A Survey of Intelligent Tutoring Systems and the Methods Used for Effective Tutoring

Fred Zelhart
Eugene Wallingford

Intelligent Systems Laboratory
Department of Computer Science
University of Northern Iowa

May 13, 1994

DRAFT

Table of Contents

Abstract

This paper shall be a broad survey of the field of artificial intelligent tutoring systems (ITS), the techniques used, and how to apply these techniques to create effective tutoring systems. Section 1 is a discussion of what makes an effective, useful tool and what has been done as far as current experimentation in different tutoring strategies. Section 2 describes some current systems by prominent researchers. Section 3 covers some quality combinations of the criterion. The final part of this paper describes a proposed combination of the criterion to make an effective tutoring system.

1 Qualities of Effective Tutors

The best model for tutoring systems is, of course, today's educational system. Education of today seems to be based on five criteria: teaching style, limited domain, feedback, student interaction, and help style. It is important to be aware that these five areas interact with each other. There are many different combinations when using the techniques of each area. Changing one area without corresponding changes to the others will more than likely result in a worse system (see discussion at end of this section).

1.1 Limited Domain

A limited domain allows the system to most effectively meet the needs of the user. Instead of trying to cover the domain of math, covering addition is a realizable project. There are only so many skills to teach about addition to provide a student with sufficient skills for everyday use. There are also only so many conceptual misunderstandings a student will have in this domain. The student may not understand multiple digit addition or may not know that 2 + 2 = 4. These problems are limited and can be recognized more easily than presenting a beginning student with trigonometry and having the system figure out if the problem is in the use of math skills or in only the trigonometry skills.

A teacher is also more adept at teaching his/her area of specialty than an outside area. For example, it is difficult for a researcher of database systems to give quality problems and information to students in a networking course. By limiting the subject area, more in-depth information can be covered. It is very difficult to stay abreast of all changes in all areas, by limiting one's self, one can focus on the area

intended. A focused teacher has more information about a specific subject and can therefore talk in specifics as opposed to generalities about this area.

A limited domain also limits the programming required to complete the system. By not teaching all of geometry, but only triangle congruency, more quality problems can be supplied. Problems can be more complex as many of the programming tasks will overlap across the problems. Experienced programmers will take advantage of reusable code as often as possible. If a system were going to cover, for example, both algebra and geometry, much programming would have to be done twice. For example, the modules that reason through the students understanding of the subject would have to be done over, the student interface would have to be rewritten, and the expert that solves the problems that the student poses would also have to be redone. The programmer would either have to do twice as much work, or sacrifice parts of the systems to poor programming to get the job done in a reasonable amount of time. Either way, the end result is a poor quality system.

1.2 Teaching Style

The style of teaching is the way that the system imparts information to the student. The techniques include telling the student what is wrong, allowing experimentation, testing, and coaching. Sometimes these techniques are used together within a single system. It is important to remember that no one style makes an effective tutor. Instead, it is the combinations of this with the other areas in effective ways that get the student to learn.

Several early systems were designed only to tell the student what to do. A problem would be presented and the student would work the problem. The system would then tell the student what was incorrect. The only other form of interaction between the student and the tutor was when the tutor told the student how to work through the problem. The telling style of teaching is similar to the conventional lecturing style of teaching. The student is given information to learn, if there is some form of interaction, the student may be told why their solution is incorrect, but more than likely, will only be told if the answer is right or wrong.

Another approach is to “coach” the student. This style allows the system to act much more like a real teacher. These types of systems give the student a problem and then “help” the student with the problem. When a false assumption on the part of the student is made, the system will attempt to correct the problem. This correction may come in the form of an explanation or as suggestion. For example the system may suggest a different action to take than what the student tried.

“Learning environments” provide everything that the student needs to explore the domain in which they are working. The student is either presented with the problem or they may create their own problem to solve. The user is allowed to

"play" with the state of the problem to see what will happen when different variables in the problem are changed. This allows the user to guide the learning; sometimes giving the student a greater sense of accomplishment. For this reason, many learning environments are combined with the coaching style of teaching to keep the student from getting too far off track. Exploration should be permitted in the learning environment, but the student must learn the correct information eventually.

The final teaching style is that of testing. Very little learning of new skills takes place with this style of teaching. It is used instead to further develop the current skills of the user. The user is presented a test to find out how much they understand the concepts to date. These tests are different from the "telling" style in that explanation of the correctness of a solution is not given. The student is only told whether their answer is correct or incorrect. The system usually assumes that the student has skills to work the problem, but that these skills are underdeveloped. This style is often the most intimidating to the student as these systems are usually not very user-friendly and have very little student interaction. This teaching style is often referred to as "drill and practice."

1.3 Feedback

How effective a system is depends heavily upon its feedback timing and style. Timing refers to when the student is given a response to the solution. When the feedback is presented to the student should be governed by what the student knows. For example, in a beginning programming tutor, the tutor may wish to correct all of the syntax errors the student is making as these mistakes are made. As the student progresses and learns more, the feedback about the syntax of the language would be annoying. Instead, the system should present immediate feedback for the higher level concepts that the student is attempting to learn, and delay the feedback for the lower level concepts. Tutors are better than teachers in this respect in that they can provide a student with timely feedback better than most teachers. Currently a human teacher needs at least several hours to correct students' assignments. This is simply because of the volume of the work and the speed at which a human can perform. Since the student will not get an assignment returned for, more than likely, at least one day, the student is allowed to think that the processes used on the assignment being corrected are valid. If this is not the case and the student is asked to do additional assignments, the false concepts will be used on the new assignment also. This causes the student to use the faulty concepts not just once, but twice. This delay of correction develops a sense of frustration within students as they are not told of the incorrect use of the processes until more assignments are completed. In a worst case scenario, the student would not find out what information is being misused until the concept had been used and acquired into the set of skills that the student commonly uses when solving problems.

The style of the feedback is also quite important to the tutoring process. If the feedback is given as natural language, it is much more effective than something that reads "WRONG: by rule #132." The more effective method would be a statement such as, "The answer is incorrect because it has already been determined that Bacterium A is not present (result of blood test #1)." This is something that the student can read and understand. Just as computers need their rule-systems in a format they can understand, so do humans. If a student cannot understand the feedback of the system, the machine becomes as valued as a can of food with no opener. Worthless and a source of frustration.

1.4 Student Interaction

How the student is allowed to interact with the system is of great importance. Whether the student is allowed full or limited access to the control structures of the system, through graphics or only text and if this interaction is "natural" or not are all items to consider *before* building the system. These items determine how the student perceives the computer in educational use as well as how the tutor is perceived. If a tutor is difficult to use, these feelings of ignorance may carry over to other systems if not to computers on the whole; making the user less willing to use not only ITSs, but computers in general.

Systems should not allow the user full access to the control of the tutoring, however, the user should be allowed as much control of the learning process as possible. The student should be allowed enough control so that mistakes can be made, but not permit false assumptions and the learning of incorrect information. Allowing the student to question when he/she wants allows the student to guide the instruction as desired. If a problem arises where the student wants to know why a certain procedure is required, he/she should be allowed to ask as long as this would not interfere with the tutoring process. This gives the best information dissemination to the student. A system may assume that the student knows some information, but should always allow a way to correct its own false assumptions of the student. A question that cannot be asked will leave the student as much in the dark using the tutor as a power-outage would.

