subjects is displayed. As can be seen the knowledge tracing and remediation model predicts the mean performance of the group quite well. The actual and expected values are quite similar across all three posttests. Students are performing somewhat better than expected in the second posttest and somewhat worse in the third posttest. The correlation of actual and expected values across subjects is negligible in the first two posttests, but is substantial in the third posttest ($r = 0.68$, $p < .0001$). In the first five sections of the curriculum students practiced with the tutor until reaching mastery criterion. Consequently there is little variability in learning probability estimates across students in these sections and little potential for a predictive relationship in the first two posttests. In the final section however, students completed a fixed set of exercises and the estimates of the students knowledge state varied more substantially. Under these circumstances, the probability estimates the student had learned the coding rules are highly predictive of individual differences in students' posttest performance.

Table 2
Actual and expected proportion of exercises completed correctly across subjects in each of the three posttests.

	Mean Probability Correct			
	Actual	Expected	r	MSE
Posttest 1	.89 (.10)	.88 (.01)	-.26	.01
Posttest 2	.89 (.13)	.80 (.003)	.20	.02
Posttest 3	.55 (.26)	.61 (.02)	.68	.06

Posttest Coding Goal Predictions

In attempting to predict accuracy at each goal in the posttest exercises, decisions must be made about how to score actions when a student has left a recognizable solution path. We used a relatively simple scheme with three rules: (1) if a student makes a mistake at one goal in the (hierarchical) goal tree, we continue tracing actions at sibling goals; (2) if a student makes a within-category error (substituting one constructor function for another or one extractor for another), we continued tracing descendent goals; and (3) if a student makes a non-categorical error, tracing of descendents is suspended, but resumed if the error is corrected. This scoring scheme is somewhat overly strict. If the student makes a single error, e.g., omitting the outermost of nested extractor functions, or perhaps inverting the the order of nested extractor functions, each action in the series may be scored as incorrect.

For each student and each posttest we computed (1) the proportion of goals correctly satisfied across all exercises and (2) the mean expected probability of correctly satisfying a goal across exercises. The average of these statistics across subjects is displayed in Table 3. (The standard deviation of these means is presented in parentheses). In addition the correlation and mean squared error of the actual and expected measures across subjects is displayed. The overall pattern of results is quite similar to Table 2. The actual and expected accuracy rates are quite similar across the three tests, although students are underperforming expectations on the third posttest. Actual and expected accuracy is uncorrelated for the first two tests again, but very strongly correlated for the third test, $r = 0.80$, $p < .0001$.

Table 3
Actual and expected proportion of goals satisfied correctly across subjects in each of the three posttests.

	Mean Probability Correct			
	Actual	Expected	r	MSE
Posttest 1	.94 (.06)	.94 (.005)	-.22	.004
Posttest 2	.94 (.09)	.93 (.002)	.10	.01
Posttest 3	.83 (.15)	.93 (.01)	.80	.03

Discussion

The model that underlies knowledge tracing again provided a good fit to accuracy data in the tutor. More importantly, the knowledge tracing model also provided a good fit to posttest accuracy data. The model predicted mean posttest accuracy across subjects quite well at both the level of complete exercises and individual programming goals. Moreover, when the probability of having learned the rules varied substantially across subjects, in the case of the third test, the model was quite sensitive to individual differences among students in posttest performance for both exercises and individual goals. It should be noted that these predictions were based on best-fitting learning and performance parameters for this group of subjects and so these results represent an upper bound on the predictive validity of knowledge tracing. We have made substantial changes in the cognitive rule set between studies to date, so we have yet to observe how well a set of parameter estimates generalizes across different groups of students.

