Advanced Research on Computers in Education
R. Lewis and S. Otsuki (Editors)
Elsevier Science Publishers B.V. (North-Holland)

Two pseudo-students: Applications of machine learning to formative evaluation

Kurt VanLehn

Departments of Psychology and Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213 U.S.A.

1. The need for on-line evaluation of instructional designs

Ideally, the design of instructional material should include ample testing of preliminary designs so that design errors can be detected and repaired before the instruction is deployed. Such formative evaluations are rarer than they should be for several reasons:

- Although the final polishing can be left off, all the little design details must be fully worked out before the instruction can be given to human students. It would be much better if formative evaluations could be conducted while the design was still in a rough, preliminary state. When design errors are detected early in the design cycle, less time is wasted working out the details of designs that will ultimately be discarded.
- Testing instruction with human students can be difficult, time consuming and expensive. Although increasing the quality of instruction is always a desirable goal in principle, the time and money needed for formative evaluation may not always be justifiable.
- Current testing methods often yield only two kinds of information, neither of which is particularly helpful to designers. First, there is a overall assessment of instructional effectiveness, such as a difference between pre-test and post-test scores. Second, there are the participants' comments as to which parts of the instruction seem to work well, and which did not.

In contrast to the situation in instructional design, consider the state of the art in mechanical and electrical design. After drafting blueprint or schematic of their design on a CAD station, designers can submit it to a simulation program for evaluation. Simulation programs provide very detailed reports on the design's performance which allow the engineer to locate design errors early in the design process. They may not find all the errors, because the simulation programs and testing regimes are not always perfect, but catching errors early is much cheaper than catching them later or not catching them at all.

The goal of the research described here is to develop simulation programs that can be used

for formative evaluation during the instructional design process. Such simulations are called *pseudo-students*, because they simulate human students learning from the given instruction. However, unlike human students, pseudo-students keep a detailed trace of the learning so that the designer can discover the causes of undesirable pedagogical outcomes. For instance, one pseudo-student, *Sierra*, helped demonstrate that many systematic arithmetic errors are caused by incomplete and poorly sequenced instruction (VanLehn, 1990a). Most of these design defects would be easy to fix now that they have been detected.

2. Types of pseudo-students

Just as there must be multiple simulation programs in mechanical and electrical engineering, there should be multiple pseudo-student programs, each adapted for a particular class of instructional design problems. An instructional design problem is framed by two important choices: the delivery mode and the subject matter. There are many different options for both. For instance, instruction can be delivered by lectures, textbooks, simulation-based trainers, printed tutorials, human tutors or peer teaching groups. Subject matter can range from text editor commands to algebraic problem solving to mechanical design. It is not practical to develop pseudo-students that can be applied to all problems in this vast two-dimensional space. However, it is feasible to address well-defined regions characterized by particular classes of delivery modes and subject matters. This paper discusses two pseudo-students, Sierra and Cascade. The subject matter of Sierra is arithmetic algorithms, and the delivery mode is standard textbook-based classroom instruction. The subject matter of Cascade is college physics problem solving, and the delivery mode is self-study of the textbook.

There are many individual differences among students, and it would be pointless for a pseudo-student to model just one kind of student. Thus, a proper pseudo-student must have parameters for representing individual differences among students. A thorough evaluation of a proposed piece of instruction requires gathering the reactions of a whole "school" of pseudo-students, where different members of the school have different parameter settings. However, if there is too much variability among the students, it may be infeasible to capture it all with mere parameter variations. Structurally different pseudo-students may be necessary. In physics problem solving, there are two rather distinct learning learning styles (Chi, Bassok, Lewis, Reimann & Glaser, 1989), so the current version of Cascade has two different learning models. Although less is known about arithmetic learning styles, it appears that there are also two approaches, a "syntactic" one and a "semantic" one (Resnick, 1982). The semantic or meaningful approach generally leads to acquiring correct knowledge of the algorithms. The syntactic approach sometimes leads to correct knowledge, but it often leads to acquiring incorrect, "buggy" algorithms. Sierra models only the syntactic approach.

Given all the recent work in machine learning, it is certainly technologically feasible to build programs that learn from the same kind of instruction that human subjects do. Moreover, it would be a simple matter to add a trace to their processes so that one could find the causes for the learning outcomes produced while running the program. In fact, it would not even be necessary to have the pseudo-student produce measurable outcomes, such as test scores, in

order to tell if it had learned correctly. One can simply inspect the knowledge structures it produces instead. If the acquired knowledge is incorrect, one can inspect the traces, find the causes of the incorrect knowledge, and suggest changes in the instructional design.

