15:42:37 OCA PAD INITIATION - PROJECT HEADER INFORMATION 10/13/87

Active

Project #: G-36-678 Cost share #: Rev #: 0
Center # : R6401-0A0 Center shr #: OCA file #:
Work type : RES
Contract#: GRANT DTD 9/25/87 Mod #: Document : GRANT
Prime #: Contract entity: GTRC

Subprojects ? : N
Main project #:

Project unit: ICS Unit code: 02.010.142
Project director(s):
KOLODNER J L ICS
CULLINGFORD R E ICS

Sponsor/division names: LOCKHEED - PALO ALTO /
Sponsor/division codes: 211 / 009

Award period: 870903 to 880228 (performance) 880331 (reports)

Sponsor amount	New this change	Total to date
Contract value	50,000.00	50,000.00
Funded	50,000.00	50,000.00
Cost sharing amount		0.00

Does subcontracting plan apply ?: N

Title: CASE-BASED REASONING: INFERENCE METHODS AND SUPPORT RESOURCES

PROJECT ADMINISTRATION DATA

OCA contact: John B. Schonk 894-4820

Sponsor technical contact

BOB YATES
(415)424-2474
LOCKHEED MISSILES & SPACE COMPANY
3251 HANOVER STREET
PALO ALTO, CA 94304-1117

Sponsor issuing office

SAME

Security class (U,C,S,TS) : U
Defense priority rating : NONE
Equipment title vests with: Sponsor

ONR resident rep. is ACO (Y/N): N
supplemental sheet
GIT X

Administrative comments -
PROJECT INITIATION
REFERENCE P.O. NUMBER SKPAH5640F ON CORRESPONDENCES AND INVOICES

EORGIA INSTITUTE OF TECHNOLOGY   OFFICE OF CONTRACT ADMINISTRATION

SPONSORED PROJECT TERMINATION/CLOSEOUT SHEET

Date 4/18/88

Project No. G-36-678   School/~~Lab~~ ICS

Includes Subproject No.(s) N/A

Project Director(s) J. L. Kolodner   GTRC/GIT

Sponsor LOCKHEED - PALO ALTO

Title CASE-BASED REASONING: INFERENCE METHODS AND SUPPORT RESOURCES

Effective Completion Date: 2-28-88 (Performance) 3-31-88 (Reports)

Grant/Contract Closeout Actions Remaining:

- ☐ None
- ☐ Final Invoice or Copy of Last Invoice Serving as Final
- ☐ Release and Assignment
- ☐ Final Report of Inventions and/or Subcontract: Patent and Subcontract Questionnaire sent to Project Director ☐
- ☐ Govt. Property Inventory & Related Certificate
- ☐ Classified Material Certificate
- ☐ Other

Continues Project No.   Continued by Project No.

COPIES TO:

Project Director	Facilities Management - ERB
Research Administrative Network	Library
Research Property Management	GTRC
Accounting	Project File
Procurement/GTRI Supply Services	Other
Research Security Services	
Reports Coordinator (OCA)	
Program Administration Division	
Contract Support Division	

Georgia Tech

Georgia Institute of Technology
School of Information and Computer Science
Atlanta, Georgia 30332-0280
404-894-3152

March 29, 1988

Ralph Barletta
Lockheed AI Center
Building 259
3251 Hanover Street
Palo Alto, CA 94304-1191

Dear Ralph:

The following papers will serve as the final report for the project on case-based reasoning that you are supporting at Georgia Tech. These papers were all submitted to the AAAI or Cognitive Science or DARPA Case-Based Reasoning Workshop during the month of March, and they summarize the work we have been doing on case-based reasoning over the past six months. As you can see, our major topics of concern were integrating case-based reasoning with other reasoning methods to achieve a planner that combines opportunistic and reactive behavior (Hinrich's paper and Turner's two papers), creating a memory for cases that run on a parallel machine (my paper), and adapting an old solution to fit a new one through a kind of analogy that creates an abstraction of the old and new case as part of the adaptation process (Shinn's two papers). We also have a student who has been looking at using case-based reasoning for scheduling a flexible manufacturing system (David Wood). When he has something written up, I will send you a copy.

