From: AAAI Technical Report SS-92-02. Compilation copyright © 1992, AAAI (www.aaai.org). All rights reserved.

Diagrammatic Reasoning of Tabular Data

Todd R. Johnson, Jack W. Smith, Kathy Johnson, Nasir K. Amra, Matthew DeJongh

Division of Medical Informatics
The Ohio State University
Email: tj@cis.ohio-state.edu

Introduction

This paper reports on the diagrammatic reasoning component of RedSoar, a knowledge system for identifying red blood cell antibodies. Expert blood bank technologists make heavy use of information recorded on paper-based diagrams while solving antibody identification problems. During problem solving there is a constant interaction with the diagram, both as a source of data and as a scratch memory for recording intermediate results. RedSoar (Johnson, et al., 1991) was designed to mimic the experts' use of these diagrams. RedSoar accesses the external diagrams in ways similar to those used by experts and displays a limited ability to learn how to use the diagrams more quickly. Our work highlights both the utility and the difficulties of using such approaches for building knowledge systems. The analysis of the diagrammatic reasoning and the approach used are potentially applicable to a wide range of domains.

Motivation

Historically, the case knowledge in most AI systems has been encoded in a database or rule-base. In RedSoar we decided to model external data access through diagrammatic reasoning for two reasons. First, we wanted to closely model human problem solving. Verbal protocols illustrated that the experts made heavy use of the diagrams, including the use of visual patterns in the data and the use of the diagram to record intermediate results. Furthermore, because of the size of the diagram and the amount of data represented it seemed unlikely that humans remember the data and manipulate an internal representation of it. Supporting this hypothesis are protocols in which the expert asked to view previously seen diagrams while working on problems that require multiple diagrams.

Second, because of other research goals, dealing with flexible problem solving, we decided to use Soar (Laird, et al., 1987) for coding RedSoar. Soar places implicit limits on the size of working memory that prohibit the encoding of the complete case in working memory. One alternative is to encode the cases in long-term memory such that only a small portion of the information is in working memory at one time. This, however, is radically different from the way humans access the data and might, therefore, require a different problem-solving approach.

Based on these considerations, we decided that the best approach was to simulate the access to external diagrams of the data using a model of visual data access. Using this approach, the complete data is stored on a simulated sheet of paper and only a fixed amount of the information is present in working memory at one time.

Red Blood Cell Antibody Identification

Whenever a person requires a blood transfusion the antibodies in their blood must be identified so that a compatible unit of donor blood can be given. If the donor blood contains antigens to the antibodies in the patient's blood, the antibodies will bind to the antigens causing a transfusion reaction which can lead to complications or death. To identify antibodies, donor red cells of known antigenic makeup are placed in vials, where the red cells in a single vial have the same antigens. A sample of the patient's serum is then added to these vials. If a mixture in a vial reacts, then the patient's blood must contain at least one antibody to an antigen present on the red cells in that vial. The tests are done under various conditions designed to enhance certain reactions. All of this information is recorded in two tables (see Figure 1), the antigram, which records the antigens on the donor red cells, and the master panel of reactions, which records the reactions and test conditions. Both tables appear on a single sheet of paper along with information about the patient. Each row of the antigram represents a red cell and each column represents a specific antigen (D, C, E, etc.). A "+" in a table cell indicates that the red cell has the corresponding antigen. A "0" indicates that the red cell lacks the antigen.

Master Panel

Test Phase	Red Cell			
	1	2	3	4
IS Albumin	0	0	0	0
37 Albumin	0	0	0	0
Coombs	2+	0	2+	0
Enzyme IS	0	0	0	0
Enzyme 37	1+	0	1+	0

Antigram

Red Cell	Antigen										
	D	C	E	c	e	M	N	S	s	Le^a	Le^b
1	+	+	+	0	+	0	+	0	+	0	+
2	+	+	0	0	+	+	+	0	+	0	+
3	+	+	+	+	0	+	0	+	0	0	+
4	0	+	0	+	+	0	+	0	0	0	0

Figure 1: A partial master panel and antigram from a red blood cell antibody identification case.

Each column of the master panel represents test results of mixing a single vial of donor red cells with the patient's serum. Each row represents the conditions in which a test was performed. A "0" in a table cell means that no visible reaction occurred. All other entries record the strength of the reaction as determined by the technologist. Reaction strengths range from "+/-" for a trace reaction to "4+" for the strongest reaction.

