*Behavior Research Methods, Instruments, & Computers*
*1990, 22 (2), 179-183*

## SESSION VIII
## TEACHING USING COMPUTERS: GENERAL ISSUES AND REVIEWS

N. John Castellan, Jr., *Presider*
Indiana University

# An implementation and empirical evaluation of an exploration environment with different tutoring strategies

FRANZ SCHMALHOFER and OTTO KÜHN
*German Research Center for Artificial Intelligence, Kaiserslautern, West Germany*

RHONA CHARRON
*McGill University, Montreal, Quebec, Canada*

and

PAULA MESSAMER
*University of Colorado, Boulder, Colorado*

An exploration environment and tutoring strategies were developed for the first few hours of learning the programming language LISP. In this environment, the amount of exploratory and receptive learning can be systematically manipulated. In an experimental study with three different learning conditions, learning in a basic exploration environment (without an automated tutor) was compared to learning with an automated tutor that provided help rather selectively, and with an automated tutor that provided help whenever possible. The results showed that the selective tutor condition was most effective: The students in this condition took the least time in acquiring knowledge and solving the criterion test tasks, while solving equal numbers of the tasks correctly.

Tutoring systems provide a new means of combining learning from instructions (receptive learning) and learning through exploration, so as to utilize the advantages of both learning methods and avoid their disadvantages. In addition, the instruction and exploration sequences can be tailored to the particular needs of the learner. Since the teaching strategy of a tutoring system can be made either more or less instruction- or exploration-based (Dede, 1986), an adequate mixture of receptive and exploratory learning must be determined for every tutoring system.

In *receptive* learning, a teacher or tutor presents information so that the learner can acquire new knowledge. In *exploratory* learning, on the other hand, the learners themselves can choose which additional knowledge they want to acquire by instigating an appropriate interaction with an environment that will supposedly yield new information. Receptive learning can be divided further, according to type of material. The tutor may present general statements in the form of a text. This method has been called "learning by being told" or "advice taking" (Mostow, 1983). The tutor may also present concrete examples of a situation, in order to represent specific solutions to a particular problem. The student can thus learn from a demonstration consisting of a sequence of examples. This learning method has been called "learning from examples" (Winston, 1975).

Exploratory learning can also be divided according to type of material. The type of material available to a learner depends on what the learner is exploring. If the learner explores an environment, only concrete situational examples will be available. Thus, when a learner generates

This research was supported by grant Schm 648/1 from the Deutsche Forschungsgemeinschaft to the first author and by an equipment grant from IBM/Canada to McGill University. Correspondence should be addressed to Franz Schmalhofer, German Research Center for Artificial Intelligence, Postfach 2080, D-6750 Kaiserslautern, West Germany.

some input to a LISP-interpreter environment, this input and the environment's response together will constitute a situational example. This method has been called "learning by discovery." If a tutor or teacher is available for interrogation by the student, much more general questions can be asked and answered. The learning material may therefore also consist of text sentences. This method is called the "question and answer" method.

## A TUTORING SYSTEM FOR LEARNING ELEMENTARY LISP

To test different combinations of receptive and exploratory learning, a tutoring system consisting of a learning environment and a teaching component was developed. The learning environment was restricted to the subject's first few hours of learning the programming language LISP.

The tutoring system is applied in the following ways: (1) Learning by being told is used to induce the knowledge needed for successful learning by exploration; (2) an exploration agenda is provided to the learner (e.g., "learn the functions FIRST, REST, and LIST"), and a simple correct example is shown for each function; (3) the student's explorations are then monitored by tracking the learning progress in terms of a model (Schmalhofer, 1986); and (4) the resulting student model, which is updated on-line with every exploration episode, is used to determine when learning from a specific example and learning by being told should be applied; learning from an example is applied when a user needs help with specific details (i.e., when he or she repeatedly produces an error), whereas learning by being told is applied to direct the learner's attention and provide more global conceptual information.

In the tutoring system, the amount of receptive learning can be manipulated in two ways: Examples or advice can always be presented whenever a student behaves nonoptimally. Alternatively, learners may be allowed to learn from their own mistakes; in this case, examples or advice will only be presented after a learner has also failed to learn from such a mistake. We will now describe the various components of the tutoring system.

