N89-19889

DEVELOPMENT OF A PERSONAL-COMPUTER-BASED INTELLIGENT TUTORING SYSTEM

Stephen J. Mueller
Computer Sciences Corporation
16511 Space Center Blvd.
Houston, Texas 77058

A large number of Intelligent Tutoring Systems (ITSs) have been built since they were first proposed in the early 1970's. Research conducted on the use of the best of these systems has demonstrated their effectiveness in tutoring in selected domains. Computer Sciences Corporation, Applied Technology Division, Houston Operations has been tasked by the Artificial Intelligence Section at NASA/Johnson Space Center (NASA/JSC) to develop a prototype ITS for tutoring students in the use of the CLIPS [1] language: CLIPSIT (CLIPS Intelligent Tutor). For an ITS to be widely accepted, not only must it be effective, flexible, and very responsive, it must also be capable of functioning on readily available computers.

While most ITSs have been developed on powerful workstations, CLIPSIT is designed for use on the IBM PC/XT/AT personal computer family (and their clones). There are many issues to consider when developing an ITS on a personal computer such as the teaching strategy, user interface, knowledge representation, and program design methodology. Based on experiences in developing CLIPSIT, this paper reports results on how to address some of these issues and suggests approaches for maintaining a powerful learning environment while delivering robust performance within the speed and memory constraints of the personal computer.

CHOOSING A DEVELOPMENT ENVIRONMENT

One of the major goals of Intelligent Computer Aided Instruction (ICAI) is to improve the quality of education and training especially for the average and below average student. For this to happen, educationally effective tutors must be made widely available to end users, be affordable, and run on inexpensive computers. Strategies for cost effective development of ICAI programs for small computers include choosing a language and an environment for developing the ICAI based on availability, language and development costs, and performance requirements.

The delivery environment chosen for CLIPSIT was the IBM PC/XT/AT family of computers using the languages CLIPS, C, and a commercially available graphics package. There were a number of reasons for which CLIPS was chosen. The first reason is that CLIPS is an expert system building tool written in C which can be compiled to create a runtime executable. A typical development strategy for ICAI is to develop a prototype first using an artificial intelligence (AI) language or programming environment and then rewrite the runtime programs in a general purpose language (such as C) to achieve optimal performance and transportability and to minimize distribution costs [2]. By using the CLIPS language, this rewriting phase could be eliminated thereby saving valuable resources.

A second reason CLIPS was chosen as the development language is its availability to developers and students. CLIPS is available at no cost to anyone currently working on a federal government contract. Since the intention is to distribute a copy of CLIPSIT along with the CLIPS language, the student would have easy access to the required software.

A third reason is the high performance and extensibility of CLIPS. External functions can easily be written in C and called from the CLIPS production rules. The ability to write user defined functions was invaluable to the teaching expert whose job it was to

manipulate the windows and menus of the user interface as well as communicate with the tutorial parser.

The IBM PC family of computers was also chosen as the development and delivery vehicle for a number of reasons. Most students have access to personal computers either at home or school or business. Availability is one of the most important requirements for realizing the utility of an ITS. Inexpensive personal computers now have the power required for many ICAI applications whereas previously most applications were developed on specialized AI workstations. Intelligent tutors are just one example of ICAI where remarkable progress is being made on personal computers. As processing power and memory capacity costs continue to decline, inexpensive machines will be capable of supporting significant ITSs. Finally, the CLIPS language is a very effective tool on the personal computer and the C language compilers necessary to take advantage of its extensibility are readily available.

CLIPSIT APPROACH

CLIPSIT is a knowledge-based system which tutors students in the concepts and syntax of the CLIPS language. The primary goal of the CLIPSIT tutor is to provide a proof of concept which can demonstrate that a usable tutor can be developed on a personal computer. Once completed, CLIPSIT will provide approximately 10 lessons with each lesson containing about 10 problems. The program currently consists of about 75 generalized rules written in CLIPS and a high quality user interface written in C and a commercial graphics package. Rules comprising CLIPSIT represent the following typical functional modules in an intelligent tutoring system: a domain expert, a diagnostic expert, a teaching expert, and a student model. The functional components of CLIPSIT are shown in (Figure 1) and described in more detail on the following pages.

