Research report 114

ITS METHODOLOGY FOR PROBLEM SOLVING AND PROGRAMMING

Robert F Thomas

(RR114)

Abstract

This report outlines a cognitive model of problem solving and programming, which forms the basis of an Intelligent Tutoring System. Within this framework the report takes a critical look at the state of the art and addresses many of the problems that current ITS systems face.

Department of Computer Science
University of Warwick
Coventry CV4 7AL
United Kingdom

December 1987

ITS Methodology for Problem Solving and Programming

Robert F. Thomas

Department of Computer Science

University of Warwick

Coventry CV4 7AL

UK

1. INTRODUCTION

There is an abundance of literature covering the design and construction of Intelligent Tutoring systems, as well as review papers in the field. This report outlines a cognitive model of problem solving and programming, and describes a tutoring system framework which addresses many of the problems that current ITS systems face. Within this framework, the report takes a critical look at the state of the art.

The complexity of issues that exists in the field of computers in education has resulted in a wide variety of different ideas both on teaching methods and how to implement them in a tutoring system. The first section of this report provides an overview of the variety of teaching philosophies that have been embodied in ITS systems, and discusses how these approaches affect the student's learning ability. The second section outlines a general tutoring system architecture that we have developed, and within this framework focusses on important design issues and discusses the contributions of other ITS projects.

2. TEACHING PHILOSOPHIES

The teaching philosophies embodied in tutoring systems range from the traditional CAI "drill and practise" systems to the learning environments such as turtle graphics (Papert 80).Yazdani (85) points out

that all tutoring systems can be placed along a continuum:

Learning Environment -> Coaching System -> Traditional CAI.

Modern tutoring systems that adopt traditional teaching methods, use formal lessons followed by repeated presentation of problems until the lesson content is learnt. They are generally implemented in shallow, rigidly structured knowledge domains, where the presentation of learning material is controlled by a simple algorithm.

Coaching systems bring more flexibility into the lesson. They provide a more informal environment in which the student is given a task requiring certain skills to accomplish it. The coach monitors the student and interrupts, either when the student makes an error, or to explain skills that would be useful in the situation.

Computer Based Learning Systems adopt the Piagetian doctrine of "learning by doing", and provide an environment in which the student can explore domain concepts. Learning about domain concepts is thus a by product of experimenting with the environment.

In order to compare these different approaches, one must look at the effects they have on the student's learning ability.

The most obvious factors that influence learning are motivational. Malone (80) has demonstrated the use of simulation, graphics and games techniques to maintain the interest of the student. Confidence is another important factor. Often students find tutoring systems difficult to understand, both in terms of the dialogue and the lesson material. Coaching systems such as WEST (Burton & Brown 82), which teaches arithmetic skills and WUSOR (Goldstein 82), which teaches statistical reasoning, have demonstrated the advantages of allowing the students to explore and test out their ideas in a learning environment that provides tuition only when it detects student errors. They minimise interruptions while the student is learning, and provide a graphical representation of the teaching domain.

SOPHIE (Burton, Brown & deKleer 82) is an example of a tutoring system that integrates the use of simulation and games techniques into the more sophisticated domain of Electrical Fault Dianosis. SOPHIE teaches troubleshooting skills by inserting a fault into the simulation of an IP-28 regulated power supply, and monitors the student as they try to locate the fault. SOPHIE also has several other teaching modes. The

Workbench allows the student to explore the effects of introducing faulty components on the system, and the GAMES mode allows either two students or two teams (promoting mutual learning) to compete against each other, by alternately generating and locating faults.Both modes allow the student to experiment with the skills they have acquired during the course.

Another important design decision is whether to leave the teacher in the HCI loop, and although ITS systems are designed to be autonomous, it is arguable whether this optimises learning.

3. ITS ARCHITECTURE

The last 20 years of research into teaching systems have focussed on several important design issues that constitute the basis of an Intelligent Tutoring System. This section discusses these issues in the context of a general ITS architecture.

The architecture of an ITS system can be broken into four interacting modules as follows:

(1) User model

(2) Control mechanism

(3) Tutoring strategies

(4) User Interface.

3.1. User Model

3.1.1. Current Approaches

The user model is the tutoring systems representation of the student it is interacting with. This model is built using data that the program elicits from the student and is used to control the tutoring module when choosing an appropriate teaching strategy. This provides the basis for an optimal method of correcting a student's mistakes. However it is the most important and also the most difficult module to design, because of the difficulty of gathering and interpreting student data. It is important, when developing both the user model and teaching strategies, to have a psychological theory, which can provide guidelines for collecting and interpreting student data, as well as representing and imparting expertise and knowledge (Anderson, Boyle, Farrell & Reiser 84). Work to date has focussed on three ways to model the student.