The input to the system should be natural to both the user and the type of problem. A problem that can be described graphically should be shown graphically to give the user the best representation of the problem. The student should not have to rely on a purely textual description of the problem even though this is the easiest form of interaction. More information about the problem can be given in a shorter amount of time by using a graphic.

The user should also be allowed to examine the graphic in detail if needed. For example, if the student has a question about a certain part, there should be a way to select that part of the graphic and then query the system as to the workings of the

part. For this type of interaction, the best form for the selection and display of the part would be graphic.

The system should allow the user to query in a natural form. If the graphic can be used to query the system about the tutorial, then it should be used. The system response would most likely be in a text form. Some variations on text are to have the information printed, displayed to the screen, and with current technology, through recorded sound. By interacting with the student through as many means as possible, the tutor will be more effective than just using only one form of interaction. It has been shown that varying the main means of input in human teachers makes a more effective teacher.

Saying this does not mean that all systems should be either textual or graphical in nature. This difference is dependent upon the subject matter being taught and the teaching style being used. No matter what teaching style and subject, the better systems seemed to be able to use a combination of graphics and text to provide information to the user. This automatically gives the user two ways of looking at the same problem. This different perspective will also help the student to learn: "If I don't understand the problem as given, I should try to look at it a different way." This is a very important problem solving skill.

1.5 Help System

The help system will probably have the most impact on whether the student wants to use the system again. If the student experiences difficulties with the system and cannot find a way out, the inclination would be to use an easier tutor next time. If the program could instead provide some assistance, feelings of intimidation would be lessened. People might even prefer to ask questions of the system as it will not judge them by the questions asked. Since it is a machine, the student will only interact with it on one topic. It is a human perception that a computer cannot or will not judge a human, which gives the student the freedom sense to ask questions.

The help provided to the user must be given in a timely fashion, be contextual, and provide correct information at the user's request. Help that is not provided when the user needs it is not really help at all, but a sort of instructional manual to be perused at a later time. Some systems do not need a full help system as there is limited interaction with students.

The help should be provided when the user needs it. This may cause the help system to be tied in with the feedback system so that once the student gets a response to their problem, the system could prompt them to use the help on the subject. This would also allow the help to be contextual: the subject that the student is working

with is where the help will start. This frees the user from searching through a help system for the desired information.

2 Specific Systems

2.1 Previous Systems

Two good systems (or at least they appear to be) are WUSOR-II and SOPHIE. They work with two different levels of students and provide two different working environments, but one can look at their examples and draw conclusions as to what is necessary for an effective tutoring system.

These two systems are given mainly as benchmarks for the examination of other systems. They are not perfect systems in all aspects, however, they are excellent examples of early efforts of ITSs that were very successful in meeting the goal of being an effective tutor. They also show good usage of combining the different parts of each criterion above in interesting ways.

A note of caution: these are observations of systems. To find out what is really required of an effective tutor, one should observe a teacher in action, or observe a tutor for several hours. The work done on these two systems was accomplished by applying knowledge gained through many hours of observational research into teaching. To find out what is truly important, the same must be done by every beginning researcher.

2.1.1 WUSOR-II

WUSOR-II is the tutoring system used in the computer simulation game WUMPUS. Using WUMPUS, students learn logic, probability, geometry, and decision theory by moving through a maze trying to defeat the WUMPUS while still avoiding various dangers of the maze. WUSOR-II was designed by Ira Goldstein and Brian Carr at M.I.T. in the mid- to late-'70s.

The game starts by placing the user randomly in a series of caves. They are told about their cave and the ones surrounding it. This information can be used to make a decision as to where to move next. The problems that a WUMPUS hunter may encounter include pitfalls (fall to a death), bats (randomly place the user in a new cave), and the WUMPUS (eats the user and ends the game). The user is to find the WUMPUS and kill it using the arrows provided. A WUMPUS can only be killed by an arrow shot from a cave adjacent to its lair. The information provided about the cave and its neighbors allows the user to make logical decisions as to where to move next. Warnings are provided in the form of sense data. One can smell the

WUMPUS two caves away, bats can be heard squeaking, and a breeze comes from the pitfalls.

WUSOR-II is a text-based coaching system. Even though the system may have been limited this way because of the technology of the time, a newer version could be written that would be graphical (showing a map of known caves and their dangers). This graphic would allow the marking of caves that are known to be dangerous and ones that are safe to travel into; freeing the student from memorizing which caves are suspected safe and which are known dangerous. The student would only have to consult the map to find the correct answer. The student can do this with the text based system by drawing the map on a piece of paper. However the system is concerned with teaching decision making and logic skills, not drawing skills, and should therefore eliminate this burden from the student.

At first glance the domain may not appear limited, but in reality it is. There are only so many different types of moves that the user can make. Also there are a finite number of dangers that can be realized and programmed in to the system. In actuality the system has no true help subsystem, but the feedback provided by the system provides the user with sufficient help in working with the tool. The tool is simplistic in nature to use anyway, so only a simple manual or teacher's instruction is all that is required for the student to start using the system. This gives younger students access to tools that would teach them early problem solving skills.

WUSOR-II has four modules to help keep track of the session: the Psychologist, the Student Model, the Expert, and the Tutor. The Expert module contains the knowledge used to infer information about the surrounding caves. This module also provides feedback to the system as to whether the student is correctly identifying possible caves to move into. If the Expert knows of a pitfall and the student chooses to move there, the Expert will flag this and the system will respond accordingly. The Psychologist uses the feedback provided by the Expert and the information in the Student Model to get an idea of what the student knows, and what has yet to be learned. This information is used by the Student Model to represent the user. The Student Model is a collection of the rules that the student should be aware of and correctly using. The Tutor is the student interaction module which tells the student what errors in judgment he/she is making and what is correct. The Tutor is also responsible for providing appropriate feedback (rewarding statements as well as critical ones). The Tutor will also take the information from the student and provide it to the Expert module. This continuous cycle, Tutor to Expert to Psychologist to Student Model to Tutor, is the main control structure of the system. Each module provides information to the next to give the user the best possible tutorial.

Some of the problems that WUMPUS has with modeling is that the same student errors may have several different possible causes. For example, the student may try

to move to a pitfall cave not aware that a breeze is the sign of a nearby pitfall, or the user may be a beginner who has not yet learned how to infer which cave may have the pitfall. The Psychologist and Tutor work together to decide if the student does not know a piece of critical information (breeze equals nearby pitfall), or if the student is a beginner lacking the logical skills necessary. These two modules solve the problem using tutoring rules supplied in the program. The problem being that these two modules may have drawn incorrect conclusions about what the student knows, and therefore would incorrectly diagnose the cause of the error. For an example dialog with WUMPUS, please refer to appendix A.