### The Basic Exploration Environment

The learning environment is based on a reduced LISP interpreter, which is written in TURBO PASCAL on an IBM/AT. It can handle the functions LIST, FIRST, REST, SET, and DEFUN, as well as any combination and any list structure. Acting in the learning environment, within an hour the student can learn: (1) the correct syntax for an input to the LISP system; (2) how a given input is evaluated and what result will be returned; (3) the number and type of arguments that a function requires; and (4) that quoted expressions, bound atoms, or function calls can be specified as arguments.

At the beginning of a learning session, examples for each function that the learner can explore are shown at the top of the screen. The learner must then generate an input to the LISP system. In order to avoid unnecessary typing errors, only characters valid in LISP (letters, digits, blank, parentheses, and quotation mark) can be typed, and only lines with balanced parentheses are accepted as inputs. In addition, colors are used to indicate the level of nesting in the expressions. Because these features help to generate syntactically correct training examples, they should reduce the number of trivial and useless syntax errors. (Such errors do not convey useful information about the system, and they reduce the efficiency of learning through exploration because they increase study time.)

The generated inputs are evaluated by the LISP interpreter, and either the result of the evaluation or an error message is displayed. If the monitor that supervises the learning process detects a sequence of training examples that do not seem to contain useful information, the learner is prompted to press a key in order to get help. Both the detection of inefficient exploration and the assistance provided are based on the monitoring of the knowledge-acquisition process.

### Monitoring of the Knowledge-Acquisition Process

Every input is analyzed immediately by the monitor Ideally, such an analysis should be conducted according to psychological principles. In particular, it should render a description closely related to the information tha learners store in their memories. Instead of storing every example individually in memory, learners remember only information that they consider generally relevant and ig nore information that is highly specific. The knowledge of the general form of correct inputs to the LISP system can be described by templates (Anderson, Farrell, & Sauers, 1984).

The monitor models a template construction process by means of an inductive learning mechanism that creates increasingly general template representations. Beginning with the first input, the LISP interpreter determines whether or not each input is syntactically correct. Syn tactically correct inputs are called *positive examples*. Th first positive example is stored in memory. When a sec ond positive example is generated, the two examples are compared from left to right, in order to construct a tem plate. As long as the respective elements of the two ex amples are equal, they are taken as constants of the tem plate. When they are different but are named as belonging to the same class, a variable is introduced to the template with the constraint that it may take as a value any mem ber of the respective classs. If the two elements that are being compared belong to different classes, whether o not they both belong to a more general superclass i checked. If they do, a variable is introduced with the con straint that the template must be bound to a member o the respective superclass. If no common superclass can be found, the generated input is used to build a separate template.

Since the generated inputs may differ in number of ele ments, generalizations are made regarding not only clas membership, but also number of elements. The sequence of input examples below shows how a template that is constructed from the first example is modified, and how a separate template is constructed from the fourth example. The input sequence is shown on the left, and the constructed templates and modifications are shown on the right. (?A denotes a single member and +A an arbitrary number of members of a class.)

```
1. (FIRST '(A B))       (FIRST '(A 'B)), (A is-atom),
                        (B is-atom)
2. (FIRST '(X (Y Z)))   (FIRST '(?A ?B)), (?A is-atom),
                        (?B is-expr)
3. (FIRST '((A B)))     (FIRST '(?A +B)), (?A is-expr),
                        (+B is-expr)
4. (FIRST FRIENDS)      (FIRST ?A), (?A is-bound-atom)
                        (FIRST '(?A +B)), (?A is-expr),
                        (+B is-expr)
```

## Teaching Strategy

Assistance to learning through exploration is provided by the tutor on two occasions: (1) when a sequence of *n* inputs that are redundant in terms of the constructed templates has been detected, or (2) when a sequence of *m* errors has been detected.