(1) Expert model

(2) Bug model

(3) Genetic graph

The expert model represents the student as a subset of the expert knowledge base. Thus student mistakes are interpreted as being caused by a lack of expert skills or knowledge. WEST (Burton & Brown 78) is a coaching system for a statistical reasoning game "how the West was won". The system monitors the student's progress through the game by comparing their performance with the computer expert. Issue recognisers spot any discrepencies and represent these in terms of missing skills. The approriate tutorial action is then taken. The main criticism of this approach is that errors cannot be attributed solely to the lack of prerequisite skills.

The bug model represents the student in terms of " buggy" behaviour. DEBUGGY (Burton & Brown 82) contains a bug database that picks up a student's inappropriate answers, and hypothesises about the type of error that the student has made. Once one particular type of bug has been isolated, the appropriate remedial action is taken. This approach has been implemented in the domain of acquiring simple arithmetic skills, but within this relatively simple domain, the presence of several interacting bugs produces complex student behaviour which is difficult to interpret. To reduce this interaction problem in more complex domains, the bugs can be interpreted on different conceptual levels, and domain knowledge and skills can be taught in a more modular way.

Goldstein (82) developed the concept of a genetic graph, which provides a history of the student's progress as they develop expertise. The genetic graph was used in WUSOR I,II, and III to model the students acquisition of probabilistic skills in a maze game called WUMPUS. The graph is a network of rules connected by links. The rules specify the procedural knowledge embodied in the domain, and are connected according to their locality and relationship in the problem domain. As implemented in WUSOR II, the graph contains four types of relationship. Generalisation/specialisation, analogy, deviation/correction, simplification/refinement. Each relationship provides a different way of tutoring the same rule, and gives the system the flexibility, firstly of being able to monitor the student's progress through the graph, and secondly to have a choice of several explanation strategies at any stage on the graph.

On a more general level, the development of these user models has highlighted several problems that must be addressed. Primarily the need for both a long term record of the student's performance, to indicate the level of skill attained, and provide a steady baseline of performance. Also a more detailed short term model, which models the student's cognitive processes, and thus provides a more accurate picture of performance.

3.1.2. Model for problem solving and programming

In the domain of teaching program design, problem decomposition and PASCAL programming skills, a psychological model, based on protocols and informal observation has been developed (Craig & Thomas 87). Diagram (3) shows the outline of the model.

(1) Long term student model (history)

(2) Short term model: 3 levels - a) Mental Models b) Problem decomposition c) Domain Knowledge

The long term model keeps a record of the students progress over the course of sessions, and builds up a general model of overall performance. This is useful to provide a baseline to refer to during a session, where due to short term factors (temporary illness, boredom etc..), the current performance of the student may be unrepresentative. It also provides an indication of both the students knowledge and experience of the problem domain. DEBUGGY (Burton and Brown 82) was developed to look at this problem of short term perturbations. DEBUGGY contains a model of the "noise" created by factors such as cognitive overload, boredom etc.. It applies this model to it's student model, to provide a more flexible model within which can account for short term fluctuations in performance.

3.1.2.1. The short term model

This is used to evaluate the student's ability to understand the current concept being taught. There is a hierarchy of three levels of student understanding, which, when combined indicate the appropriate tutorial action to take.

3.1.2.2. Mental Models

Mental Models refer to the students conceptual model of the idea or problem that is presented. For a

novice, this usually takes the form of an analogy, for example the use of water as an analogy for electricity, or in programming, conceptualising variable assignments as pigeon holes. As the novice progresses, the model becomes more complex, as the student learns more features that must be incorporated.

The following two examples highlight the importance both of a psychological theory in which to base the user model, and the need for a hierarchical user model, to accurately understand the depth of student misconceptions.

WHY (Stevens A., Collins A., Goldin S.E. 82) was developed to teach students the "causal model" underlying the climate of Oregon or Ireland. The system model was based on tutoring dialogues and experiments, and provides an interesting parallel to the results we have observed with novices learning pascal concepts. They isolated 16 common "conceptual bugs" that students had in this domain, which can be interpreted either as being due to a mental model of insufficient complexity, or an inappropriate analogy. As an example, one conceptual bug is the "squeezing causes condensation" bug. The student thinks that putting pressure on the air mass will cause condensation. The reason for this bug is that students often use the analogy of thinking that an air mass is like a sponge that soaks up water. When tutoring the student, it is important to repair the student's model at the right level, in order to minimise confusion.