2.1.2 SOPHIE

Developed at Bolt Beranek and Newman, Inc., by John Seely Brown, Richard Burton, and others, SOPHIE (SOPHisticated Instructional Environment) is designed to provide an environment to explore ideas instead of receive instruction. The system permits the user to experiment with ideas for electrical circuits, test them, and debug the models. The system contains knowledge necessary to debug the circuits, and to help the user analyze the circuits being discussed. This means that the student can create his/her own problems to work and still be provided the correct answer by the program.

These student created problems are created to help solve other problems that the system provides. The student may want to run an experiment to find out the results of combining the parts of a circuit in a specific way. This environment permits that and also has the capability to correctly debug the student proposed circuit so that the tutorial is uninterrupted.

The system provides a problem and asks the student to analyze a circuit by finding the fault from the given information and circuit diagram. The system develops a "pre-plan" of what is wrong with the circuit so it can judge if the student is making progress towards understanding the concepts. If the student is just "playing around," the system will interrupt and redirect the work towards finding the correct solution.

To start with, the student is given a graphical representation of the circuit in question as well as textual information about the circuit. This information tells the user what the control settings are for the circuit, and what controls are available for use. The student is then allowed to question the system to the circuit's various attributes, such as voltage at certain points, and the student must provide a "fix" for the circuit. This may be something simple like throwing a switch, or as complex as replacing parts of the circuit. For an example session of SOPHIE, see appendix B.

SOPHIE does allow the user to respond and query in a form of natural language. SOPHIE deals with natural language by skipping words it does not understand and

infers the meaning of the sentence from the remaining parts. If the question is too foreign, SOPHIE will give the student a complete statement rephrasing the input and asking the student if this is what was intended. This way the student does not have to learn an entire new language to use the system. SOPHIE is designed to work around the abilities of the user. In fact, this system would probably be usable even if the user were to speak in slang, just for the simple fact that the system would ask for complete meaning before answering.

SOPHIE is also capable of generating problems that have to do with certain areas if the student needs specific help. For example, if the student is having difficulties understanding parallel circuits and how switches affect these, SOPHIE can generate such a problem for the student. SOPHIE and the student will then work through the problem together to find out where the difficulties lie. This allows the user to control the learning process permitting him/her to focus on these areas of understanding that need the most help.

2.2 Current Systems

This section shall try to explain the current, popular systems GUIDON (including the subsequent development of GUIDON-WATCH), PROUST, and Anderson's Geometry tutor. These systems are not only the most widely known and used systems, but they combine qualities of the criteria in different ways making them very interesting to study. Some of the combinations fall short of their intentions, others manage to work well together.

2.2.1 GUIDON

GUIDON is a system developed by Clancey as an effort to better educate users of MYCIN. Although MYCIN is capable of helping doctors in diagnosis, it is not effective at conveying this information to students in the field of medicine. GUIDON uses the knowledge of MYCIN to give the users cases to solve, and to get solutions for these cases. GUIDON will answer questions and provide reasons explaining why a specific answer is correct or incorrect. An advantage of GUIDON is that it was designed to "shell out" the concepts of teaching. The theory was that any program that was set up similar to MYCIN could use GUIDON to teach its diagnosis processes. Factoring out the knowledge necessary only for tutoring gave this system a base in teaching that the others do not have. Other systems must also spend development time on the subject as well as the tutorial.

GUIDON is also capable of remembering student interaction across sessions. What John Doe learns in lesson one will be remembered for the next time that John Doe uses the system; keeping users from being treated as beginners every time a new session is started. Users are allowed to advance naturally just as if the system were an actual organic tutor. This allows the system to provide better feedback for the

user. If a student is consistently missing a concept across patients, GUIDON is capable of diagnosing this and guiding the student to a case that will make them work on this particular skill. By keeping track of what the student has already learned, the system could also refer the student to a previous case which would help to explain the concept.

The tutoring session is managed through two different sets of rules: a set for discourse and one for tutoring. Rules of discourse (d-rules) are those rules that define how to interact with the natural language of the student. These are the rules that will accept the student's questions and define them appropriately for the system. Tutoring rules (t-rules) define the differences between forgetting an item and not knowing it. These are also the rules that will suggest to the student other ways of doing things. For example, if the student has chosen to do a biopsy on a patient and one is not required (or if only an x-ray is needed), these rules tell GUIDON to inform the student of the misjudgment and ask for reconsideration. This action will then go through the d-rules to be properly phrased for the student. Using these two sets of rules together gives GUIDON great power in the area of student interaction.

Some of GUIDON's shortcomings are addressed by its companion system GUIDON-WATCH. GUIDON is totally text-based, which, initially does not seem a bad idea, but reading straight text can be tedious and it is often difficult to "go back" in such a system. In GUIDON-WATCH there are windows for the student to interact with the system and windows that show what is currently being examined by the system. This allows the student to see what choices are being eliminated as the system works through the diagnosis. This further reinforces the proper methods of diagnosis to which the user is being exposed.

Another of GUIDON's problems is that it is always testing the user. Although testing is the best form to convey this information to the user, it is also the most intimidating to students. Many people do not like being in positions of high pressure and close scrutiny. This is offset by the knowledge that, as a doctor, this is the type of environment the user will be working in anyway, so he/she should get used to it. The ability of GUIDON to "talk" to the user in feedback form is also a great help by getting the user to feel more at ease when working with the system.

GUIDON attempts to use as much natural language as possible, however it is not capable of a full understanding of English. This would be a tremendous undertaking on its own, let alone combined with a tutorial system. GUIDON is capable of understanding several statements and questions. This is done by managing generic dialogs with the student. The system will match the generics, which are stored as rules, to the output and control features of the program. Through these means, the user gets the feeling that he/she is actually working in a natural language.

GUIDON-WATCH's authors found that they were presenting too much information to the user. The original designers allowed for the display of all the information necessary for the debugging of the system. This information would be useful only to the student who also knew about the workings of the system, to others it was merely confusing. Students were not aware of why they should be exposed to this extra information and therefore felt some frustrations with the system. The designers of the system recognized this during the testing of the system with actual medical students. They have since redesigned the work to allow the user to choose from a "designer" and a "user" mode. This way the designers of the system could still examine all of the necessary information without "flooding" the normal users with unnecessary information.

GUIDON-WATCH also performs window management on its own. The designers found that this helped by eliminating tedious tasks for the user who really should not be troubled with the extra work. The system manages the display of text and graphics in resizable windows as well as allowing the text to be "scrolled." What windows were displayed and where they appeared was also a task of GUIDON-WATCH. If a process will help the student to learn better, allow the student access to the procedure, but do not force extra work upon the learner.

The major drawback preventing many programmers from writing windowed versions of software is the number of hours spent writing the code to do the work of the windowing system. Many programmers have found that programming with graphics is much more difficult than using a text based system. No longer can the system just display information; it must now show the information the way the user wants. This could result in miniature windows as well as full screen ones that have more room to display the information in them than they need. What used to be just a bunch of sentences on the screen with a specified number of characters per line is now a variable width font displayed in a sizable window. These changes have to for when designing the system. GUIDON-WATCH took on even more by having the diagnosis process be graphical as well. This management of graphics takes up by far more time than most people would originally wish to put into a system. This team of programmers seems to have done quite well with their task.

