Perceptual Chunks in Geometry Problem Solving: A Challenge to Theories of Skill Acquisition

Kenneth R. Koedinger & John R. Anderson
Psychology Department, Carnegie Mellon University

ABSTRACT

In current theories of skill acquisition it is quite common to assume that the input to learning mechanisms is a problem representation based on direct translations of problem instructions or simple inductions from problem solving examples. We call such a problem representation an *execution space* because it is made up of operators corresponding to the external actions agents perform while executing problem solutions. Learning proceeds by modifications and combinations of these execution space operators. We have built a model of geometry expertise based on verbal report evidence which contains operators which can be *described* as modifications (e.g., abstractions) and combinations (e.g., compositions) of execution operators. However, a number of points of evidence lead us to conclude that these operators were *not derived* from execution space operators. In contrast, it appears these operators derive from discoveries about the structure and properties of domain objects, particularly, perceptual properties. We have yet to develop a detailed and integrated theory of this "perceptual chunking", but we present the expert model is a challenge to current theories of skill acquisition.

1. Introduction

The process of skill acquisition is generally described as involving two phases as shown in Figure 1. In the knowledge acquisition phase, the system uses information about the problem domain, e.g., problem descriptions, problem constraints, example solutions, etc., to build some kind of basic problem space[1], essentially, a set of simple condition-action operators that it can use to attempt to solve problems in the domain. In the knowledge tuning phase, the basic problem space is elaborated through problem solving practice so that the system becomes more effective and efficient. The elaborated problem space may incorporate heuristics that control the system's search or may be an abstracted version of the basic problem space in which operators make larger steps allowing for faster solutions.

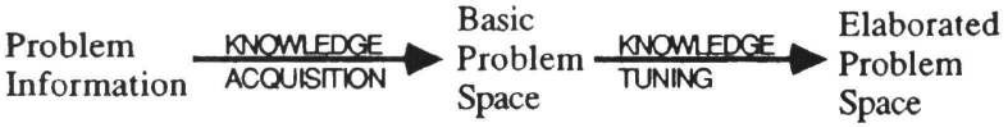

Figure 1. A framework common to many theories of skill acquisition and learning.

The framework in Figure 1 is characteristic of a number of theories of skill acquisition. In ACT* (Anderson, 1983), knowledge acquisition is modelled by a mechanism called proceduralization while knowledge tuning is modelled by composition, generalization, and discrimination. In Soar (Newell, in press), knowledge acquisition has been modelled by a program called TAQ while knowledge tuning is modelled by Soar's chunking mechanism. Other research efforts have focussed on one or the other of these phases. For example, knowledge acquisition has been modelled in the UNDERSTAND program (Hayes and Simon, 1974) which built a problem space from a natural language description, and in a program by Neves (1978) which built a problem space from example solutions. Knowledge tuning has been modelled in terms of macro-operator learning (Korf, 1987) and in terms of problem space abstraction (Sacerdoti, 1974; Unruh, Rosenbloom, and Laird, 1987).

While this framework has certainly proven useful, we argue that it is inadequate for a complete and general theory of skill acquisition. We support this argument with empirical data and a model of expert geometry problem solving which cannot plausibly be learned within this framework. The basic argument is as follows. We have found that geometry experts skip steps in developing proof plans. By

[1]We don't mean to suggest that a system can have only one problem space associated with a domain. Thus, when we refer to a system's problem space for a domain, one can think of it as the collection of all problem spaces for that domain.

itself this behavior is not contrary to the standard framework – it might be explained, for example, by ACT*'s composition or Soar's chunking mechanism. However, a closer look at the details of this step-skipping behavior brings such explanations into question. In particular, we identified a regularity in the kinds of steps experts skip which cannot be easily explained in terms of compositions or chunks of consecutive production or operator applications. We present a schema-based model, called DC, which accounts for this regularity. While DC's schemas could be represented in terms of production rules, it is difficult to imagine how current production rule learning models could produce the organization inherent in these schemas.