Since redundant examples are usually generated when a learner does not know what else can be learned about the system, information about more general or presently yet unexplored features of the system should be provided. Such information can be provided best in the form of a short text. The template that matches the last example is examined in terms of whether it can be generalized further, or whether it already describes some unit of the target knowledge. If further generalization is possible, a verbal description of more general inputs is presented. If no further generalization of the particular template is possible, a general verbal description of the yet unlearned functions is presented. The verbal descriptions provided as help are all prestored, so they can simply be selected for presentation. The help information for some redundant examples is shown below:

example 1: (FIRST '(A B))
example 2: (FIRST '(X Y))
help information: The argument for the function FIRST can be a list of any complexity.
example 3: (FIRST '((A B) (C D)))
example 4: (FIRST '(X Y))
help information: The argument for the function FIRST can also be a bound atom or a function call.

Such information should help the learner open a new exploration space, in which new informative examples can be constructed. When a sequence of errors is detected, the learner presumably has wanted to perform a task, but has failed to construct a completely correct solution out of prior knowledge. Since the learner wanted to generate a particular example, a demonstration of a correct form that is closely related to the negative example should be helpful. To accomplish this goal, the last incorrect input of the error sequence is corrected and then presented to the learner. The correction is accomplished through analysis of the incorrect input from left to right. Parentheses and quotes are deleted or added if needed, with the following restriction: If a symbol is identified as being a function name, the input is corrected whenever possible in such a way that the function name yields a function call. Below, one can see how some incorrect inputs are corrected. The corrected examples provide a solution to some task. Presumably this solution is at least related to the solution that the learner has attempted.

Incorrect:

```
1. (FIRST (FIRST '(A (B))))
2. (FIRST (REST '(A B)) (REST C D))
3. (LIST (FIRST '(A B) (REST '(C D))))
4. I AM HERE
```

Corrected:

```
1. (FIRST (FIRST '((A) (B))))
2. (FIRST (REST '(A B (REST C D))))
3. (LIST (FIRST '(A B)) (REST '(C D)))
4. '(I AM HERE)
```

## EMPIRICAL EVALUATION OF TUTORING STRATEGIES

### Method

In the following empirical study (Charron, 1989), three different learning conditions were compared. Learning in the basic exploration environment (with no tutor) was compared with both learning with a selective tutor and learning with a constant tutor. The selective and the constant tutor differed in teaching strategy. For the selective tutor, the parameters were set to $n = 3$ (assistance after three similar inputs) and $m = 2$ (correction after two errors). The constant tutor was specified by setting $n = 2$ (assistance after two similar inputs and $m = 1$ (every error is corrected).

**Table 1**
**Mean Results of the Learning Phase**

| | Condition | | |
|---|---|---|---|
| | No Tutor | Selective Tutor | Constant Tutor |
| Exploration time (min) | 17.6 | 11.8 | 17.8 |
| No. of inputs | 24.50 (8-59) | 17.67 (11-22) | 25.33 (5-62) |
| No. of errors | 7.16 (1-22) | 3.33 (0-6) | 7.50 (0-19) |
| No. of tutor assistances | | 1.00 (0-6) | 7.50 (0-19) |

Note—Ranges of values given in parentheses.

**Subjects**. Eighteen students from McGill University, who were paid $10 for their participation, were randomly assigned to one of the three conditions.

**Procedure**. After having read an introduction sheet, that outlined the experiment, all subjects read a text about data representations in LISP (Atoms and Lists) and about how the LISP interpreter evaluates expressions. After that, knowledge about the LISP functions FIRST, REST, and LIST was to be acquired under the three different experimental conditions. For each function, a single example was presented at the top of the screen. All subjects were instructed to interact with the exploration environment until they felt that they had acquired sufficient knowledge for solving related programming tasks. Among other data, the number and duration of interactions with the exploration environment were recorded. Upon completion of this learning phase, all subjects were tested with an identical set of six simple programming tasks. The programming tasks required the learners to extract and combine various elements from lists. Two sample programming tasks are as follows:

Generate an input to the LISP system using the list (A B C) so that the atom A is returned.

Generate an input to the LISP system using the list (A B C) so that the atom B is returned.

## Results

The results of the learning phase of the experiment are shown in Table 1. Subjects in the selective tutor condition spent less time in exploring the LISP functions than did the subjects in the no-tutor and constant-tutor conditions. This time difference can be explained by the fact that the selective tutor subjects generated fewer inputs and made fewer errors. Also, a smaller number of tutor corrections was required in the selective tutor conditions. However, none of these group differences were significant.

