

**PERTUTOR: AN INTELLIGENT TUTORING SYSTEM  
IN THE DOMAIN OF PERCENTAGES**

by

**LILLIANA SANCHO-CHAVARRIA**

**A THESIS**

**Presented to the Department of Computer  
and Information Science  
and the Graduate School of the University of Oregon  
in partial fulfillment of the requirements  
for the degree of  
Master of Science**

**December 1991**

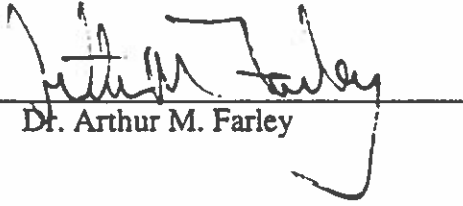
APPROVED:

Arthur M. Farley  
Dr. Arthur M. Farley

An Abstract of the Thesis of  
Lilliana Sancho-Chavarría for the degree of Master of Science  
in the Department of Computer and Information Science  
to be taken December 1991

Title: PERTUTOR: AN INTELLIGENT TUTORING SYSTEM IN THE DOMAIN OF  
PERCENTAGES

Approved: \_\_\_\_\_

  
Dr. Arthur M. Farley

The focus of this study is on a tutorial system in the domain of percentages. We present PerTutor, a system developed under the framework of Intelligent Tutoring Systems. The purpose of the system is to provide reasonable explanations and guidance and to adapt the level of difficulty of problems to each individual's knowledge about percentages. PerTutor is comprised of four main modules: the expert, the student model, the tutor, and the user interface. It relies on previous tutorial interactions to determine both the explanations and the level of difficulty of the problems that will be presented next. An initial implementation of the system demonstrates that PerTutor exhibits some degree of flexibility and intelligence. It generates problems of different levels of difficulty, identifies different ways in which students can approach a problem, and responds according to each situation. Improvements for a future version of the system are also discussed.

## VITA

NAME OF AUTHOR: Lilliana Sancho-Chavarría

PLACE OF BIRTH: Cartago, Costa Rica

## GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon  
Instituto Tecnológico de Costa Rica

## DEGREES AWARDED:

Master of Science, 1991, University of Oregon  
Bachelor's Degree in Computer Science, 1987, Instituto Tecnológico de Costa Rica

## AREAS OF SPECIAL INTEREST:

Artificial Intelligence  
User Interfaces  
Computers in Education

## PROFESSIONAL EXPERIENCE:

Instructor, Department of Computer Science, Instituto Tecnológico de Costa Rica,  
Costa Rica, 1986-1989

Instructor, Instituto Latinoamericano de Computación, Costa Rica, 1987-1989

Computer Consultant, Private Practice, Costa Rica, 1988-1989

## AWARDS AND HONORS:

Fulbright-LASPAU Scholarship, 1989-1991

## PUBLICATIONS:

Aguero, U. & Sancho, L. (1990). La computadora como herramienta educativa.  
Tecnología en Marcha. 9(4), 3-7.

## ACKNOWLEDGMENTS

I wish to express my sincere gratitude to my adviser, Dr. Arthur Farley, for his continuous support, assistance, and encouragement throughout the development of this thesis.

I would also like to thank the Fulbright-LASPAU organization, without their support my Master's program would have been only a dream. I would like to extend my appreciation to my adviser at LASPAU, Mrs. Julie Leitman, for her assistance and her willingness to help.

I am also grateful to my family, especially my mother, for teaching me the value of education.

Finally, I would like to thank my friends in Eugene, for the support that they have given me, especially during the last stages of this project.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION .....	1
The Domain of Percentages .....	2
Existing Software for the Instruction of Percentages .....	3
II. OVERVIEW OF THE USE OF COMPUTERS IN EDUCATION .....	7
Evolution of Computer Assisted Instruction .....	7
Intelligent Tutoring Systems .....	10
III. A PERCENTAGE TUTOR.....	16
The Expert Module .....	16
The Student Model .....	25
The Tutor .....	26
The User Interface .....	30
IV. EXAMPLES FROM AN INITIAL VERSION OF THE SYSTEM .....	35
V. DISCUSSION .....	46
APPENDIX: SOME DETAILS ABOUT THE IMPLEMENTATION.....	50
REFERENCES .....	57

## LIST OF TABLES

Table	Page
1. Examples of Different Types of Problems.....	18
2. Rules for the Generation of the Percent and the Total .....	20
3. Equations to Solve Percentages Problems .....	23
4. Rules for the Level of the Next Problem .....	27
5. Specific Tutorial Rules by Type of Error and Number of Trial when It Is the First Time the Error Occurs in the Session .....	28
6. Specific Tutorial Rules by Type of Error and Number of Trial when the Error Has Already Occurred in the Session .....	29
7. Examples of Possible Operations or Clauses for Problems of Type I ....	53

## LIST OF FIGURES

Figure	Page
1. An Example of a SUPERPILOT Program .....	8
2. Components of an Intelligent Tutoring System .....	12
3. A View of the User Interface .....	31
4. The File Menu .....	32
5. The Edit Menu .....	33
6. The Special Menu .....	33
7. The Windows Menu .....	34
8. User Interface After Problem Statement Has Been Given .....	35
9. Student Gave the Correct Solution to the Problem in the First Trial.....	36
10. Use of the Operations Window .....	37
11. Student Is Missing One Step .....	37
12. Student Computed the Missing Step and Gave the Correct Answer.....	38
13. Tutor Determines that Steps Do Not Correspond to Any Partial or Complete Solution of the Problem .....	39
14. Tutor Determines that Steps Do Not Correspond to Any Partial or Complete Solution of the Problem (Continued).....	39
15. End of the Tutor's Explanation for Example 3.....	40
16. Student Gives Correct Answer after Failing First Attempt.....	40
17. Tutor Determines that Steps Correspond Neither to Partial Nor to Complete Solution of the Problem (Short Explanation).....	41
18. Tutor Determines that Steps Correspond Neither to Partial Nor to Complete Solution of the Problem (Short Explanation Continued).....	42
19. Student Divides Rather than Multiplying.....	43



	Page
20. Student Corrects the Buggy Procedure, and Multiplies Instead of Dividing .....	43
21. Student Does the Same Error, Divides Instead of Multiplying .....	44
22. Tutor Bases Explanation on Previous Problem .....	44
23. Student Solves Problem after Tutor's Reference to Previous Problem....	45
24. Elastic Ruler for Percentages .....	48
25. A General Data Flow in PerTutor .....	50
26. Student's Interaction in a Problem .....	54
27. A Trial Interaction .....	55

## CHAPTER I

### INTRODUCTION

Percentages have become a necessary concept with which to understand modern society. Nowadays, it is common to hear about the percentage of unemployment, percentage of taxes, percentage of people in agreement or disagreement on a certain issue, percentage of discount, the percentage of inflation, and even the percentage of cotton in a T-shirt.

Although the understanding of percentages seems to have become important, research has shown that many students have little comprehension of problems that involve percentages. In 1978, the National Assessment of Educational Progress reported that only about 50% of seventeen-year-olds and about 33% of the thirteen-year-olds could convert  $\frac{9}{100}$  to a percentage, and also that only about 50% of the seventeen-year-olds and about 20% of the thirteen-year-olds could state the percentage of games won by a team that won five of twenty games (Wiebe, 1986). In a study of 669 first and second-year algebra students, Blohm and Wiebe (1980) found that the average score of students when asked to solve problems in the form "A% of B =" and "B - (A% of B) =" was only 32%.

These results suggest that the traditional methods for teaching percentages are not effective. Glatzer (1984) claims that the problem is that the emphasis is placed on mechanical procedures and the concept and applications of percentages are very often left out. However, the above results also suggest that there has been little success in teaching the mechanical procedures to solve percentage problems, and it is clear that students have difficulty in understanding how to solve the equations for percentage problems. Therefore,

not only the concept of percentage needs to be emphasized but also the methods by which percentage exercises are solved.

Most teaching methods present the material uniformly to all students without considering individual differences and paces of learning. This usually happens as a result of the amount of students per teacher. The lack of individual instruction might also affect the student's understanding of a subject. Computers, on the other hand, can be viewed as instruments that can help teachers to fill the gap of individualized learning. The claim is that computer tutors have the potential to provide instruction that is dynamically adapted to the learner (Ohlsson, 1987).

This study presents a proposal for an intelligent tutorial system for percentage problems, which we have called PerTutor. It consists of a system that gives reasonable explanations and guidance, adapting the level of difficulty of the exercises to each individual's degree of knowledge about percentages. Description of an initial implemented version of the system is also presented.

### The Domain of Percentages

Percentage problems usually deal with finding one of three variables, either the percentage, the part, or the whole (total); consequently, we can identify three basic types of percentage problems. Type I problems seem to be the most common of all three. In Type I problems both the percent and the total are given, and the student is asked to obtain the amount of the part; for instance, what is 30% of 50 ? In Type II problems the percent and the percentage are given, and the student is asked for the total; for instance, 4 is 10% of what ? Finally, in Type III problems the percentage and the total are given, and the student is asked for the percent; for instance, what percent is 15 of 50 ? Further types of problems

can be created by dealing with parts directly and their effects on the total amount (e.g. mixture problems).

In order to solve a basic percentage problem, a student must identify the type of percentage problem and apply the corresponding equation. However, there is more than the solution to the equations involved in the understanding of the concept of percentage. Students should also be able to understand when a result actually makes sense.

In addition to the three basic types of problems, we could consider a fourth type that can give the student a broader view for the understanding of the concept of percentages. Type IV problems involve situations in which the student is required to do an addition or subtraction after the percentage has been computed. The following is a typical situation. The store has a bicycle on sale at 20% off. If the original price of the bicycle was \$90, how much would it cost now ? Notice that the level of difficulty of the problems increases from Type I to Type IV problems.

Textbooks typically present percentages as follows: the concept of percentage, changing fractions and decimals to percentages, changing percentages to decimals and fractions, finding percentages, and solving percentage problems and applications. An alternative approach to teaching percentages is to present problems in sequence from Type I problems to Type IV problems, providing the appropriate guidance when the student commits mistakes, and advancing according to the level of knowledge of the student. It is this approach we will investigate with one tutorial system.

#### Existing Software for the Instruction of Percentages

Most of the existing software for the instruction of percentages falls into the category of drill and practice (see chapter II). Three recent percentage programs were

studied: MEEC's Automotive Math II, Gamco's Percent, and QED's Proportions and Percents.

Automotive Math II is a program for the instruction of decimals and percents. For the instruction of percents, the system presents a main menu from where the student chooses to see examples, to do a practice, or to do a test. Both the examples and practice parts are divided into sections for multiplication (type I), division (type II), or mixed operation (type III) exercises. In the examples section, the system displays a problem statement and then a step by step solution of the problem. In the practice section, the maximum number of exercises is pre-established, but the student has the option of choosing a number of exercises less than the maximum amount. All data in the program is generated randomly, so the system has no control on the level of difficulty of the problems being generated. In the practice section, a problem statement is given, and the student has to type in the final answer to the problem. The system requires a very precise response in order to identify it as a correct answer; in other words, the student has to be careful with the amount of decimal places included in the final answer. If the student did not give the correct answer, the system will ask the student to give data for each of the steps involved in the solution of the problem. In the case that the student can not solve the problem, the system will give the solution to it.

Gamco's Percent also has one section for each type of problem. Each section consists of ten problems. The data is generated randomly, but with certain rules to generate problems of different levels of difficulty. For instance, let's consider the distribution of data for Type I problems. In 20% of the problems of Type I, the percent is a whole number from 10% to 99%. In 10% of the problems, the percent is a whole number less than 10%. In 30% of the problems, the percent is a whole number greater than 99. In 10% of the problems, the percent is equivalent to  $\frac{1}{3}$ ,  $\frac{2}{3}$ ,  $\frac{1}{6}$ , or  $\frac{5}{6}$ . In 10% of the problems, the percent is a mixed number less than 100%. And in 20% of the problems, the percent is

less than 100% and has one or two decimal places. Similar distributions are used for the other types of problems. Although this program makes an effort to take into account the level of difficulty of the problems, the program shows a great amount of randomness, and does not tailor the level of difficulty of the problems to the student's needs.

In Gamco's Percent, the student gives the solution to the problem by typing the final answer. If the answer is incorrect, the system will give one more trial to student; if still the student did not give the correct answer, the system will display the answer and require the student to type it in. This program does not provide tutoring when the student makes mistakes, instead it provides a help option which the student can select to obtain a general explanation about percentages.

QED's Proportions and Percents is the largest of the three systems, and provides more guidance than the other two systems. It contains 24 sections, and the level of difficulty increases from section 1 to section 24. The first sections provide exercises in the domain of proportions as an introduction to the topic of percentages. Some other sections contain exercises by type of problem and by common situations where percentages can be used. For example, section 10 consists of exercises that include discounts and taxes. Students can work in any section. If the student makes many mistakes in a section, the system would recommend a change to a previous section; it is the student's choice to do so. Explanations in this program are very elaborate, and solutions are given step by step. The general quality of the system is good; however, as in other percentages systems, the data is generated completely at random, and the system does not take into account the student's procedure to solve the problem.

In general, the systems studied exhibit some degree of flexibility by allowing the students to choose the type of problems they want to solve; however, all interactions in the systems are pre-established. QED's Proportions and Percents was considered superior to the other two programs. All of the systems studied disregard the procedure used by the

student when trying to solve a problem, which is a critical factor in determining the kind of assistance required by the student. Additionally, the systems can not identify partially correct solutions; that is, they can only determine if the solution was right or wrong. Another drawback of these systems is that they generate the data completely at random, and do not take into account that the level of difficulty of the problems varies depending on the data; consequently, the level of difficulty of the problems is not adjusted according to the perceived student state of knowledge about percentages. It is clear that the amount of domain knowledge available to these systems is very limited.