Note that the tables shown in the figure contain less data (only four red cells) than is usually collected for a case. Usually eight or more red cells are used and 27 or more antigens appear in the antigram.

Soar and External Access

RedSoar is encoded in Soar, a cognitive architecture de-

signed to support intelligent activity. Soar is based on the problem-space computational model (PSCM) (Newell, et al., 1991) in which all problem solving is viewed as search for a goal state in a problem space. Knowledge about when operators are applicable to a state can be specified independent of knowledge about which operator to select. Operator selection knowledge, called *search-control knowledge*, is expressed in terms of preferences for or against applicable operators. If at any time during the problem solving the search-control knowledge is insufficient to indicate which operator to select, a subgoal is set up to generate additional knowledge so that a single operator can be selected. This subgoal is achieved by searching another problem space. Operators can either be implemented by directly available knowledge or by using an operator-specific problem space. Implementation in a problem space is similar to using a subfunction to implement an operator in LISP. Hence, subgoaling results in a traditional goal/subgoal stack in working memory where the topmost goal represents the highest level goal.

After looking at a diagram to locate a particular piece of information, people learn the sequence of actions it takes to get the information so that the next access is faster. Learning in Soar occurs through a process called *chunking*. Whenever a subgoal produces a result for a higher level goal, recognition knowledge that can directly produce the result *without entering the subgoal* is added to long-term memory. For example, if Soar subgoals and uses depth-first search to determine which one of several possible operators to take, then once the results of the search are known, recognition knowledge, called a *chunk*, is added to long-term memory such that the next time a similar situation arises, the chunk will directly specify which operator to take without entering the subgoal to do depth-first search.

In Soar all access to the external environment is done through the top goal. To be more precise, input and output channels can be attached to the state associated with the top goal. An input channel is constantly dumping information about the external environment into the state. An output channel sends information from within Soar to an output system. Input and output channels are connected to LISP functions that are external to the Soar architecture. These functions are responsible for monitoring the environment and sending the information to working memory (as is the case for an input channel) and for processing information sent out through an output channel.

Analysis of the Diagrammatic Reasoning

In our analysis of the diagrammatic reasoning in antibody identification we identify several necessary models or representations (see Figure 2). The antigram and master panel, D, are a model of the real situation, S, i.e., the test tubes containing serum and red cells. I_d is an internal model of the diagram and I_s, called the *situation model*, is an internal model of the situation, S. Because of size limitations on working memory I_d is assumed to be a partial representation of D. I_s represents not only the perceived data, but also inferences about S, such as hypotheses about the antibodies present in the patient's serum.

The agent, whose internal environment is shown in the figure, must have knowledge to build and use I_d and I_s. We identify four bodies of knowledge for dealing with these models: 1) knowledge to interpret the diagram to produce I_d; 2) knowledge to relate I_d to I_s; 3) knowledge to manipulate I_s; and 4) knowledge to determine how to change the contents of I_d to contain required information, i.e., knowledge about accessing parts of D.

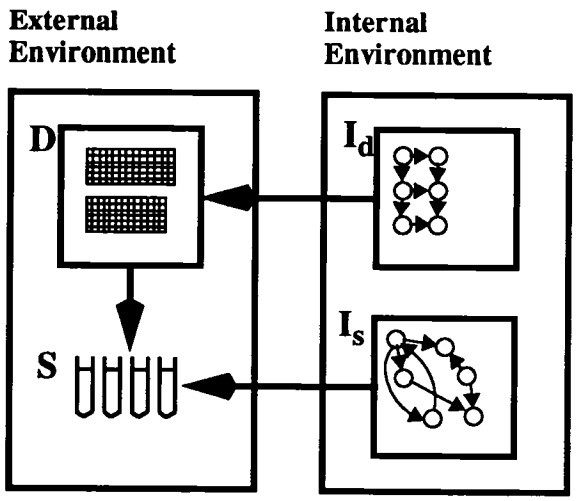

Figure 2: The models or representations used in antibody identification and how they relate to the real situation, S. The arrows point to the referent of the model. D is the diagram recording the test results and the known data (the antigens on the donor red cells). I_d is an internal model of the diagram. I_s is an internal model of S.