The results in Table 2 show that the subjects in the two tutor conditions required significantly less time to solve the programming tasks than did the subjects in the no-tutor condition [$F(2,14) = 4.6$, $MS_e = 47.3$, $p < .05$]. There was no significant difference in the number of correctly solved programming tasks; all subjects solved about 50% of the programming tasks correctly. When the results of the learning and testing phases are considered together, it can be said that the most efficient learning occurred in the selective tutor condition, followed by the constant-tutor condition. The worst performance was observed in the no-tutor condition.

**Table 2**
**Results of the Test Phase**

| | Condition | | |
|---|---|---|---|
| | No Tutor | Selective Tutor | Constant Tutor |
| Total time (min) | 30.45 | 18.45 | 18.49 |
| Percent correct | 47 | 50 | 61 |

## COMPARATIVE DISCUSSION

In this discussion, other tutoring approaches (e.g., An derson's LISP-tutor) will be compared to the present one and to the results of our research. In learning a program ming language like LISP, students are normally expectec to study a textbook and consequently solve a number o related programming problems. Intelligent automatec tutoring systems are typically applied for assisting learner in solving such programming problems (Anderson & Skwarecki, 1986). The tutor determines which set o programming tasks is to be solved. For each program ming task, a tutor has one or several goal structures, eacl of which determines a valid solution to the problem Whenever a learner makes a syntactic error or the learner's programming actions deviate from the tutor' goal structure, the tutoring system immediately provide help. The learner will therefore always follow one of the tutor's goal structures for solving the task.

Three important assumptions underlie this approach (1) It is better to have the teacher or tutor determine whicl tasks need to be solved than to allow the student to be creative and make his or her own selection. (2) All the possible or good solutions to a task can be generatec on-line or are known beforehand. (3) Presumably stu dents cannot learn from their own errors. Therefore students' errors should be immediately corrected whe they arise.

The preceding assumptions, however, may not hold fo all learning situations. Since learners may prepare them selves for performing quite different tasks with a system different learners have different learning needs. Learner may also come up with new solutions to a problem, whicl have not been represented in the tutoring system. Sinc the errors that students make in performing tasks ar directly related to the students' own goals, error message may themselves stimulate personally useful reasonin processes. Under some circumstances, students may there fore better learn from their own errors rather than bein presented correct solutions that are only vaguely relate to their own thought processes. Rather than make the learner abandon (erroneous) thoughts completely, the presentation of a related correct input may sometimes bet ter allow the learner to straighten out errors and thus con tinue along a line of personal reasoning.

The present empirical study showed that the partial tu tor condition was the most effective. These learners re quired the least amount of time to acquire the knowledg to solve the criterion test tasks and to solve an equal num ber of programming tasks correctly. Overall, tutoring wa successful, in that the learners in both tutor condition performed better than those in the no-tutor condition. Th results thus suggest that a partial tutoring strategy use with an exploration environment may be quite success ful, because this combination includes the advantages o both types of learning.

## REFERENCES

Anderson, J. R., Farrell, R., & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science*, **8**, 87-129.

Anderson, J. R., & Skwarecki, E. (1986). The automated tutoring of introductory programming. *Communications of the ACM*, **29**, 842-849.

Charron, R. (1989). *The influence of different degrees of assistance in automated tutoring*. Unpublished master's thesis, McGill University, Montreal.

Dede, C. (1986). A review and synthesis of recent research in intelligent computer-assisted instruction. *International Journal of Man-Machine Studies*, **24**, 329-353.

Mostow, D. J. (1983). Machine transformation of advice into a heuristic search procedure. In R. Michalski, J. G. Carbonall, & T. M. Mitchell (Eds.), *Machine learning* (pp. 367-403). Palo Alto, CA: Tioga.

Schmalhofer, F. (1986). The construction of programming knowledge from system explorations and explanatory text: A cognitive model. In C. R. Rollinger & W. Horn (Eds.), *GWAI-86 and 2nd Austrian Artificial Intelligence Conference* (pp. 152-163). Heidelberg: Springer.

Winston, P.H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision* (pp. 157-209). New York: McGraw-Hill.