The rest of this document describes PerTutor, a tutorial system for the instruction of percentages. The system will attempt to solve some of the problems that often appear in current educational software. Chapter II presents a brief overview of the use of computers in education and discusses Intelligent Tutoring Systems as the framework used to develop the proposed system. Chapter III presents the design of the system. Chapter IV provides example interactions with the initial implemented version of the system. Finally, chapter V evaluates the system, and discusses some issues for future research in the field. An Appendix presents details of data structures used to implement the percentage tutoring system.

## CHAPTER II

### OVERVIEW OF THE USE OF COMPUTERS IN EDUCATION

#### Evolution of Computer Assisted Instruction

Computers have been thought to be useful for education since the 1950's (Yasdani, 1987). Computer Assisted Instruction (CAI) started as drill and practice programs, which led the students through a linear path and presented and evaluated the same material. The purpose of these systems was to reinforce knowledge. However, students could advance through the material by trial and error without obtaining enough feedback and without learning. Drill and practice systems were too rigid and provided little evidence of the advantage of using computers in education.

In the 1960's, the approach was to use the student's response to control the material that would be shown next. This idea led to the development of branching programs, which would contain a limited number of pre-established scripts that would be triggered by the student's responses. The main benefit from this idea was that programs would display at least some degree of flexibility. However, programming these branching scripts was a very laborious task. As an attempt to provide teachers with a simple programming language to develop educational software, authoring languages emerged (Gagné, Wager, & Rojas, 1981). SUPERPILOT is an example of an authoring language. It has instructions to present text, accept responses, perform computations, display special effects, and control program flow. Figure 1 presents an extract from a SUPERPILOT program. The instruction T: is used to display text, the instruction A: is used to accept the student's



responses, the instruction M: is used to match the teacher's response with the student's response, and the control instruction J: is used to branch to various places within the program, depending on the student's answer. The program begins by asking the student about the amount of planets in the Solar System. If the student responds 9, which is the correct answer, the program will say that it is the correct answer and will branch to the third segment. Otherwise, the program will branch to the second segment where the program explains that there are 9 planets and asks the student if she knows their names. If the student knows the names, the system will branch to the fourth segment; otherwise, it will say the names, and then will branch to the fourth segment. The rest of the program operates in a similar fashion.

```

* first
T: How many planets are there in the Solar System ?
A:
M: 9
JN: second
T: Correct! There are 9 planets in the Solar System.
J: third

*second
T: No, there are 9 planets in the Solar System.
T: Do you know their names ?
A:
TN: Their names are: Mercury, Venus, Earth, Mars, Jupiter, Saturn,
    Uranous, Neptune and Pluto.
JN: fourth

*third
T: Which is the biggest planet ?
A:
M: %Jupiter%

```

Figure 1. An example of SUPERPILOT program.

SUPERPILOT also provides instructions that support graphics and audio. The graphics editor is quite rudimentary, and it takes a long time to prepare the graphics.

Although it might be easier to learn and use than regular programming languages, SUPERPILOT requires a significant level of skill from the teacher for it is actually a non-structured programming language. The idea of providing teachers with tools to develop instructional interactions was good; however, similar problems to the ones encountered in SUPERPILOT are also found in other authoring languages. Most of these problems are due to fact that the computer technology of the 1960's did not facilitate the development of educational software.

In the 1970's, generative systems evolved (Woods & Hartley, 1971; Sleeman & Brown, 1982). These systems had the capacity of generating some of the teaching material. Woods and Hartley proposed models to generate teaching material for arithmetic computation tasks; particularly addition. They first developed an experimental model of task difficulty, which would be used for the generation of the examples. They found that the probability of successfully adding a column, regardless of the number of columns that form the example, varies both with the size of numbers and with the number of numbers to be added. From the model, they derived a formula to express the effects of number of rows and size of numbers on the probability of success and time taken to complete an example. The formula is used by the computer to extrapolate and interpolate from experimental data and generate examples at other difficulty levels. Thus, the computer would generate examples for a pupil to work at a specified level of success.

Since the idea was to use the computer for individualized instruction, an important issue was how to adjust the level of difficulty of the examples to the student's competence. This problem was approached by allowing the computer to operate in an optimistic or pessimistic mode. When the level of success of the student is greater than the specified probability of success, the program becomes optimistic and gives a greater weight to past performances at lower difficulty levels (where the student should have been more successful). The pessimistic mode, on the other hand, operates in an opposite way. The

computer would have to readjust continually the limits of the difficulty class in which the student was working in order to generate suitable problems. Generative systems are also called adaptive systems because they were devised to select problems at the level of difficulty appropriate to the student's performance.

As the above example shows, generative systems could actually function for personalized instructional purposes. Their strength is in their capacity to generate teaching material, and they have been mostly used in the domain of arithmetic. The application of this technique in other domains could be more difficult because randomly generated data might produce incoherent situations. In addition to that, most of the criteria they follow to provide individualized learning is based on parametric summaries rather than on explicit representation of the student's state of knowledge in the domain (Sleeman & Brown, 1982).

By 1980, the research community seemed to agree that computer-based instructional systems must exhibit some degree of intelligence. The notion of knowledge became central to the development of instructional systems. It was deemed for the systems to show flexibility and adapt to the needs of individual students as a human tutor would. At this point, research in computer-based instructional systems bonded to the field of Artificial Intelligence because the representation of knowledge within intelligent systems was a common interest. Intelligent Tutoring Systems emerged as an attempt to deal with the shortcomings of generative systems (Sleeman & Brown, 1982; Yasdani, 1987).

### Intelligent Tutoring Systems

Intelligent Tutoring Systems go a step beyond traditional CAI (Burns & Capps, 1988). In traditional CAI, both the specific domain and the teacher's knowledge are pre-programmed, so traditional CAI systems are neither flexible nor adaptable to the individual

behavior of the students. The idea of ITSs is then to develop domain-specific tutoring systems capable of displaying some degree of intelligence; that is, systems that approximate the behavior of an expert human tutor and are adaptive to aspects of the student's behavior rather than responding according to a fixed pattern (Bregar, Farley, & Bayley, 1986).

The notion of knowledge communication is central to ITSs. Wenger (1987) defines the task of teaching and learning as a task of knowledge communication. This refers to an instructional situation that involves a tutor (either a tutoring system or a human tutor) and a student, where the object of communication is knowledge or expertise in some domain. The power of a tutoring system is found in the amount and types of knowledge that it has.

Wenger identifies four main components of an ITS. They are the expert module, the student model, the pedagogical model (or tutor), and the user interface. Figure 2 indicates the interactions among the components of an ITS. The expert module contains a representation of the knowledge to be communicated. An ideal expert module would be able to perform well in the domain, as a human expert would; that is, it would generate problems and their complete solutions, including intermediate steps. It would also generate different solution paths in order to compare its solutions to the student's solution. Anderson (1988) describes the expert module as the backbone of any ITS, for it provides the domain intelligence. He claims that an adequate expert module must have an abundance of knowledge, and consequently, a great deal of effort needs to be expended to discover and codify that knowledge.

There are basically three approaches to encoding expert knowledge. One approach refers to "black box" expert systems, which encode the knowledge without actually codifying the underlying human intelligence. Black box systems generate the correct input-output behavior, but the internal computations that produce the behavior are not available for inspection. A second approach refers to "glass box systems", which are able to articulate knowledge about the domain. Glass box systems are built using methodologies

from the knowledge engineering field; that is, a knowledge engineer and a domain expert formalize key concepts, and design a system (usually a rule-based system) to represent and communicate the knowledge. Finally, a third approach is to make the expert module a simulation of the way a human uses the knowledge. This is the most demanding approach, but is also the one that more closely represents an ideal expert module.

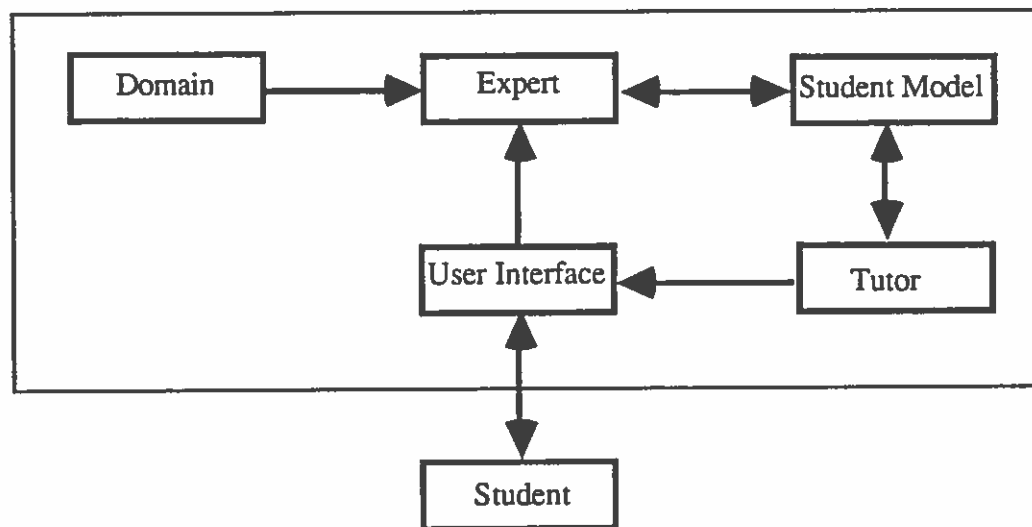


Figure 2. Components of an Intelligent Tutoring System.

The student model represents a student's current state of knowledge about the domain (Wenger, 1987). Its main purpose is to allow the system to adapt to the individual needs of the student. It may be consulted with several purposes. One of them is to decide when a student can move to a next type of exercise or a new topic. Another is to offer unsolicited advice when the system determines that the student needs it. A third purpose is to generate adequate problems for the student. Finally, the student model is used to adapt the explanations and provide appropriate guidelines. The student model is constructed

through a process of diagnosis, in which observable behavior is used to infer the student's current state of knowledge. Both the student model and the diagnostic module are tightly related, since the student model is the data structure and the diagnosis is a process that manipulates it (VanLehn, 1988). One of the major problems for the construction of the student model is that it needs to be updated dynamically, since hypothesis about the student's current state of knowledge might not hold anymore after the student has learned. The construction of the student model is a difficult task, and even human tutors sometimes are not able to do an accurate analysis of the student's deficiencies.

The pedagogical model, or tutor, is composed of pedagogical decisions (Wenger, 1987). It comprises the tutorial methodology that will determine the tutorial interactions; that is, it determines the manner in which topics are treated and presented. In this regard, didactic decisions are based on references to both the student model and the domain knowledge. The literature shows that didactic decisions are also based on the type of instructional environment that is being used. ITSs usually take the pedagogical approach of problem-solving monitors, mixed-initiative dialogues, or coaches (Sleeman & Brown, 1982; Wenger, 1987). Monitors maintain a high degree of control over the instructional interaction. They monitor the student's activities and adapt to the student's responses, but do not yield control to the student. An example of this type of environment is ACE, a problem solving monitor that analyzes complex explanations in the domain of nuclear magnetic resonance spectra (Sleeman & Hendley, 1982).

Mixed-initiative dialogues share the control with the student, and they both exchange questions and answers. The best example might be SCHOLAR (Wenger, 1987), which is a tutoring system developed by Carbonell in 1970. SCHOLAR can conduct a mixed-initiative dialogue about the geography of South America. The language used consists of simple English sentences, and both the tutor and the student can ask and answer questions.

Coaches offer guided discovery. The student controls the activities and the system intervenes when it seems necessary. The most significant systems developed under this framework are WEST (Burton & Brown, 1979) and WUSOR (Goldstein, 1979). WEST's domain is based on the educational game "How the West was won". The purpose of the game is the exercise of arithmetic skills and the development of a strategy to win the game. The computer generates three random numbers which the student uses to compose an arithmetic expression, and the result indicates the number of spaces that the student can move. Both the student and the expert module will solve the given problem. The system recognizes the behaviors of both the student and the expert, and compares them in order to obtain a differential model. This allows to recognize the weaknesses of the student and provide appropriate assistance.

WUSOR is a coach for the computer game WUMPUS. The game consists of a journey through caves, where there are dangers such as deadly pits and bats, and the terrible Wumpus is hiding. The system warns the player about the dangers in nearby caves; the player has to decide and plan where to go next in order to avoid the dangers. The student wins the game by shooting one of her five arrows into the Wumpus' lair. The system intervenes when the student has not chosen the optimal move. Coaching systems provide a form of guided discovery learning, and assume a constructivist position (Burton & Brown, 1979). That is, the student has the chance of learning from his mistakes and discovering the means to recover from them.

The user interface allows communication between the student and the tutoring system (Wenger, 1987). Since the user interface is responsible for the final communication of knowledge, it is also responsible for the effectiveness of the whole tutoring system. That is, the interface can make the topics more or less understandable, and it can also make the system more or less attractive to the student. User interfaces for tutoring systems can take various forms such as natural language, graphical interfaces, or direct manipulation. For

instance, SOPHIE, a simulation system in the domain of troubleshooting electronic circuits, provides a robust user interface at the level of natural language. STEAMER, on the other hand, which is also a simulation-based system, provides a direct manipulation user interface of a steam plant. Good user interfaces for tutoring systems are built keeping in mind the goals and concepts that are important to users and to the domain being tutored (Miller, 1988).

The four components described above represent a framework for the development of ITSs. The more complete and powerful each of these components is, the more intelligence the tutoring system could exhibit. A great deal of research needs to be done in this area. There is hope that ITS will provide flexible and adaptable computer-based tutoring systems. The research presented in the rest of this document is based on this framework.