2.2.2 PROUST

Designed by W. Lewis Johnson, PROUST is a program that is for use in helping novice programmers in debugging their systems. The program takes in a PASCAL program and then proceeds to analyze it for errors. These errors are based on what problems most novice students have in understanding the syntax of PASCAL. The main idea behind the system is that it would take away the problem of dealing with the syntax of the language from the student. The student would only have to be concerned with applying the concepts that are being taught in the class, not the concepts inherent in the language.

When the system is fed a program, it tries to determine what the student intended the program to do. For this purpose, the system has descriptions of the problems that the student will have to solve, must be capable of decomposing the goals and hypotheses of the student, and has control strategies to work around encountered problems that are not currently accounted for in the programming.

PROUST is capable of disambiguating variable names and procedures by making multiple passes of the program. This allows PROUST to find all variable names and types before trying to use them in the analysis of the entered program. It will wait to match any of the patterns with the program until this initial scan is done. This appears to help in the diagnosis of bugs that are often encountered in systems. Earlier systems often tried to disambiguate on-the-fly which caused further misunderstandings on the part of the system. By performing an initial scan, the whole program is analyzed instead of only analyzing each part separately.

One of PROUST's drawbacks is that it does not provide feedback to the student until the program has been run. The student has already typed in the program to be evaluated. If the system already has knowledge about the problem and the expected patterns, it should also be capable of interacting with the student to diagnose the problems immediately. This would teach the student correct programming procedures immediately, not through delayed feedback, which may cause the student to make incorrect assumptions about the programming techniques. The authors of the system acknowledge this problem and have stated that they wish to incorporate the current system with a better tutorial.

This expectation goes beyond the original intentions of the system as it was designed to accept input in the fashion shown, but this is not an effective feedback method. Yes, the student is given the feedback within a reasonable amount of time, however, feedback is not presented until all errors have been made. The student must instead wait until the complete analysis of the program is done.

The program assumes that the student will be willing to read through the analysis of the program, an example of which is printed in Appendix C. If the program is complex and erroneous, this output listing could be lengthy. The information is also displayed to the screen causing the user to write out what is wrong with the program so that he/she could go back and correct the errors. Many persons will not be willing to write out the errors and then go back and fix them. The program should be capable of providing a means to the student so these errors can be displayed while editing of the program. This way the student does not have to keep track of the extra information. If the purpose of PROUST is to take away the syntax concerns from the student, it should also take out any lower-level stumbling blocks such as editing and correcting in a complex environment.

It should be noted in fairness to the system, the authors do say that several students using the system showed significant improvement over past performances. Even though the system is not exemplary, it does still teach students the needed information. The problem being: it could teach more information better. This is why the combinations of the criterion are so important.

2.2.3 Anderson's Geometry tutor

Anderson's Geometry tutor is an excellent example of how to write an effective tutorial system. There are multiple ways of presenting the student with the information, it deals with a limited domain, and appears to supply quality, timely feedback that is of a well formatted nature. The problems supplied also seem to be of a nature that effectively convey the information to the student.

Anderson started work on this and a similar system at comparatively the same time. He also applied this information in local educational institutions. The LISP tutor was designed for use in beginning programming courses at Carnegie-Mellon University. The Geometry tutor was used in local area high schools. Anderson observed how these classes were being taught to get a basis for his designs and applied the tactics used by some of the teachers observed.

The Geometry tutor displays a triangle congruency problem for the student in a frame. The goal is supplied at the top of the screen with the knowns being shown at the bottom. This gives the appearance of working "upwards" towards the goal. The proof for the problem is displayed as the student works through the proof in the remainder of the screen. As the student works through the proof, the diagram associated with the proof is updated. The system also updates the display of the parts of the proof to include "visual" links between the parts and the rules that were applied to derive the information. For example, if the student were able to prove that two sides were equal because their angles are equal, the display would update to show the congruency and lines would be drawn from the two parts to the rule (angles equal) to the new information (sides equal). Appendix D shows a mock-up of this interaction.

This display gives the student several ways to view the problem and keeps the information up-to-date. Since the system updates the picture for the student, there is no chance for the student to mismark the problem causing the student to draw false conclusions. Many students will draw diagrams inaccurately, or mislabel the information; since this task is taken away from the student, the likelihood of the student incorrectly working the problem due to drawing errors is lessened to zero. This is a great example of limiting what the student must do to what they need to learn. By seeing the graphic updates, the student will learn the relationships without being bothered by the tedious tasks.

Command functions are accessed through the menus of the program. These functions are the parts of the proof language of geometry. By selecting operands for the command, the program will evaluate if these operands are valid for the command function chosen (i.e. trying to prove two sides equal and selecting only one side causes an error). If the operands are valid, the system will attempt to perform the function. After the user inputs the conclusion and it is verified, the graphical display of the problem and the display of the proof are both updated.

The system provides feedback to the student based on correctness and applicability to the problem. For example, if the student were to try and prove something that would not help him/her reach the goal, the system would respond by saying that he/she is correct, however, that would not help with the problem. This type of feedback is very important for two reasons: reward and redirection. The system informs the student about the validity of the response, providing a sense of reassurance in understanding the concept. The feedback also redirects the student down a new pathway so roadblocks and feelings of frustration are not encountered. The elimination of frustration and acknowledgment of correct work keeps the student wanting to use the system and in turn makes them want to learn the subject. So that the user can see the benefits of the work done without having to perform additional tasks.

The student model used in the Geometry tutor keeps track of concepts that the student should currently understand. The student should understand any concept that has been correctly used. With this situational knowledge, the tutor tailors its feedback to fit the knowledge of the student (similar to GUIDON). If the student has proven that a concept is known yet it is not applied, then the feedback will be of a form reminding the student of the information. If the student is unsuccessfully applying a concept, the tutor will go ahead and provide help to the student about the process. This keeps the student from feeling intimidated about asking for help, yet still correctly identifies the cause of the error.

The student is also allowed to ask the system for help on certain items. For example, if the meaning of a known is not understood, it can be selected and the help system queried. The help that the system provides will automatically go to the topic that has been selected. This “context sensitive” help has proved to be more successful with users than a system where the user searches for the desired information. Sometimes the topic is not known, and therefore the list of supplied topics in the help does them no good when searching. If the user could instead select the item that is unknown, and let the computer search for the information, future use of the help system would be much more likely.

This is important if the user is being introduced to the system. If the help is intimidating to the student, the whole system will be intimidating. This is similar to the student who is afraid to ask questions of any teacher, only because one teacher

was unable to provide an acceptable means of assistance. ITSs should make up for deficiencies of teachers, not emphasize that which is already unacceptable. Just as with human teachers, the criterion have to be balanced carefully or the whole system falls apart.

