Student Modeling as Strategy Learning

Pat Langley
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213 USA

1. Introduction

One of the major prerequisites for intelligent computer aided instruction (ICAI) is the ability to construct a model of the student's knowledge. Some progress in this area has been made through carefully studying students' behavior, and deriving common "bugs" or "mal-rules" that lead to errors. Unfortunately, this approach requires that each new domain be analyzed in detail before an intelligent tutor for that domain can be undertaken. More recently, researchers have attempted to develop generative theories to explain the origin of such bugs. Such theories would help predict bugs in new domains, and lay the foundation for more general ICAI systems.

However, recent AI learning research suggests an alternate approach to student modeling. In this paper we describe SAGE, a system that learns search heuristics, and discuss its application to constructing models of students' behavior on mathematics problems. SAGE is stated as a production system, and learns by determining the conditions under which various operators should be applied. In modeling students' strategies, the system instead determines the conditions under which students apply these same operators, whether correctly or incorrectly.

2. Learning Search Heuristics

One of the central insights of AI is that intelligence involves *search*, and a corollary of this insight is that learning often involves the acquisition of heuristics for directing search down profitable paths. We have explored the process of learning search heuristics through SAGE, a program that begins a task with weak, general methods, and that acquires domain-specific, powerful methods as a function of experience. The system has been tested on a number of puzzle-solving tasks such as the Tower of Hanoi and Slide-Jump, as well on simple algebra problems (Langley, 1982).

SAGE is stated as an adaptive production system. In other words, its procedures are cast as condition-action rules, and it modifies its behavior by constructing new condition-action rules. The system consists of two main components. The first of these contains very general rules for assigning credit and generating new rules; this component is responsible for learning from mistakes. The second component is domain-specific, but always takes the form of rules for proposing moves through some problem space. Initially, these rules contain only the *legal* conditions for making a move. As a result, SAGE sometimes makes good moves and sometimes makes bad moves; in other words, at the outset the system must *search* for a solution. However, as experience is gained, SAGE generates more conservative versions of its move-proposing rules with additional conditions. Eventually, the program arrives at a set of heuristics that propose only useful moves and that avoid bad moves entirely; that is, SAGE learns to direct its search down desirable paths.

SAGE incorporates a number of methods for assigning credit to desirable moves and blame to undesirable ones. These include techniques for noting loops, unnecessarily long paths, dead ends, and illegal moves. However, in this paper we will focus only on the most general credit assignment heuristic: learning from complete solution paths. This method is straightforward. SAGE employs its initial move-proposing rules to search a problem space, eventually finding one or more optimal solution paths. The system then retraces its steps, marking those moves lying on the paths as good instances of the heuristics it is trying to learn. In addition, moves that lead one off the path are labeled as bad instances, since they do not lead to a solution. As good and bad instances are identified, this

information is passed to SAGE's learning mechanism, which attempts to generate heuristics for directing search.

SAGE learns through a process of discrimination. Given a bad instance for a move-proposing rule, the system retrieves the last good instance of the same rule and searches for differences between the two situations in which the rule was applied. SAGE finds all differences between these two situations, and constructs a more conservative variant of the rule for each difference that it finds, with the new conditions based on those differences. Since many differences may occur, and since some of these may be spurious, SAGE does not automatically assume that all of these variants are useful. Instead, it requires that a variant be relearned in a number of different contexts before it is allowed to direct search. This is accomplished by associating a *strength* with each rule. These strengths are initially low, but they are increased whenever a variant is relearned; once a variant's strength exceeds that of its parent, the variant is applied whenever it matches. Of course, even variants based on relevant differences may still be overly general and lead to further errors. In such cases, the discrimination process is applied recursively and still more specific rules are constructed. This process continues until SAGE can solve the problems it is given without search.

3. Modeling Subtraction Strategies

The application of SAGE to modeling students' strategies is straightforward. Given a student's answers to a set of problems and the legal operators for that domain, SAGE should be able to find solution paths which give the same answers as did the student. From these solution paths it should be able to determine the conditions under which that student applied the operators, using its discrimination learning algorithm. In other words, the program should be able to learn a strategy that mimics the student's behavior, and the resulting set of rules would be equivalent to a model of that student's strategy.

We have performed initial tests of SAGE's student modeling capabilities in the subtraction domain, drawing upon earlier analyses by Brown and Burton (1978), Brown and VanLehn (1980), and Young and O'Shea (1981). These researchers have identified and modeled a variety of subtraction errors, such as: (1) always borrowing, whether necessary or not; (2) borrowing when the top number is larger, but not when it is smaller; (3) subtracting the smaller number from the larger, regardless of position; and (4) pattern errors, such as 0 - N = 0 and 0 - N = N. Our approach to modeling these errors is most similar to that of Young and O'Shea, who explained many bugs in terms of missing rules.