The money has been quite useful to us. Thank you for your support.

Sincerely,

[illegible]

Janet L. Kolodner
Associate Professor

Enclosures

Retrieving Events from a Case Memory: A Parallel Implementation[1]

Janet L. Kolodner
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

Perhaps the most important support process a case-based reasoner needs is a memory for cases. In this paper, we describe a parallel retrieval algorithm that can be used to retrieve cases from a hierarchically organized memory for cases given the description of some new case as a retrieval probe. We also describe the structure of the memory it works on. The organization of cases in memory is based on previous work by Schank and Kolodner. The retrieval algorithm is a concept refinement search algorithm and is based on work by Riesbeck and Martin that is implemented in DMAP. It is implemented in a program called PARADYME (Parallel Dynamic Memory) that is designed to work alongside a case-based problem solving program. There are four parts to PARADYME: a hierarchically-organized memory for cases, a concept refinement retrieval process, heuristics for choosing the best out of several retrieved cases, and heuristics for respecifying a retrieval probe when it is over- or under-specified.

1 Introduction

Perhaps the most important support process a case-based reasoner needs is a memory for cases. The memory must make cases accessible when appropriate retrieval cues are provided to it and it must incorporate new cases into its structures as they are experienced, in the process maintaining accessibility of the items already in the memory. It must be able to handle cases in all of their complexity, and it must be able to manage thousands of cases in its memory. In this paper, we discuss a parallel retrieval scheme for a conceptual memory based on previous research into memory organization and retrieval methods (e.g., Kolodner, 1983, Kolodner & Cullingford, 1986, Schank, 1982, Reiser, 1983, Martin & Riesbeck, 1986). While the abstract principles of the previous work remain the same, the details have been modified in several ways. The model to be presented, called PARADYME, has four parts:

1. a hierarchical organization of knowledge and cases
2. a parallel memory retrieval process that uses a concept refinement approach to retrieval
3. a set of transformation rules that transform and elaborate a retrieval probe to get a better "best match" than is possible from the original set of cues

[1]This research was supported in part by DARPA under Contract No. N0039-87-C-0026 to Thinking Machines, in part by NSF under grant No. IST-8608362, and in part by Georgia Tech, the Georgia Tech Foundation, and Lockheed AI Center. This work was done while the author was on sabbatical at Thinking Machines Inc., Cambridge, Mass. Thanks to Thinking Machines for providing machine and programming support for the project. Programming was done by Eyal Yaari. Thanks also to Alex Kass, Phyllis Koton, Chris Owens, Chris Riesbeck, and Robert Thau for enlightening comments made during discussion of my scheme. A shorter version of this paper was submitted to AAAI-88.

4. a set of heuristics that choose the best matching case from those that are activated

PARADYME is implemented on the Connection Machine, a SIMD parallel machine, in a program by the same name. Because we want PARADYME to be able to work along with a problem solving system, we have given it knowledge and cases from a case-based reasoning system that is under development. Thus, PARADYME currently uses JULIA's (Hinrichs, 1988, Kolodner, 1987a, b, Shinn, 1988) knowledge structures and cases. Because the cases are some that JULIA has processed, the cases are full problem solving experiences represented in their entirety. JULIA's, and therefore PARADYME's, domain is meal planning.

2 Background

There are several requirements we put on a memory for cases:

1. Best matching cases must be retrieved using a set of retrieval cues that provide a partial description of the item to be retrieved.
2. Memory should return small numbers of cases rather than large numbers. If large numbers of cases match an underspecified description, then either a prototype, a generalization, or a request for more information should be returned by memory.
3. Retrieval should be fast. It is done in the context of reasoning and we want reasoning to be fast. Therefore, it is preferable to have the hard work done at memory update time rather than at retrieval time. Retrieval processes should be fairly uncomplicated.
4. Retrieval time should not increase as the memory grows.
5. Generalizations and cases should be equally accessible.