The second body of knowledge, relating I_d to I_s, is needed because the agent must be capable of smoothly working with both the diagrammatic representation, I_d, and the situation model, I_s. For example, if the situation model contains a representation of red cell 4 and the agent needs to extract information about red cell 4 from the diagram, red cell 4 as represented in the situation model must be linked to some representation in I_d that corresponds to red cell 4 in the diagram. This binds the internal representation of red cell 4 to perceptual data.

The implementation, based on this analysis, is described in the next section.

Diagrammatic Reasoning in RedSoar

There are two main constraints on the design of RedSoar's diagrammatic reasoning component, both designed to limit the size of working memory:

1. The external diagram must be selectively accessed (limited bandwidth); and
2. Only a partial model of the external diagram can be kept in working memory at one time (limited storage).

To satisfy these constraints RedSoar is designed to use visual markers, markers that can be placed on the external data and then moved about (Ullman, 1984; Chapman, 1989). Figure 3 illustrates the design. The external data is shown at the top with two visual markers placed on the master panel. Once a visual marker is placed, it deposits, in working memory, whatever it is placed on. Thus if a marker is placed on a 9, that number is entered into working memory as the value of the marker. Likewise, if the marker is moved or if the data it is placed on changes, these changes will be immediately reflected in working memory. The external data is actually encoded in LISP arrays to simulate the paper-based diagrams used by technologists. The transfer of information and the movement of visual markers is handled by LISP procedures connected to the corresponding input and output channels. Hence the LISP code simulates a visual system and a motor system.

The motor system accepts the following commands for placing and moving markers:

Find: Place a marker on an object with specific properties.

Walk: Move a marker one table cell in a specified direction.

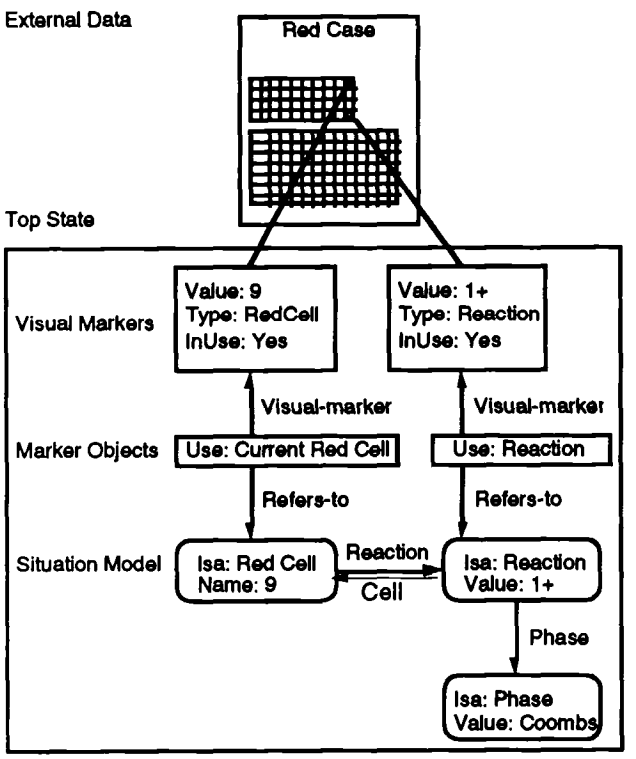

Figure 3: RedSoar's diagrammatic reasoning component.

Warp: Place a given marker at the same location as a second given marker.

Intersect: Place a marker at the point where two given markers intersect.

As shown in Figure 3, the top state in Soar contains the visual markers and marker objects that together correspond to I_d, and the situation model, I_s. Each visual marker is represented by an object in working memory that records the data that the marker is placed on and the type of the data. As stated earlier this information is placed in working memory by the visual system. The system also keeps track of which markers are in use. The marker objects relate objects in the situation model to visual markers corresponding, in some way, to those objects. Each marker object also records the specific use to which the visual marker is being put.

Since a constraint on this work is that working memory be kept small, we have placed a limit on the number of visual markers. There are ten markers that must be allocated and deallocated during problem solving. At the same time, objects in the situation model that correspond to deallocated visual markers can be removed if they are no longer needed, as would be the case when the external data is used to compute some result. This helps keep the size of working memory even smaller.

