Acquiring Computer Skills by Exploration versus Demonstration

Franz Schmalhofer and Otto Kühn[1]

McGill University / Canada, and Universität Freiburg / West Germany

It is well known that more effective learning can be achieved by tailoring the learning episodes to the particular needs of an individual rather than presenting the same sequence of instructions to all learners. There are two ways by which this can be achieved: a tutor can adjust its instructions to the learner's previously acquired knowledge or it may simply allow the learners themselves to determine the sequence of learning episodes. In the first case, the individualization is determined by the prior learning histories with the global learning goals being assumed identical for all learners. The LISP tutor of Anderson, Boyle & Reiser (1984) is an example for such a system, in which the learning goals are determined by the tutoring system, and the instructions which assist the learners in performing the respective task are adjusted to the learners' knowledge. Further individualization can be achieved by having the learners learn by exploration: the learners themselves can now set their own learning goals according to their specific interests. Student-driven exploration can be enhanced by providing instructions on those occasions where learning by exploration fails. For example, the redundancy checkers discussed by Brown (1984), can be used to detect weaknesses in a student's exploration.

In a number of cases learning by exploration may be more appropriate. For example, in application systems such as text editors or spreadsheets depending on their needs users may learn different parts of the system. Under such circumstances it can be quite frustrating to the learners when they are taught system components in which they are not interested. Contrary to instruction-based learning, in which advanced learners may still be presented with introductory materials due to the difficulties of knowledge diagnosis, learning by exploration allows the learners themselves to decide what to learn. Under certain circumstances learning by exploration may well be more effective then learning from instruction. For example Carroll et al. (1985) have shown that a text editing system may be learned more effectively by exploring it than by studying a conventional manual.

The advantages of learning by exploration may be caused by a number of different factors. Learners can selectively acquire that knowledge which they consider most important. They can be more active and set their own learning goals. In order to achieve their learning goals, they can engage in problem solving (Robert, 1986). This may lead to procedural and problem solving oriented knowledge representations which are better suited for solving computer tasks. Successfully solving these problems may be quite motivating for the learner. Since learning by exploration originates from the student's own domain knowledge, the newly acquired knowledge becomes inherently connected and interwoven with the prior knowledge. Therefore, it may be better remembered.

However, each of these advantages may also turn into a disadvantage. A student could have insufficient domain knowledge to set appropriate learning goals. Because of insufficient domain knowledge the students may not be able to determine which knowledge is really important. They may acquire suboptimal procedures for achieving their goals or in the extreme case no successful procedures at all. Problem solving processes may not always be successfully completed and can be more time consuming than learning from instructions. This causes frustration for the learner. A student's lack of domain knowledge can thus put severe limitations on what can be learned by exploration.

Although learning by exploration and learning by instructions (or, more specifically, demonstrations) differ in a number of interrelated ways, one difference appears to be most fundamental. While instructional materials are determined by the teacher, who is very knowledgeable of the domain, in exploration the learning episodes are generated by the students who know about their particular knowledge desires. The advantages and disadvantages of learner versus teacher generated learning episodes were investigated in an experiment in which 80 students from the University of Freiburg acquired some elementary LISP programming knowledge. The results of this study show how the effectiveness of

[1] This research was supported by grant Schm 648/1 from DFG (German Science Foundation)

learning by exploration depends upon the amount of relevant knowledge which the learner can utilize for generating learning episodes. These results suggest a particular combination of the two learning methods. On this basis, a supervised exploration environment for acquiring some elementary LISP knowledge was developed, implemented on an IBM PC/AT, and consequently evaluated through two learners' think aloud protocols.

Experiment

Forty students who had some previous experience with computers (either they had taken a BASIC programming course or had used some application software such as word processors or database programs) and 40 students without any prior computer experience were first instructed about some fundamental LISP concepts (atoms and lists). Then they acquired additional knowledge about some simple LISP functions either by exploration or by learning from demonstration examples. In particular the function FIRST, which extracts the first element from a list, and the function SET, which binds a LISP-expression to some symbol, were learned. Simple LISP functions were used as the learning domain because modularity is a prerequesite of explorability and the LISP functions satisfy this requirement. Also, the subjects of the study should not have had any specific domain knowledge. Whereas this was true for LISP, they may have already been familiar with text processing.