No one would trust such suggestions unless the pseudo-student had been very carefully validated with human students. Even if its learning processes were be consistent with what is known about human psychology, one would still need assurances that its predictions about learning outcomes are in fact accurate predictions of the learning outcomes from human students. In short, an important phase in the development of a pseudo-student is comparing its learning outcomes to human students.

This introduces another important dimension along which pseudo-students can differ. They can produce different learning outcomes for comparison with human students. They might predict achievement test results, transfer, the time require to learn to criterion, error profiles, or any other outcome that is both measurable in both human and machine behavior. For instance, Sierra predicts systematic errors, and Cascade predicts problem solving protocols.

In summary, there are four dimensions along which pseudo-students differ: (1) the subject matter area, (2) the delivery mode, (3) the class of learning styles or approaches, and (4) the outcome measure used for assessing the accuracy of the pseudo-student as a model of human students.

3. Sierra

There are two big problems in teaching arithmetic procedures. The problem that has received the most research is how to get students to acquire a meaningful understanding of the procedures. However, a more basic problem is that some students do not learn the correct procedure at all. Instead they acquire a buggy procedure, which may resist detection and remediation for years (Brown & Burton, 1978; VanLehn, 1982). Buggy procedures cause errors that are systematic: the student will produce the same incorrect answer when given the same problem twice. Sometimes the class of problems that cause incorrect answers is very small relative to the whole class of arithmetic problems, so detecting a buggy procedure can be difficult. Worse still, there is evidence that bugs often spontaneously resurface even after apparently successful remediation (Resnick & Omanson, 1987). These difficulties in detecting and remediating bugs have led researchers to suggest that instruction be redesigned so that bugs never have the opportunity to be learned (Resnick & Omanson, 1987). This is the design problem that motivates Sierra.

Sierra was originally intended as psychological model of the learning processes that cause bugs and as an exercise in a new method of psychological investigation called competitive argumentation (VanLehn, Brown & Greeno, 1984). Now that it has been built, its more practical benefits have become apparent. This paper explores its potential as a pseudo-student.

There are several assumptions about learning that are deeply ingrained in Sierra. The most important one is that only the kinds of learning that can lead to bugs are modeled. In particular, Sierra does not include anything like the kinds of meaningful procedural learning that are ascribed to spme good students. That kind of learning presumably produces only correct procedures. If we want to detect instructional design errors that can cause errors, it does no good to include a kind of learning that is not sensitive to those errors. We want to design instruction so that even students who cannot or will not learn meaningfully can at least get a correct procedure out of the instruction.

A second assumption is that students who acquire bugs are driven mostly by the examples and exercises given to them, and they make little use of the natural language explanations that textbooks and teachers offer them. This assumption is consistent with many laboratory and field studies (see VanLehn, 1986, for a review).

In order to actually use an mathematical procedure, one must know more than the procedure itself. One must also understand the syntax of the mathematical notation. For instance, the rows and columns are important in subtraction, but diagonals are not. For many kinds of mathematics, one must also know the addition, subtraction and multiplication tables (e.g., that 3+5=8). Sierra models only the process of learning procedures, so it is assumed that prerequisite knowledge of notation and number facts has been mastered.

These assumptions imply that Sierra's learning task is to induce a procedure from examples of its operation. This learning task has been widely studied in the machine learning literature (see VanLehn, 1987, for a review), and there are two basic methods for solving it. One is explanation-based learning (EBL). EBL begins by explaining the first of the given examples. Its explanation is based on a complete understanding (called a domain theory) of the subject matter. The explanation is generalized slightly and stored in memory. The next time it encounters a problem that is similar to the example's problem, it can retrieve the generalized example and re-use it solution. This is more efficient than deriving the solution from first principles. Thus, EBL improves the overall efficiency of the problem solver, allowing it to solve problems that might otherwise be beyond its reach.

The second main learning technique in machine learning is called similarity-based learning (SBL). SBL compares examples to each other, notes their similarities and differences, and forms generalizations.

Neither EBL nor SBL are exactly right for modeling how humans learn mathematical procedures. The problem is that most procedures are taught incrementally. First a simple version of the procedure is taught. For instance, the first version of a subtraction procedure might only handle problems that do not require borrowing (e.g., the simple procedure can solve 38-13 but not 31-18). When this is mastered (more or less), students are introduced to a more complicated version of the procedure, such as one that can handle simple cases of borrowing (e.g., 31-18) but not more complicated cases (e.g., 301-8). This incremental teaching process of continues until the complete procedure has been presented.