Retrieving appropriate cases from a case memory is essentially a massive search problem that requires retrieval of a best match rather than an exact match. Given a partial description of a situation, it is up to the case memory to recall the case from memory that best matches the new situation. In our initial work on this problem, we chose to take our inspiration from people (Kolodner, 1983, 1984, Schank, 1982). The models that came from these studies, Schank's (1982) dynamic memory and Kolodner's (1983, 1984) CYRUS, hypothesized several things:

1. Memory categories are associated with concrete types of situations. Each category holds general information about the contents of such situations, the relationships between characters, props, and actions in such situations, and the causal and temporal consequences and antecedents of the situations. These categories are arranged in abstraction hierarchies and packaging hierarchies (Schank, 1982).

2. These memory categories, called MOPs, also organize indexing structures. Indexes associated with each category differentiate items in the category from each other. When several items share the same set of indexes, a more specialized category (a subMOP) is formed and items are organized in the same way within those categories. Physically, in our implementations, items in categories were organized in multiple redundant discrimination nets (Kolodner, 1984, Lebowitz, 1983).

3. Items are found in memory by first choosing a small set of categories to confine search to and then using the features of the specified event to designate which branches of the organizational structure should be traversed. Traversal happens in parallel among indexes at the same level of memory, and traversal finishes when an appropriate item or set of items with a subset of the specified features is found. (Kolodner, 1984 explains in more detail.)

4. There are several circumstances under which such search does not succeed, and there are retrieval strategies to deal with each of these search problems. One kind of strategy identifies categories for search if none is designated in a retrieval probe. Another elaborates retrieval probes if memory traversal fails before a particular event is found. Another creates context for the retrieval probe and directs traversal functions to search in a different part of memory for items with this created context.

5. Memory update functions choose indexes for events by choosing those features from an event that specialize or violate norms of the category the event is being indexed in and lead to unanticipated consequences within that category. These functions create specialized categories by a similarity-based induction method: When several items are indexed by the same feature or set of features, the similarities of those items is extracted and a new category is formed.

The search method embodied here is a "concept refinement" method, which provides much more control over the portions of memory that get activated than does an intersection search.[2] In concept refinement search, a concept is not "turned on" until its parent in the abstraction hierarchy is accessed and some feature that specializes it with respect to the parent is specified. CYRUS (Kolodner, 1983, 1984), IPP (Lebowitz, 1983), and MOPTRANS (Lytinen, 1986) did this through a "locked network" in which traversal to a lower level of an abstraction hierarchy could not be done unless the higher level had already been accessed and the label associated with the index to the item at the next lower level was specified. DMAP (Riesbeck & Martin, 1986; Martin & Riesbeck, 1986), which our method is based on, implements concept refinement in another way. An item can be accessed if one of its antecedents in the abstraction hierarchy is activated by the probe, if that abstraction predicts another concept, and if some specialization of the predicted concept is specified in the probe. DMAP's method has the advantage of not requiring a redundant indexing scheme. The predictions DMAP makes are linguistic, but we have generalized them for searching a conceptual memory for events.

[2]MBR (Stanfill, 1987) is a massively parallel search technique that uses intersection search. It also runs on the Connection Machine. Its representations are both flat and monolithic (homogeneous). MBR broadcasts to each item in its memory in parallel. Its major activity is running a similarity metric to measure how close each of the items in memory is to what it is looking for. MBR has been run on large databases but never on hierarchical, heterogeneousm, or distributed structures.

In the scheme to be presented, we have created a memory system that upholds the principles presented above in a parallel implementation. While memory remains hierarchical, we have decoupled the retrieval procedures themselves from memory's organization. Memory organizes generalizations but does not require that the organization be used to access memory. While in CYRUS' implementation, indexes were used to block passage through memory, insuring that only relevant nodes were accessed, in PARADYME, predictions made by memory's knowledge structures identify which cases are good candidates for retrieval. We have also changed representations significantly. While in CYRUS, events were monolithic structures, PARADYME has a distributed representation.