2. The Basic Phenomenon: Step-Skipping

We analyzed 12 protocols coming from the concurrent verbal reports (Ericsson and Simon, 1984) of four subjects solving one problem and one subject solving eight problems. Two of the subjects were mathematics graduate students, two were psychology researchers with extensive experience in geometry, and one was a Pittsburgh area high school geometry teacher.

In analyzing these protocols we were surprised to find that the steps subjects took in the process of planning a proof do not correspond with the rules of geometry: the definitions, postulates, and theorems. In contrast, most previous models of geometry theorem proving have worked in a problem space based on these rules (Gelernter, 1963; Goldstein, 1973; Anderson, Boyle, & Yost, 1985). We call this the *execution space* because these rules correspond with the steps that are written down in the final execution of a proof plan. While the the steps subjects wrote down or stated in explaining their final solution correspond with the execution space, they skipped many of these steps in planning a solution.

TABLE 1
A Verbal Protocol for a Subject Solving the Problem in Figure 2.

		****** Planning phase ******
B1:	We're given a right angle – this is a right angle,	Reading given: rt ∠ADB
B2:	perpendicular on both sides [makes perpendicular markings on diagram];	Inference step 1: AC ≅ BD
B3:	BD bisects angle ABC [marks angles ABD and CBD]	Reading given: BD bisects ∠ABC
B4:	and *we're done*.	Inference step 2: △ABD ≅ △CBD
		****** Execution phase ******
B5:	We know that this is a reflexive [marks line BD],	In this phase, the subject refines and
B6:	we know that we have congruent triangles; we can determine anything from there in terms of corresponding parts	explains his solution to the experimenter.
B7:	and that's what this [looking at the goal statement for the first time] is going to mean ... that these are congruent [marks segments AD and DC as equal on the diagram].	

Figure 2 shows one of the problems we used and its solution in proof tree format. Table 1 contains the protocol of a subject solving this problem. The subject's verbalizations are shown in the left column of Table 1. The right column contains a summary of the steps the subject mentions. The protocol is divided up into 1) a *planning phase* in which the subject is searching for a solution and 2) an *execution phase* in which he executes the previously outlined solution by reporting it to the experimenter.

This expert had a plan for solving this problem in 13 seconds at the point where he said "we're done". We can describe his planning as follows. In block B1 of the protocol he reads the first given. Next at B2, he makes the inference that the lines AC and BD are perpendicular – "perpendicular on both sides", which follows from the first given. At B3, he reads the second given and then at B4 he says "we're done" which appears to indicate that he has made an inference from the second given which proves the goal. Further inspection of his explanation of the solution, particularly at block B6, makes it clear that this inference was that the two triangles ABD and CBD are congruent.

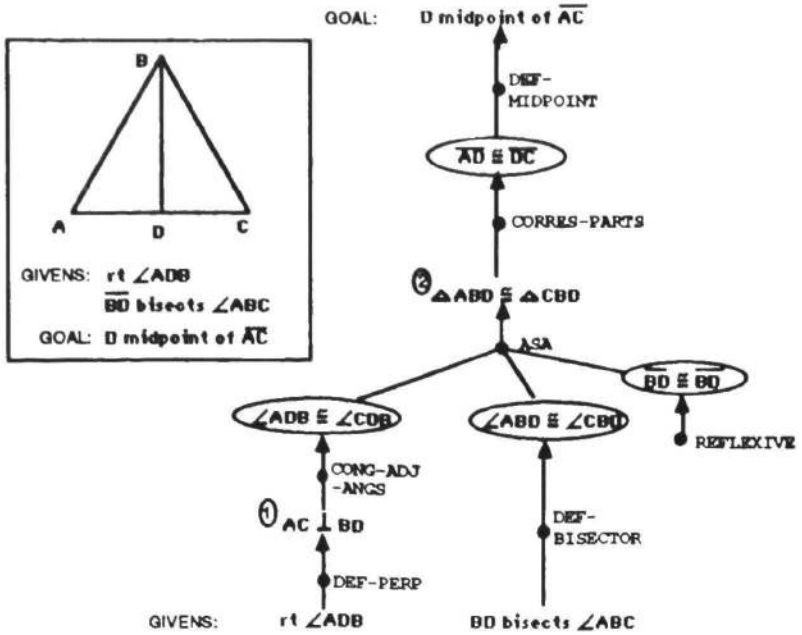

Figure 2. A problem (in the box) and its solution. The numbered steps are ones a subject mentioned during planning (see Table 1), while the circled steps are ones he skipped.