Exploration condition: In the exploration condition the learners could enter LISP expressions with an editor providing help for generating syntactically correct inputs. A LISP interpreter evaluated these expressions and gave appropriate feedback. The exploration condition was divided into three blocks so that each of the functions which were to be learned would be noticed by the subjects. For each block the subjects generated either 8 inputs (first and second block) or 16 inputs (third block). At the beginning of each block one or two simple meaningful inputs to the LISP system were presented, namely

1. "(FIRST '(A B))",
2. "(FIRST (FIRST '((A B) C)))",
3. "(SET 'FRIENDS '(JACK JOHN))
   (FIRST FRIENDS)".

Instruction condition: In the instruction condition 32 appropriately selected examples were presented where each block started with the example which was also presented in the exploration condition. In each condition 32 training examples were thus generated or presented. The exploration subjects who entered the presented inputs had to create another 28 inputs on their own, whereas the subjects in the demonstration condition were presented with 32 examples and could not generate any examples by themselves.

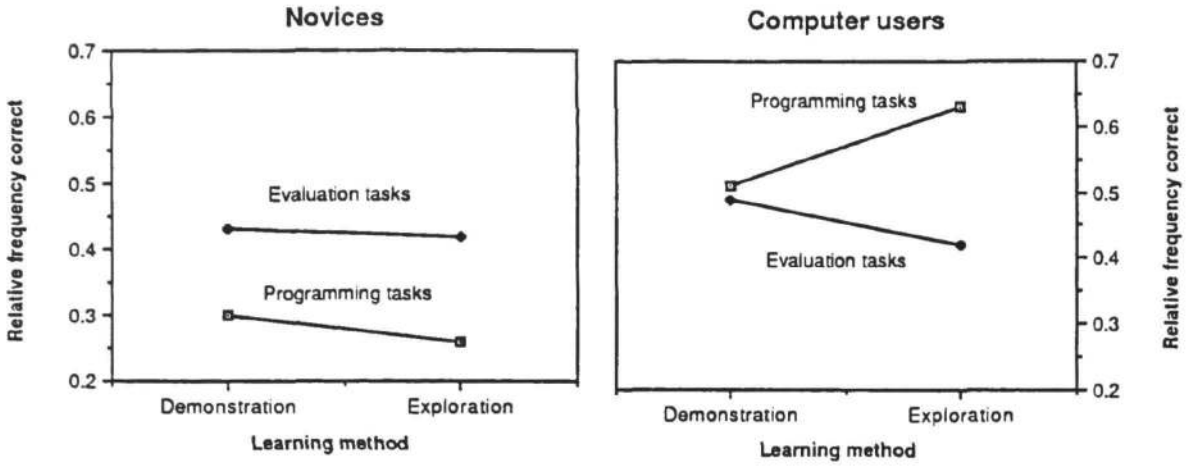
Programming and evaluation tasks: The acquired knowledge of each learner was tested by 10 programming tasks in which the subjects had to generate an input to the LISP-system in order to obtain some prespecified result. The inputs were evaluated by the LISP-interpreter and the result was shown to the subjects. If the result of the subjects' input was not the result that was to be achieved, the subjects were given two more trials to achieve the correct result. Thereafter, the subjects' knowledge about the LISP-system was examined by evaluation tasks, in which inputs to the LISP-system were presented, and the subjects, rather than the LISP interpreter, had to generate the results. The whole experiment took between 1.25 and 3 hours. For a more detailed description see Kühn & Schmalhofer, 1987.

Results :

For the novices and the computer users the relative frequencies of correct solutions in the programming and the evaluation tasks as a function of instruction method are shown in Figure 1. A (2x2x2) ANOVA with the factors prior knowledge, instruction method and test task showed that overall the two tasks were about equally difficult ($F(1,76)=0.31$), and that computer users performed better ($F(1,76)=21.6$, $MSE=0.35$, $p<.001$). More interestingly, novices performed better in the evaluation tasks and computer users performed better in the programming tasks, resulting in a prior knowledge by test task interaction ($F(1,76)=20.5$, $MSE=0.14$, $p<.001$). In addition, learning from demonstrations was more useful for correctly solving the evaluation tasks and learning by exploration was more effective for the programming tasks, resulting in an instruction method by test task interaction ($F(1,76)=5.13$, $MSE=0.14$, $p<.05$). Supposedly, in the exploration condition the generation of inputs was trained which is an important component for successfully solving programming tasks. The instruction groups had some

advantage in the evaluation tasks, possibly because the self generated learning examples provide less complete information about the system than the examples selected by a teacher.

