Theoretical and Empirical Motivations for the Design of ANGLE: A New Geometry Learning Environment

Kenneth R. Koedinger and John R. Anderson

Psychology Department
Carnegie Mellon University
Pittsburgh, PA 15217

Cognitively Sound Tutor Design

Quite often the design of Intelligent Tutoring Systems (ITS) has been a seat-of-the-pants venture in which designers have relied mainly on their beliefs and intuitions to guide their decisions. Expert component design has sometimes been aided by consultation with domain experts, but typically the major emphasis has been on getting the program to work. The situation with respect to the design of the interface and tutoring components is worse: theoretical justification or empirical testing of design decisions are rare at best.

Of course seat-of-the-pants design can lead to innovation, but it more often leads to bad artifacts (Norman, 1989). Unfortunately for the ITS field, little follow-up empirical work has been done with ITSs to determine whether they are actually useful. Even less work has been done to identify what aspects of their design account for their usefulness.

Some researchers claim that the most important factor in the success of an ITS is the quality of the expert component or cognitive model that underlies it (Anderson, 1988; Lesgold, et. al., in press). The ITS presented in this paper, called ANGLE, takes this claim to heart as it is based on a second generation cognitive model of geometry proof problem solving. In addition, with ANGLE we are in a position to test this claim as ANGLE can be compared with the Geometry Tutor, an existing ITS based on a less accurate cognitive model.

ANGLE's expert component has grown out of detailed problem solving research, in particular, protocol analysis. In addition, ANGLE's interface design has been theoretically driven by three goals: 1) students' interface actions should mirror the actions of the cognitive model, 2) the unique ability of computer interfaces to allow the dynamic manipulation of display objects should be exploited, and 3) ease of use should not be compromised in pursuit of the preceding goals.

Background

The Geometry Tutor (Anderson, Boyle, & Yost, 1985) resulted from basic problem solving research and the application of the ACT* theory (Anderson, 1983). It has proven effective in the high school classroom – students using the Geometry Tutor in their classroom outperformed those in a normal classroom by about 1 standard deviation (Anderson, Boyle, Corbett, & Lewis, in press). The apparent reason for the tutor's success was the shift in the focus of instruction from the product of problem solving, i.e., the definitions, postulates, and theorems that make up the steps of a proof, to the process of problem solving, i.e., how can I, as a student, generate those steps. In other words, while textbook instruction provides the facts or *what* of geometry, the Geometry Tutor shows students *how* they can use these facts to solve problems. The strategic advice the tutor provides reflects the search control heuristics in the cognitive model.

While it appears the Geometry Tutor's success was largely a result of explicating *a* process or method for problem solving, it turns out that this is not the only effective method for doing geometry proofs. In fact, recent research on skilled geometry problem solving (Koedinger & Anderson, in press) has identified a geometry proof method (called DC) which is not only a more accurate account of human problem solving, but is also more effective. This paper describes this model and its fundamental role in the design of ANGLE: **A** **N**ew **G**eometry **L**earning **E**nvironment.

DC: A New Approach to Geometry

Koedinger and Anderson (in press) report on the DC model in detail: describing a computer simulation of it and evaluating it in comparison with other methods. This section reviews DC and its evaluation as background and motivation for its use in ANGLE.

The DC Method

Like most previous models of geometry theorem proving (Gelernter, 1963; Goldstein, 1973; Anderson, Boyle, & Yost, 1985), the expert component of the Geometry Tutor works in a problem space based on the formal rules of geometry: the definitions, postulates, and theorems. We call this the *execution space* because these rules correspond with the steps

To be presented at the Knowledge-Based Environments for Learning and Teaching, AAAI Spring Symposium Series, Standford University, March 27-29, 1990.

that are written down in the final execution of a proof plan.

The DC model grew out of the analysis of verbal reports given by skilled subjects as they solved geometry proof problems. We were surprised to find that while the steps subjects wrote down in their final solution corresponded with the execution space, they skipped many of these steps earlier in the process of planning a solution. The problem solving protocols of all the skilled subjects had this flavor where there were phases of planning where steps were skipped and phases of execution where these steps were filled in. It was clear that subjects were not searching step-by-step in the execution space. Rather, subjects were planning in some other more abstract problem space (Newell & Simon, 1972; Sacerdoti, 1974) using knowledge that allowed them to focus on the key inferences and ignore the minor inferences. We have characterized the nature of this knowledge in a computer simulation called the Diagram Configuration model (DC).