## CHAPTER III

### A PERCENTAGE TUTOR

In chapter I, we presented basic concepts about the domain of percentages. In chapter II, we presented a basic framework to develop Intelligent Tutoring Systems. In this chapter we present the proposal of an Intelligent Tutoring System in the domain of percentages, PerTutor. The system will be divided in modules according to the framework discussed in chapter II. Here we present elements of the system without reference to implementation issues. In the Appendix we present details of our implementation, including a description of the data structures used.

#### The Expert Module

An expert module for an ITS in the domain of percentages should be able to propose and solve percentage problems. That is, it would be able to recognize the type of a problem and devise possible solutions for it. In PerTutor, the expert will be able to generate problems of several levels of difficulty. On the other hand, an ideal expert in the domain of percentages would require certain amount of mathematical knowledge that would enable it to solve the problems. For practical purposes, the proposed expert module will not require a great amount of knowledge about solving equations. This is because we will take advantage of the information that the expert already knows from the generation module.

The expert module of PerTutor lies between a black box and a glass box expert system. It will be capable of showing its solution to the problem, step by step, whenever it is required, so that the student can see how it solves the problem. From that perspective, the expert would be a glass box expert system. However, since the amount of general arithmetic knowledge the system will have is limited, the expert will not be able to elaborate about the algebraic foundations that led to the results. From that perspective, the expert is a black box system. This should not be considered a disadvantage, because the student will still have access to the solution, that is in fact the important element. Our system is meant to be a percentage tutor, not an algebra tutor.

The expert module would contain three parts: generation of problems, solution of problems, and diagnosis of the student performance, which will be discussed in the following sections.

### Generation of Problems

A problem in the domain of percentages is usually presented as a problem statement that describes a hypothetical situation, which contains some data and involves a question. Hypothetical situations would have to be given to the system; otherwise, the amount of knowledge the expert would need to know in order to generate them by itself would be immense. Thus, the system will have to have access to a set of frames that describe suitable situations for percentage problems. Such frames contain text and placeholders for data such as the percent, the percentage and the total. There are four types of frames (i.e., one for each type of problem). During the generation of problems, particular frames are chosen randomly within each type of frame.

A key issue for the generation of problems is the level of difficulty of the problem to be presented next. Problems can have different levels of difficulty that depend on the

question asked and the data given. As discussed in chapter I, problems of Types III and IV are considered more complex than, for instance, problems of Type I.

Table 1 presents examples of each type of problems. As we can observe, although the first three situations in the example involve the same data, the level of difficulty of the problems increases from the problem of Type I to the problem of Type III, and the problem of type IV involves a situation that is even more complex. Moreover, problems of the same type can also have different levels of difficulty. In this case the level of difficulty depends on the data that each problem involves, particularly on the values for the percent and the total. For example, a percentage problem of Type I that asks for 50% of 100 seems less

TABLE 1: Examples of different types of problems

Problem Type	Problem Statement
I	Joe plays with the basketball team of his school. In the last game, Joe scored 25% of the points, and the team won with a total score of 80. So, how many points did Joe score ?
II	Certain Electronics Store has a nice tape recorder on sale. The price tag says that it is 25% off, and that is \$20 off. What was the original price of the stereo ?
III	Jane is going to visit her brother that lives 80 miles away from her home. In the way she notices that she has already travelled 20 miles. What is the percent of miles that she has already travelled ?
IV	Peter is participating in a 90 mile bicycle race. He was running at a steady speed of 20 miles per hour, and has already run 40 miles. Peter is anxious to get to the end, and so he just increased his running speed on 30%. How fast is Peter running now ?

complex than a problem of the same type that asks for 50% of 280, 60% of 72, or 34% of 15; in the same way, a percentage problem of Type III that asks what percent is 20 of 80 seems less complex than a problem that asks what percent is 28 of 80.

The observation that the level of difficulty also depends on the data involved in the problem, suggests that we generate values that correspond to different levels of difficulty for the percent and the total. We propose a set of generation rules that will produce values for problems of a given type. These rules are described in Table 2. They generate data for problems with a low level of difficulty to problems with a higher level of difficulty. Since it seems important to work with simple numbers when learning the methodology to solve percentages, the set of rules was designed to assure that only integer values would be generated for all the percent, the percentage, and the total. The set presented in Table 2 includes neither rules to generate percents greater than 100 nor rules to generate mixed numbers. Such rules, however, can be derived by using the same pattern and can be easily added in future versions of the system.

The use of a rule depends on the tutor's criteria. The tutor will decide which rule should be applied to generate the next problem. If the tutor decides to keep showing problems of the same level of difficulty as the current problem, then the same rule will be used. If the tutor decides to advance to a problem with a higher level of difficulty than the current problem, then the next rule in the list should be used. If the tutor decides to show a problem with a lower level of difficulty than the current problem, then the previous rule in the list should be used. When all rules have been applied, the tutor could assume that the student has mastered the current type of problem. The expert can apply again the generation rules for the next type of problem. Notice that the level of difficulty of the next problem to be presented depends highly on the information provided by the student model and the decisions taken by the tutor. The criteria to decide whether to keep the same level of difficulty, advance to the next level, go back to a previous level, or change the type of

problem will be discussed in detail in the section regarding the tutor. By now, let us concentrate on the generation of the data for the problems.

TABLE 2. Rules for the generation of the percent and the total

Rule	Percent	Total	Description
1	100	$y, y=10$ or $y=100$	100% of 10, or 100% of 100
2	100	$2*y, 1 \leq y \leq 50$	100% of an even number between 2 and 100
3	100	$y, 1 \leq y \leq 100$	100% of a number between 1 and 100
4	100	$y, 1 \leq y \leq 1000$	100% of a number between 1 and 1000
5	50	100	50% of 100
6	50	$10*y, 1 \leq y \leq 10$	50% of a number between 10 and 100
7	50	$2*y, 1 \leq y \leq 50$	50% of an even number between 2 and 100
8	50	$2*y, 1 \leq y \leq 500$	50% of an even number between 2 and 1000
9	50	$y, 1 \leq y \leq 100$	50% of a number between 1 and 100
10	50	$y, 1 \leq y \leq 1000$	50% of a number between 1 and 1000
11	25	$4*y, 1 \leq y \leq 25$	25% of a number between 4 and 100
12	25	$4*y, 1 \leq y \leq 250$	25% of a number between 4 and 1000
13	75	$4*y, 1 \leq y \leq 25$	75% of a number between 4 and 100
14	75	$4*y, 1 \leq y \leq 250$	75% of a number between 4 and 1000
15	$10*x, 1 \leq x \leq 10$	$10*y, 1 \leq y \leq 10$	the percent is a multiple of 10 between 10 and 100, and the total is also a multiple of 10 between 10 and 100

TABLE 2. (Continued)

Rule	Percent	Total	Description
16	$10*x, 1 \leq x \leq 10$	$10*y, 1 \leq y \leq 100$	the percent is a multiple of 10 between 10 and 100, and the total is a multiple of 10 between 10 and 1000
17	$5*x, 1 \leq x \leq 20$	$20*y, 1 \leq y \leq 5$	the percent is a multiple of 5 between 5 and 100, and the total is a multiple of 20 between 20 and 100
18	$5*x, 1 \leq x \leq 20$	$20*y, 1 \leq y \leq 50$	the percent is a multiple of 5 between 5 and 100, and the total is a multiple of 20 between 20 and 1000
19	$x, 1 \leq x \leq 100$	$10*y, 1 \leq y \leq 10$	the percent is a random number between 1 and 100, and the total is a multiple of 10 between 10 and 100
20	$x, 1 \leq x \leq 100$	$10*y, 1 \leq y \leq 100$	the percent is a random number between 1 and 100, and the total is a multiple of 10 between 10 and 1000
21	$x, 1 \leq x \leq 100$	$y, 1 \leq y \leq 1000$	the percent is a random number between 1 and 100 and the total is a random number between 1 and 1000

Since the critical values for the generation of problems are the percent and the total, a simple strategy would be for the expert to generate problems of Type I, and then apply the corresponding equation to transform those problems into the actual type of the problem being presented. In this way, the generation rules can also be applied across problem types. This strategy would consist of three steps. In the first step the tutor would generate the values for the percent and the total. In the second step the expert would obtain the percentage. And, in the last step, the expert would replace the values obtained in the previous steps into the corresponding placeholders of the frames that describe hypothetical

situations. This is a simple strategy and it seems to be adequate to generate a variety of problems with different levels of difficulty.

### Deriving the Solution of a Problem

The expert module should contain the knowledge that would enable it to solve problems in the domain of percentages. The more knowledge it has, the more ways it could solve a problem. Percentage problems can be solved by applying the corresponding equations for each type of problem (see Table 3). These equations and basic algebraic knowledge constitute the basic knowledge that the expert needs to have. In addition to the three basic equations, experts in the domain of percentages might use alternative methods to solve percentages problems. This happens when they are able to recognize special situations that could be solved by using shortcuts. For instance, the problem of obtaining 50% of 80 can also be solved by dividing 80 by 2 since experts know that 50% is equivalent to one half. Problems involving percentages such as 20%, 25%, and 75% are commonly solved in similar ways. Although these alternative methods might be helpful in deriving the solution to a current problem, their use is not necessary because the system could apply the standard equations to get the same result. However, those alternative methods are quite important in the process of identifying the student's solution.

PerTutor will contain both knowledge about the basic equations and knowledge about the special cases. For practical purposes it will only use the equations in Table 3 as the basic methods to solve the problems; in this way, the system can demonstrate consistency each time it needs to present the solution to the student. It will use the knowledge about alternative methods to be able to recognize a larger number of ways in which students could solve problems.

TABLE 3. Equations to solve percentages problems

Type of Problem	Equation
1	$(\text{percent} / 100) \times \text{total}$
2	$\text{percentage} \times 100 / \text{percent}$
3	$\text{percentage} \times 100 / \text{total}$

### Diagnosis of the Student's Performance

Diagnosis can occur at two levels. First, it can occur when trying to identify what the student just did. Second, it can occur when trying to identify a general pattern in the behavior of the student. At the first level, the diagnosis module would receive not only the final answer to the problem but also each step used by the student when attempting to solve the problem. The user interface will provide a mechanism for the student to do computations and give each intermediate step as part of the solution. The expert would analyze the given solution to determine if it is completely correct, partially correct or completely incorrect.

The assessment of the student's responses can be accomplished by developing a "buggy" theory similar to the one developed by Brown and Burton (1979). They developed a theory of the student's misconceptions or bugs in basic arithmetic skills. They also developed a system to teach students and student teachers how to diagnose bugs as well as how to provide a better understanding of the underlying structure of the arithmetic skills.

In the domain of percentages, we can use a similar idea with the purpose of developing a catalog of common percentage errors. The diagnosis module would access the catalog when attempting to recognize the student's solution to the problem. The



advantage of this strategy is that the expert can look for specific usual problems. The disadvantage of this method is that the system will not be able to recognize unusual errors. This situation actually represents a tradeoff between the amount of time required to construct an ideal expert module and the advantage of obtaining a running version of the system. The use of a catalog of common errors is an effective and efficient method for doing diagnosis in simple domains.

Different solutions can be considered completely correct. One is when the student gives each correct step of the corresponding equation and the correct result. A second is when the student has a history of solving the problems correctly by mental computation; that is, with no need of intermediate steps. In such case, the expert can assume that the student knows the procedure to solve the problem. And three, when the student uses a correct shortcut to solve the problem.

Solutions are considered partially correct if they can be fixed by making minor changes to the intermediate steps. There are also several cases. One is when the data in the operations is correct but the operators are incorrect, for example, when the student multiplies instead of dividing. A second case is when the student switches the order of the operands, for example, dividing 100 by the percent instead of dividing the percent by 100. A third is when the student uses the correct data, but both the order of the operands and the operator are wrong. A fourth case is when the student needs to do just one more step to complete the solution. A fifth is when the final answer is correct, but the student's operations are incorrect. The sixth case is when the final answer is incorrect but the student's operations are correct.

Solutions are considered totally incorrect when the final answer is incorrect and the expert can not identify what the student did. Two cases describe this situation. One is when the student does not show the steps to solve the problem. The second case is when

the error involves more than performing minor changes in the intermediate steps given by the student.

In addition to the above situations, the diagnosis module would need to check for answers that are obviously too big or small to be correct, including negative answers. This additional information would be useful to the tutor at a later stage.

Notice that, since the student will be capable of giving the solution of the problem step by step, the diagnosis module should be able to recognize several ways in which the student could introduce the data. The diagnosis module should know about commutative and associative properties. For example, to give an answer for the question what is 25% of 80, the student could perform the intermediate steps in several ways, by multiplying 0.25 by 80 or viceversa, by first dividing 25 by 100 and then multiplying the result by 80, or by dividing 80 by 100 and then multiplying the result by 25. Similar situations occur for all other types of problems.

The second level of diagnosis is when the expert tries to identify a pattern in the behavior of the student that will be useful to make pedagogical decisions. This level of diagnosis is related to the student model, and will, therefore, be discussed in the following section.

### The Student Model

In the tutorial system we are proposing, the student model consists of the student's history through the tutorial session. It contains the intermediate steps and the answers given by the student when attempting to solve each problem, as well as the assessment of each attempt, indicating whether the solution was right or wrong. If the solution was wrong, it should also indicate what the error was.

As was previously pointed out, the student model is a data structure representing previous interactions between the student and the system. The diagnostic module will consult the student model with the purpose of determining patterns in the behavior of the student. It is not our intention to derive a general pattern of the behavior of the student. This has proven to be a very difficult task even for human tutors. Instead, the diagnostic module should identify particular features in the performance of the student, and derive statistics from the history of the student interactions, that could allow the tutor to make decisions. Examples of information that the diagnostic module could derive are the level of help that the student has received in the last three problems, the percentage of times the student has correctly solved problems, the number of times the student has correctly solved problems in the first trial, and the number of times the student has had the same error (such as multiplying instead of dividing). Having this information, it will be the tutor's task to decide what course the tutorial interaction will take.