Of the four verbalizations in the planning phase, two indicate his reading and encoding of the given statements and two indicate inferences. In other words, he came up with a solution plan in two steps. In contrast, the final solution to this problem requires seven steps as shown in Figure 2.

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 using knowledge that allows 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).

3. The Diagram Configuration Model (DC)

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*. Two examples are illustrated in Figure 3.

Diagram configuration schemas have four attributes: 1) the configuration, 2) the whole-statement, 3) the part-statements, and 4) the ways-to-prove. The *whole-statement* and *part-statements* attributes of a schema 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 main action of a diagram configuration schema comes from the *ways-to-prove* attribute. This attribute contains different ways to "prove the schema". Saying a schema is "proven" is a brief way of saying that the whole-statement and all the part-statements of the schema can be proven. Each of the ways-to-prove is a list of part-statements, indicated by their number, which are sufficient to prove the schema. 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 schema can be proven.

The essential idea behind diagram configuration schemas is that skilled geometry problem solvers can recognize certain configurations in problem diagrams and they know that if certain statements about a configuration have been proven, *all* the statements about the configuration can be proven. Instead of planning proofs one statement at a time, diagram configuration schemas allow skilled problem solvers to plan multiple proof steps in a single thought.

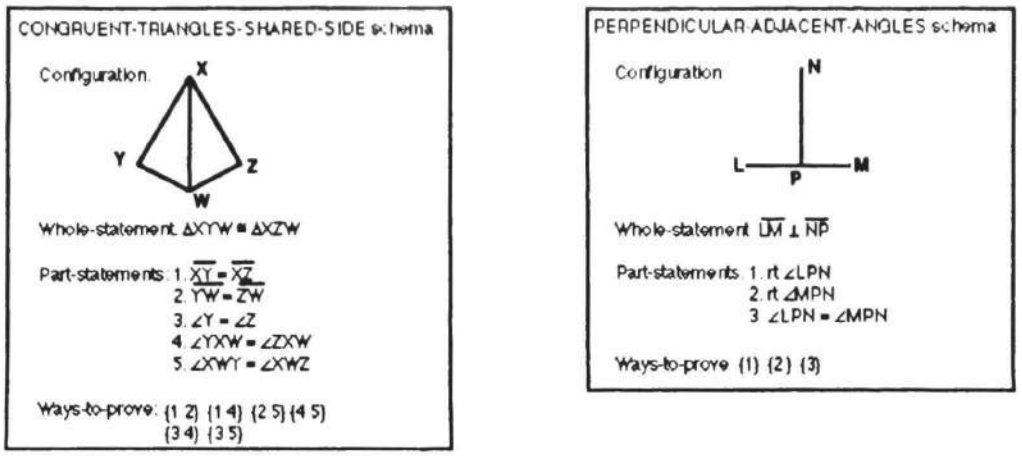

Figure 3. Two diagram configuration schemas.

In Table 1, we saw a skilled subject plan a seven step proof in two steps. We can explain his planning in terms of DC. DC visually parses a problem diagram into instances of the various configurations it knows about. Inside the rounded boxes in Figure 4 are the configurations DC recognizes in the problem diagram in Figure 2. Attached to each configuration are the part-statements which refer to it. Notice that certain part-statements are associated with more than one configuration. For example, ∠ADB ≅ ∠CDB is a part-statement for both the PERPENDICULAR-ADJACENT-ANGLES and TRIANGLE-CONGRUENCE-SHARED-SIDE schemas.