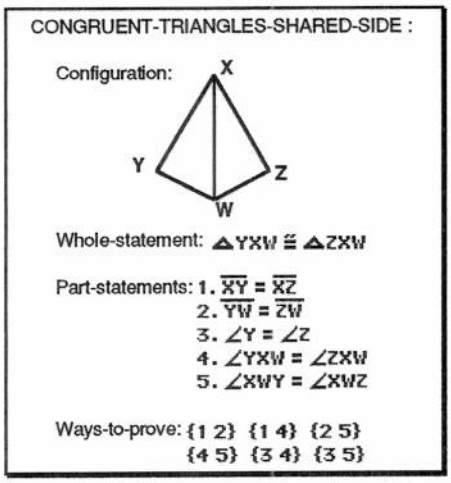

Figure 1. A diagram configuration schema.

Model Description. The core idea of DC is that the knowledge of skilled geometry problem solvers is organized around certain prototypical geometric figures we call *diagram configurations*. Clustered around each diagram configuration are related geometry facts. We call such clusters of geometry information *diagram configuration schemas*.

Diagram configuration schemas have four attributes as shown in Figure 1. The *whole-statement* and *part-statements* attributes contain statements which refer to the geometric figure stored in the *configuration* attribute. The whole-statement refers to the configuration as a whole, while the part-statements are relationships among segment and angle parts of the configuration. The *ways-to-prove* attribute contains possible subsets of the part-statements which need to be proven (or given) in order to "prove the schema". Whenever a schema is "proven" the whole-statement and all the part-statements of the schema can be proven. Thus, these schemas have a pattern completion character such that if some portion of the part-statements are proven, the rest of the part-statements can be proven. For example, one of the ways-to-prove of the TRIANGLE-CONGRUENCE-SHARED-SIDE schema is {1 2} which indicates that if part-statements 1. XY=XZ and 2. YW=ZW are proven, the whole-statement and the rest of the part-statements are proven. Such a use of this schema corresponds with five steps in the execution space: the application of the REFLEXIVE property and the SIDE-SIDE-SIDE postulate to prove the whole-statement, $\Delta XYW \cong \Delta XZW$, followed by three applications of the CORRESPONDING-PARTS rule to prove the other three part-statements (3, 4, and 5). This chunking illustrates the power of DC's abstract planning.

Besides abstract planning, another feature essential to the success of the DC method is the use of the problem diagram as a guide for generating candidate inferences. Figure 2 shows a proof problem in the ANGLE interface – the problem diagram is in the upper-left corner, the problem givens are on the bottom, and the problem goal is on the top. A number of instances of the TRIANGLE-CONGRUENCE-SHARED-SIDE schema can be seen in this diagram, for example, $\Delta ACK \cong \Delta BCK$ and $\Delta GCK \cong \Delta HCK$. While it is not the case that any configuration that appears in a problem diagram can be proven (e.g., $\Delta GCK \cong \Delta HCK$ in the problem in Figure 2), it is a fact that, given a properly drawn diagram, any configuration which can be proven appears in the diagram (e.g., $\Delta ACK \cong \Delta BCK$). By only considering configurations which appear in the diagram, DC effectively prunes out a tremendous number of logically well-formed statements that might otherwise be considered (see analysis below).

Contrary to many problem solving systems, DC's power does not come from clever search control heuristics applied to the basic problem space. Instead, it comes from an abstract problem space representation. Thus, a simple depth-first search is sufficient to find proofs in DC's problem space.

Empirical Validation. To provide empirical support for DC, we analyzed 12 protocols coming from the concurrent verbal reports (Ericsson and Simon, 1984) of five skilled subjects. The protocols were divided into 1) *planning episodes* in which subjects developed an initial sketch for part or all of the solution and 2) *execution episodes* in which they reported the details of part or all of their final solution. We identified the steps in subjects' final solutions that they mentioned during the planning.

We found that subjects mentioned only 37 of the 98 total steps in the final solutions of the 12 problems – they skipped 61. If DC is a good model of abstract planning in geometry, the planning steps of skilled

subjects should tend to correspond with DC-schemas. Indeed, we found that 23 of the 29 steps the model predicts will be mentioned were actually mentioned, while 55 of the 69 steps the model predicts will be skipped were actually skipped. A Chi square test ($X^2(1) = 30.3$) indicates it is unlikely that the model's fit to the data is a chance occurrence ($p < .001$).