The student interaction with the tutor occurs through a two-window user interface which presents a problem description in one window and accepts student responses in the other. Student responses are analyzed by the tutor and compared against expected student responses. The student is notified immediately of any discrepencies found in his syntax or logic by the tutor. Student responses which deviate from the expected response are compared to anticipated errors in a bug catalogue. Those bugs which are not present in the bug catalogue are handled by special diagnostic rules. Each problem presented to the student is stored in a separate file along with the appropriate lesson text and is presented upon request by the student model. These files contain such information as expected responses, expected errors and error messages, teaching strategies, and problem description facts which describe the allowable variations to the "correct" expected responses the student is permitted to make. At the completion of a problem, all facts pertaining to the current problem are purged and a new problem is presented. By modularizing problems, a general set of (domain) rules can exist to analyze each specific set of problem facts. Modularization has the additional benefit of eliminating the need for problem specific "buggy rules" [3] which would greatly increase the size of the rule base and inhibit performance.

DOMAIN EXPERT

The domain expert contains the knowledge that the student needs to learn. This knowledge is used to solve problems generated by the teaching expert in order to provide a basis for error analysis by the diagnostic expert. As tokens are entered by the student, (tokens are definable characters that serve to delimit the contents of an input string) they are checked against the predefined expected response for that token. More specifically, the domain expert reads in problem description facts as well as a skeleton solution containing the expected student responses. As long as the student response continues to match the expected response, the student is left alone. Should the student response differ from the expected response, additional analysis is made to determine if the response is indeed an error or simply another equally valid approach to solve the problem. When the student response is determined to be valid, the fact database is updated to reflect the current approach. However, if the student response is determined to be an error, then an error fact is created for the diagnostic expert to analyze.

Many features are built into CLIPSIT which provide the student with the freedom to use his own creative instincts to solve a problem. This is a tremendous advantage over forcing the student to follow a strict one-to-one mapping between a student response and an expected system response. One example of the flexibility of the tutor is in regard to the naming of pattern variables in CLIPS. Should the student choose to name a variable something other than what the system expects, this should be (and is) allowed. Also,

in many cases, the order of patterns in a rule is arbitrary. By reading the problem description facts from the problem file, the domain expert knows which patterns can be interchanged. Additionally, there are times when the tokens in a pattern can be arranged arbitrarily such as when the student is applying logical field constraints. All of these variations to the expected system responses are supported given that the student's solution is an equally valid solution to the problem.

Since a fast response time is of utmost importance for a tutor to hold the student's attention, only one skeleton solution is provided. That is, there is only one acceptable logic path for the student to follow. In order to make one skeleton solution span the entire set of possible solutions, the problem must be very narrow in scope. All of the problems in CLIPSIT are worded in such a way that there is basically only one way to solve the problem given the normal creative variations mentioned above. By limiting the scope, the number of solutions becomes very manageable and the expected student responses can be predicted with great accurracy. The obvious benefit of this approach is the improved response time obtained by requiring less rules and facts to describe and analyze the problem.

DIAGNOSTIC EXPERT

The diagnostic expert provides error analysis for the student responses. Once an error has been encountered by the domain expert, that error is intercepted by the diagnostic expert to try and determine the cause of the student's misconception. Based on the error analysis, the diagnostic expert hypothesizes what misconceptions the student may have. These hypotheses are recorded in the current state of the student model.

There are three sources of error diagnosis in CLIPSIT: a bug catalogue containing a list of expected errors for a particular problem, a set of logic checking rules which uses the problem description information to analyze the consistency of the student response, and a catch-all set of rules which detects simple syntax errors and misspellings.

The bug catalogue is generated from experience gained in teaching classes on CLIPS and noting various student errors and misconceptions. The advantage of this approach is that, although cumbersome, empirical data such as this can be easily obtained from these classes.[4] Approximately fifteen CLIPS classes have been presented to date to draw upon for information. The bug catalogue consists of a table of expected errors for each token in the student response. For every expected error in the bug catalogue, there exists a corresponding diagnostic message. In this way, a very specific diagnostic message is presented to the student directly applicable to the current problem.

The logic checking error rules are basically inverses of many of the domain rules. These rules analyze the expert skeleton solution and problem description facts and can recognize inconsistencies in the student thought process. For example, if the student had used a variable to represent the price of a pair of shoes in one pattern, and then tried to use that variable to later represent a coat, the error would be caught. Every diagnostic rule has the ability to generate specific error messages for the particular student response being analyzed. The general logic checking rules contain a skeleton error message with appropriate slots to be filled in with diagnostics specific to a particular problem.