3 Representing Knowledge and Cases

Representation in PARADYME is similar to that described in Schank's *Dynamic Memory* (1982). That is, the details of any particular event (case) are distributed throughout memory in two ways. First, they are distributed in an abstraction hierarchy associated with the kind of situation the event is an instance of (i.e., its type). Thus, a particular Mexican meal with death chili as its main course will have its description distributed in "meal", which says this kind of event has several eating scenes (only some are shown in Figure 1), that the participants want to satisfy hunger, that the main event is ingesting the main dish of the main course, etc.; "mexican meal", which says this kind of event has food of mexican cuisine, that a particular set of spices can be expected, that the drink of choice is Mexican beer, that food tastes spicy, etc.; and "death-chili-meal", which gives the details of this meal, e.g., who the eaters are, that they are mostly people who like very spicy food, where the meal took place. The center of Figure 1 illustrates this.

Second, details of events (cases) are distributed throughout abstraction hierarchies associated with scenes of the event. Schank (1982) called this a packaging hierarchy. In the case of meals, its scenes include food preparation, eating the appetizer, eating the main course, etc. Thus, details about what was served in the death-chili-meal appear in memory in knowledge structures describing meal scenes. The fact that the main course was death chili is distributed through the abstraction hierarchy of "meal-main-scene', as shown on the right side of Figure 1, while the fact that the appetizer was guacamole is distributed through "meal-appetizer-scene's" hierarchy, on the left side of Figure 1. "Meal's" other scenes are not shown.

We distribute representations in this way to allow the case-based reasoner to use small chunks of cases in its reasoning rather than having to wade through large cases and to allow generalization across scenes common to several kinds of situations (Schank, 1982). A retrieval probe might activate a full event with its scenes or only the representation for a particular scene. With a flat representation (i.e., no abstraction), generalizations must be recreated each time they are needed. With a monolithic structure (i.e., all aspects of the event in one hierarchy), it would be hard to make generalizations across different types of events.

There are several things to notice about this representational scheme that will be significant in judging the retrieval process. First, general knowledge (e.g., about meals and mexican meals) and details of particular cases are organized in the same structures. Thus both are equally accessible and accessible by the same retrieval methods. Second, we provide retrieval algorithms with a