### The Tutor

One of the advantages attributed to the computer as an instructional tool is its potential capacity to conduct individualized learning. However, the computer per se is not capable of doing so. Instead, is the tutoring software which provides the means to accomplish individualized learning for it embraces the methodology that will establish the tutorial interactions.

The tutor will require certain knowledge to respond appropriately in a variety of situations. As a general criteria, the proposed tutor will give students a maximum of three trials to solve each problem. After each trial, the tutor will analyze the situation to produce the appropriate interaction. We propose a set of rules that contain specific criteria that the tutor will follow after each trial. There will be two classes of tutorial rules. One class

corresponds to the kind of problems that will be presented next. For this class of rules there are three possibilities: advance to the next level, go back to the previous level, or remain in the same level of problems. The conditions that should be met for each possibility are described in Table 4.

TABLE 4. Rules for the level of the next problem

Rule	Conditions
Advance to next level	Student has correctly solved, in the first trial, two consecutive problems of the current level.
Go back to previous level	Student could not solve the last two problems, and the tutor had to give its solution.
Remain in same level	When neither of the two above rules applies.

The other class of rules corresponds to the rules of what to say and show after each tutorial interaction. In this case, the student's answer to the current problem should also be taken into account. If the student's answer is correct, the tutor will make a rewarding statement, which it will choose from a set of rewarding statements, and will apply its criteria to decide whether the student should remain in the same level of problems or should advance to the next level. If the student's answer is incorrect, the tutorial interaction will depend on aspects such as the kind of error, the frequency with which the student has had the same error, the current trial, and the pedagogical decisions previously taken by the tutor.

Tables 5 and 6 summarize a set of possible tutorial rules. Table 5 describes possible actions that the tutor could follow when it is the first time that the student gives a

TABLE 5. Specific tutorial rules by type of error and number of trial when it is the first time the error occurs in the session

Kind of Error	Strategy per Trial
Wrong operator	<ol style="list-style-type: none"> <li>1. Give a hint: Check the Operator.</li> <li>2. Show basic information about percentages.</li> <li>3. Solve the problem.</li> </ol>
Wrong order of operands	<ol style="list-style-type: none"> <li>1. Give a hint: what is wrong about the operands.</li> <li>2. Show basic information about percentages.</li> <li>3. Solve the problem.</li> </ol>
Wrong operator and wrong order of operands	<ol style="list-style-type: none"> <li>1. Give a hint: check how to combine the data.</li> <li>2. Solve one of the errors.</li> <li>3. Solve the problem.</li> </ol>
Student is only missing one step to complete the solution	<ol style="list-style-type: none"> <li>1. Tell the student that one step is missing.</li> <li>2. Indicate the operator that the step involves.</li> <li>3. Solve the problem.</li> </ol>
Student gave correct solution without indicating the intermediate steps	<ol style="list-style-type: none"> <li>1. Say that it is OK to do that.</li> <li>2. Say that it is OK to do that.</li> <li>3. Say that it is OK to do that.</li> </ol>
Student gave a wrong solution and did not show intermediate steps	<ol style="list-style-type: none"> <li>1. Ask to show steps.</li> <li>2. Say, again, that it is useful to show steps.</li> <li>3. Solve the problem.</li> </ol>
Result is correct but intermediate steps incorrect.	<ol style="list-style-type: none"> <li>1. Ask to show intermediate steps.</li> <li>2. Ask again for procedure.</li> <li>3. Show the correct operations.</li> </ol>
Result is incorrect but intermediate steps correct	<ol style="list-style-type: none"> <li>1. Indicate that there are student's steps that correspond to the actual solution.</li> <li>2. Ask to check the steps and find correct operations.</li> <li>3. Show correct operations.</li> </ol>
Answer is negative	<ol style="list-style-type: none"> <li>1. Say that answer can not be negative.</li> <li>2. Give a basic explanation about percentages.</li> <li>3. Solve the problem</li> </ol>
Answer is too big	<ol style="list-style-type: none"> <li>1. Explain why answer should not be so big.</li> <li>2. Explain why answer should not be so big.</li> <li>3. Explain why answer should not be so big.</li> </ol>
Solution incorrect, and operations can not be identified	<ol style="list-style-type: none"> <li>1. Show basic explanation about percentages.</li> <li>2. Solve first operation.</li> <li>3. Solve the problem.</li> </ol>

TABLE 6. Specific tutorial rules by type of error and number of trial when the error has already occurred in the session

Kind of Error	Strategy per Trial
Wrong operator	<ol style="list-style-type: none"> <li>1. Give a hint: again you are not using correct operator.</li> <li>2. Show how same error was previously solved.</li> <li>3. Solve the problem.</li> </ol>
Wrong order of operands	<ol style="list-style-type: none"> <li>1. Give a hint : something wrong about operands.</li> <li>2. Show how same error was previously solved.</li> <li>3. Solve the problem.</li> </ol>
Wrong operator and wrong order of operands	<ol style="list-style-type: none"> <li>1. Give a hint: check how to combine the data.</li> <li>2. Solve one of the errors.</li> <li>3. Solve the problem.</li> </ol>
Student is only missing one step to complete the solution	<ol style="list-style-type: none"> <li>1. Tell the student again that one step is missing.</li> <li>2. Indicate the operator that the step involves.</li> <li>3. Solve the problem.</li> </ol>
Student gave correct solution without indicating the intermediate steps	<ol style="list-style-type: none"> <li>1. Give only rewarding statement.</li> <li>2. Give only rewarding statement.</li> <li>3. Give only rewarding statement.</li> </ol>
Student gave a wrong solution and did not show intermediate steps	<ol style="list-style-type: none"> <li>1. Ask again to show intermediate steps.</li> <li>2. Solve first operation.</li> <li>3. Solve problem.</li> </ol>
Result is correct but intermediate steps incorrect	<ol style="list-style-type: none"> <li>1. Say that answer is correct but operations incorrect</li> <li>2. Ask to show the intermediate steps.</li> <li>3. Show operations.</li> </ol>
Result is incorrect but intermediate steps correct	<ol style="list-style-type: none"> <li>1. Give hint: some steps are correct.</li> <li>2. Ask to check the procedure and find correct operations.</li> <li>3. Show correct operations.</li> </ol>
Answer is negative	<ol style="list-style-type: none"> <li>1. Say that answer can not be negative.</li> <li>2. Give a basic explanation about percentages.</li> <li>3. Solve the problem</li> </ol>
Answer is too big	<ol style="list-style-type: none"> <li>1. Explain why answer should not be so big.</li> <li>2. Explain why answer should not be so big.</li> <li>3. Explain why answer should not be so big.</li> </ol>
Solution incorrect, and operations can not be identified	<ol style="list-style-type: none"> <li>1. Show basic explanation about percentages.</li> <li>2. Solve first operation.</li> <li>3. Solve the problem.</li> </ol>

particular wrong answer. Notice that, the tutor's first attempt would be to give only a general hint, so that the student would have a chance to check her work. In some other situations, the tutor will give more specific guidance, solve part of the problem, give explanations about percentages, or show how the problem should be solved. To avoid monotony, the tutor will have to know different ways in which a hint, an explanation or a rewarding statement can be said; therefore, it needs to have access to a data base of possible statements, and it needs to check what has been said previously, before deciding what to say.

Table 6 describes possible actions that the tutor could follow when it is not the first time that the student gives a particular wrong answer to the problem. For these cases, the tutor will give hints, solve part of the problem or show how the problem should be solved; additionally, the tutor will be able to make references to previous problems for which the student made the same kind of error. The tutor will show how that problem was solved, so that the student can learn from her mistakes. Notice that the tutor's final selection depends on the combination of rules that fired and on previous tutorial interactions.

### The User Interface

The user interface should consist of elements that will provide a fluent way of communication to solve percentages problems. From the point of view of the student, the user interface should be clear and easy to use, and it should also facilitate the problem solving process. From the point of view of the expert, it should provide a means to collect information for the process of diagnosis. The user interface of PerTutor will consist of functional windows and menus.

The user interface will require a window to display the problem statement, a window to give the final answer, a window for the tutor, and a transcript window. The

functions of the Problem Statement and the Answer windows are obvious. The Problem Statement window is the place where the expert will display the statement of the problem. The Answer window is where the student will type the final. The Tutor window is the place where the tutor will communicate with the student; all hints, messages, and guidelines will appear in this window. The Transcript window will keep a record of the operations performed by the student. It will be useful for the student to see how he is solving the problem. The tutor and the Transcript windows will contain scroll bars so that the user can refer to previous interactions in the session. The scroll bar in the Problem Statement window is used in case that the problem statement is too long to fit in the window. Additionally, the system will provide a calculator window which will be useful to both the student and the tutor; the student can use it to compute intermediate steps, and the tutor can use it as a tool to obtain the procedure performed by the student while solving the problem. Figure 3 presents a view of the user interface.

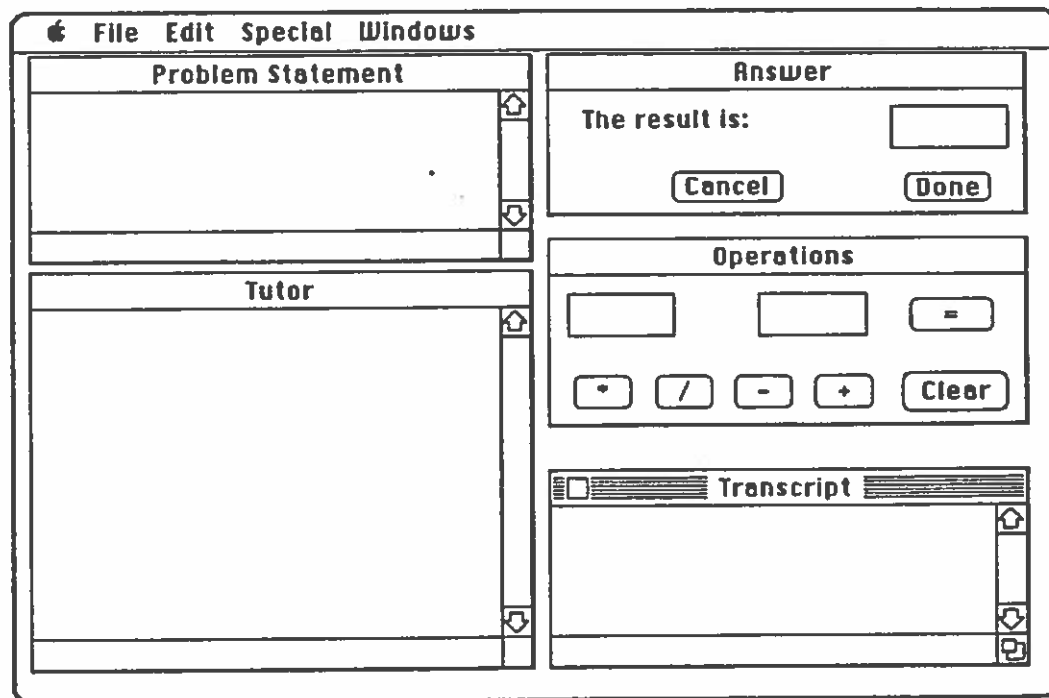


Figure 3. A view of the user interface.



In a typical situation, the system will first display the problem statement in the Problem Statement window. During each trial, the user could compute intermediate results through the Operations window. Each intermediate step is computed by entering the data in the slots of the Operations window, selecting the operator, and then clicking in the equals button. Each step will be recorded in the Transcript window. The user can enter the final answer by typing it in the Answer window and then clicking in the button labelled "done". Once the answer has been entered, the system will record it in the Transcript window, and the tutor will respond accordingly in the Tutor window.

The menus will supply additional functions. The File menu will allow the student to initiate, save and retrieve tutorial sessions, and it will also allow to exit the system. Figure 4 shows the File menu. The purpose of the command New is to start a new session. The commands Open and Save work together. The command Open will open a session that had already been saved. After this command, the system will retrieve the status of a previous tutorial interaction, and the student can proceed the session from where she had left the work. The Save command will store all data and status of the current session. This command allows the tutor to remember a tutorial session. The command Close closes a tutorial session. And, the command Quit is used to exit the system.

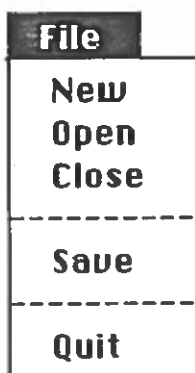


Figure 4. The File menu.

The purpose of the Edit menu (see figure 5) is to provide a fast way to modify and introduce data. Since it is common that the result of a step will be used in another step, the student can use the commands Copy and Paste to copy data from a previous operation into a new operation. In this way, the student can also reduce the possibility of typos when introducing the data. The Cut command is provided as an alternative way to erase data.



Figure 5. The Edit menu.

The Special menu provides three special functions. The Help command gives the student the possibility to ask for help at any time. The Commands Show Performance and Print Performance will be used to obtain statistics about the performance of the student through the session. Show Performance will display the information on the screen, and Print Performance will do it on paper. The Special menu is described in Figure 6.

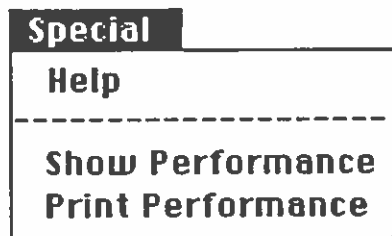


Figure 6. The Special Menu.

Finally, the Windows menu (see figure 7) lists the names of all windows of the user interface. It provides an alternative way of selecting the windows.