The application of SAGE to student modeling is best understood through an example. The system is given an initial set of rules incorporating operators for a domain. Table 1 shows the system's initial rules for subtraction.[1] These include operators for finding the difference between two numbers in a column, for decrementing a number by one, for adding ten to a number, and so forth. However, the rules in which these operators occur contain only the most general conditions. When the system is given a subtraction problem and a students' answer to the same problem, it must search in order to arrive at a similar answer. Once it has found a solution path to the same answer, SAGE employs its discrimination process to determine the conditions on various operators that will produce similar behavior in the future.

For example, suppose the system is given the problem 34 - 21, along with the student's (correct) answer of 13. Based on the solution path leading to this answer, SAGE would give the difference-finding operator precedence over the decrement and add-ten operators (since they were

---

[1] We have chosen to paraphrase the rules in English for the sake of clarity. Each line in the table corresponds to a single condition or action in the actual productions. Words in italics correspond to variables in the actual rules.

never applied). In addition, the system would note that the difference-finding operator was called upon to find the difference between 4 and 1, but not the inverse difference between 1 and 4. Comparing these two situations, the discrimination routine would find two differences. In the good instance, 4 is *larger* than 1, while in the bad instance, 1 is not *larger* than 4. Similarly, in the good instance, 4 is *above* 1, while in the bad instance, 1 is not *above* 4. Accordingly, SAGE would construct two variants, one including a condition based on the *larger* relation, and the other containing a condition based on the *above* relation.

Table 2. Initial production system for subtraction.

---

find-difference
If you are processing *column*,
  and *number1* **is in** *column*,
  and *number2* **is in** *column*,
then find the difference between *number1* **and** *number2*,
  and write this difference as the result for *column*.

add-ten
If you are processing *column*,
  and *number1* **is in** *column*,
  and *number2* **is in** *column*,
  and *number1* **is above** *number2*,
then add ten to *number1*.

decrement
If you are processing *column*,
  and *column2* **is left of** *column*,
  and *number3* **is in** *column2*,
  and *number3* **is above** *number4*,
then decrement *number3* **by 1**.

shift-column
If you are processing *column*,
  and you have a result for *column*,
  and *column2* **is left of** *column*,
then process *column2*.

At this point, one can infer that the student employs at least one of these conditions in deciding when to apply the difference-finding operator, but we cannot tell which of the conditions (or both) is used. However, suppose we next examine a problem in which borrowing is required, such as 43 - 25. If the student gives the correct answer of 18, then SAGE would infer (after finding a solution path and discriminating) that the student's differencing rule contains both of the above conditions. However, if the student gives the answer 22 instead, then the variant including the *larger* relation would be retained in favor of the variant including the *above* relation. The resulting model would always subtract the larger number from the smaller regardless of position, and would explain the student's failure to borrow in terms of the missing *above* condition. We have described only a small part of the model-building process, since the conditions on other operators must also be determined; however, this example should give the reader an idea of the basic approach.

4. Directions for Future Research

Although SAGE has been tested in a number of domains as a strategy learning system, our application of the program to student modeling is still in its initial stages. The most obvious priority is to test the system more fully in the subtraction domain. After our analysis of subtraction errors has progressed, we plan to test SAGE in the domains of algebra and symbolic integration. Like subtraction, these areas are mainly procedural, but they are sufficiently different to provide an interesting test of the system's generality. Finally, we hope to provide SAGE with the ability to generate diagnostic problems. Given a set of competing hypotheses as to why an error has occurred, the system would then be able to design critical experiments to determine which hypothesis best explained the student's behavior. Together with SAGE's techniques for discovering the appropriate conditions on operators, this method should lead to a general and robust system for constructing models of students' mathematics strategies. In conclusion, though our research is still in the initial stages, we are confident that it will lead to insights about the nature of student modeling, the nature of strategy learning, and the relation between them.

Acknowledgements

This research was supported by Contract N00014-83-K-0074 from the Office of Naval Research. I would like to thank Derek Sleeman for discussions that led to many of the ideas presented in this paper.

References

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

Brown, J. S. and VanLehn, K. Repair theory: A generative theory of of bugs in procedural skills. *Cognitive Science*, 1980, *4*, 379-427.

Langley, P. Strategy acquisition governed by experimentation. *Proceedings of the European Conference on Artificial Intelligence*, 1982, 171-176.

Sleeman, D. Inferring (mal) rules from pupils' protocols. *Proceedings of the European Conference on Artificial Intelligence*, 1982, 160-164.

Young, R. M. and O'Shea, T. Errors in children's subtraction. *Cognitive Science*, 1981, *5*, 153-177.