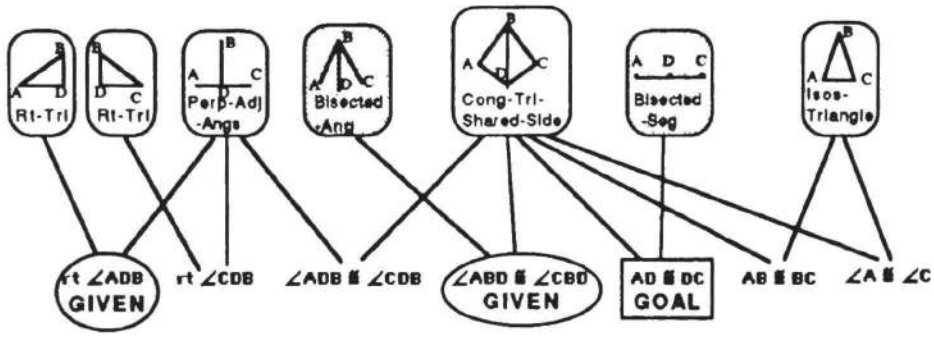

Figure 4. The configurations and associated part-statements that appear in the problem diagram in Figure 2.

At block B1 in Table 1, the subject reads and encodes the first given rt ∠ADB which we have circled in Figure 4. By "encodes" we mean he determines what the statement means. In this case, by encoding the given rt ∠ADB we believe the subject understands this to mean that the measure of ∠ADB is 90 degrees. At block B2, he makes an inference corresponding with proving the PERPENDICULAR-ADJACENT-ANGLES schema. The result of this inference is that he knows the other two part-statements rt ∠CDB and ∠ADB ≅ ∠CDB are true. At block B3, he reads and encodes the second given BD bisects ∠ABC. This statement is the whole-statement for the BISECTED-ANGLE schema and he encodes it by considering the part-statements of this schema as given. This schema has only one part-statement, ∠ABD ≅ ∠CBD, which is marked as given in Figure 4. When the subject reads the goal statement (B7), we claim he encodes it in a similar way, thinking of the corresponding part-statement AD ≅ DC.

Following B3, the subject knows that the four part-statements on the left in Figure 4, rt ∠ADB through ∠ABD ≅ ∠CBD, are true – only the three on the right remain unknown. Two of the known statements, ∠ADB ≅ ∠CDB and ∠ABD ≅ ∠CBD, correspond with the one of the ways-to-prove of the TRIANGLE-CONGRUENCE-SHARED-SIDE schema, namely (4 5). At B4, the subject makes an inference which we claim corresponds with proving this schema. His explanations at B6 and B7 support this claim. If, in fact, he is proving the TRIANGLE-CONGRUENCE-SHARED-SIDE schema at B4, he should know the three other part-statements AD ≅ DC, AB ≅ BC, and ∠A ≅ ∠C are true. It seems clear that he

knows this from B6, "... we have congruent triangles; we can determine anything from there ...." The fact that he first looked at the goal statement at B7 provides further evidence. It indicates that earlier in the protocol, at B4, he had some other way of detecting that he was done with the proof. Assuming his inference at B4 corresponds with proving the TRIANGLE-CONGRUENCE-SHARED-SIDE schema, he knows, at this point, that all the part-statements in all the configurations that appear in the diagram are true. Thus, the use of this schema explains how he knows he can prove any goal statement no matter what it is.

We now turn to a general description of DC. DC has three processing stages: 1) diagram parsing, 2) statement encoding, and 3) schema search. In the computer simulation each stage is done to completion before the next begins, however, we believe that human problem solvers integrate these processes. Our simulation approach allows us to evaluate the contribution of the diagram parsing process makes to limiting search independent of the schema search process. Note that DC is intended as a model of the panning phases of skilled subjects and not the execution phases. A model of the execution phases would involve finding solutions, either by retrieval or by search in the execution space, to the series of trivial one to three step subproblems that result from planning.