The tutor lets the student work as he/she sees the problem. The user can work from the goal down to the knowns, from the knowns to the goal, or mix these two methods as needed to prove the problem. This freedom alleviates many of the student's fears about proofs. For example, having to write the proof on paper in an organized fashion keeps many students from concentrating on the proof at hand. Emphasis is placed instead on trying to go directly from one end to the other. Today's textbooks do not help much by showing proofs in one direction only. This system frees the student from the worry of a "correct looking" proof. The system will take care of the display, the student need only be concerned with the learning. Skill acquisition is more important than learning how to write the proof. Students can always go back and re-write their solutions at a later date in a fashion acceptable to a teacher. The important item is that the student learn how to do a proof. Let learners at least learn how and have some success at the problem solving techniques before being loaded down with more items than they are ready to handle.

The primary driving force behind Anderson's systems is that of model-tracing. Model-tracing tries to predict what the student is going to do based on what the student has done in the past. The models do allow for "buggy" rules so that the errors of the student can be more accurately predicted. The "buggy" rules are the rules showing what the student may do wrong. These help the system explain why the student is incorrectly applying concepts so that better feedback can be given to the user.

The system does not account for time lapse on the part of the student, expects the student to know some of the domain before working with the system and is unable to work with multiple causes for the same problem. The system should be able to deal with the student response within a certain amount of time. If the student is taking too much time to work through a problem, the system should interrupt the student and ask if some difficulty is being encountered and offer assistance. This would keep the student from getting "hung up" on items that are not of great importance. The main item that the system (and its designers) must always consider is that the student is less likely to ask for help than anyone ever plans; students do not like to be told that they are wrong, therefore the system must predict and work with the student; anticipating the problems and allowing the student to work through a solution with the system. The Geometry tutor allows the student to make errors and supplies feedback based on these errors, however it does not anticipate that the student may be contemplating the problem, or that the student may have no clue and be afraid of asking for help. Without this consideration, it is possible that the system will sit in a loop forever.

The system's expectation of domain knowledge can be overlooked as this system is intended to be used in a classroom setting where the student will have been given previous instruction and access to a teacher who would be capable of answering the student's questions. Before running any tutorial system, the student is usually expected to have some sort of beginning knowledge. The knowledge expected of the beginning user should not be so great as to cause impedance to the learning process.

3 What would be the Best System?

A type of "ultimate" tutoring system will now be discussed. No specifics will be given for this system, it is intended to show some ways of combining the criterion to come up with the most effective system possible. This is not necessarily a realizable system in the immediate future, however it will be used as a template for the development of the system described in section 4.

3.1 Teaching Style

It seems, for the systems surveyed, that the most popular teaching style was that of the coach. The coach is more like an organic teacher, it is capable of inferring students' mistakes, and has a "built-in" style of feedback that is conducive to education. The coach is usually combined with a learning environment to allow student control of the learning. The learning environment should not be so free as to allow the student to wander to a completely new path that will have no benefiting consequences, but they should be permitted and even encouraged to explore the subject being taught.

The coach responds to the student in a style of discourse that is familiar to the student. The coach will provide some instruction to the student so that the problem can be worked, but the coach will not just give the answer to the student; the student must still work the problem out. If the student encounters a stumbling block, the system can be queried to find the correct route to travel next. Coaches are normal styles for this type of education. They provide encouragement to students while keeping them on a track towards a correct solution.

The learning environment style needs to be coupled with this type of teaching style so that the user is allowed to control his/her learning. This takes the user from the passive side of learning to the active side. People are more likely to learn information if they are actively involved in the process. Learning environments give this outlet to the user. The user can explore and find solutions alone, or with the help of the system. The system is responsible to model the problem state so that the correct information can readily be displayed. This relieves some of the burden from the user.

Learning environments are also more widely applicable as there are several areas where a "living experiment" would not be possible. If the student is learning how to control a nuclear reactor and needs to become familiar with the process of emergency shutdown due to core breach, many people would be more comfortable with this education taking place as a simulation with a computer. The consequences of simulating a core breech in real life with a working reactor could be dire if the student did not properly shut down the reactor in time. The learning environment provides a reactive situation to the student simulating an actual occurrence. This simulation is easier, more controllable, and is maintained without the student, allowing more concentration on the task to be learned.

3.2 Student Interaction

As the reader can infer from the above, several means of student interaction should be available and the student should be allowed as full access to the system as possible. These not only draw the learner into the educational process, but also provide a feeling of worth to the learner by controlling the state of the process. To sit idly by and just absorb information is boring (the reader need only remember 100 person lecture halls where no interaction is permitted). The learner should be an integral part of the learning process, not just a tool used to get a paycheck.

Part of the problem representation or problem solution should be graphical in nature. This gives the user another means of looking at the problem and gives them more information in less space. As stated above, a picture about something may convey information in a way that the student would not otherwise get from a textual representation of the problem. If this picture is also used in the solving of the problem, the student will feel more comfortable using the picture to represent problems in the future. The student will also get a "hands on" feeling for working with the problem. By manipulating the picture, the student may get a sense that they are actually working with the type of problem being simulated on the computer.

With the abilities of current technology, the system should also provide a means of reading the text to the student. This allows more students access to the system. No longer is fast reading required for the student to work problems. Now the problem can be spoken. This will also affect the student by encouraging him/her to read along with the spoken problem representation, possibly increasing his/her reading and comprehension skills. It would of course also be a prerogative of the system to say that the student must now read the problem without the sound. If the student relies too heavily on the spoken input, reading and comprehension skills will never be learned. This, like many things in learning, is a difficult balance to achieve, much less program.

The student should also be allowed to query the system about the solution at any time. The student should be allowed to query as far as the system is capable of understanding. Without this capability, the system will leave many open holes for the teacher to fill in for the student. Yes, teachers should still be an active part of the learning process too, however they should be used as resources, not assignment aids. The computer should be an enhancement to the classroom, not a hindrance. If the computer leaves too many questions for the teacher to answer, the teacher will no longer wish to use the system. Instead, some other form of educational interaction will be used.

In GUIDON, natural language would be the choice of interaction when the system needs to accept input from the student. This permits a natural interface between the machine and the student. If the student must spend extra time learning how to use the machine, then educational time that could be spent tutoring is lost. The system should of course dialog with the student in the natural language also. The student should not be forced to learn a new way of reading just to learn. Learning should be an extension of the skills and concepts that are already known, not a totally new experience to be thrust into with no means of comprehension.

If there are multiple ways of looking at the problem (i.e. windows or multiple views of a picture) the system should allow the student to select the one that is wanted. The system should not shove a viewpoint onto the student, but should permit natural action and reaction. If the student wants to look at the problem "backwards" from the way that the system was programmed to examine it, let him/her. There is a time, as always, when the student must conform to certain standards, however if the problem can be conceptualized in a different format that makes the problem solving easier, then let it be realized on the machine.

Along with this, the machine should manage as much of the tedious tasks as possible. If there are multiple views of the problem, the student should be allowed to select the ones that are desired to be seen, but the system should manage the relationships of the graphic within the display and the text within areas, so that the information is in a readable format for the user. There is such a thing as too much freedom when learning. If the user is trying to do things that would end up ruining the tutorial session, the system must recognize this and not permit it. Otherwise, the student would only be playing a game with the machine, and not learning from the tutor.