During the development of SOPHIE I, Burton, Brown and deKleer distinguish between several types of knowledge that an expert uses to troubleshoot a faulty component:

(a) Knowledge of the overall functional organisation of the device.

(b) Strategies to choose next measurement to reduce hypothesis space.

(c) Tactics concerning the ease of making measurements.

(d) Understanding of electronic laws, and circuit components.

It can be seen that the types of knowledge from (a) to (d) follow the hierarchical model of knowledge representation outlined above. However the authors do not encompass these ideas into any psychological model, to aid learning. Indeed in their review paper of SOPHIE I,II, and III, they suggested two areas in which the design could be improved. Firstly, that an expert model of understanding and a learning model should have been developed in the initial stages, to aid the design process. Secondly that there was a need

for more levels of description, when explaining to a novice an expert's troubleshooting procedure.

3.1.2.3. Problem Decomposition

Problem decomposition refers to the ability of the student to take a problem and break it down in such a way that it can be solved using the skills and domain knowledge that they have acquired. The important aspects of human problem solving are problem representation, goal decomposition and goal hierarchy (Anderson 84).

The problem representation is the student's "mental model" or conceptualisation of the problem. It is vital that this mental model is correct, because it provides the state space in which the problem is manipulated, and also defines the operators used in the problem. In the area of PASCAL programming, Craig & Thomas (87) found that the wording of a problem was an important factor governing the student's ability to represent the problem as a mental model. In particular the way the problem was defined had a strong influence on the operators that the student's used to solve the problem, both in terms of subgoals and PASCAL concepts. A more detailed analysis of results, and an outline of the theory are given in the technical report.

Goal decomposition refers to the breakdown of a problem into smaller, independant subproblems. This is a top down process, which creates a hierarchy of subgoals. When an expert decomposes a problem, s/he uses two important strategies. The first is to judge the independance of subgoals, and the second is to pick out the most important subgoals and decompose these first. These are important because they ensure that any refinements that have to be made to the goal hierarchy are kept to a minimum.

SPADE (Miller 78) is an example of a tutoring system that teaches problem solving skills. It provides a planning and debugging environment in which to develop turtle graphics programs. The psychological model on which the system is based has evolved from the analysis of student protocols and suggests five stages of program development.

(a) Problem understanding

(b) Procedure defining

(c) Procedure testing

(d) Localising bugs

(e) Repairing bugs

During the planning stage the student uses a list of defined operators to build a plan representation of the problem, which is displayed as a graph. The student is encouraged to take a top down, left to right problem solving method, but can choose to leave subgoals until a later stage (which is often the case).

SPADE diagnoses errors using four sources of evidence.

(a) Warnings that the student has put into the goal boxes

(b) Problem description checked (to ensure that code has been written to achieve each subgoal)

(c) Trace running process (eg to check if previously accomplished steps have been undone etc..)

(d) code scanned for unusual patterns

The SPADE system highlights several problems that occur when designing a system to teach problem solving and correct students mistakes. The use of a planning language means that the student has to learn the syntax and semantics of this, as well as the coding language. Although SPADE is a programming environment rather than a tutoring system, it outlines the difficulty of assessing both the students problem solving techniques and their plans. Miller points out that because SPADE-0 does not use any information in the problem description, it is severely restricted in it's ability to provide advice on goal decomposition and debugging, despite "better human engineering, extending the collection of plan types, or adding new features to the framework".

A very important aspect of an expert's problem solving strategies is that they distinguish between the importance of subgoals. The hardest subgoals are tackled first, to minimise any reorganisation that may be required during subsequent plan refinement.

PROUST (Johnson & Soloway 85) is a commercially available system that finds non syntactic bugs in PASCAL programs. Once found, it offers advice on correcting the bug, and suggests how the bug arose. The system uses a problem description, which the human tutor types in, using a problem description language.

Proust contains three levels of knowledge

(1) Problem Description

(2) Programming Knowledge (a) Goals (b) Plans

(3) Bug Library

The system contains a number of problem descriptions, in the form of pseudo code. When a student has written a syntactically correct program as a solution to a question, Proust initially parses the program. It then takes the problem description it has been given for the program, and taking each subgoal at a time, compares the students subgoal solution to a goal frame, and a plan frame. The goal frame contains information about the particular subgoal, along with a list of the plan frames that could be used to achieve the goal. It tries to match each of the plan frames with the student's code segment, and if they match, no bugs are present. If no match can be found, PROUST tries to interpret differences between the template plan and the student's code in terms of bugs. It contains a number of bug rules which are activated under certain conditions, and generate an English explanation of the error. Experimental results for PROUST show that on a simple program (the rainfall problem) it gave good analysis results (fully analysing 75% of programs).However taken in context, this problem contains only 3 subgoals, and all the programs that PROUST was given were syntactically correct.