3.1. Diagram Parsing and Schema Instantiation. Diagram parsing is the process of looking for configurations in the problem diagram and instantiating the schemas associated with any of the configurations identified. The diagram is input as ordered lists of the points that appear in each line and xy-coordinates for each point. From this representation, DC's diagram parser can directly recognize any occurrence of a *basic configuration*. Basic configurations are recognizable purely from their form, for example, the ADJACENT-SUPPLEMENTARY-ANGLES configuration is recognized when the end of one line meets another line somewhere in the middle. Other configurations are specializations of basic configurations in which some relationships among the parts are constrained. For example, the PERPENDICULAR-ADJACENT-ANGLES configuration is a specialization of the ADJACENT-SUPPLEMENTARY-ANGLES configuration in which the component angles are equal. To recognize these specialized configurations, DC uses appearances in the diagram to estimate the slopes of lines and sizes of segments and angles. This is a heuristic procedure which, in the case of an overspecialized diagram, can result in extra irrelevant, but harmless schemas.

DC's diagram parser also recognizes and associates certain pairs of basic configurations. For example, two basic TRIANGLE configurations can be paired if they appear to be congruent. DC recognizes apparent congruence by checking if the sides of the triangles can be paired so that each pair of sides are the same estimated size. The pairing of congruent triangles is treated explicitly in geometry textbooks, however, other pairings of basic configurations that DC forms are not commonly discussed. These pairings amount to visual ways of cueing certain inferences that are proven in the execution space using algebra. The corresponding schemas allow problem solvers to skip over the details of algebra sub-proofs which are a large source of combinatoric explosion in the execution space (see Koedinger & Anderson, in press). These schemas are called *whole-part congruence* schemas and were also identified and discussed by Greeno (1983).

The final result of diagram parsing is a network of instantiated schemas and part-statements as illustrated in Figure 4. It is interesting to note that although no problem solving search is done in this first stage, in effect, most of the problem solving work is done here. The resulting network is finite, in fact, usually quite small, and is fully instantiated. Searching it is fairly trivial.

3.2. Statement Encoding. Before search is started, the given and goal statements of the problem must be read into the system. Statement encoding corresponds to problem solvers' comprehension of the meaning of given/goal statements. We claim that problem solvers comprehend given/goal statements in terms of part-statements. When a given/goal statement is already a part-statement, DC encodes it directly by appropriately tagging the part-statement as either "known" or "desired". However, there are two other possibilities. First, if the given/goal statement is one of a number of alternative ways of expressing the same part-statement, it is encoded in terms of a single canonical form. For example, measure equality and congruence, as in $m\overline{AB} = m\overline{CD}$ and $\overline{AB} \cong \overline{CD}$, are encoded as the same part-statement. Second, if the given/goal statement is the whole-statement of a schema, it is encoded by appropriately tagging all of the part-statements of that schema as either "known" or "desired". For example, recall DC's encoding of the goal and second given of the problem discussed above and shown in Figure 2.

3.3. Schema Search. The network that results from diagram parsing contains a set of diagram configuration schemas which are possible consequences of the problem givens. In schema search, DC attempts to prove enough of these schemas so that the goal statement is proven in the process. This search amounts to looking for a path through the network that connects the given part-statement(s) with the goal part-statement(s) subject to the ways-to-prove of each schema in the path.

DC is performing a search through the space defined by its diagram configuration schemas. We call this the *diagram configuration space*. Previous models of geometry problem solving performed search in the execution space and required heuristics to guide choices in this large search space (Gelernter, 1963; Goldstein, 1973; Anderson, Boyle, & Yost, 1985). In contrast, the diagram configuration space is small enough that DC can effectively plan proofs without extra heuristics to aid search in this space. DC can perform a brute force forward search of the diagram configuration space by arbitrarily choosing any schema which can be proven at each step in problem solving. DC's default control scheme is slightly more elaborate – see Koedinger and Anderson (in press).

The feasibility of this simple control scheme is demonstrated by a task analysis we did of one of the more difficult problems the subjects solved. 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 at the most eight states.