Something that has not been considered with some of the systems surveyed is to have the interaction equivalent to the skills of the user. For example, if an elementary student were to be working with the mathematics tutor, it would be wise to allow the interaction to be through a touch screen. Small children do not have the dexterity, nor the physical size to adequately cope with a normal keyboard.

Giving them interaction through touch-screens would allow something more natural for the learning process.

If the user is going to be manipulating the environment with a mouse, allow many of the common routines to be programmed through the mouse. For example, if there is a list of commonly used functions, this menu appears when the user clicks the alternate mouse button (usually the right mouse button). This would display the menu for the user, and the function would then be chosen. This type of interaction takes away the tediousness of trying to locate the desired command from a series of menus on the screen. Instead, the common menu is brought up with a "flick of the wrist."

The interaction with the user should also be relatively quick. If the user must wait three minutes for the system to work through the sentence of the query, by the time the information is received, the user will be tired of waiting. The user could have instead gone to the instructor and arrived at the information much more quickly. The system should also reason through problems in a reasonable amount of time. Since the teaching style of choice is the learning environment, the system should be capable of reasoning through the problems that the student is going to pose to the tutor. If the tutor takes too long to reason through the student's methodologies and problems, the student will again be tired of waiting and not wish to use the tutor for education again. Just as timely information is required, so is timely feedback.

3.3 Feedback

The feedback should be concise for the user and timely in nature. As soon as an error is detected by the system, it should inform the user of the problem, and try to provide a means for the user to find the correct answer. These are principles pulled from Anderson's Geometry tutor. Reward the student for the correct information and redirect the process of problem solving towards the correct solution. The feedback should not "give" the answer to the user. The user must still work through the problem and find the solution, however the tutor should make this as frustration free as possible. A discouraged student will not wish to pursue the subject further, the tutor should be capable of recognizing this and provide a means to prevent the attitude from occurring.

3.4 Help System

The best form of help that has been observed is that of context sensitive help. This help will search itself for the user to find the information asked for. The user does not have to be familiar with the workings of the system to perform the search. The item desired need only be selected and the help command executed. This frees the user from learning new vocabulary from the outset of the use of the system. If the user must learn new words just to learn how to use the system, the user will again

experience feelings of frustration. The user will learn the vocabulary while using the help system. If organized correctly, the system will provide the subject and topic cross reference so that the next time the information is needed, the user can just straight search the help system. This is a form of passive learning, but one that is effective nonetheless. The reader should by now realize that the quality tutoring systems are not just concerned with tutoring, but with supplying educational resources to the user in an accessible and readily available format. If the student has trouble getting to the information, then he/she will not wish to use it.

3.5 Limited Domain

To make an effective and realizable system, the domain of the problem should be relatively limited. This limitation should be with respect to the problem domain and the subject area as well. Anderson's geometry tutor was limited in respect to geometry as well as with the specific limitation of triangle congruency. This permitted the program to have enough information to effectively tutor the student.

4 Plans for a Future System

The system will be capable of complete tutoring for the average student of the domain and subject. To this end, the system will have to be capable of describing the problem to the student, accepting user input, analyzing the problem, representing the student, and providing help to the student. All of these subsystems will have to work together to adequately perform the tutoring task.

4.1 Problem Description

To describe the problem, the system will use a combination of graphics, sound, and text. This allows the student to represent the problem in three different ways. The hope is that the student will be able to pull information from one of the means of description that would not otherwise be accessible. For example, many people have difficulties reading a problem and understanding it. By allowing the system to read the problem to the person as well as having a picture of the problem, these persons may be able to gain the information necessary for the task at hand.

The domain of the problem will dictate what type of graphic the student is presented with, whether the student sees a graphic of the workings of the system or if the system displays a representation of the control panel for the system being taught. These are only two examples, but the domain will still be the deciding factor of what type of graphic is displayed. The graphic must be natural to the problem, not just something displayed to satisfy the need for graphical interaction. The student should not waste time wondering why the problem is represented in the

manner chosen; the manner should be natural to the problem and the student as to allow the student to concentrate on the problem at hand.

The system will have a subprogram which will manage the graphic used. This subsystem will be responsible for the display of the data represented in the problem, as well as updating the graphic to reflect any changes in the problem. This system will interact with the help system by informing the help system what part of the graphic has been chosen. This will allow context sensitive help for the user, and better description of the problem in general. The graphic system must also interact with the problem analysis module so as to provide the selected parts of the graphic to the command functions. This will permit the student to use the graphic while working the problem instead of manual entering in the information for the functions. This should allow a more natural interface than is normally used in tutoring systems, which normally do not allow the user to use the graphic as part of the input system.

Text is the common denominator for the representation of many problems. The picture of the problem will not necessarily be enough for the student to understand what the problem is, otherwise the experience level would be that of expert and the tutor would not be needed by this student. The text should accurately describe the problem and be brief enough that the student is not overwhelmed by the information.

Of course, the reading level of the text should be close to the educational level of the student. It would be pointless for the student to have to use a dictionary along with the tutoring system. The system is there to provide assistance to the student, not feelings of frustration and anger. As the student progresses through the use of the system, the system may change the amount of description given. If the student is adequately acquiring skills, the system may modify the text so as to keep the student challenged. For example, if the student has shown understanding of a concept, the system will take reminders to use this skill out of the text displayed. This will keep testing the student to see if concepts are being learned, or if the problem description is being followed.

A separate system may be needed to “read” the text to the student, but it will only find the reference to the correct sound file and then play the file back through the means of the computer. This subsystem will not be responsible for decoding the words and matching them to sounds, rather, each problem and display part that has text will also have a reference to a sound file. This will be much easier for the programming, as well as the flow of the text will be much more natural for the user to comprehend. A natural sounding voice is much better for student interaction than a series of pre-recorded sounds that are chained together and sound artificial.

4.2 User Interaction

A subsystem for user input will be in charge of accepting the input, putting it into a format that the system can understand, and forwarding it to the correct system. The student may input through the graphic, the keyboard, or the menus may be used to access different commands. This system will factor out the differences and put them into a format necessary for the system.

The input system must be capable of distinguishing a question from a statement or command. The questions must be forwarded to the coaching subsystem so as to be answered. The coach may also forward these questions to the help system if they are of an informational nature. The statements must be forwarded to the control subsystem so that the action may be performed. This subsystem may also rephrase the information given from the different sources (graphic, keyboard, menus) and put them into a form for the student to read and display this information elsewhere on the screen. This would allow the user to see how the question is being interpreted by the system.

The interaction subsystem will also be responsible for displaying output to the student. This may be in the form of graphical information (updating the problem graphic) or in text format. This part of the system will manage the display of the text in a format readable for the user. The system will notify the graphic system that updating is required instead of updating the graphic directly.

4.3 Student Model

Tracking the current knowledge of the student in relation to the domain will be the task of the student knowledge subsystem. This system will interact with the feedback system to provide information about the knowledge of the user. This may modify the feedback in ways as described in the GUIDON section. This subsystem must maintain a complete reference to what the student currently understands and, possibly, where the student's knowledge should be by the end of the program run.