A typical problem-solving sequence from RedSoar is shown below. The operators dealing with visual markers appear in boldface type.

```
0  G: G1
1  P: P4 (Identify)
2  S:S5
...
656 O: O1235 ((I1233) Determine-Accounts-For)
Determining what Anti-K can account for.
657 ==>G: G2494 (Operator No-change)
658   P: P2498 (Determine-Accounts-For)
659   S: S2499
660   O: O2500 (Make-Marker-Obj Reaction-Level-Antigen)
661   O: O2501 (Make-Marker-Obj Reaction-Level-Red-Cell)
662   O: O2502 (Make-Marker-Obj Reaction-Level-Gene)
663   O: O2504 (Find Antigen K Reaction-Level-Antigen)
664   O: O2508 (Find Red-Cell 537A Reaction-Level-Red-Cell
665   O: O2519 (Intersect Reaction-Level-Gene
                Reaction-Level-Antigen Reaction-Level-Red-Cell)
666   O: O2523 (Get-Reaction-Expectations)
667   ==>G: G2525 (Operator No-change)
668     P: P2526 (Get-Reaction-Expectations)
669     S: S2528
670     O: O2529 (Determine-Zygosity Antigen K)
671     ==>G: G2530 (Operator No-change)
672       P: P2532 (Determine-Zygosity)
673       S: S2533
674       O: O2534 (Make-Marker-Obj Dz-Allele-Marker)
675       O: O2536 (Find Allele Little-K Dz-Allele-Marker)
676       O: O2539 (Intersect Dz-Allele-Marker Dz-Allele-Marker
                    Reaction-Level-Red-Cell)
677     O: O2546 (Determine-Antigen-Strength)
678     O: O2547 (Predict-Reactivity)
...
718   O: O2735 (Determine-Explanation)
Anti-K explains 3+ on cell 537A phase COOMBS.
Anti-K explains 1+ on cell 537A phase ENZYME37.
```

This trace is interpreted as follows. On line 656 an operator has been selected to determine what the antibody Anti-K can account for since Anti-K has been hypothesized to be in the real world situation. To implement the *accounts-for* operator a subgoal is entered (lines 657-659). The first operator in the subgoal is *make-marker-obj* (line 660) which creates a marker object to be used to link a visual marker pointing to the antigen K in the diagram with the corresponding antibody object (Anti-K) in the situation model. At the same time, the operator allocates a free visual marker to the marker object. Then two more marker objects are made for the red cell and the space where gene presence is indicated (the "+" or "0" in the antigram). Next, (line 663) the name of the antigen is used by the *find* operator to place the antigen marker on the antigram. Similarly, the name of the red cell is used to *find* where the red cell marker goes on the antigram. Then, the red cell marker and the antigen marker are given to the *intersect* operator and the result of the intersection is marked by the gene marker. There was a "+" under the gene marker, so it is indicated in the model that K is on red cell 537A and an operator to predict the reactivity of the cell is selected (*get-reaction-expectations*, line 666). *Get-reaction-expectations* is implemented in a subgoal by first determining the zygosity of the antigen on the cell (line 670), then determining the antigen strength (line 677), and finally predicting the reactivity (line 678). The operator *determine-zygosity* is also implemented in a subgoal using another marker to fine the allele for Anti-K. A marker object is made for the allele (line 674). The allele (little-K) is found on the antigram (line 675) and its marker is intersected with the red cell marker (line 676). The allele marker was actually moved to point to the gene presence indicator rather than creating a new marker object for the intersection. After processing the rest of the cells to see if the antigen was on them and predicting the reactivity for each cell, the operator *determine-explanation* is selected to make the final determination of the reaction strength for each cell and this information is used to see how complete the model of the situation is (i.e. have all the reactions been explained).

Chunking takes place whenever a result is returned to a supergoal. Since the motor system operates by way of commands placed on the top state (S5 on line 2), the intermediate results of the determine-accounts-for subgoal (the marker values) are returned to S5. Each command, X, to the motor system produces one chunk of the form:

If *conditions* then issue motor command *X*.

Hence, once the system has learned how to do the actions required in *determine-zygosity* for a particular antigen, it should not need to enter the subgoal (lines 671-676) to determine the zygosity for other antigens. In fact, if the same case is run using the chunks produced from determining the zygosity for the first antigen, the system will never enter the subgoal, not even for processing the first antigen. In other words, the system has learned recognition knowledge that lets it quickly scan the diagram for the information it needs. We have achieved some success toward this behavior but many problems remain. These results and others are described in a later section.