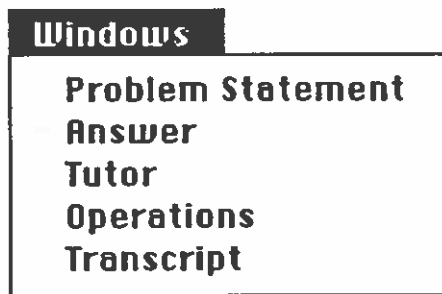


Figure 7. The Windows Menu.

Although the menus enhance the functionality of the system, we have restricted the implementation of the user interface of the first running version of the system to, basically, the five windows shown in figure 3. The implementation of most commands in the menus will be left for a future version.

To better understand how these facilities work together, we now turn our attention to an example session of the system in action.

CHAPTER IV  
 EXAMPLES FROM AN INITIAL VERSION  
 OF THE SYSTEM

This chapter discusses a series of examples that will be helpful to illustrate how PerTutor works. The first example demonstrates a simple case. This is when the student gives a correct solution to the problem in the first trial. Figure 8 shows how the screen looks when a problem is presented. As was previously explained, the problem statement appears in the Problem Statement window. The Tutor window is prepared to keep a record of the tutor's interaction with the student, and the Transcript window is prepared to keep

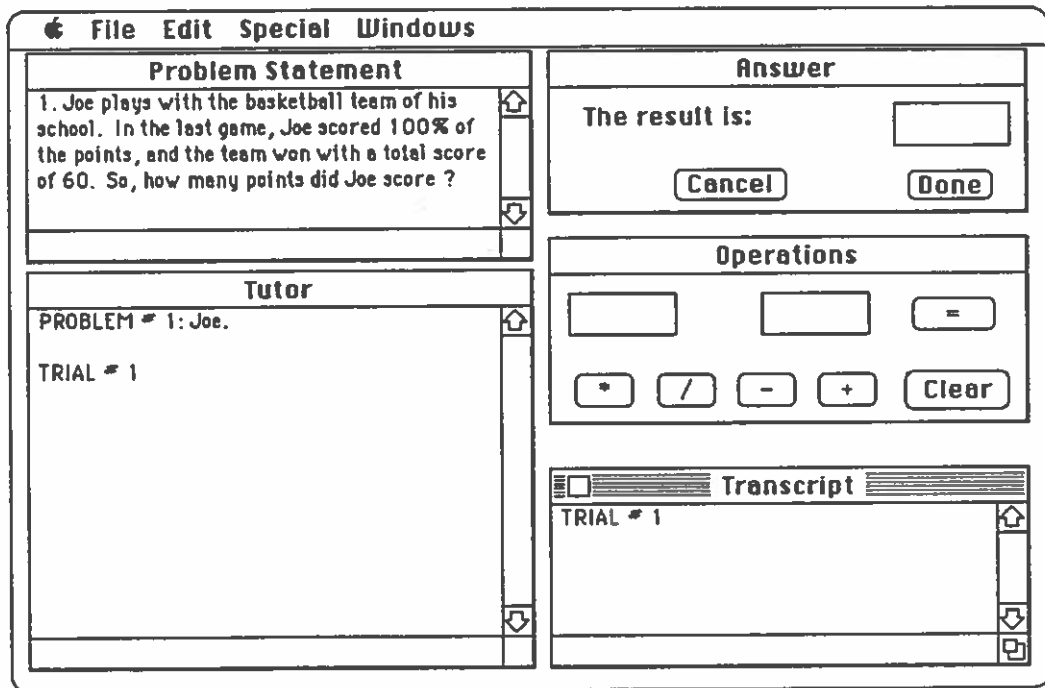


Figure 8. User interface after problem statement has been given.

the record of the student's interaction with the system. By combining the information in these two windows, the student could refer to a previous episode in the tutorial session.

The Operations window, or calculator, appears clear when a new problem is presented, and it is always cleared after the system has recorded the current operation in the Transcript window; in this way, the calculator will be ready for the next operation that the student needs to compute.

In solving this problem, the student only indicates the final answer. Figure 9 shows the tutor's response. Notice that the tutor recognizes that the student did not compute intermediate steps to solve the problem.

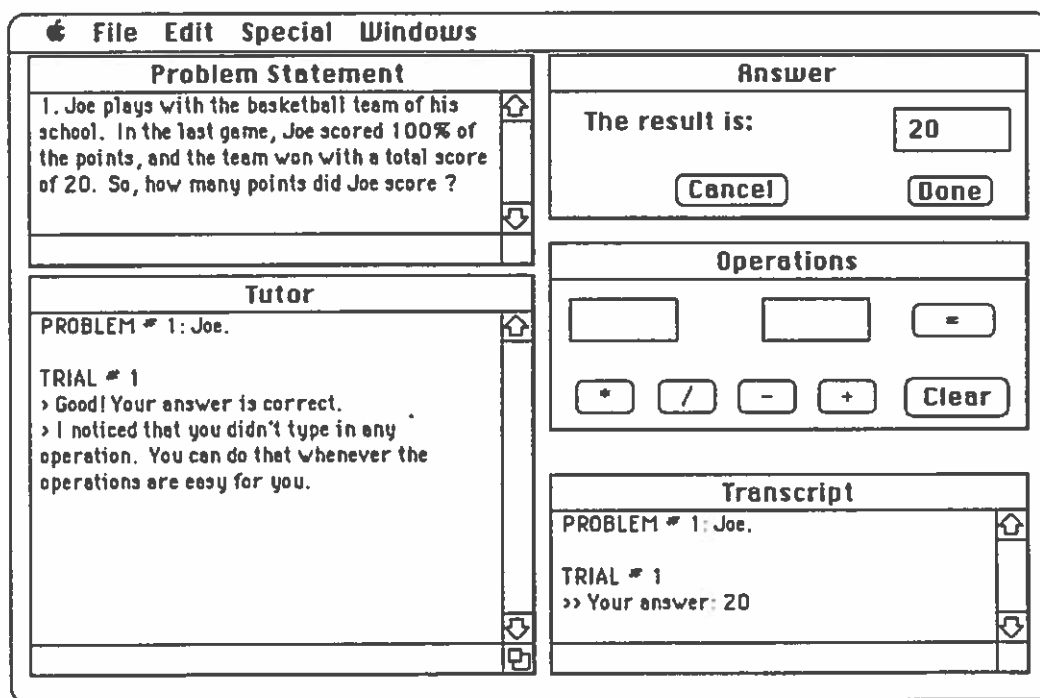


Figure 9. Student gave the correct solution to the problem in the first trial.

The second example demonstrates how the expert identifies a partially correct solution to the problem. Figure 10 shows the student using the calculator to input an intermediate step. Figure 11 describes the situation after the student has provided the

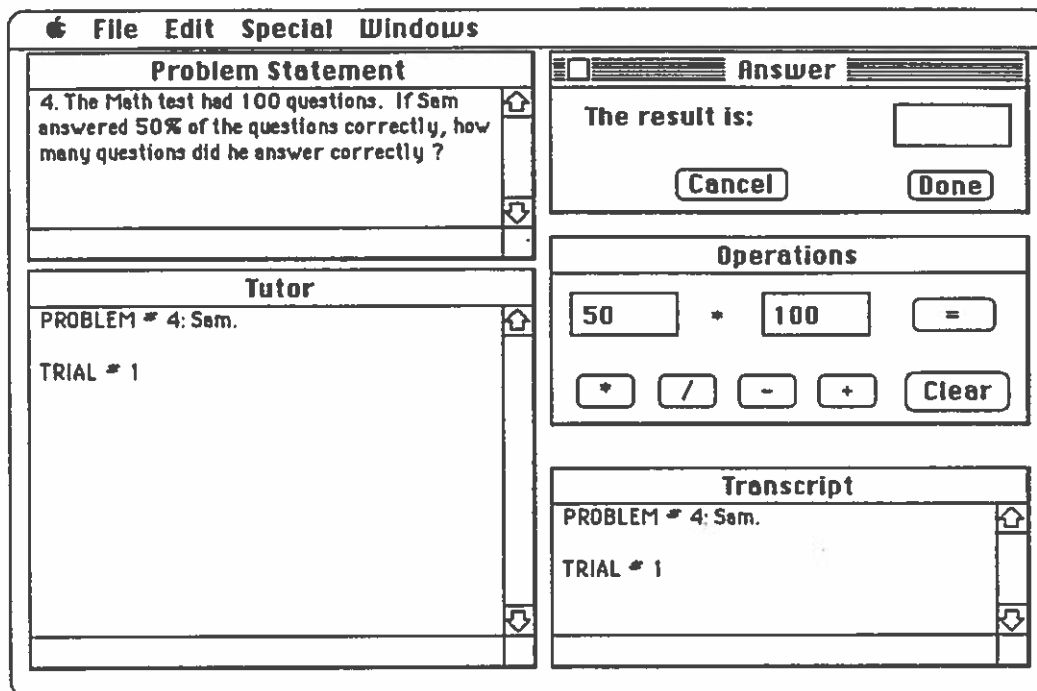


Figure 10. Use of the operations window.

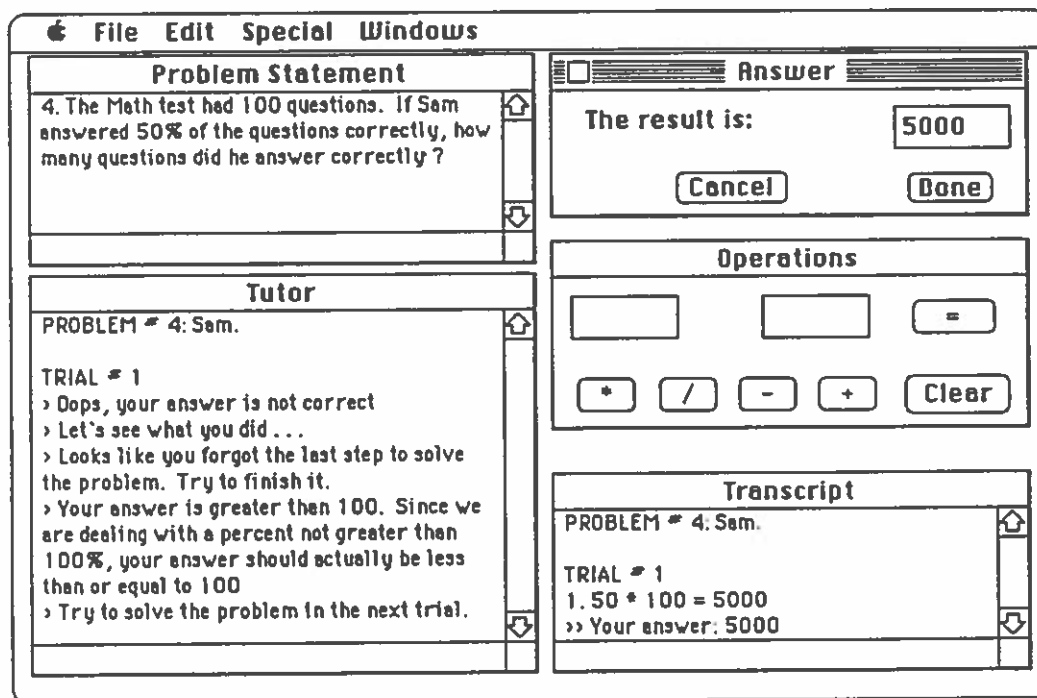


Figure 11. Student is missing one step.

answer, and the tutor has derived a diagnosis. The student gave the result after step 1 as the answer to the problem. The tutor considers that this is as a partially correct solution, but that it still needs to be divided by 100. Notice how both step 1 and the answer have been recorded in the Transcript window. To finish this second example, figure 12 shows how the system responds after the student has corrected the error and given the final result.

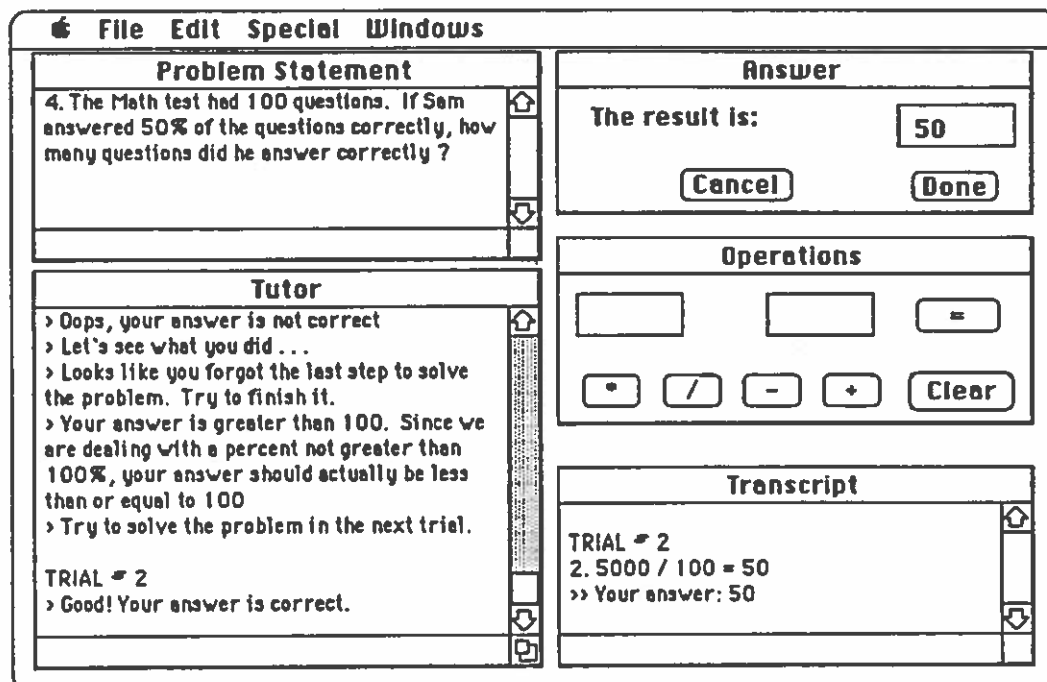


Figure 12. Student computed the missing step and gave the correct answer.