Computational Effectiveness. The relative effectiveness of DC compared to models that work in the execution space is demonstrated by a task analysis we did of one of the more difficult problems the subjects solved (shown in Figure 2). The shortest solution to this problem in the execution space is 7 steps and we estimated that a breadth-first search for this solution visits more than a million states. The shortest solution to this problem in the diagram configuration space is 3 steps and a breadth-first search for this solution visits only 8 states in the worst case. Of course, the immense size of the execution space can be somewhat reduced by adding search control heuristics, for example, as was done in previous models including the Geometry Tutor expert. However, our experience has been that DC easily solves problems that are quite difficult for the Geometry Tutor expert even with its heuristics.

ANGLE

In this section we discuss the implications of DC for the design of ANGLE. DC also has implications for the design of other instructional factors, including curriculum organization, text materials, and classroom lessons. While we feel that such factors are crucial to the success of a complete ITS package, limited space prevents us from discussing them here. Suffice it to say that a major role of these supplementary materials is to provide students with the declarative information they need to use the DC method. ANGLE, in turn, provides an environment where students can learn to understand and appropriately use DC-schemas in the context of problem solving.

Tutor Design: From DC to ANGLE

The Implicit Plan Problem. One of the difficulties involved in building an ITS is finding a way to communicate about the thinking that students do between their observable problem solving actions. We call this the "implicit plan" problem. If aspects of student planning cannot be articulated in the system interface, it is not possible for the system to monitor this planning nor provide relevant advice. For example, in data collected from the Geometry Tutor, it was found that the better performing students were taking significantly more time at the beginning of a problem before making their first interface action. Presumably, they were trying to develop a solution plan. In contrast, the poorer students would often blindly try any rule that could be applied to the problem givens.

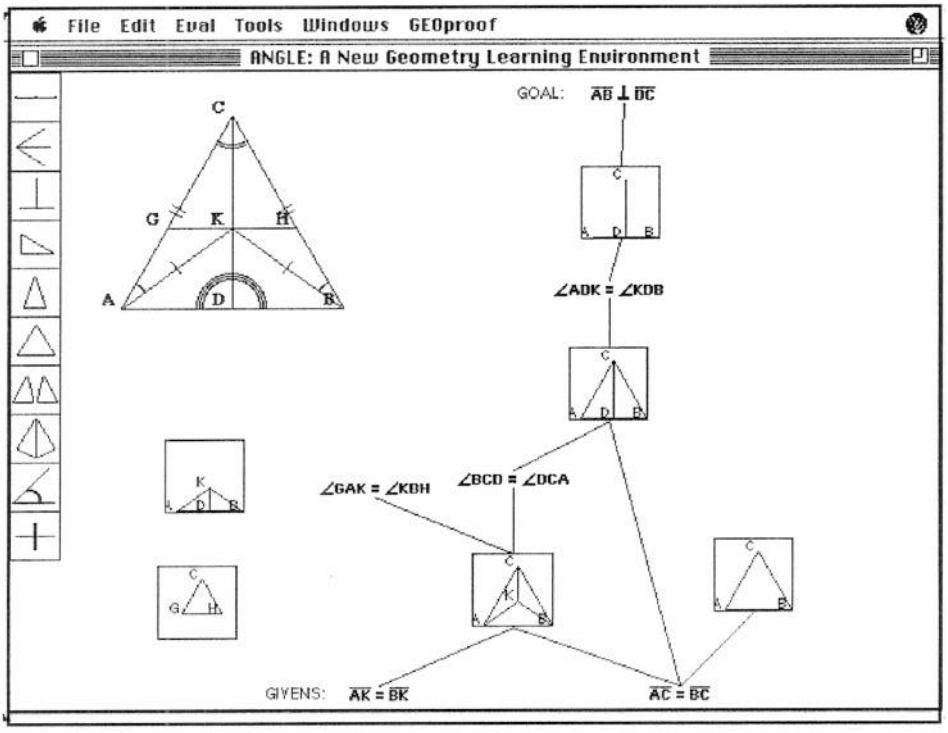

Figure 2. Tutor window after a solution was found. Notice off-path moves and schema instances that were identified but never proven.