Figure 1: Proportions of correctly solved test tasks



In order to analyse the relation between the subjects (programming and evaluation) performance and the studied examples, the training examples generated by the subjects in the exploration condition as well as the training examples presented by the tutor in the instruction condition were classified as belonging to one of four categories which were definied as follows: 1) positive examples that contain new information about the system, 2) redundant positive examples, 3) "near misses" (Winston, 1987), i.e. negative examples that are very similar to positive examples and thus convey information about the system, and 4) all other inputs were classified as "far misses". This classification was performed with a LISP programme, which constructed generalized templates representing the knowledge that may be acquired from the training examples (see Schmalhofer, 1986). The classification was performed according to the model of knowledge acquisition of the learning environment that is presented in the next section of this paper.

Table 1 shows the relative frequencies of the four types of examples generated by the novices, the computer users and the tutor. It can be seen that the exploration subjects generated fewer negative examples than were presented in the instruction condition. Both novices and computer users generated more redundant inputs than were presented in the instruction condition. Also, computer users generated more positive new examples than novices ($t(38)=2.45$, $p<.05$). Furthermore, a considerable proportion of the generated inputs in the exploration condition were far misses which do not provide useful information.

Although the training examples generated in the exploration condition were of poorer quality than those presented in the instruction condition, the computer users of the exploration condition performed better in the programming tasks than the computer users of the instruction condition. For the programming tasks, the advantage of generating training examples apparently can outweigh the disadvantage due to the usually poorer quality of selfgenerated training examples.

Table 1: Proportions of 4 types of training examples as generated by novices and computer users, or presented in the instruction condition

Type of example :	*Novices*	*Computer Users*	*Instruction*
positive new	*0.23*	*0.31*	*0.38*
positive redundant	*0.28*	*0.31*	*0.13*
"near misses"	*0.24*	*0.19*	*0.50*
"far misses"	*0.26*	*0.19*	*0.00*

Thus, the knowledge that can be acquired from learning by exploration depends upon the quality of the generated training examples, which itself depends upon the subjects' prior knowledge. Two multiple regression analyses were conducted for the proportions of correctly solved programming and evaluation tasks with the proportions of the first 3 types of training examples (positve new, positive redundant, and near misses) and prior knowledge (with the dummy coding 0 for novices and 1 for computer users) as predictors. Initially, all 4 predictors were entered into the regression equation and then insignificant predictors were dropped stepwise. The results of these analysis are shown in Table 2.

Table 2: Prediction of correctly solved tasks from prior knowledge and 3 types of generated training examples in the exploration condition

	programming tasks		evaluation tasks	
	ß-weight	correlation	ß-weight	correlation
prior knowledge	0.290***	0.605***	0.104*	0.457**
positive new	1.328***	0.621***	0.399*	0.433*
positive redundant	-0.421^{+}	-0.038		

*** = p<.001, ** = p<.01, * = p<.05, + = p<.10

It can be seen that the proportion of positive new training examples is a good predictor of the performance in the two tasks, even after the effect due to differences in prior knowledge has been taken into account. The results furthermore show that the more redundant training examples were generated the less programming tasks could be solved in the test phase. The results demonstrate that the effectiveness of learning by exploration depends on the learners' domain knowledge and their ability to generate appropriate training examples .

Combining exploration and instruction in a tutoring component

The experimental results indicate that learners with sufficient prior knowledge benefit from exploring a computer system, and that instructions are more efficient when that knowledge is not yet available. Thus instructions can be used to induce the knowledge which is necessary for more successful exploration. The correlation between the type of generated training example and task performance shows that the generated training examples can be used to diagnose whether the learner has sufficient prior knowledge for successful exploration. On the basis of such a diagnosis specific instructions can be presented to provide the information which is needed to make exploration successful.

Rather than starting learning by instruction it thus seems feasible to have all learners start by exploring the system, monitor their exploration behavior and provide instructions as needed. Thereby the learners can engage in active learning and determine themselves what to learn. Instructions are only presented when they are needed to maintain this active learning process. The advantages of learning by exploration can thus be utilized while its disadvantages are avoided. Such an exploration environment was developed for learning some elementary LISP functions.

The basic exploration environment:

The learning environment is based upon a reduced LISP interpreter which is written in TURBO PASCAL on an IBM PC/AT. It can handle the functions LIST, FIRST, REST, SET, DEFUN as well as any combination and any list structure. By acting in the learning environment the student should learn:

- the number and type of arguments which a function requires
- the correct syntax for an input to the LISP system
- how a given input is evaluated and what result is returned
- that quoted expressions, bound atoms or function calls can be specified as arguments.