Our third and fourth examples demonstrate how the tutor responds when the student gives an incorrect solution that can not be easily fixed. Figures 13 through 16 comprise the third example. In Figure 13 we observe that the student performed some data manipulations that did not lead to the solution of the problem. The system determines that the steps do not correspond to any partial or complete solution, and therefore decides that the student needs basic instruction about percentages. The tutor then gives a basic and somehow detailed explanation (see figures 13, 14, and 15).

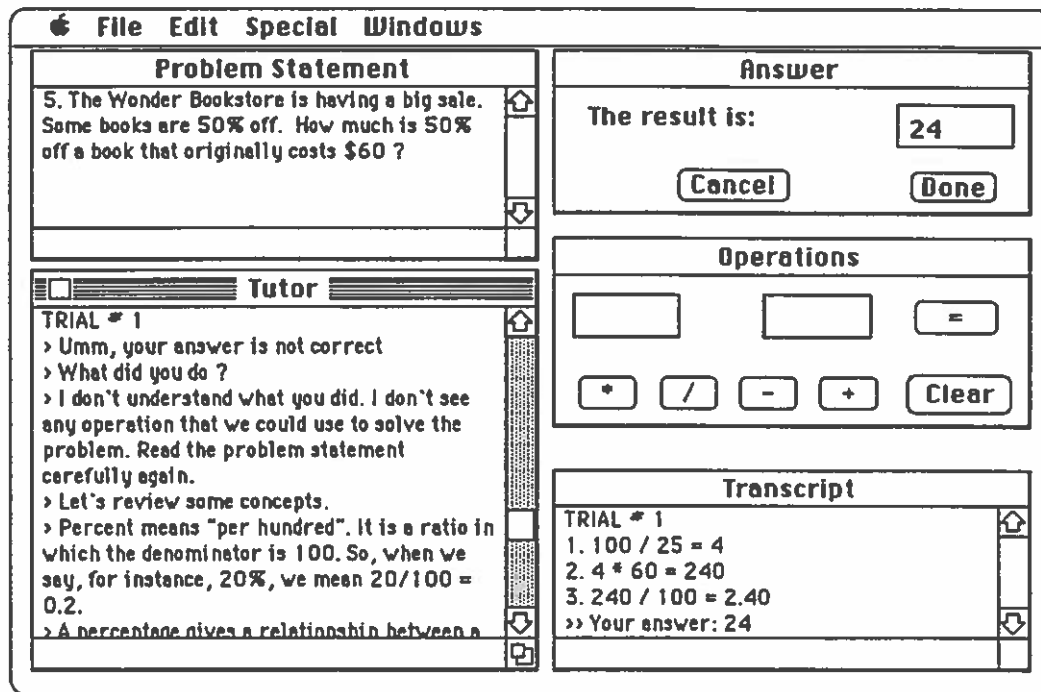


Figure 13. Tutor determines that steps do not correspond to any partial or complete solution of the problem.

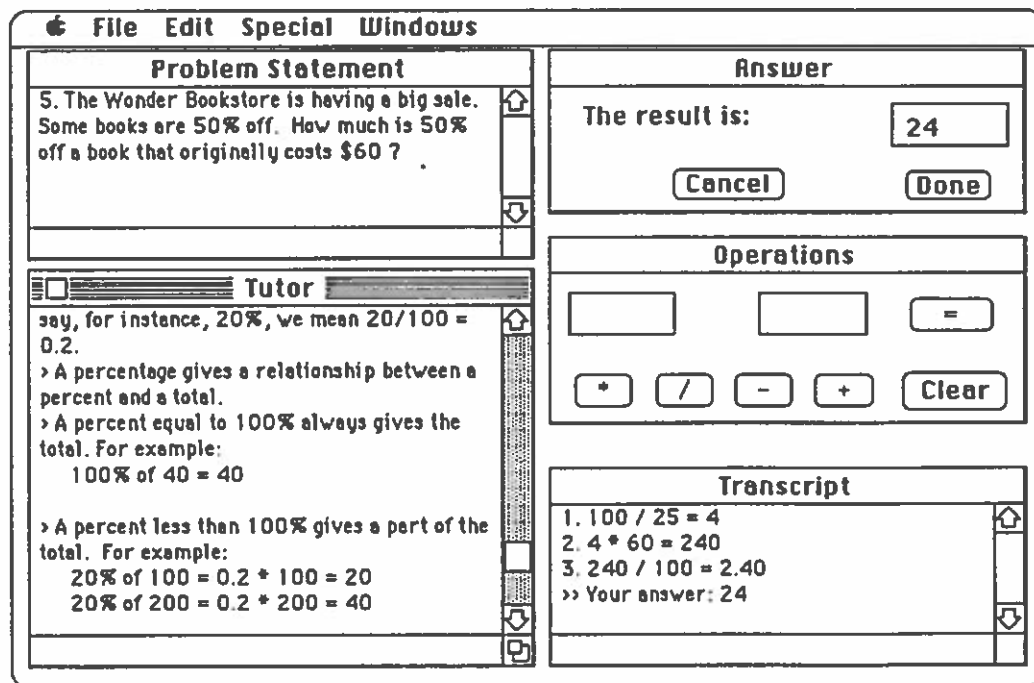


Figure 14. Tutor determines that steps do not correspond to any partial or complete solution of the problem (continued).



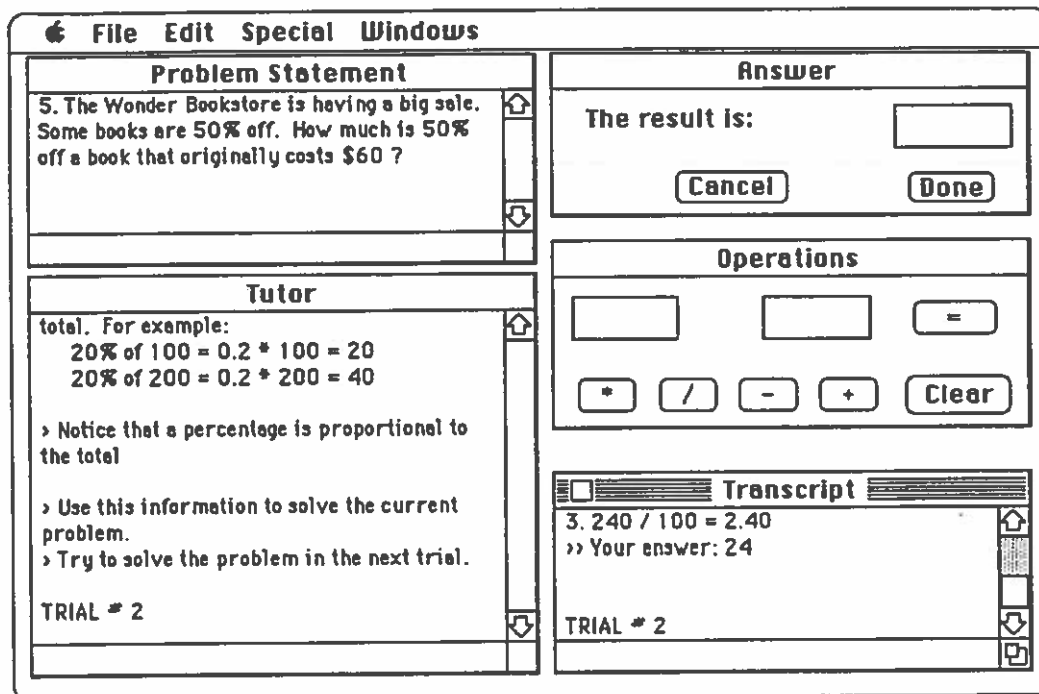


Figure 15. End of the tutor's explanation for example 3.

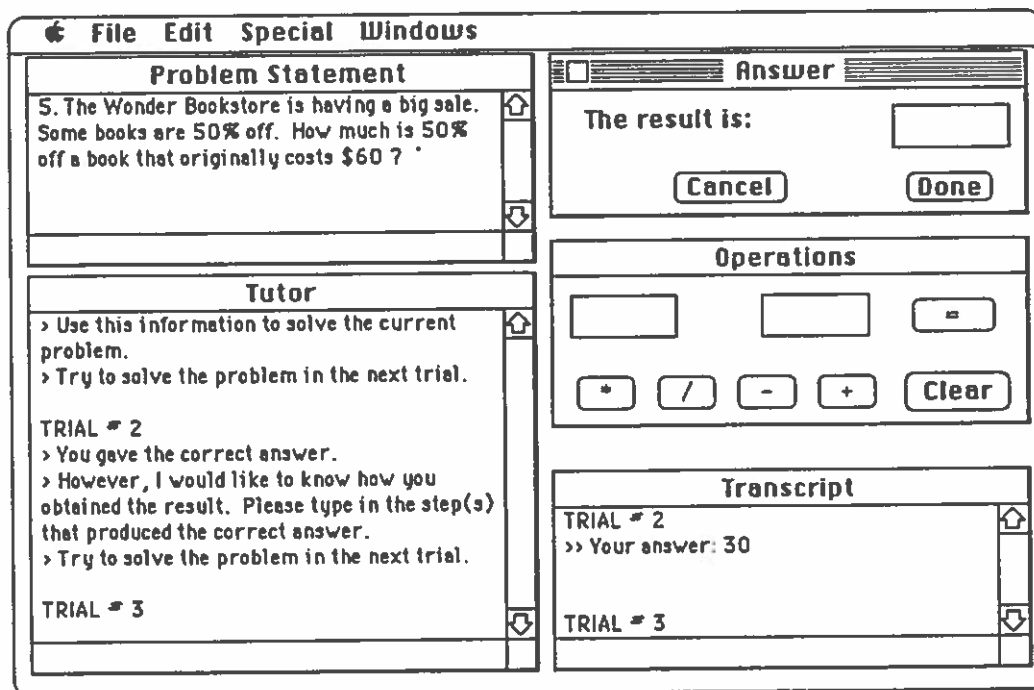


Figure 16. Student gives correct answer after failing first attempt.

Figure 16 is a continuation of the student interaction with the system when attempting to solve the same problem. After failing to obtain the solution in the first attempt, the student gives the correct result in the second trial without apparent use of intermediate steps. Such situation makes the tutor suspicious about how the student obtained the result, and therefore considers that it is necessary to request the procedure in order to make sure that the student knows how to solve the problem.

Figures 17 and 18 illustrate our fourth example. This case is similar to the one discussed in example 3. Since the solution given by the student does not indicate that the student knows how to solve the problem, the tutor also decides to give a basic explanation about percentages. However, the tutor knows that it had already given a basic explanation to the student (that is, the explanation for problem 5 in example 3), and decides to give a different basic explanation. In this case, the explanation is shorter.

The screenshot shows a window titled "File Edit Special Windows" with four main panels:

- Problem Statement:** "8. Mrs. Gardener wants to plant a flower garden. She wants to plant 40 flowers, and wants 25% of them to be roses. So, how many roses will she plant?"
- Answer:** "The result is: 63" with "Cancel" and "Done" buttons.
- Operations:** A calculator interface with input fields, a "=" button, and buttons for "\*", "/", "-", "+", and "Clear".
- Tutor:**
  - TRIAL # 1
  - > Oops, your answer is not correct
  - > Let's see what you did . . .
  - > I don't understand what you did. I don't see any operation that we could use to solve the problem. Read the problem statement carefully again.
  - > Remember that a percentage gives a relationship between a percent and a total.
  - > Also remember that, for the kind of exercises we are dealing now, the percentage is a part of the total
  - > Try to think what could be a result that could
- Transcript:**
  - TRIAL # 1
  - 1.  $25 / 40 = 0.63$
  - 2.  $.63 * 100 = 63$
  - >> Your answer: 63

Figure 17. Tutor determines that steps correspond neither to partial nor to complete solution of the problem (short explanation).

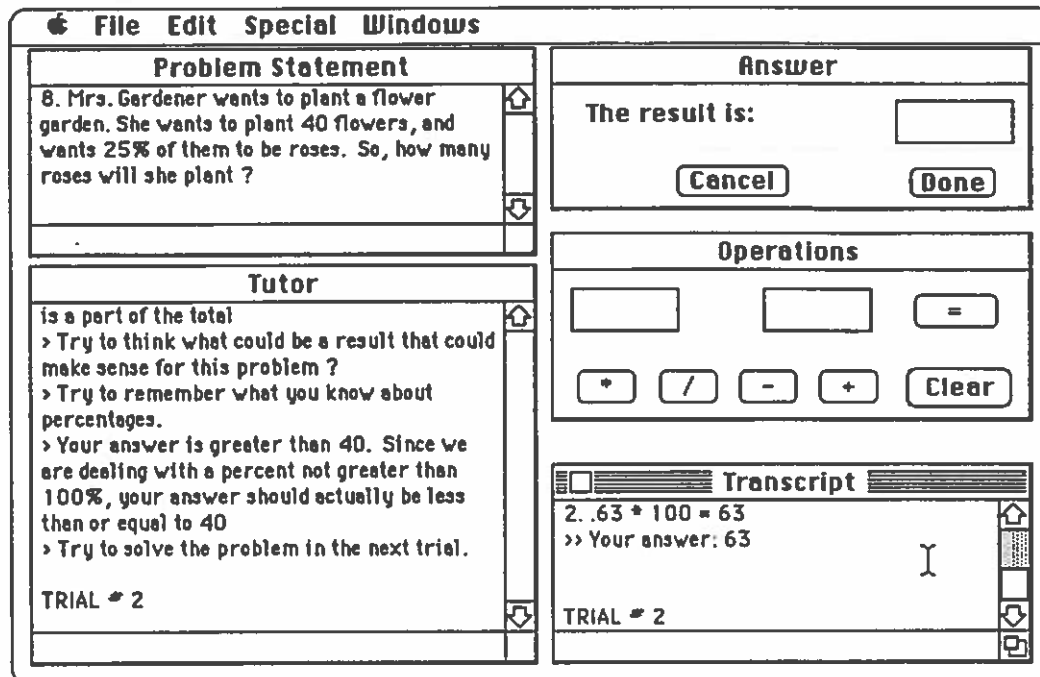


Figure 18. Tutor determines that steps correspond neither to partial nor to complete solution of the problem (short explanation continued).