In support of this interpretation, a common complaint about the Geometry Tutor is that it does not provide very good global feedback. The feedback focuses locally on the *next* proof step the student might take rather than more globally at the next few steps or an overall plan. Many critics have the intuition that proof ideas can be born at a more global level. Our current research on geometry experts has identified this more global level and has characterized it in terms of DC's diagram configuration schemas. Thus, ANGLE can address the implicit plan problem – we can reify the planning process in the interface and so, open it up for discussion and instruction.

Reifying Planning. Some ITS designers have addressed the problem of communicating about the abstract planning occurring above the level at which solution steps are executed. Examples of the resulting tutoring systems include Bridge (Bonar & Cunningham, 1988), GIL (Reiser, et. al., 1988), and Sherlock (Lesgold, et. al., 1988). The basic approach is to develop a command language, usually menu-based and possibly graphical, which makes this planning level concrete. Conveniently, we do not need to invent such a command language to reify DC's abstract problem space. Essentially, it already exists in the form of the problem diagram.

The ANGLE Interface

Figure 2 shows ANGLE interface after the completion of a solution. Students enter proof steps in this tutor by selecting a diagram configuration from the icon menu on the left of the screen. Next, they mouse-click on lines in the problem diagram to indicate a particular instance of this configuration. The lines they select are highlighted as shown in Figure 3. The figure shows the selection of an instance of the TRIANGLE-CONGRUENCE-SHARED-SIDE schema – this particular instance is the first step along the solution path shown in Figure 2. After the student mouse-clicks on the Done button, the schema instance appears on the screen and the student can drag it (using the mouse) over into the proof tree. A schema is proven by selecting statements (using the mouse) that justify the schema – a line is drawn between the schema and each statement selected. The system checks whether or not the selected statements correspond with one of the ways-to-prove of the schema. The current prototype gives error feedback when the selections are not necessary and sufficient to prove the schema. A more sophisticated feedback scheme is in the works.

The statements shown in Figure 2, e.g., **∠BCD = ∠DCA**, are entered using the bottom two icons in the icon menu. The bottom-most icon is used to mark segments equal and the one above it is used to mark angles equal. A statement is proven by indicating the schema that justifies it. When a statement is proven, the result is marked on the diagram using the conventional hash marks.

ANGLE illustrates one of the potential advantages of computer tutors, namely the dynamic and manipulatable nature of computer interfaces. ANGLE allows students to move schemas and statements around on the screen (all attached lines are maintained). On one hand, this feature is simply a way for students to keep their proof graph neatly organized. On other hand, students appear to use this feature in a more cognitively meaningful way. Without any explicit direction from us, the students who have worked with the prototype system have used different localities on the screen to implicitly record the status of the schema instances they have selected. When a schema instance is initially selected, students place it above the givens and any previously proven statements. Sometimes they discover they cannot prove the schema at the moment either because 1) more statements need to be proven before the schema can be, or 2) the schema cannot be proven and/or is irrelevant to the proof. In case (1), students have left the schema in the proof tree area unlinked but posted as a goal for future problem solving. In case (2), students have moved the schema off to the left (like the schemas for ISOS ΔGCH and $\Delta AKD \cong \Delta BKD$ in Figure 2) to indicate that it is no longer useful.

A second example of the potential advantages of a computer interface over paper and chalkboard concerns the way different schema instances can be highlighted in the context of the overall diagram as shown in Figure 3. Such temporary highlighting is impossible to do on paper, but may be quite valuable in facilitating the learning of the perceptual skill of extracting particular objects from a complex background. Finding the particular schema instance shown in Figure 3 is one of the difficulties we've observed subjects experience in solving the problem shown.

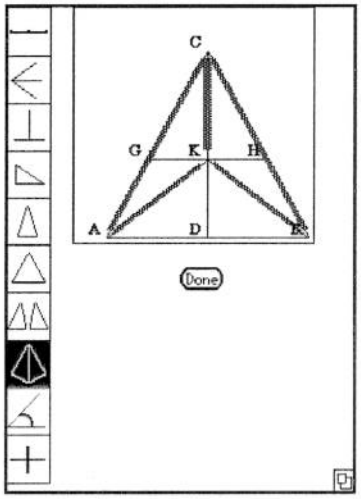

Figure 3. The selection of an instance of the TRIANGLE-CONGRUENCE-SHARED-SIDE schema.