The incremental nature of procedural teaching makes SBL inappropriate because SBL has no way to utilize material that it learned earlier. When an SBL algorithm is trying to master the second lesson, it has no way to use the procedures that it acquired during the first lesson. Clearly, this is not how students learn procedures.

On the other hand, EBL is also inappropriate. For procedural EBL, the domain theory is a procedure. EBL creates specialized versions of the procedure adapted for particular types of problems, which may make problem solving more efficient for those problems. However, it cannot actually learn the procedure itself. Yet that is exactly what mathematics students are doing, so EBL is not an appropriate model of their learning.

In order to build Sierra, a new machine learning technique was invented (actually, it was invented concurrently by several investigators -- see VanLehn, 1990a, for discussion). It combines EBL and SBL. The basic idea is to explain an example as much as possible using the current procedure. If the explanation cannot be completed, which will happen whenever the lesson is introducing a new version of the procedure, then the algorithm finds the smallest gap in the explanation such that if that gap were filled, the explanation would be complete. It collects such gaps from several examples, then it uses SBL to assess their similarities and differences, and make a generalization. The generalization is procedural, but it is not a whole procedure for solving the problem; rather it is solves only the little bit of the problem spanned by the gaps. So this piece of procedure is called a subprocedure. The subprocedure generated by SBL is added to the old procedure, creating a new procedure that is be able to explain all the lesson's examples because the new subprocedure suffices to fill all the gaps left in the old explanations. This technique is called explanation completión (EC).

In order to use Sierra, one provides two formal encodings. The first represents of the prerequisite notational and factual knowledge knowledge. The second represents the instruction. Since Sierra ignores natural language, the instruction consists of a sequence of lessons, where each lesson contains only examples and exercises. An example is a sequence of problem states. The first problem state is the initial problem to be solved, and the remaining states show the effects of consecutive writing actions. For instance, here is an example of a subtraction problem being solved:

	2	2	2
$3^{1}4$	$\not{3}^{1}4$	$\not{3}^{1}4$	$\not{3}^{1}4$
$-\ 1\ 7$	$-\ 1\ 7$	$-\ 1\ 7$	$-\ 1\ 7$
		7	1 7

Exercises are just unsolved problems, such as the initial state above.

Sierra represents individual differences among students by being non-deterministic. In the learning algorithm, there are several choice points. For instance, there might be more than one way to generalize the set of gaps. Some students may pick one generalization, and other students might pick a different generalization. When Sierra encounters such a choice point, it makes a copy of itself on another computer. One copy takes one of the choices, and the other copy takes the other choice. Although Sierra runs best if it has many computers

dedicated to it, it can run on a single computer by saving copies of itself on disk and running them later.

In order to measure the accuracy of Sierra as a model of human students, its learning was compared that of 1,147 young students (ages 6 to 9) who were learning subtraction from two standard American textbooks. A cross-sectional design was used. Students at various stages in their training were all given the same diagnostic test. Their answers were analyzed by Debuggy (Burton, 1982) and a bug profile was developed for each student. Meanwhile, the examples and exercises were extracted from the textbooks and formalized. These two instructional designs were given to Sierra, which learned 63 distinct procedures from it. These procedures "took" the same diagnostic test as the students, the answers were analyzed by Debuggy, and bug profiles were developed. Depending on how exactly how one compares the two sets of bug profiles, Sierra modeled between 52% and 85% of the student's bugs (VanLehn, 1990a). Although there is clearly room for improvement, this degree of accuracy is much better than Sierra's predecessors and probably good enough that it could be used as a tool for formative evaluation. After all, mechanical and electrical simulators are not perfect models, yet they are still useful to designers nonetheless.

If Sierra had been used as a design tool, what would it have suggested to the authors of the two subtraction lesson sequences? Its major suggestion would be to avoid premature problem solving. Many of the bugs were caused, according to Sierra, by having students solve problems before they had mastered the version of the procedure needed for solving them. These bugs could have been prevented by, for instance, giving more examples or having a tutor supervise the practice of the initial exercises. In particular, many bugs were generated by testing itself. On many tests, including our diagnostic test, students are asked to solve problems that are beyond their level of training. This causes them to reach impasses, which they surmount by inventing simple subprocedures that are usually incorrect (Brown & VanLehn, 1980). The result is that they learn a wrong way to handle these problems, which in turns may make it more difficult to learn the right way later.

Sierra made several more local suggestions. For instance, both textbooks introduce the borrowing subprocedure over a multi-lesson sequence that gradually increases the complexity of the problems. This caused many bugs. Sierra would build an overly specific procedure during the first lesson, then have troubles generalizing it on later lessons. It would be better to introduce the subprocedure in a lesson that contains both simple and complex problems. This would allow Sierra to induce the right level of generality during the initial lesson.