4.4 Coach

The coaching subsystem will also interact with the student knowledge system to correctly update the information recorded by the student knowledge module. The coach and student knowledge modules together will diagnose why a student incorrectly solves a problem, or is following incorrect steps. This may require the student knowledge module to have information about common errors made in the domain so that the coach can get accurate information about what a student is likely to do.

The phrasing of the feedback will be a job of the coaching part of the system. This part of the system will heavily interact with the student knowledge module to correctly analyze the procedures that the student is performing. The coach must also work through the problem and understand the correct way to solve the problem. This way the coach can use this as a reference when providing feedback to the student.

A subsystem of the coaching module will be responsible for student feedback. This subsystem must phrase the information correctly for the student, and take into account the current knowledge state of the user so that the feedback is as effective as possible. This subsystem must also have some knowledge of human discourse so as to effectively carry out this task. The system will probably use a set of templates for the discourse, as this will be the easiest way to interact with the student and program in the amount of time being considered. This will be one of the most important modules as it is the one that the student gets all impressions of the system from. If this module fails, the system will fail with the student.

Working through the problem will be accounted for by a subsystem of the coach. This may be considered the equivalent of the expert modules in many of the surveyed systems. This system must reason through the problem and provide an accurate trace of the methods used to the tutor, so that the tutor has a basis with which to compare the student.

4.5 Help System

A help system will be required on the assumption that not much information will be given to the student before using the system. This system will be responsible for correctly defining terms to the student. The help should be context sensitive as well as organized for search and browsing.

The student may put in a term in the interaction box and then ask the system what the meaning behind it is. This will allow the student to find out what certain terms that the tutor is using mean without having to take the time to find a dictionary. Any unfamiliar terms should be valid input for the system. The student may also click on parts of the display unsure of what the area is for, and the help system will provide an appropriate answer to the user.

4.6 Conclusion

The reader should now be able to see how the criterion can be used together and what their interactions are. There are other ways to design systems than the ones shown, but the designers take these criteria into consideration when starting the plans for their system. The way that the criterion are combined will decide the effectiveness of the tutor, and how well it is received by students.

Programmers should not only look at the viewpoints expressed here, but also observe educators who teach the subject for the tutor. A dialogue needs to be established between the designers and the educators. Teachers have specific ways of teaching and are very creative resources. They may have ideas about the representation of the problems, have some knowledge about student difficulties, or be able to provide sample problems for the tutor. Even though a machine is being used to educate people, people should still be used as the starting point for the tutorial.

Even the information provided in this paper is not enough to use as a design base. Techniques are described in this paper, but a more thorough understanding of them can be achieved through observation. The reader is ***highly*** encouraged to do this observation during the process of writing a new tutorial system. The reader should also allow educators to review the system periodically to check the effectiveness of goals being met. The best evaluators for education are students and teachers. Making use of the living resources at hand will help the programmer more than any amount of reading.

Appendix A

Sample Dialog with WUSOR-II Coach [4]

(player's responses are preceded by **)

Hello, Timmy. There are 15 caves, 2 bats, 2 pits, and 1 WUMPUS. You are now at cave 15 with neighbors 4, 14, and 0. Brrr! There is a draft. You are near a pit. What a stench! The WUMPUS is near. What now?

**** 4**

You are now at cave 4 with neighbors 15, 14, and 2. Brrr! Squeak! A bat is near. What now?

The goal of the Coach is to tutor a beginner in the relevant logical, probabilistic, and strategic knowledge needed to play the game. For example, the Expert informs the Tutor that cave 14 should be treated as more dangerous than cave 0 or 2, since there is multiple evidence (from the drafts in caves 15 and 4) that cave 14 contains a pit. If the player now moved to cave 14, coaching might take place as follows:

**** 14**

Timmy, it isn't necessary to take such large risks with pits. One of caves 2 and 14 contains a pit. Likewise one of caves 0 and 14 contains a pit. This is multiple evidence of a pit in cave 14, which makes it quite likely that cave 14 contains a pit. It is less likely that cave 0 contains a pit. Hence, we might want to explore cave 0 instead.

Do you want to take back your move?

Appendix B

Sample interaction with SOPHIE [4]

(user's responses preceded by **)

Welcome to SOPHIE, a reactive learning environment. The piece of equipment you are studying is a regulated power supply. Would you like to start with a faulted instrument that needs fixing?

**** Yes**

The instrument has a fault and its present control settings are:

CC	**1.0**	**CURRENT CONTROL**
CR	**HIGH**	**CURRENT RANGE SWITCH**
LOAD	**20**	**LOAD RESISTANCE IN OHMS**
VC	**1.0**	**VOLTAGE CONTROL**
VR	**HIGH**	**VOLTAGE RANGE SWITCH**

**** What is the output?**

The output voltage is 11.7 volts.

Notice that "output" was disambiguated as "output voltage."

**** What is it in the working instrument?**

In a working circuit the output voltage is 19.9 volts.

In case the student does not know what the required value of measurement would have been, he can ask for any measurement in a properly functioning instrument. The student then requests an internal measurement.

**** What is the voltage between R8 and Q1?**

The voltage at N1 is 47.6 volts.

Other interaction:

**** Replace Q6**

The student believes that he has discovered the problem (in transistor Q6) and tries to fix it. Before a component is replaced, however, the system queries him as to how he thinks it is faulted.

I am going to ask you some questions about how Q6 is faulted. Are any junctions shorted?

**** I don't know**

Are any terminals open?

**** Yes**

Which terminal(s) are open?

**** The base**

Q6 is not faulted in that manner. Q6 not replaced.

After learning that Q6 does not have an open base, the student asks if this was at least a reasonable hypothesis.

**** Is it possible that Q6 has an open base?**

Before the system's evaluation procedures can be applied to the student's idea or hypothesis, the hypothesis must be completely specified. To help the student out, the system suggests a plausible complete specification of it in an abbreviated form familiar to the student.

Is the complete fault mode EB/EC/BC = Open/OK/Open what you intended?

**** Sure**

That fault is consistent with all of the measurements you have made.

The hypothesis is reasonable, but other measurements will indicate that another component is faulty.