Conclusion and Future Work

We are planning to compare ANGLE to the Geometry Tutor in an evaluation study. We are optimistic that the ANGLE tutor can exceed the 1 standard deviation improvement achieved by the Geometry Tutor and perhaps approach or even exceed the 2 standard deviation improvement gained by individual human tutoring (Bloom, 1984). In addition to the interface benefits mentioned above, the major advantage of ANGLE may derive from the cognitive science research that inspired it.

We can think of a tutor as a model of problem solving which students can emulate. To the extent that the tutor's problem solving method is a good one and students successfully emulate it, then they will be good problem solvers as well. Because DC is a more powerful method than the one in the previous tutor, we believe that students who successfully emulate it will be even better problem solvers than those who successfully emulated the problem solving method taught by the Geometry Tutor.

Besides being a more powerful problem solving method than one that focuses on single steps, our basic research suggests that the DC method may be easier to learn. Koedinger and Anderson (1989) argue that DC-schema's are not learned, in any direct way, from the formal rules of the domain – what we call the execution space. Instead, they appear to be learned by identifying useful categories of domain objects (i.e., the configuration) and learning their properties (i.e., the part-statements and ways-to-prove). These categories carry the load of recognition: indicating *when* particular knowledge should be brought to bear and having the effect of drastically reducing the search space. The properties carry the load of inferencing – indicating what can be concluded (part-statements) and under what conditions (ways-to-prove).

We believe this type of learning, that is, abstraction from domain objects, is a more natural extension of students' prior knowledge. Thus, contrary to the concern that DC represents an expert method that will be too difficult for students to understand, we believe the DC method will be easier to learn. The instruction in the Geometry Tutor is focussed on the formal rules of geometry which are totally new and unfamiliar to students. In contrast, the focus of instruction in ANGLE would be on diagram configurations and their properties. Pre-geometry students already have some prior perceptual intuitions about geometric figures. Diagram configuration schemas can be taught by building off this familiar ground. Clearly students will have to refine their perceptual knowledge to learn how to encode problem diagrams into diagram configurations. However, having students plan proofs by selecting diagram configuration icons and instantiating them in the problem diagram will aid them in learning this encoding.

References

Anderson, J. R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R. (1988). The expert module. In M. C. Polson & J. J. Richardson (Eds), *Foundations of Intelligent Tutoring Systems.* Hillsdale, NJ: LEA.

Anderson, J. R., Boyle, C. F., & Yost, G. (1985). The geometry tutor. In *Proceedings of the International Joint Conference on Artificial Intelligence-85.* Los Angelos: IJCAI.

Anderson, J. R., Boyle, C. F., Corbett, A., & Lewis, M. (in press). Cognitive modelling and intelligent tutoring. *Artificial Intelligence.*

Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher, 13,* 3-16.

Bonar, J. G., & Cunningham, R. (1988). Intelligent tutoring with intermediate representations. Paper presented at ITS-88. Montreal.

Ericsson, K. A., & Simon, H. A. (1984). *Protocol Analysis: Verbal Reports as Data.* Cambridge, MA: The MIT Press.

Gelernter, H. (1963). Realization of a geometry theorem proving machine. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and Thought.* New York: McGraw-Hill Book Company.

Goldstein, I. (1973). Elementary geometry theorem proving. MIT AI Memo 280.

Koedinger, K. R., & Anderson, J. R. (in press). Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science.*

Koedinger, K.R., & Anderson, J.R. (1989). Perceptual chunks in geometry problem solving: A challenge to theories of skill acquisition. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: LEA.

Lesgold, A. M., Lajoie, S. P., Bunzo, M., & Eggan, G. (In press). SHERLOCK: A coached practice environment for an electronics troubleshooting job. In J. Larkin, R. Chabay, & C. Scheftic (Eds.), Computer Assisted Instruction and Intelligent Tutoring Systems: Establishing Communication and Collaboration. Hillsdale, NJ: LEA.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Norman, D. A. (1989). *The Psychology of Everyday Things.* New York, NY: Basic Books.

Reiser, B. J., Friedmann, P., Gevins, J., Kimberg, D. Y., Ranney, M., & Romero, A. (1988). A graphical programming language interface for an intelligent LISP tutor. *Proceedings CHI'88.*

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence, 5,* 115-136.