Although PROUST has been developed to provide the basis for a PASCAL tutoring system there are several issues which it does not address:

(1) The only way it has of understanding the student is the syntactically correct PASCAL program. This provides little information on the conceptual level of understanding either about PASCAL concepts or problem understanding and program design.

(2) PROUST analyses student programs in terms of the subgoal breakdown that it has for the problem. All but the most trivial problems have several possible goal structures. PROUST enforces one particular way of decomposing a problem rather than allowing the student to develop their own equally valid solutions.

(3) PROUST uses the subgoal structure that the tutor typed in to analyse the student's programs, and it

has been demonstrated, during PROUST's evaluation (Johnson 86), that the greater the difference between the student's plan structure and PROUST's, the less it was able to make any analysis of the student's program. A study of novice programmers (Craig & Thomas 87) has outlined several areas where PROUST would have difficulty. (a) Novices have great difficulty with syntax. (b) Novices do not develop a plan, but often type straight into the computer, not understanding what they are doing. (c) Novices often misinterpret or do not understand the question they are trying to solve.

3.1.2.4. Domain Model

Domain knowledge is the amount of context related knowledge that the student has memorised. For example the syntax of a computer language. This is the bottom level of the learning process in the sense that the student understands the area on a conceptual level, and can solve the problems, but lacks specific details about the domain.

3.2. Control Mechanism

The control mechanism and the user model are the vital parts of the tutoring framework. The control mechanism accesses the user model in order to pinpoint at what level the student is experiencing problems, and then to use this information to choose a suitable teaching strategy. An important design consideration is to have a fast control structure, which quickly analyses the user model, and decides upon an appropriate strategy. The two main problems that occur with current ITS systems are speed and complexity. With many representations, the search space that must be traversed to find an appropriate response, is enormous. when

PROUST uses a frame based approach, where a plan structure is hypothesised at the top level, and depending on how slots in the plan are filled, lower level frames are activated accordingly. As outlined above, PROUST has a 3 level control structure. WUSOR uses a semantic net representation to embody the genetic graph, and tutoring strategies are selected as the network is traversed. Although it is useful in simple domains, it becomes unwieldy in more complex areas.

3.3. Teaching Strategies

The teaching strategy affects both the efficiency and quality of student learning. The most important issues are :

(1) When to interrupt

(2) What tutoring role to adopt.

3.3.1. Knowing when to interrupt

Knowing when to interrupt has a strong effect on student. Too many interuptions and the learning process is disrupted as well as the student becoming demotivated. Too few and the same effects occur. WEST (Burton & Brown 82) contains tutoring heuristics, which tell the coach when to interrupt and what to interrupt with. These heuristics are in the form of a principle hierarchy, which calls up these rules of thumb when conditions fire.

3.3.2. Tutoring roles

A tutoring system needs to provide a comprehensive range of tutoring strategies, which address different levels of understanding, and also provides different ways of teaching a particular concept. Extending the user model that has been outlined, diagram (4) outlines the different strategies that may be used.

(1) Mental models; Repair analogy; Propose new analogy

(2) Problem solving; Correct problem representation; Correct problem decomposition; Correct goal hierarchy

(3) Domain knowledge; Teach domain knowledge; Correct domain knowledge.

In some cases these can be integrated, and the following systems demonstrate the tutoring strategies that have been implemented and tested.

Woolf & McDonald (78,84) developed MENO-Tutor to use as a generic research tool experimenting with tutoring rules. It has been tried in several different domains, including understanding weather and PASCAL programming. The system's tutoring component consists of a hierarchy of decision units, on three abstraction levels.

(1) Pedagogic states

(2) Strategic states

(3) Tactical states

The top level decides on the choice of approach, which would include when to interrupt the student, and when to introduce a new topic, test the students knowledge, or teach. The second level refines the top level decision into a particular strategy. For example, the "tutor state" may produce an "explore competency", "teach data" or "describe domain" strategy. The bottom level produces tactics or courses of action, which are used to guide the interaction. The "describe domain" strategy could be implemented as one of five courses of action.