4. Evidence for DC: Step-Skipping Regularity

In evaluating DC, it is worthwhile to consider whether the step-skipping behavior of skilled subjects could be explained in terms of an alternative abstract problem space. We consider two possible alternatives both based on modifications of the execution space. First, an abstract space can be created from the execution space by an *abstraction* process where the conditions (if-part) of execution operators are generalized, for example, by dropping a clause which, ideally, refers to some detail which can be temporarily ignored (Sacerdoti, 1974). Such "minor" clauses in the execution operators of geometry are rare – dropping clauses most often results in operators that can propose future states which cannot be proven. Such incorrect plans can cause significant efficiency problems, however, this is not our major criticism of the adequacy of this abstraction method for modelling skilled geometry problem solving. Rather, this method is inconsistent with the observation that the abstract plans of our skilled subjects were always correct. That is, the abstract inferences they made, like the ones in Table 1, never produced unprovable statements. Thus, it appears unlikely that their abstract operators have been learned through a "clause-dropping" type abstraction process.

A second approach to building an abstract problem space is by *composition* of consecutively applicable execution operators. This general approach has received numerous instantiations, e.g., ACT*'s composition (Anderson, 1983), Soar's chunking (Laird, et. al., 1987), Korf's macro-operator learning (Korf, 1987). Although most of these approaches have some stipulations of the appropriate context in which composition can occur, there is little in them that indicates whether or when some pairs of consecutively applicable operators are more likely to be composed than other pairs. Thus, we would not expect any regularity in the kinds of steps that would be skipped in an abstract problem space of composed execution operators. However, such a regularity is exactly what we observed of subjects.

We analyzed the protocols of all our subjects as illustrated in Table 1 and Figure 2. In particular, we divided each protocol into segments corresponding to planning and execution phases and we annotated the protocol with the inference steps subjects verbalized. We made a proof graph of each subject's final solution and then identified each step in this solution the subject mentioned while planning.

Our claim is that the steps taken in planning tend to correspond with diagram configuration schemas. In other words, we predicted that subjects would tend to mention statements which are whole-statements of diagram configuration schemas and tend to skip those statements which are not. For certain schemas, like the algebra-related schemas, which do not have whole-statements, we predicted subjects would only mention one part-statement of the schema, in particular, the one which concludes the inference. As an example, these predictions exactly match the planning behavior of the subject in Table 1. In the eleven other cases, the predictions were not as perfect, however, they tended to be correct. Figure 5 shows the results from all twelve cases. Clearly, there is a regularity in the steps being

skipped and DC captures a lot of this regularity. A Chi square test ($X^2(1) = 41.5$) indicates it is unlikely that the model's fit to the data is a chance occurrence ($p < .001$).

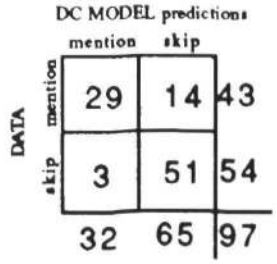

Figure 5. DC's account of the step-skipping behavior.

In addition to the evidence of regularity in step-skipping, we found other evidence in the problem solving protocols inconsistent with an abstract planning model based on compositions of execution operators. In the process of executing an abstract plan, subjects could not always immediately fill in the steps they had skipped during planning. However, if subjects learned abstract planning operators from previously compiled execution operators, the knowledge to fill in the skipped steps should be readily available. Since these execution operators remain necessary to execute proof plans, there is no reason why they would be forgotten.

Finally, there are computational reasons to question the composition-based explanation of step-skipping. On one hand, diagram configuration schemas can be viewed merely as a more compact notation for a set of macro-operators or composed production rules. On the other hand, these schemas indicate a particular organization of macro-operators and this organization may be difficult to achieve in typical composition mechanisms. To illustrate the point, consider the TRIANGLE-CONGRUENCE-SHARED-SIDE schema in Figure 3. This schema can be represented as 6 macro-operators whose left-hand sides correspond to the 6 ways-to-prove of the schema and whose right-hand sides contain 5 actions which correspond with the 5 part-statements of the schema. The collection of such macro-operators for each schema, call it S, is a restricted subset of the space of possible macro-operators. S is restricted in two ways. First, S does not contain any of the possible macro-operators which could make inferences between statements which are whole-statements of schemas, for example, it doesn't contain an operator that could infer perpendicularity directly from triangle congruence in a problem like the one in Figure 2. Second, S does not contain any of the 2, 3, or 4 action macro-operators that would be learned on the way to a 5 action macro-operator like the ones corresponding with the TRIANGLE-CONGRUENCE-SHARED-SIDE schema. To achieve DC's simplicity in search control and match to the human data, a composition mechanism would need to prevent a proliferation of unnecessary macro-operators. It is not clear how this restriction could be implemented in current mechanisms[1].