The last example demonstrates an important feature of PerTutor, which is that the tutor can base its explanation on a previously solved problem. Figures 19 and 20 show a student's error when attempting to solve problem 11. The student uses a buggy procedure; that is, instead of multiplying in step 2, he is dividing. The tutor responds with a hint about the error, which allows the student to solve the problem in the next trial (see figure 20). As we can observe in figure 21, the student does the same thing in problem 15. The tutor recognizes the error, remembers that it had happened before, and gives an appropriate hint. However, the student is still not able to solve the problem in the second trial, and the tutor decides to base its explanation on the solution given for problem 11 (see figure 22). The tutor recalls how the student was able to reach the solution in the previous problem. In figure 23, we see that the student finally obtains the correct solution

The screenshot shows a window titled "File Edit Special Windows" with four main panels:

- Problem Statement:** "11. The Math test had 20 questions. If Sam answered 35% of the questions correctly, how many questions did he answer correctly?"
- Answer:** "The result is:" with a text box containing "0.02" and buttons for "Cancel" and "Done".
- Operations:** A calculator interface with input fields, an equals sign button, and buttons for "+", "/", "-", and "Clear".
- Transcript:** A scrollable area showing the student's work:
 

```
TRIAL # 1
1. 35 / 100 = 0.35
2. 0.35 / 20 = 0.02
>> Your answer: 0.02
```

The Tutor panel shows feedback for the first trial:

```
PROBLEM # 11: Sam.

TRIAL # 1
> Oops, your answer is not correct
> Let's see what you did ...
> Let's concentrate on steps 1 and 2

> Hint: check the operator.
> Try to solve the problem in the next trial.
```

Figure 19. Student divides rather than multiplying.

The screenshot shows the same window as Figure 19, but with updated content:

- Answer:** The text box now contains "7".
- Transcript:** Shows the second trial:
 

```
TRIAL # 2
3. .35 * 20 = 7
>> Your answer: 7
```
- Tutor:** Shows feedback for the second trial:
 

```
TRIAL # 2
> Good! Your answer is correct.
```

Figure 20. Student corrects the buggy procedure, and multiplies instead of dividing.

The screenshot shows a window titled "File Edit Special Windows" with four main panes:

- Problem Statement:** "15. Mrs. Gardener wants to plant a flower garden. She wants to plant 80 flowers, and wants 65% of them to be roses. So, how many roses will she plant?"
- Answer:** "The result is: 0.01" with "Cancel" and "Done" buttons.
- Operations:** A calculator interface with input fields and buttons for "+", "/", "-", "+", and "Clear".
- Transcript:**
  - TRIAL # 1
  - 1.  $65 / 100 = 0.65$
  - 2.  $0.65 / 80 = 0.01$
  - >> Your answer: 0.01

The Tutor pane is empty in this view.

Figure 21. Student does the same error, divides instead of multiplying.

The screenshot shows the same window as Figure 21, but with updated content in the Tutor and Transcript panes:

- Problem Statement:** Same as in Figure 21.
- Answer:** Same as in Figure 21.
- Operations:** Same as in Figure 21.
- Tutor:**
  - > Let's concentrate on steps 1 and 2
  - > Again, you are not using the correct operator
  - > In problem 11, Sem problem, you had the same error that you have now. Let's review it.
  - > In that problem we wanted to find 35% of 20, and you solved it like this:
  - >  $35 / 100 = 0.35$
  - >  $0.35 * 20 = 7$
  - > So, the answer was: 7.
  - > Try to solve the problem in the next trial.
- Transcript:**
  - TRIAL # 2
  - 1.  $65 / 100 = 0.65$
  - 2.  $0.65 / 80 = 0.01$
  - >> Your answer: 0.01

Figure 22. Tutor bases explanation on previous problem.

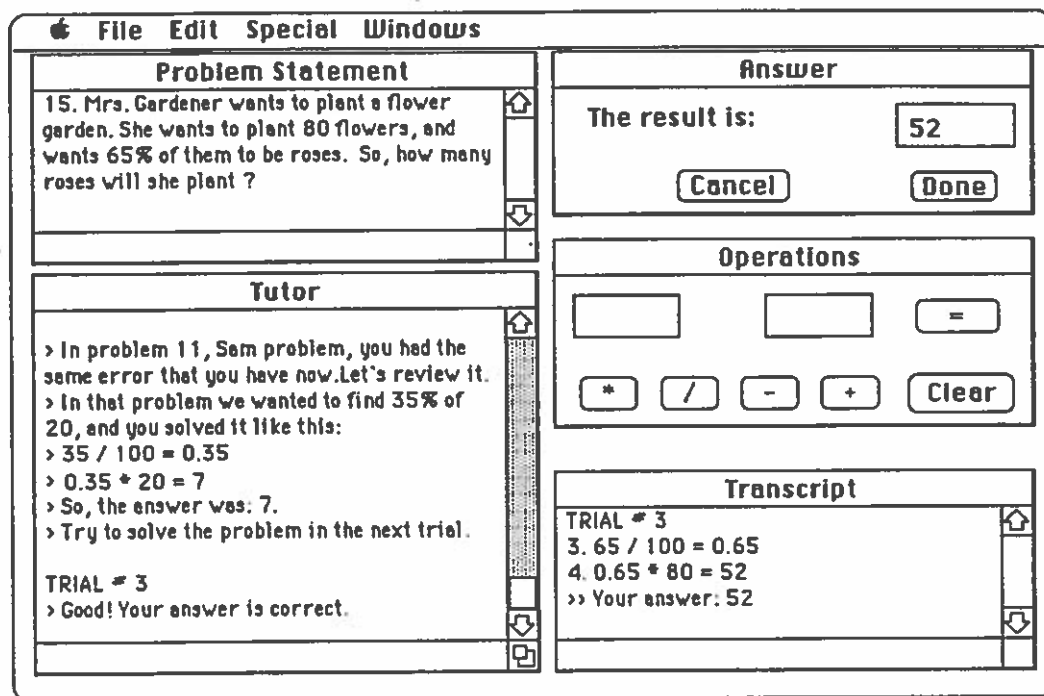


Figure 23. Student solves problem after tutor's reference to previous problem.

The intention of the above examples has been to provide an overview of the most important features of PerTutor. There are other situations that the system is capable of recognizing and responding. For instance, when the student multiplies instead of dividing, when the student gives a result that is negative, when the steps performed by the student are correct but the final answer is incorrect, when the tutor solves only the first step for the student, etc. The system is characterized by a significant level of flexibility. In the next chapter, we will evaluate and discuss particular characteristics of the system in more detail.

## CHAPTER V

### DISCUSSION

We have presented in this study an intelligent tutoring system in the domain of percentages. One of our purposes has been to construct initial versions of the four components that comprise an intelligent tutoring system (i.e, the expert module, the student model, the tutor, and the user interface). In chapter II we discussed the features required for ideal components. In our first version of the system, we attempted to cover a significant number of these issues, and, as we demonstrated in chapter IV, the system is capable of exhibiting some degree of flexibility and intelligence.

The expert module of our system is characterized by its capacity to generate problems of different levels of difficulty, and by its capacity to recognize alternative ways in which a student could approach the solution to the problems. There are several elements, however, that we would like to improve in future versions of the expert module. The first one is regarding the generation of problems. The system assumes that the pupil's state of knowledge about percentages, when the session starts, is at the beginners level. In other words, the system begins by delivering problems of low level of difficulty, and moves up through the sequence of levels in accordance with the student's performance during the tutorial session. In this sense, the expert provides individualized learning for students advance to the next level of problems as they are ready to do so. Our concern is that students might have different levels of knowledge when the session starts, and so, the system should account for that situation. Therefore, a future version of the system should have the capacity to evaluate the student at an early stage of the session, and determine the

student's level of expertise. The evaluation could be carried out by a method similar to binary search. Under this approach, the system would begin presenting a problem of medium level of difficulty, and, depending on the student's performance, the system would jump up or down in the sequence until the level of knowledge of the student is identified. Then, the student can advance through the remaining levels towards the most difficult problems. With this method, the system would demonstrate a higher level of adaptability.

A second issue regarding the expert is the process of diagnosis. During the diagnosis, PerTutor has access to a pre-established catalog of errors that can occur when solving percentages problems. Errors such as multiplying instead of dividing, or inverting the order of the operands are contained in the catalog. This idea is similar to the one explored by Brown and Burton in their BUGGY project (1979). In contrast to the BUGGY project, PerTutor is not capable of cataloging new bugs; therefore, the diagnosis is solely based on the information that has been initially recorded in the catalog. The drawback of this implementation is that all possible mistakes have to be anticipated, and that the system needs a catalog of bugs for each type of problems. An improvement in the process of diagnosis would be to provide the expert with more general knowledge from which to derive and recognize possible mistakes. As is common in expert systems, this knowledge would be contained in a set of production rules intended to determine the difference between the expert's solution and the student's solution to the problem. The specific set of rules would have to be investigated for a future version of the system.

In PerTutor, we also explored the student model. The system represents the student model as the history of the student's interactions through the session. This implementation gives a significant amount of information that the tutor can examine when deciding how to direct the student in her learning. It is our belief that this implementation is also suitable for future versions of the system. It is the tutor's task to make a good use of the information recorded in the student model.



The tutor module in PerTutor is capable of responding to a variety of situations. The strategies that it uses depend on the history of the student, on whether the answer to the last problem was right or wrong, and, if it was wrong, on the frequency of the error. In general, the tutor's explanations are clear. Nevertheless, the tutor could be enhanced by extending its data base of possible answers and statements, so that it can display more versatility during its interaction. Additionally, to make the system even more clear, the tutor could use graphical techniques to give the explanations. For example, it could use instruments such as graphic bars and charts.

Wiebe (1986) proposed several graphical instruments to teach percentages. One of the instruments contains two rulers, a regular ruler and an elastic ruler. The regular ruler represents the whole and the part, and the elastic ruler represents the percent. The elastic ruler is used to calibrate the whole into 100 equal parts. Students can stretch the elastic ruler and align 0% with 0 and 100% with the whole in order to obtain the relation between the whole, the percent, and the part. In figure 24 we demonstrate the functionality of these rulers when the question is how much is 60% of 50. We think that this method is clear, and that it could be adapted for a future version of PerTutor.

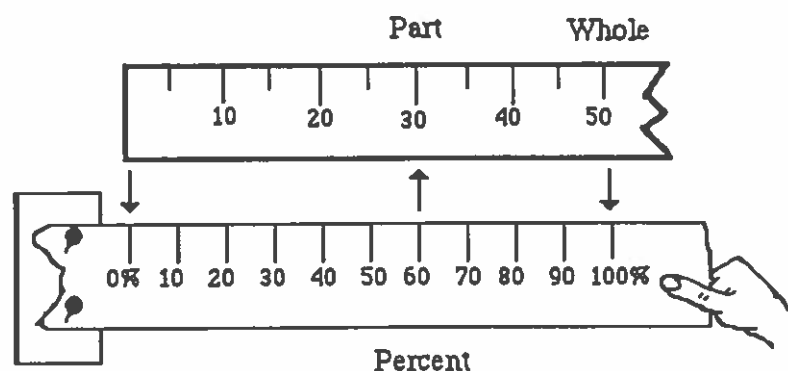


Figure 24. Elastic ruler for percentages.

Another aspect that should be considered for the tutor module is the environment in which the tutorial session will be carried out. The current environment is clear and easy to use. But, a possible improvement would be to develop an environment that would intrinsically motivate the student to solve the problems. A game environment could provide such motivation. After observing that many percentages problems take place in money-related situations, we consider that a challenge sales game could be appropriate. The basic framework of the game would be a market in which the student would visit different stores and be offered different deals. The game would contain levels of complexity just as percentage problems do. Under this environment, the tutor would yield some of the control of the session to the student, so that the student can choose which stores to visit. This kind of environment would make the system more attractive (and fun) to the student.

In the case that a new form of interaction between the student and the system is added, the system will require some changes in the user interface. It would contain pictures and graphics to represent the market scenario, and it would require characters to represent the student and the tutor. The user interface of the current system would be embedded into the new sales environment.

After proposing all of the above improvements to PerTutor, this project might seem very ambitious. In fact, it is. All of the above modifications and extensions would require a great deal of effort, but they would increase significantly the effectiveness of the system. Nonetheless, some of the most relevant aspects of the project have already been explored in one way or another. The level of flexibility shown by the first version of PerTutor is certainly promising. Perhaps, the most important lesson that we have learned from this experience has been that the development of high quality educational software is an ambitious enterprise, but it is possible.

## APPENDIX

## SOME DETAILS ABOUT THE IMPLEMENTATION

To better understand how PerTutor works, we will present and discuss several general aspects of the implementation, as well as the most important data structures. Figure 25 describes the general data flow in PerTutor.