(a) Describe general knowledge

(b) Describe specific knowledge

(c) Describe dependant knowledge

(d) Propose analogy

(e) Suggest example

During the lesson, the control structure is continually changing, at all levels of the hierarchy. Once a course of action has been decided upon, it is sent to the language generator module where the appropriate response is generated.

WUSOR contains four variations on an explanation, one of which is chosen on the basis of general teaching heuristics, such as "vary your explanation" and "avoid strategies that have been consistently unsuccessful in the past". The four explanation types are similar to those described in MENO.

(a) Explanation

(b) Generalisation

(c) Analogy

(d) Refinement

3.4. User Interface

The user interface provides an important way of presenting information to the student. The computer has the advantage over traditional teaching mediums in it's flexibility of offering complex graphics and text interfaces. LOGO and turtle graphics have demonstrated the power of providing an environment which the student can quickly become familiar with (Papert 80).

In the domain of program design and problem solving, the use of graphics, to show the modularity and hierarchical structure, both of a problem solution and a program structure, are powerful teaching aids for problem solving and program design. SPADE is a good example of both the advantages created by using graphics, and the problems associated with unrestricted user input. SPADE allows the user to create a hierarchy of planning boxes on the screen, which is a useful way to show the development of a goal hierarchy. However, once the student has specified the type of box, they are free to describe it's exact purpose by typing in text. As a result SPADE cannot interpret their comments, in order to "understand" the purpose of the goal boxes, and thus cannot understand the students problem solving strategy.

There is a trade off between the freedom given to a student to express the interaction in English, and the complexity of the parser needed to analyse the dialogue, and also the size of the model needed to interpret the results (Carbonell 70). Increasing the size of the vocabulary also magnifies the opportunity for errors, misunderstandings and ambiguities. This problem is particularly acute for CBLS systems which try not to restrain the student's communication.

4. SUMMARY AND CONCLUSIONS

This review has provided a general framework for an Intelligent Tutoring System on two levels. Firstly it addresses the general educational approaches that have been embodied in the tutoring systems, and their effect on student learning. Secondly, it has provided a modular approach to the design of an ITS, outlining important issues, and problems highlighted by current systems. This has been with particular reference to the domain of program design and PASCAL programming. This paper provides the basis for the development of a PASCAL and program design tutor, which aims to integrate a sound psychological learning model with a comprehensive tutoring model and a sophisticated graphics interface.

REFERENCES

Anderson J.R. (84) Cognitive psychology and its implications Academic Press, London.

Anderson J.R., Boyle C.F., Farrell R.G., Reiser B.J. (84). Cognitive principles in the design of computer tutors. proc. sixth annual conference of the Cognitive Science Society, Boulder Co.

Barr A., Feigenbaum E. (80) The handbook of Artificial Intelligence v2, chap. 9, W.H.Freeman San Francisco

Burton R.R. (82) Diagnosing bugs in a simple procedural skill. *In* Sleeman D., Brown J.S. (82)

Brown J.S., Burton R.R., de Kleer J. (82) Pedagogical, natural language and knowledge engineering techniques in SOPHIE I,II & III. *In* Sleeman D.,Brown J.S. (82)

Carbonell J. (70) A.I. approaches to C.A.I. I.E.E.E. trans. man machine systems.

Dede C. (86) A review and synthesis of recent research in intelligent computer-assisted instruction Int. J. Man-Machine Studies 24, 329-353

Ford L. (84) Intelligent computer aided instruction. See Yazdani M., Narayaman A. (84)

Goldstein I.P. (82) The genetic graph: a representation for the evolution of procedural knowledge *In* Sleeman D., Brown J.S. (82)

Green T.R.G., Payne S.J., Van der Veer G.C. (ed) (83) The psychology of computer use, Academic Press, London.

Johnson W.L. (86) Intention-based diagnosis of novice programming errors, Pitman, London.

Malone T.W. (81) Towards a theory of intrinsically motivating instruction, Cog. Science 4, 336-369

Miller M.L. (78) A structured planning and debugging environment for elementary programming Int. J. Man-Machine Studies 11, 79-95

Papert S. (80) Mindstorms - Children, computers and powerful ideas Harvester Press

Sleeman D., Brown J.S. (ed) (82) Intelligent Tutoring Systems Academic Press, London.

Woolf B., McDonald D.D. (84) Building a computer tutor : design issues I.E.E.E.- Computer Sept. 1984

Yazdani M., Narayaman A. (84) A.I. and human effects Ellis Horwood Ltd.

Yazdani M. (86) Intelligent tutoring systems survey A.I. Review Vol. 1, 43 - 52.