5. Discussion and Conclusion

We have posed the DC model as a challenge to current theories of skill acquisition characterized by the framework in Figure 1. The problem with this framework is not so much with the mechanisms of knowledge acquisition and knowledge tuning, but rather in the assumed form of the basic problem space which is the interface between them. It is commonly assumed that this basic problem space is made up of operators which correspond to the external actions problem solvers take in solving problems (the execution space) and that the bulk of learning is in terms of this problem space. In contrast, it seems that the human knowledge acquisition system occasionally modifies its problem space for a domain – not by modifying the operators as models of the knowledge tuning already do, but by changing the representation of problem states, for example, by creating new perceptual chunks.

That such changes in the problem state representation occur is supported by other research. In their work on the learning of the Tower of Hanoi puzzle, Anzai and Simon (1979) identified the perceptual chunking of disks into "pyramids" as crucial to learning the advanced pyramid subgoal strategy.

---

[1]One might consider whether this restriction could be achieved within the Soar architecture by having a hierarchy of problem spaces corresponding with the desired organization. However, this approach begs the question – how would this hierarchy be learned in the first place?

Research on the nature of expertise has identified the possession of perceptual chunks as a special characteristic of expertise in a number of domains (for example, see Chase and Simon, 1973). The role of these perceptual chunks in problem solving has not been well established. The DC model serves as a detailed demonstration of how perceptual chunks can be used in problem solving and, at the same time, as a challenge to current theories of skill acquisition.

A first order challenge is to specify a knowledge acquisition mechanism which is capable of perceptual chunking and of changing the basic problem space representation appropriately. Theories of categorization or Soar's data chunking (Rosenbloom, et. al., 1987) are possible candidate mechanisms. A second order challenge is to specify a knowledge tuning mechanism which can deal with the shifting nature of the basic problem space as it is changed by the acquisition of new chunks. Perhaps meeting this challenge will require rethinking the two phase framework.

REFERENCES

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

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: International Joint Conference on Artificial Intelligence.

Anzai, Y., & Simon, H. A. (1979). The theory of learning by doing. *Psychological Review, 86*, 124-140.

Chase, W. G., & Simon H. A. (1973). The mind's eye in chess. In W. G. Chase (Ed.) *Visual Information Processing*. New York: Academic Press.

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.

Greeno, J. G. (1983). Forms of understanding in mathematical problem solving. In S. G. Paris, G. M. Olson, & H. W. Stevenson (Eds.), *Learning and Motivation in the Classroom*. Hillsdale, NJ: Erlbaum.

Hayes, J. R., & Simon, H. A. (1974). Understanding written problem instructions. In L. W. Gregg (ed.), *Knowledge and Cognition*. Potomac, Md.: Erlbaum.

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

Korf, R. E. (1987). Macro-operators: A weak method for learning. *Artificial Intelligence, 27*, 35-77.

Neves, D. M. (1978). A computer program that learns algebraic procedures by examining examples and by working test problem in a textbook. In *Proceedings of the 2nd Conference on Computational Studies of Intelligence*. Toronto: Canadian Society for Computational Studies of Intelligence.

Newell, A. (in press). *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA.

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

Rosenbloom, P. S., Laird, J. E., & Newell, A. (1987). Knowledge level learning in Soar. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 499-504.

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

Unruh, A., Rosenbloom, P. S., & Laird, J. E. (1987). Dynamic abstraction problem solving in Soar. In *Proceedings of the AOG/AAAIC Joint Conference*, Dayton, OH.