There are some indications in the data that knowledge tracing can be refined further. First, the actual accuracy rates are substantially more variable across subjects than the expected accuracy rates. In part this is expected due to sampling variability, but it may also result in part from holding learning and performance parameters constant across subjects. In using best fitting parameter estimates for the group as a whole the model will tend to underestimate the knowledge state of students who are doing well and overestimate the knowledge state of students doing poorly, thus reducing the variability in expected accuracy rates.. We have observed related phenomena in earlier studies (Corbett & Anderson, 1992a,b). By dynamically adjusting the four learning and performance parameters as the student works through the tutor exercises it should be possible to improve the model's sensitivity to individual differences.

Predictive accuracy may also be enhanced by incorporating a retention factor into knowledge tracing. On average students performed slightly below expectation on the third posttest, but this effect is largely attributable to forgetting. Each posttest was cumulative and eight of the exercises on the third posttest were similar to the exercises on the second posttest. Students averaged only 60% correct on these exercises. This is a substantial drop from the 89% success rate achieved with similar exercises on the second posttest rate and substantially less the the 77% success rate predicted by the model. Meanwhile students performed slightly better than predicted on the remaining seven exercises in the third test that reflected the final two tutor sections (50% actual success vs 41% predicted success). This suggests that we need to incorporate a retention factor into knowledge tracing, particularly since forgetting may contribute to individual differences in posttest performance.

In summary, the general pattern of results is encouraging confirmation of the knowledge tracing procedure. The model's success in predicting posttest performance at the level of individual goals (rule applications) suggests that knowledge tracing need not be limited to immediate feedback environments that constrain students to remain on recognized solution paths. While immediate error correction simplifies the process, student modeling can be pursued in less constrained environments to the extent that (1) the relative independence of task subgoals can be specified and (2) the tutor can recognize when the student recovers from an error and moves back

onto a solution path. In the case of programming in particular, such flexibility also makes it possible to trace students' program testing and debugging skills. If the code generation model embedded in the current tutors is supplemented with a code testing and debugging model, then students' movement off a recognizable code generation path would become the occasion to monitor their error detection and recovery skills.

References

Anderson, J.R., Conrad, F.G., & Corbett, A.T. (1989). Skill acquisition and the Lisp Tutor. *Cognitive Science*, 1, 467-505.

Anderson, J.R., Corbett, A.T., Fincham, J.M., Hoffman, D., & Pelletier, R. (1992). General principles for an intelligent tutoring architecture. In V. Shute an W. Regian (eds.) *Cognitive approaches to automated instruction.* Hillsdale, NJ: Erlbaum.

Atkinson, R.C. (1972). Optimizing the learning of a second-language vocabulary. *Journal of Experimental Psychology*, 96, 124-129.

Corbett, A.T., & Anderson, J.R. (1989). Feedback timing and student control in the Lisp Intelligent Tutoring System. In D. Bierman, J. Breuker & J. Sandberg (eds.) *Artificial Intelligence and Education: The Proceedings of the Fourth International Conference on AI and Education.* Springfield, VA: IOS.

Corbett, A.T., & Anderson, J.R. (1990). The effect of feedback control on learning to program with the Lisp Tutor. In The Proceedings of the Twelfth Annual Conference of the Cognitive Science Society. Hillsdale, NJ: Erlbaum.

Corbett, A.T., & Anderson, J.R. (1991). Feedback control and learning to program with the CMU Lisp Tutor. Paper presented at the Annual Meeting of the American Educational Research Association.

Corbett, A.T., & Anderson, J.R. (1992a). Knowledge tracing in the ACT Programming Tutor. *The Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Corbett, A.T., & Anderson, J.R. (1992b). Student modeling and mastery learning in a computer-based programming tutor. In C. Frasson, G. Gauthier & G. McCalla (eds.) *Intelligent Tutoring Systems: The Proceedings of the Second International Conference on Intelligent Tutoring Systems.* New York: Springer-Verlag.

Goldstein, I.P. (1982). The genetic graph: A representation for the evolution of procedural knowledge. In D. Sleeman and J.S.Brown (eds.) *Intelligent tutoring systems.* New York: Academic.

Acknowledgement

This research was supported by the Office of Naval Research, grant number N00014-91-J-1597 from .