Although Sierra would in principle be a useful tool for instructional designers, Sierra was exploratory software, pushing the limits of machine learning and artificial intelligence. It was slow, unfriendly, and written in a now-obselete computer language (Interlisp-D). It could perhaps be developed into usable software, but it would take a significant effort. More importantly, much has been learned about human learning since Sierra was completed in 1985. Some of those results should be incorporated into a revised version.

4. Cascade

The primary goal of the Cascade project is to understand the cognitive processes of students who are learning physics problem solving by studying examples and solving exercises. One of the products of this project will be a simulation of the learning processes. Eventually this simulation could become a physics pseudo-student.

Several recent studies have shown that a major difference between good learners and poor learners is how they study the examples. Chi, Bassok, Lewis, Reiman and Glaser (1989) were the first to discover this difference. They took protocols of 8 students as they read a chapter on translational dynamics in a standard college textbook, studied 3 examples, and worked 19 problems. Subjects were categorized as either Good or Poor students on the basis of a median split on their scores while solving the problems. Since students were not significantly different on pre-tests, we can infer that the Good students were better at learning than the Poor students. Chi et al. analyzed the protocols to see if the Good students were acting differently from the Poor students as they studied. Two significant differences were found. First, the Good students tended to explain the examples very thoroughly to themselves. They would check each line of the example's solution to see if it could be derived from the preceding lines. The Poor students, on the other hand, tended to simply paraphrase the line, then move on to the next one. Often, they just read the line without even paraphrasing it. There was a second significant difference found by Chi et al. When students commented about their understanding of the line they were studying, the Good students would tend to say that they did *not* understand the line, while the Poor students tend to say that they *did* understand it. These two findings have been replicated in two other domains: Lisp (Pirolli & Bielaczyc, 1989) and electrodynamics (Ferguson-Hessler & de Jong, 1990). The two findings are consistent with the hypothesis that a major difference between Good and Poor students is that only the Good students explain the examples to themselves, checking their knowledge by seeing if it is complete enough to regenerate the example's solution. Chi et al. call this strategy *self-explanation*.

The main goal of the Cascade project is to model the learning processes of the Good and Poor subjects. Chi et al. graciously gave us copies of their protocols, which are being used to evaluate the accuracy of the models. The project is in progress, so only a preliminary report can be given.

The model of the Poor students is quite simple. As they read the examples, they retain only those lines that have equations. These are added to the stock of equations obtained by reading the chapter's prose. When the Poor student model solves examples, it selects an equation that contains the sought quantity, then recurses to find values for the other quantities in the equation. This means-ends style of problem solving is quite common and has been modeled before (Larkin, 1986; Bundy et al., 1979).

We have tried three models of self-explanation so far. One was a simple form of explanation-based learning (van Harmelen & Bundy, 1988). It required such a complete version of the domain knowledge that it now seems quite implausible. For instance, we had

to assume that the student already knew about certain forces laws (i.e., that a surface exerts on an object a force that is perpendicular to the surface) before they started studying the examples even though the actual text first introduces these laws in the midst of the examples. Of the 111 rules in the knowledge base, 17 were introduced for the first time in the examples. Most of these rules were simply used without comment. Since EBL merely operationalizes knowledge that it already possesses, it could not explain how students learn these 17 rules.

The next model was based on explanation completion (EC), although of a much simpler kind than the EC algorithm used in Sierra. When a gap is encountered, the EC algorithm selects a template or "explanation pattern" (Schank, 1986), fills in its slots so that it will bridge the gap, and saves this structure as a new rule. This model sufficed to learn 4 of the 17 rules. The current model is a combination of EC and EBL, called explanation-based learning of correctness (EBLC). It suffices to learn all 17 rules (Vanlehn,1990b)..

The next step in the research will be to find a way to combine the Good student model and the Poor student model. This is not as easy as it may seem. The Good student model "thinks" like a physicist, in terms of forces and accelerations. The Poor student model "thinks" like a novice, in terms of variables and equations (Larkin, 1986). Although we are fairly sure that this is an accurate characterization of the 8 human subjects towards the end of the experiment, we are quite uncertain about where it developed. We doubt that the students came to the experiment with a preset algebraic or physical approaches to the material since their scores on the pretests are not significantly different. However, we do not see yet how self-explanation can cause one group to develop a physics approach while the lack of self-explanation causes the other group to develop an algebraic approach.