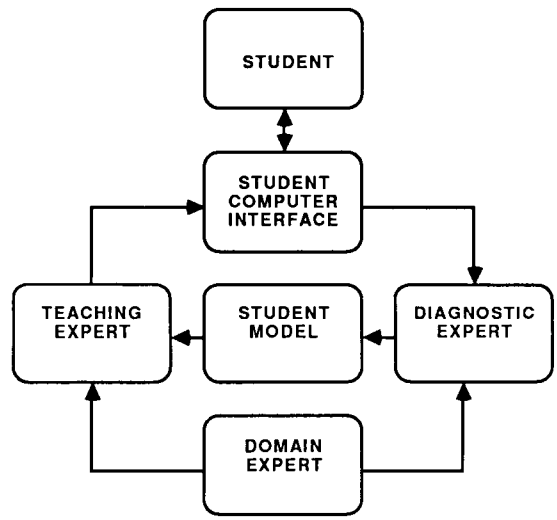

Figure 1. TYPICAL INTELLIGENT TUTORING SYSTEM (ITS) ARCHITECTURE

The third set of diagnostic rules provides a low level of analysis capable of detecting simple syntax errors and misspellings. Their main purpose is to provide a catch-all for errors which have filtered through the tutor and remain undiagnosed. The objective of the knowledge engineer is to minimize the number of times these rules are required to execute by improving the effectiveness of the bug catalogue and logic checking rules.

The error messages provided by the three diagnostic rule sets vary in their levels of specificity. The expected error rule set provides the most specific messages followed by the logic checking rules and finally the catch-all rules. Since the goal of the diagnostic system is to provide the most meaningful diagnosis available for a student response, a hierarchy of priorities among the three diagnostic rule sets was created. Whenever an error is detected, there is the possibility that a rule in all three diagnostic rule sets could be activated. Since the nature of the error messages becomes more general as the priority of the diagnostic rule set decreases, it is important to enable the rule with the best (most specific) message . By implementing this priority system the student is always assured of getting the best diagnosis possible. One advantage of this three level error detection system is that the knowledge engineer can continue to collect expected errors for the bug catalogue as the tutor matures and easily add them to the system at any time. Improving the bug catalogue's ability to detect errors allows the tutor to divert the error analysis from less specific logic checking and catch-all rules to the most specific expected error rules.

TEACHING EXPERT

The teaching expert is a set of specifications of what instructional material the tutor should present and how the material should be presented. All problems in CLIPSIT have a tutoring strategy specified within the problem description facts. Although this approach restricts adapting the problem presentation to more closely match the ability of a particular student, the immediate benefit of less code and faster response time achieved by not having to compute a presentation strategy, is a viable tradeoff on personal computers. There are three basic strategies available to the knowledge engineer in CLIPSIT for presenting a problem. Selection of one of these three strategies depends on the complexity of the problem to be presented and the current lesson.

The first strategy is to provide the student with examples and a list of candidate responses available for use in the solution. Since the first hurdle of the student is determining what the immediate goal is, that is, determining the desired response from the instruction [5], a list of solution components provides the student with adequate goal reinforcement. Additionally, many teachers agree that certain classes of students are competent to solve sets of problems only after an example is done for them. By reinforcing the goal with examples, performance improves significantly. From the standpoint of the diagnostic expert, this approach is also beneficial since it now has a clear understanding of what the correct solution must be. For example, if the student is given a problem description and asked to generate a fact to represent some aspect of the problem, the student could generate many valid representations by choosing endless different names for the same item. By naming the tokens in the problem for the student, a finite and manageable solution set is now available and the student has gained valuable insight by example.

A second strategy for presenting the problem is to use a template. Problems can be presented to the student as a puzzle where the tutor supplies some of the solution pieces and the student supplies the rest. The earlier lessons and problems would provide more pieces of the puzzle than the later problems. The template is very useful, once again, for reinforcing the students understanding of what the problem is really asking for. It also provides a means for the teaching expert to lead the student down the desired solution path, thereby providing for faster and easier error diagnosis.

The third strategy, if it can be called that, is to simply turn the student loose to take whatever approach he or she chooses. Since this method is by far the most difficult to provide diagnostic analysis for, the problem must be worded in such a way that the correct solution can be readily anticipated by the tutor.

All strategies have the benefit of coaching messages generated by the diagnostic rules. A student is given three attempts to provide the anticipated response for the solution. If the student has not determined the correct response by the third attempt, he is given the answer.

STUDENT MODEL

The student model is the representation of the student's understanding of the domain knowledge as perceived by CLIPSIT. The student model is used to assess the student's comprehension of the problem goals and to make decisions about what strategy should be followed to correct any perceived student misconceptions. By comparing the student performance against the expected responses of the tutor, (also known as the "overlay model" [6]), CLIPSIT can determine which teaching goals the student has failed to grasp.