At the beginning of the learning session only the names of the functions which the learner can explore are shown on 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 that are valid in LISP (letters, digits, blank, parentheses and the quote) 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. As these features help generating syntactically correct training examples, they should reduce the number of trivial syntax errors

that result from typing errors and limitations of a learner's memory. Such errors don't convey useful information about the system to be studied, and they reduce the efficiency of learning by exploration by increasing the study time (see Köhne & Weber, 1987; Waloszek, Weber, Wender, 1986)

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

Monitoring of the knowledge acquisition process

Every input of a student is immediately analyzed by the monitor. Ideally such an analysis should be conducted according to psychological principles. In particular it should render a description which is closely related to the information which the learners store in their memories. It is known that rather than individually storing every example in memory learners only remember those things which they consider generally relevant and ignore the very specific information. 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 an inductive learning mechanism which creates increasingly general template representations. For the first and all other inputs the LISP interpreter determines whether or not it is syntactically correct. Syntactically correct inputs are called positive examples. The first positive example is stored in memory. When a second positive example is generated the two examples are compared from left to right in order to construct a template. As long as the respective elements of the two examples are equal they are taken as constants of the template. When they are unequal but are named as belonging to the same class, a variable is introduced into the template with the constraint that it may take as a value any member of the respective class. If the two elements that are being compared belong to different classes it is checked whether or not they both belong to a more general superclass. If this is the case, a variable is introduced with the constraint that it must be bound to a member of 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 the number of elements, generalizations are made not only with respect to the class membership, but also with respect to the number of elements. The latter generalizations are performed as follows: If during the comparison from left to right either in the input or the template all elements have been processed, while in the other additional elements are available, the class memberships of these elements are determined, and if a common supertype can be found, an additional variable is added to the generalized template which can match any number of elements of that class; if no common class can be determined, different variables are used. For a sequence of input examples Table 3 shows the template, which is constructed from the first example and how this template is modified and a separate template is constructed from the forth example. (?A denotes a single member and +A an arbitrary number of members of a class.)

Table 3: Templates formed from a sequence of inputs

input sequence	Constructed templates and modifications
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)

A think-aloud study showed, that three causes of negative examples can be distinguished: 1) accidental errors such as misspelling a function name or forgetting to type a quote, 2) errors that are made to determine what elements of a template are necessary and whether it can be further generalized, and 3) errors that occur when a learner attempts new and more complex inputs.The verbal protocols showed that for incorrect inputs of type 1) or 2), the error message provided by the system contained sufficient information for the learner. Therefore, no intervention of the tutor is required. It can therefore be assumed that a negative input of type 1) or 2) is likely to be followed by a positive input. When learners make errors while trying something new, the system error message may not provide enough information. In this case,

a learner will produce a sequence of errors. Since only type 3 errors are likely to occurr in a sequence without intermittent positive examples, they can be easily detected by the monitor.

Providing examples and text-information to assist exploration

Assistance to learning by exploration is provided by the tutor on two occasions: 1) when a sequence of n (=2 in the current version of the tutor) redundant inputs has been detected, or 2) when a sequence of m (=2) 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 not yet explored features of the system should be provided. Such information can best be provided in the form of a short text. The template that matched the last example is examined as to whether it can be further generalized or whether it already describes some unit of the expert 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 not yet learned functions is presented. The verbal descriptions that are provided as help are all prestored so that they can simply be selected for presentation. Table 4 shows the help information for some redundant examples.

Table 4: Help information for some redundant inputs

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

When a sequence of errors is detected, it can be assumed that the learner wanted to perform a more complex task and did not know how to correctly specify the parameters of the task. Based upon the PUPS theory (Anderson & Thompson, 1987), it may be suspected that such errors occur because the specific form is unknown. A form can best be taught by giving an example. Since the learner wanted to generate a particular example the correct form for that particular example should be presented. To accomplish this goal, the last incorrect input of the error sequence is corrected and then presented to the learner. The correction is accomplished by analyzing the incorrect input from left to right. Parenthese 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 a way so that the function name yields a function call. Table 5 shows how some incorrect inputs are corrected.