Appendix C

Sample Interaction between PROUST and Student [14]

Code Entered

```
program Rainfall (input, output);

var
	Rain, Days, Totalrain, Raindays, Highrain, Verain: real;

begin
	Rain := 0;
	repeat
		writeln (‘Enter rainfall’);
		readln;
		read (Rain);
		while Rain < 0 do
			begin
				writeln (Rain:0:2, ‘is not possible, try again’);
				readln;
				read (Rain);
			end;

		while Rain <> 99999 do
			begin
				Days := Days + 1;
				Totalrain := Totalrain + Rain;
				if Rain > 0 then
					RainDays := Raindays + 1;
				if Highrain < Rain then
					Highrain := Rain
			end;
	until Rain = 99999;

	Averain := Totalrain / Days;

	writeln (Days:0:0, ‘valid rainfalls were entered’);
	writeln (‘The average rainfall was’, Averain:0:2, ‘inches’);
	writeln (‘The highest rainfall was’, Highrain:0:2);
	writeln (‘There were’, Raindays:0:0, ‘in this period’);
end.
```

System Feedback

NOW BEGINNING BUG REPORT

>> Now Reporting CRITICAL Bugs in the SETUP part of your program:

You left out the initializations for the variables HIGHRAIN, DRYDAYS, RAINDAYS, and TOTALRAIN. Programs should not fetch values from uninitialized variables! If you don't initialize these variables to the values that you want, you will have no way of knowing what values they will be initialized to.
(TO CONTINUE, PLEASE PRESS CARRIAGE RETURN)

>> Now Reporting MINOR Bugs in the SETUP part of your program:

The initialization at line 7 appears to be unnecessary.
The statement is question is:
    RAIN := 0;
(TO CONTINUE, PLEASE PRESS CARRIAGE RETURN)

>> Now Reporting CRITICAL Bugs in CONTROL part of your program:

You used a WHILE statement at line 19 where you should have used an IF. WHILE and IF are NOT equivalent in this context; using WHILE in place of IF can result in infinite loops.
The statement in question is
    WHILE RAIN <> 99999 DO
(TO CONTINUE, PLEASE PRESS CARRIAGE RETURN)

>> Now Reporting CRITICAL Bugs in the OUTPUT part of your program.

The maximum and the average are undefined if there is no valid input. But lines 34 and 33 output them anyway. You should always check whether your code will work when there is no input! This is a common cause of bugs.

You need a test to check that at least one valid data point has been input before line 30 is executed. The average will bomb when there is no input.
BUG REPORT NOW COMPLETE

Appendix D

Graphic Display of Anderson's Geometry tutor
(Before and after student interaction)

System as seen upon startup.

System after student correctly applies transitive property and definition of bisector.

Note how the graphics for the problem is displayed in an area apart from the rest of the problem, how the goal and knowns are separated, and the graphic updating of the problem diagram to match the proof.

References

1. Anderson, John R., C. Franklin Boyle, Albert T. Corbett, and Matthew W. Lewis, “Cognitive Modeling and Tutoring,” Artificial Intelligence and Learning Environments, MIT Press, Cambridge, MA, 1990, pages 7-49.

2. Anderson, John R., James G. Greeno, Paul J. Kline and David M. Neves, “Acquisition of Problem-Solving Skill,” Cognitive Skills and Their Acquisition, edited by John R. Anderson, Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ, pages 191-230.

3. Anderson, John R., and Edward Skwarecki, “The Automated Tutoring of Introductory Computer Programming,” Communications of the ACM, 29(9):842-849.

4. ”Applications-Oriented AI Research: Education,” Chapter IX, The Handbook of Artificial Intelligence, edited by Avron Barr and Edward A. Feigenbaum, Addison-Wesley Publishing, Reading, MA, 1982, pages 224-294.

5. Baril, Daniel, Jim E. Greer, and Gordon I. McCalla, “Student Modeling with Confluences,” Proceedings of the Ninth National Congress on Artificial Intelligence, Volume 1, pages 43-48.

6. Chandrasekaran, B., Todd R. Johnson, and Jack W. Smith, “Task-Structure Analysis for Knowledge Modeling,” Communications of the ACM, 35(9):124-137.

7. Clancey, William J., and Elliott Soloway, “Artificial Intelligence and Learning Environments: Preface,” Artificial Intelligence and Learning Environments, MIT Press, Cambridge, MA, 1990, pages 1-6.

8. Clancey, William J., and Bruce G. Buchanan, “Intelligent Computer-Aided Instruction,” Rule-Based Expert Systems, edited by Bruce G. Buchanan and Edward H. Shortliffe, Addison-Wesley Publishing, Reading, MA, 1985, pages 455-463.

9. Clancey, William J., “Use of MYCIN’s Rules for Tutoring,” Rule-Based Expert Systems, edited by Bruce G. Buchanan and Edward H. Shortliffe, Addison-Wesley Publishing, Reading, MA, 1985, pages 464-489.

10. Clancey, William J., “Extensions to Rules for Explanation and Tutoring,” Rule-Based Expert Systems, edited by Bruce G. Buchanan and Edward H. Shortliffe, Addison-Wesley Publishing, Reading, MA, 1985, pages 531-568.

11. du Boulay, Benedict, and Christopher Sothcott, “Computers Teaching Programming: An Introductory Survey of the Field,” Artificial Intelligence and Education, Volume One, Ablex Publishing, Norwood, NJ, 1987, pages 345-372.

12. Fischer, Gerhard, Andreas C. Lemke, and Raymond McCall, “Towards a System Architecture Supporting Contextualized Learning,” Proceedings of the Eighth National Congress on Artificial Intelligence, Volume 1, pages 420-425.

13. Greer, Jim E., and Gordon I. McCalla, “A Computational Framework for Granularity and its Application to Educational Diagnosis,” Proceedings of the Eleventh International Joint Congress on Artificial Intelligence, Volume 1, pages 477-482.

14. Johnson, W. Lewis, “Understanding and Debugging Novice Programs,” Artificial Intelligence and Learning Environments, MIT Press, Cambridge, MA, 1990, pages 51-97.

15. Lawler, Robert E., and Masoud Yazdani, “Introduction,” Artificial Intelligence and Education, Volume One, Ablex Publishing, Norwood, NJ, 1987, pages ix-xii.

16. Murray, William R., “A Blackboard-Based Dynamic Instructional Planner,” Proceedings of the Eighth National Congress on Artificial Intelligence, Volume 1, pages 434-441.

17. Ohlsson, Stellan, “Some Principles on Intelligent Tutoring,” Artificial Intelligence and Education, Volume One, Ablex Publishing, Norwood, NJ, 1987, pages 203-237.

18. Richer, Mark H., and William J. Clancey, “GUIDON-WATCH: A Graphic Interface for Viewing a Knowledge-Based System,” Artificial Intelligence and Education, Volume One, Ablex Publishing, Norwood, NJ, 1987, pages 373-4112.

19. Sleeman, Derek, “PIXIE: A Shell for Developing Intelligent Tutoring Systems,” Artificial Intelligence and Education, Volume One, Ablex Publishing, Norwood, NJ, 1987, pages 239-267.

20. Soloway, Elliott, “Learning to Program = Learning to Construct Mechanisms and Explanations,” Communications of the ACM, 29(9):850-858.

21. White Barbara Y., and John R. Frederiksen, “Causal Model Progressions as a Foundation for Intelligent Learning Environments,” Artificial Intelligence and Learning Environments, MIT Press, Cambridge, MA, 1990, pages 99-157.

22. Woolf, Beverly, Darrell Blegen, Johan H. Jansen, and Arie Verloop, “Teaching a Complex Industrial Process,” Artificial Intelligence and Education, Volume One, Ablex Publishing, Norwood, NJ, 1987, pages 413-427.

23. Woolf, Beverly, “Intelligent Tutoring Systems: A Survey,” Exploring Artificial Intelligence, ed. by Howard E. Shrobe, Morgan Kaufmann Publishers, San Mateo, CA, 1988, pages 1-43.