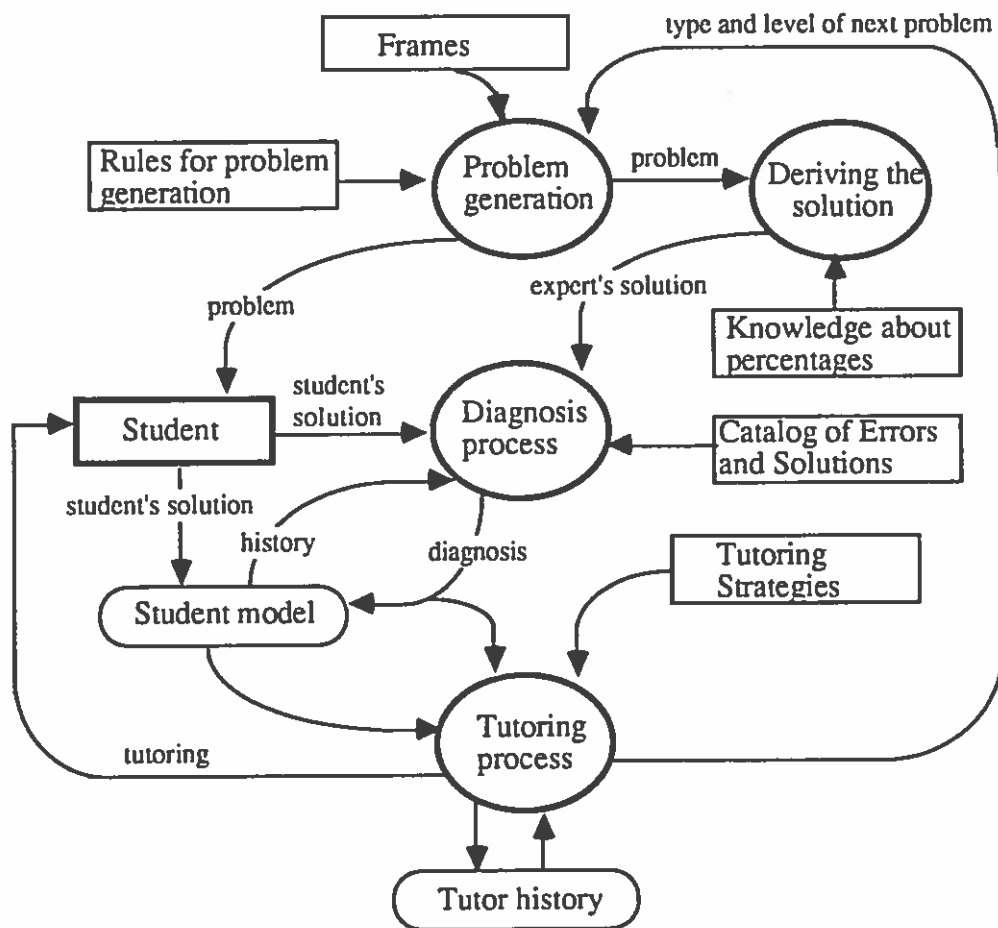


Figure 25. A general data flow in PerTutor.

As we can observe, the problem generation module has access to a file of rules, and a file of frames or templates for the problem statement. The file of rules contains the rules for the generation of problems discussed in chapter III. The frames file contains the text for the problems, the subject and the object involved in the statement, and the range of values for which the statement can be appropriate. To generate a problem, the problem generator module also requires the type and level of difficulty of the problem to be presented next. These are chosen a priori by the tutor module. The system starts by presenting problems of the lower level of difficulty. The data for each problem is recorded through the following data structure:

```

type
  problemData = record
    percent,      { the percent }
    whole,        { the whole or total }
    part : real   { the part }
  end;

  problemInfo = record
    problemType,      { type 1, 2, 3 or 4 }
    generationRule,   { rule applied to generate the values }
    problemLevel,     { level of difficulty of the problem }
    problemFrame: integer; { frame number }
    data : problemData { the actual values used in the problem }
  end;

var
  problemsHistory: array[1..maxProblems] of problemInfo;

```

The data for the problems is kept in an array structure due to the facility that such structure provides to retrieve the data when making reference to a previous problem in the session.

The process of diagnosis begins after the student has given the solution to the problem. This process uses the student's solution, the expert's solution, a catalog of possible errors and solutions, and the student's history. Both the student's and the expert's solutions are composed of the list of operations (steps) and the final answer. The

operations given by the student during each trial are recorded in a linked list that has the following structure:

```

type
  stepPointer = ^userStep;
  userStep = record
    optor: char;      { the operator }
    opnd1, opnd2,    { the operands }
    result : real;   { the result }
    clauseId : integer { identification of the operation after diagnosis }
    next : stepPointer { pointer to next operation }
  end;

  operationsList = record
    first,           { pointer to first operation in the list }
    last : stepPointer { pointer to last operation in the list }
  end;

```

Notice that, in addition to the data given by the student, the userStep structure includes a field for the diagnosis of the operation. This field will be better understood after discussing the data in the catalog of errors and solutions.

Since most percentages problems can be solved in at most two steps, the catalog of possible errors and solutions is a list of two step combinations of operations. In the system, each catalogued operation is known as a clause. Table 7 gives examples of possible clauses for problems of type I. A combination can be formed by clauses 1 and 2, and other combination by clauses 7 and 12. The first combination gives a correct solution, whereas the second gives a partially correct solution to a problem of type I. An actual combination in the catalog is formed by references to the possible clauses, which are contained in a separate file (not shown in the diagram of figure 25).

During the process of diagnosis, the system first tries to identify each of the steps given by the student. Then, it looks for a one step solution, a two step solution, or a partially correct solution. Occasionally, the diagnosis module needs to obtain information from the student model. For example, when the student gives a correct answer without showing intermediate steps, the expert checks the student model with the purpose of

determining if the student could actually know the steps; in that case, the diagnosis indicates either that the answer is correct or that the student still needs to show the steps.

TABLE 7: Examples of possible operations or clauses for problems of type I

Clause Code	Operation	Error
1	percent / 100	No error
2	clause1 x whole	No error
7	percent x 100	Incorrect operator
12	clause7 x whole	Error is in clause 7

The data structure that represents the student model is central to the system. It contains data from the student's interactions during the solution of each problem. Similar to the structure containing the history of the problems, the student model is represented as an array of records. Each record in the array contains the steps used by the student when attempting to solve the problem during each trial. The representation of the student model as an array facilitates traversing the structure in any direction, as well as obtaining the data of a particular problem when the tutor wants to make a reference to a previous problem.

The data structure that comprises the student model is as follows:

```

type
  problemPerformance = record
    studentSolved: boolean;           { indicates whether or not the student
                                      solved the problem }
    trials: array[1..3] of trialPointer { data of each of the trials }
  end;

var
  theHistory: array[1..maxProblems] of problemPerformance;
```

Figure 26 gives a graphical description of the problemPerformance record. The record indicates whether or not the student solved the problem, and it contains the data obtained from the student during each trial.

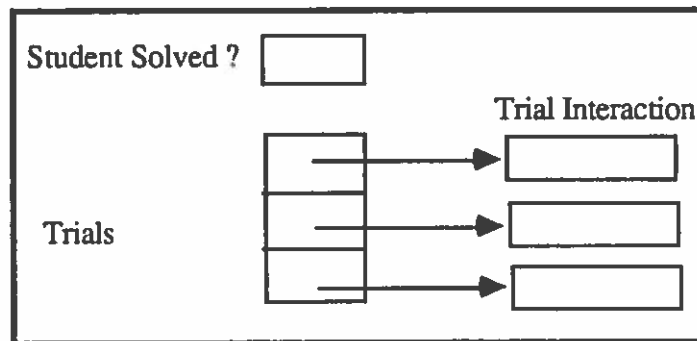


Figure 26. Student's interaction in a problem.

For each trial, the system records the solution given by the student; that is, the intermediate steps and the final result. Since each student interaction is related to a diagnosis and to a response from the tutor, we have decided to embrace in the same data structure the information provided by the student, the corresponding diagnosis, and the reaction of the tutor. Therefore, the student model and the tutor history shown in figure 25 are actually represented within the same data structure. We have chosen to do so in order to avoid duplication, and to facilitate making references to the data in the program.

Figure 27 gives a graphical description of the information recorded for each trial. The student's operations correspond to the description of the operationsList structure given above. The student's answer field represents the student's final answer to the problem during the trial. The diagnosis fields contain the corresponding identification of the diagnosis rules that fired during the diagnosis process. At most two rules can fire, one that

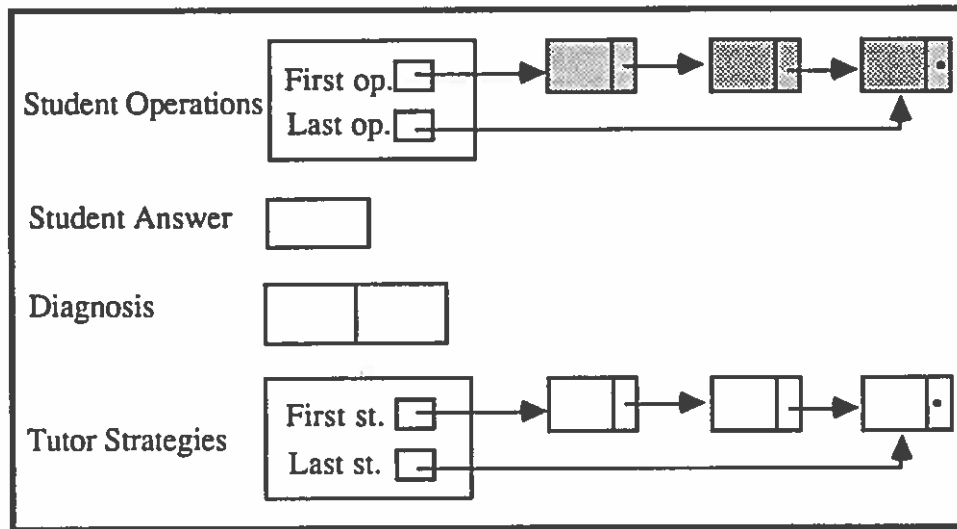


Figure 27. A trial interaction.

indicates the actual error, and a second that indicates a side effect; for example, the diagnosis could be wrong operator, and a side effect could be negative answer.

The reaction of the tutor is represented as a linked list of strategies. It is composed of the strategies discussed in chapter III. They include the actions taken by the tutor and the messages given to the student. The data structure that represents a trial interaction is as follows:

```

type
  diagnosisType = array[1..2] of integer; { to hold the identification of the fired
                                          diagnosis rules }
  strategyPointer = ^strategy;
  strategy = record
    code : char;           { identification of the strategy }
    message : integer     { identification of the displayed message }
    next : strategyPointer { pointer to next strategy }
  end;
  strategiesList = record
    first,                { pointer to first strategy in the list }
    last : strategyPointer { pointer to the last strategy in the list }
  end;

```





## REFERENCES

- Anderson, J.R. (1988). The expert module. In M.C. Polson & J.J. Richardson (Eds.), Foundations of intelligent tutoring systems (pp. 21-53). Hillsdale, NJ: Erlbaum.
- Blohm, P.J., & Wiebe, J.H. (1980). Effects of representation and organizational features as text on success in mathematical problem solving (Report No. CS-005-735). Sanasota, FL: American Reading Conference. (ERIC Document Reproduction Service No. ED 195 930)
- Bregar, W.S, Farley, A.M., & Bayley G. (1986). Knowledge sources for an intelligent algebra tutor. Computer Intelligence, 2, 117-129.
- Burns, H.L., & Capps, C.G. (1988). Foundations of intelligent tutoring systems: An introduction. In M.C. Polson & J.J. Richardson (Eds.), Foundations of intelligent tutoring systems (pp. 1-19). Hillsdale, NJ: Erlbaum.
- Burton, R.R., & Brown, J.S. (1979). An Investigation of computer coaching for informal learning activities. International Journal of Man-Machine Studies, 11, 5-24.
- Gagné, R., Wager W., & Rojas, A. (1981). Planning and authoring computer-assisted instruction lessons. Educational Technology, 21, 17-26.
- Glatzer, D.J. (1984). Teaching percentage: Ideas and suggestions. Arithmetic Teacher, 31(6), 24-26.
- Goldstein, I.P. (1979). The genetic graph: a representation for the evolution of procedural knowledge. International Journal of Man-Machine Studies, 11, 51-77.
- Miller, J.R. (1988). The role of human-computer interaction in intelligent tutoring systems. In M.C. Polson & J.J. Richardson (Eds.), Foundations of intelligent tutoring systems (pp. 143-189). Hillsdale, NJ: Erlbaum.
- Ohlsson, S. (1987). Some principles of intelligent tutoring. In R.W. Lawler & M. Yasdani (Eds.), Artificial intelligence and education: Learning environments and tutoring systems (pp. 203-237). Norwood, NJ: Ablex.
- Polson, M.C., & Richardson, J.J. (1988). Foundations of intelligent tutoring systems. Hillsdale, NJ: Erlbaum.
- Sleeman, D., & Brown, J.S. (1982). Introduction: Intelligent tutoring systems. In D. Sleeman & J.S. Brown (Eds.), Intelligent tutoring systems (pp. 1-11). London: Academic Press.

- Sleeman, D., & Hendley, R.J. (1982). ACE: A system which analyses complex explanations. In D. Sleeman & J.S. Brown (Eds.), Intelligent tutoring systems (pp. 1-11). London: Academic Press.
- Yasdani, M. (1987). Intelligent tutoring systems: An overview. In R.W. Lawler & M. Yasdani (Eds.), Artificial intelligence and education: Learning environments and tutoring systems (pp. 183-201). Norwood, NJ: Ablex.
- VanLehn, K. (1988). Student modeling. In M.C. Polson & J.J. Richardson (Eds.), Foundations of intelligent tutoring systems (pp. 55-78). Hillsdale, NJ: Erlbaum.
- Wenger, E. (1987). Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge. Los Altos, CA: Morgan Kaufmann.
- Wiebe, J.H. (1986). Manipulating percentages. Mathematics Teacher, 79(1), 23-26.
- Woods, P., & Hartley, J.R. (1971). Some learning models for arithmetic tasks and their use in computer based learning. British Journal of Educational Psychology, 41(1), 38-48.