RedSoar can modify the diagram (the master panel or antigram) by drawing a slash through a selected cell. The motor system supports this with a slash operator that takes a marker as an argument and draws a slash on the diagram. The change is immediately reflected in working memory via input through the visual marker. During problem solving, slashing is used to indicate those antibodies that have been ruled-out.

Results

The results obtained so far with RedSoar's diagrammatic reasoning have been mixed. The advantages are:

1. The approach, as expected, allows the problem-solving behavior of human experts to be more closely modeled. Because the data is arranged in the same order as it is presented to a human expert, the program's behavior displays some of the same sequences of actions as the experts.
2. The techniques can be applied to a wide range of tasks in which data is recorded in a tabular format. It might even be possible to generalize RedSoar's visual system to cover any tabular representation of data.
3. The approach is successful at limiting the amount of information in working memory.
4. The approach forces us to consider problem-solving methods that work with limited information and interact with the environment for additional information.
5. The system has been able to learn short sequences of output commands that significantly decrease the time required to access the diagram; however, significant problems remain as discussed below.

There are four problems with the current approach:

1. The visual system is difficult to use from a programmers point-of-view. Despite providing high-level operators to facilitate programming, the visual markers and marker objects require extra programming overhead. Unlike accessing a database, accessing a diagram using visual markers can require a large sequence of operators. The current design of the system does not attempt to encapsulate or hide the external access, thus the encoded methods are tightly tied to output commands and visual markers. An alternative is to encapsulate much of the access in subgoals so that it is largely hidden from the programmer of higher level problem spaces.
2. The visual system is specialized for the RedSoar domain. The visual system knows about the values of the table cells and the types of values.
3. The learning results have been disappointing. Despite much effort, the system cannot be used with learning enabled for extended periods of time. We have been able to fix some of the problems, but many remain to be solved. In particular the creation of redundant chunks appears to be causing many of the current problems; however, there are no clear solutions to this problem. Redundant chunks lead to multiple output commands that are semantically equivalent but syntactically different and it is difficult to detect and handle these cases. Problems also result from the changing values of visual markers as they are moved across the diagram. In such cases, the current version of Soar tends to build erroneous chunks that can never fire.
4. There is a sharp separation between the visual system and central cognition. In particular, all information transfer is done using low bandwidth channels and no learning can occur in the visual system. If a knowledge system is to learn to recognize additional tables and symbols then either the visual system must be designed to deliver extremely primitive information to working memory (so that central cognition can process it), or the visual system must learn to recognize and deliver to working memory higher level symbols.

Conclusion

The study of diagrammatic reasoning in antibody identification has resulted in a method for reasoning about tabular data that compares favorably to the way experts perform. Also, it appears that this approach is applicable to any domain in which tabular data is used for problem solving. While many issues about external data access have been clarified, there have been several questions raised. It is not clear how complex the objects returned from the visual system should be or how and where learning to recognize objects and patterns takes place. In response to the problems, we are currently studying chunking and how to facilitate use of the approach. An experimental modification of Soar's chunking mechanism appears to solve many of the problems with learning and we are exploring alternative problem space designs as well as additional modifications to the Soar architecture to solve the remaining problems.

Acknowledgments

We thank the members of the Division of Medical Informatics and the members of the Soar community for their comments and discussion on this work. This research is supported by National Heart Lung and Blood Institute grant HL-38776 and National Library of Medicine grants LM-04298 and F37LM00003.

References

Chapman, D. (1989). Penguins can make cake. *AI Magazine*, 10(4), 45-50.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33, 1-64.

Johnson, K. A., Johnson, T. R., Smith, J. W., Jr., DeJongh, M., Fischer, O., Amra, N. K., and Bayazitoglu, A. (1991). RedSoar—A system for red blood cell antibody identification. In *Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care*, (pp. 664-668). Washington D.C.: McGraw Hill.

Newell, A., Yost, G., Laird, J. E., Rosenbloom, P. S., and Altmann, E. (1991). Formulating the problem space computational model. In R. F. Rashid (Eds.), *Carnegie-Mellon Computer Science: A 25-Year Commemorative Reading*, MA: ACM-Press: Addison-Wesley.

Ullman, S. (1984). Visual Routines. *Cognition*, 18, 97-159.