Table 5: Correction of negative examples

incorrect:	corrected:
(FIRST (FIRST '(a (b))))	(FIRST (FIRST '((a) (b))))
(FIRST (REST '(a b)) (REST c d))	(FIRST (REST '(a b (rest c d))))
(LIST (FIRST '(a b) (REST '(c d))))	(LIST (FIRST '(a b)) (REST '(c d)))
I AM HERE	'(I AM HERE)

A preliminary evaluation of the appropriateness of the provided instructions was performed by having two subjects, who were first instructed about data representations in LISP, think aloud while learning the elementary LISP functions by exploration. They were instructed to explore as long as they found it to be a useful learning experience, which was about 1.5 hours. In order not to interfere with the learners' usual exploration, no tutor assistance was provided to the learners. The sequences of the subjects interactions with the LISP-system were recorded and then used to determine the first learning episodes of each block where assistance would have been provided by the tutor. Only the first occasions of each of the 16 blocks in which the tutor would have provided assistance were used for evaluating the appropriateness of the tutor's assistance. In one block the tutor would not have provided assistance, so that only 15 assistances were to be evaluated. In 10 cases explanatory text and in 5 cases corrected examples would

have been presented. The tutor's assistance was then compared to the learners' verbalisations, which were used to judge whether the tutor assistance would have been adequate. The tutor's assistance was judged as helpful in 6 cases, as neutral in 6 cases, and as inappropriate in 3 cases. In two of the 6 helpful cases the tutor's corrected example would have prevented an error path of nine episodes, which may be judged to be quite effective.

Discussion

Most Intelligent Tutoring Systems are more instruction- than exploration-based and provide the student with rather little possibility to learn by exploration. However, for acquiring computer knowledge learning by exploration may be quite fruitful. The studied material may be better remembered when learning by exploration because students themselves can select what to learn and because the learning material originates from the students' own memory, and for whatever reason self-generated information is better remembered (Slamecka & Katsaiti, 1987). With respect to performing simple programming tasks this prediction was confirmed for learners who had some very general prior domain knowledge (computer users), but not for complete computer-novices. Learning by exploration allows to practise the generation of possible system inputs, which appears to be an important computer skill component. We suggest that learning by exploration can become even more effective when a learner's exploration is monitored so that often occuring problems can be detected and bypassed by giving appropriate instructions. A supervised exploration environment for the learning of some LISP-basics was described. In this environment explanatory text and an error-correction facility are used to provied learners with general information as well as with information about the specific forms of system-inputs. A preliminary empirical evaluation was performed for an implementation on a PC.

We believe that supervised exploration may be particularly suited for learning the specific forms for coding some already known procedures in a new environment. This may occur when writing a program in a new programming language or using some new application software. Since the commands of a system may be separately learned, they can be learned by exploration and may be even better learned by supervised exploration. However, instruction-based learning may be necessary for acquiring more complex knowledge. Learning by supervised exploration is therefore only proposed as a component for Intelligent Tutoring Systems.

References

Anderson, J.R. Boyle, C.F. & Reiser, B. (1985). Intelligent Tutoring Systems. Science, 228, 456-462.

Anderson, J.R., Farrell, R., & Sauers, R. (1986). Learning to program in LISP. Cognititve Science, 1984, 8, 87-129.

Anderson, J.R. & Thompson, R. (1987). Knowledge Representation in the PUPS Theory. Technical Report, Department of Psychology, Carnegie-Mellon University.

Brown, J.S. (1984). Learning -by-doing revisited for electronic environments. In White, M.A. Ed, The Future of Electronic Learning. New Jersey: Lawrence Erlbaum Associates.

Carroll, J.M., Mack, R.L. & Lewis, C.H. (1985) Exploring exploring a word processor. Human-Computer-Interaction, 1985, 1, 283-307.

Köhne, A. & Weber, G. (1987). Struedi: A LISP-Structure Editor for Novice Programmers. In: Bullinger, H.J. & Shackel, B. (ed.). Human-Computer Interaction INTERACT'87. Elsevier Science Publishers B.V. (North-Holland).

Kühn, O. & Schmalhofer, F. (1987). Erlernen der Computerbenutzung: durch gezielt sequenzierte Instruktion oder durch Explorieren? In: Schönpflug, W. (ed). Software Ergonomie 87, Teubner Verlag, Stuttgart, 387-397.

Slamecka, N.J. & Katsaiti, L.T. (1987). The Generation Effect as an Artifact of Selective Displaced Rehearsal. Journal of Memory and Language, 26, 589-607.

Waloszek, G., Weber, G. & Wender, K.F. (1986). Entwicklung eines intelligenten LISP-Tutors (Bericht No. 2). Braunschweig: Technische Universität, Institut für Psychologie.

Winston, P.H. (1984) Artificial Intelligence. Second Edition. Addison-Wesley, Reading, Massachsetts.