It might seem that the major news for education is simply that physics students should be encouraged (or coerced) into explaining examples to themselves. Perhaps all the detail one gets from a pseudo-student analysis is not useful for this instructional design problem. On the other hand, it could be that getting students to self-explain is only the beginning. Self-explanation is an improvement over paraphrasing examples, but there will probably be ways to improve it. For instance, it is clear already from the Cascade simulations that self-explanations often are much simpler if one employs "sloppy" physics inferences, such as blurring the distinction between instantaneous velocity and velocity over a period of time, or between weight as a scalar and weight as a vector. These kinds of sloppy inferences can become difficult habits to break. Indeed, the conceptual blurrings just mentioned are a common source of errors among physics students. It may take careful design in order to find example sequences that block the development of such bad habits. A pseudo-student can help with this design problem.

5. Conclusions

Two pseudo-students have been briefly described here. Space does not permit discussion of a third one, which is being developed in the domain of electronics by David Kieras (personal communication). From a practical perspective, pseudo-students have both near-term and long-term potential. In the near-term, they help us understand how students learn by proving detailed computational models of the learning processes. This often brings new pedagogical insights even though the models themselves do not find direct use in the education community. Felicity conditions for skill acquisition (VanLehn, 1990a; VanLehn, 1987) are an example of such an insight. In the long term, pseudo-students can probably play an important role in developing high quality instruction. Even though not all instruction needs to be high quality, because its life cycle might be very short or it might be used by only a few student, there are still many applications that demand high quality instruction. Pseudo-students could be a useful tool for providing it.

6. Acknowledgements

This research was supported by the Office of Naval Research's Cognitive Sciences Program, contract N00014-88-K-0098, and by the Office of Naval Research's Computer Sciences Division, contract N00014-86-K-0678.

7. References

Brown, J. S. & Burton, R. B. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2, 155-192.

Brown, J. S. & VanLehn, K. (1980). Repair Theory: A generative theory of bugs in procedural skills. *Cognitive Science*, *4*, 379-426.

Bundy, A., Byrd, L., Luger, G., Mellish, C. & Palmer, M. (1979). Solving mechanics problems using meta-level inference. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence*. Los Altos, CA: Morgan-Kaufmann.

Burton, R. B. (1982). Diagnosing bugs in a simple procedural skill. In D. H. Sleeman & J. S. Brown (Eds.), *Intelligent Tutoring Systems*. New York: Academic. 157-183.

Chi, M.T.H., Bassok, M., Lewis, M., Reimann, P. & Glaser, R. (1989). Self explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, *13*, 145-182.

Ferguson-Hessler, M.G.M. & de Jong, T. (1990). Studying physics tests: Differences in study processes between good and poor performers. *Cognition and Instruction*, *7*(1), 41-54.

Larkin, J. H. (1986). Understanding, problem representation, and skill in physics. In S. F. Chipman, J. W. Segal & R. Glaser (Ed.), *Thinking and learning*. Hillsdale, NJ: Erlbaum.

Pirolli, P. & Bielaczyc, K. (1989). Empirical analyses of self-explanation and transfer in learning to program. In G. Ohlson & E. Smith (Ed.), *Proceedings of the Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum.

Resnick, L. (1982). Syntax and semantics in learning to subtract. In T. Carpeter, J. Moser & T. Romberg (Ed.), *Addition and subtraction: A developmental perspective*. Hillsdale, NJ: Erlbaum.

Resnick, L. B. & Omanson. S. F. (1987). Learning to understand arithmetic. In R. Glaser (Ed.), *Advances in Instructional Psychology*. Hillsdale, NJ: Erlbaum.

Schank, R.C. (1986). *Explanation Patterns: Understanding mechanically and creatively.* Hillsdale, NJ: Erlbaum.

van Harmelen, F. & Bundy, A. (1988). Explanation-based generalisation = partial evaluation. *Artificial Intelligence, 36,* 401-412.

VanLehn, K. (1982). Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. *The Journal of Mathematical Behavior, 3*(2), 3-71.

VanLehn, K. (1986). Arithmetic procedures are induced from examples. In J. Hiebert (Ed.), *Conceptual and Procedural Knowledge: The Case of Mathematics.* Hillsdale, NJ: Erlbaum.

VanLehn, K. (1990). *Mind Bugs: The origins of procedural misconceptions.* Cambridge, MA: MIT Press.

VanLehn, K. (1990). Explanation-based learning of correctness: Towards a model of the self-explanation effect. Submitted to the 1990 Conference of the Cognitive Science Society.

VanLehn, K., Brown, J. S., & Greeno, J. G. (1984). Competitive argumentation in computational theories of cognition. In W. Kinsch, J. Miller & P. Polson (Ed.), *Methods and Tactics in Cognitive Science.* Hillsdale, NJ: Erlbaum.