Every problem file in the problem set available to the student contains information about the teaching goals it is trying to present. Certain problems are designated by the problem description facts for each lesson as being either required or remedial. As the student progresses through the required problems, any deficiency inferred by the diagnostic rules is recorded in the student model. On the basis of the student's performance, the system selects the next problem to present. If sufficient deficiencies have been recorded for a particular goal, a remedial problem is immediately presented to the student before the next required problem.

Clancey et al. [7] listed four major information sources for maintaining the student model: a) student performance progress observed by the system; b) direct questions asked of the student; c) assumptions based on the student's learning experience; and d) assumptions based on some measures of the difficulty of the subject matter material. The prototype CLIPSIT student model relies soley on observing past performance. A past performance history of the student's misconceptions is carried over to later lessons. This information is then used to select remedial problems when the student demonstrates misconceptions about goals presented in previous lessons.

USER INTERFACE

The design of the interface can make or break the effectiveness of a tutor, regardless of the clever design of the other components. If the user interface is confusing or non-supportive of the tutored domain, the effectiveness of the instruction will be diminished or lost entirely. A powerful user interface has been developed for CLIPSIT. The interface, written in C and utilizing a commercial graphics package, provides two windows for system text and user input. Window dimensions are controllable by both the tutor and student. Cursor keys or a mouse may be used to scroll the windows. Additional types of pop-up windows are available for menus, student responses, and diagnostic and help messages.

A series of functions are available which pass student responses to those portions of the tutor which handle error detection and instructional strategies. These input/output functions provide a general method for capturing student responses token-by-token, moving the cursor between the windows, permitting limited student editing, writing text to windows, and special message formatting.

SUMMARY

The basic tradeoff which greatly affected the design of CLIPSIT was the issue of response time versus capabilities, that is, the ability of the tutor to respond to a student response fast enough for the student to avoid confusion and frustration versus the amount of flexibility and power that could be implemented. The intent was to create the most powerful teaching package available within a 640k memory constraint and with an average response time of 1-2 seconds. It was felt that a response time greater than 2 seconds between student responses would leave the student confused about what (if anything) was happening with the tutor rather than concentrating on the problem at hand. This fast response time was achieved and even surpassed using a PC/AT class of computer. Response times for the PC/XT class of computers usually met this constraint but would struggle once the student took a solution path much different from the one anticipated.

In order to minimize response time, the key strategy implemented was to represent as much knowledge about the problem as possible in the problem facts rather that make the system use many production rules to reach similar conclusions. For example, each problem file contains facts specifying how the problem should be presented to the student, that is, whether to use a template or not and how much help should be provided. This approach greatly simplifies the teaching expert function. The problem description facts also specify the possible variations to the solution that the student may make. These facts greatly simplify the domain expert.

One of CLIPSIT's greatest strengths is that all production rules are generic in nature. By eliminating problem specific rules, the size of the rule base decreases and typically execution speed improves. All problem specific facts are stored in separate problem files and loaded only when necessary. This minimizes the size of the fact base as well as the number of partial rule instantiations. An additional benefit of storing problem facts in an individual file and using generic rules is maintainability. Separate problem files can be generated by someone unfamiliar with CLIPSIT or rule based expert systems. Additional problems can be easily added, removed, or altered with no changes to the generic rule base.

REFERENCES

1. "CLIPS" is an acronym for "C Language Integrated Production System" and was developed by the Artificial Intelligence Section, Mail Code FM72, NASA/Johnson Space Center, Houston, Tx 77058.

2. Wallach, Brett "Development Strategies for ICAI on Small Computers," in Kearsley, G.P., ed.,Artificial Intelligence and Instruction: Applications and Meathods (Reading, MA : Addison Wesley Publishing Co. ,1987).

3. Brown, J.S., & Burton, R.R., "Diagnostic models for procedural bugs in basic mathematical skills," Cognitive Science, 2, 155-192.

4. Wenger, Etienne, "Basic Issues," Artificial Intelligence and Tutoring Systems, M. Morgan ,ed.,(Los Altos CA : Morgan Kaufmann Publishers, 1987).

5. Matz, M. "Towards a process model for high school algebra errors," in D. Sleeman & J.S. Brown,eds., Intelligent Tutoring Systems (New York : Academic Press 1982).

6. Carr, b., & Goldstein, I.P., "Overlays: A theory of modeling for computer aided instruction," Artificial Intelligence Laboratory Memo 406 (Logo Memo 40), Massachusettes Institute of Technology, 1977.

7. Clancey, W.J., Barnett, J.J., & Cohen, P.R., "Applications-oriented AI research:Education," in A. Barr & E.A. Feigenbaum, eds., The handbook on Artificial Intelligence (Vol 2) (Los Altos, CA : William Kaufmann, 1982).