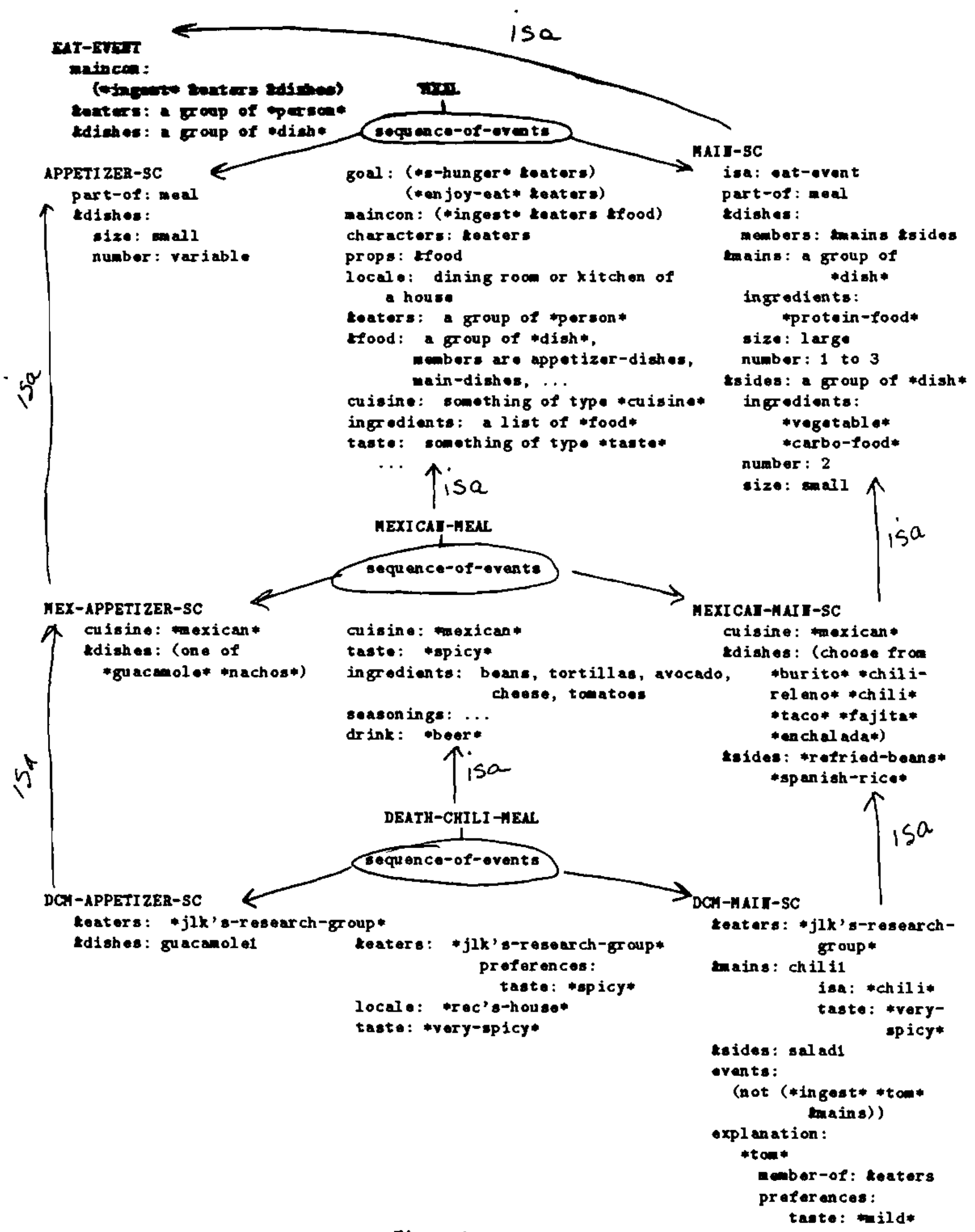
isa
EAT-EVENT
maincon:
(*ingest* &eaters &dishes)
&eaters: a group of *person*
&dishes: a group of *dish*
MEAL
sequence-of-events
APPETIZER-SC
part-of: meal
&dishes:
size: small
number: variable
goal: (*s-hunger* &eaters)
(*enjoy-eat* &eaters)
maincon: (*ingest* &eaters &food)
characters: &eaters
props: &food
locale: dining room or kitchen of a house
&eaters: a group of *person*
&food: a group of *dish*, members are appetizer-dishes, main-dishes, ...
cuisine: something of type *cuisine*
ingredients: a list of *food*
taste: something of type *taste*
...
MAIN-SC
isa: eat-event
part-of: meal
&dishes:
members: &mains &sides
&mains: a group of *dish*
ingredients: *protein-food*
size: large
number: 1 to 3
&sides: a group of *dish*
ingredients: *vegetable* *carbo-food*
number: 2
size: small
isa
isa
isa
MEXICAN-MEAL
sequence-of-events
MEX-APPETIZER-SC
cuisine: *mexican*
&dishes: (one of *guacamole* *nachos*)
cuisine: *mexican*
taste: *spicy*
ingredients: beans, tortillas, avocado, cheese, tomatoes
seasonings: ...
drink: *beer*
MEXICAN-MAIN-SC
cuisine: *mexican*
&dishes: (choose from *burito* *chili-releno* *chili* *taco* *fajita* *enchalada*)
&sides: *refried-beans* *spanish-rice*
isa
isa
isa
DEATH-CHILI-MEAL
sequence-of-events
DCM-APPETIZER-SC
&eaters: *jlk's-research-group*
&dishes: guacamole1
&eaters: *jlk's-research-group*
preferences:
taste: *spicy*
locale: *rec's-house*
taste: *very-spicy*
DCM-MAIN-SC
&eaters: *jlk's-research-group*
&mains: chili1
isa: *chili*
taste: *very-spicy*
&sides: salad1
events:
(not (*ingest* *tom* &mains))
explanation:
tom
member-of: &eaters
preferences:
taste: *mild*

Figure 1

natural way of knowing if there are a large number of cases that partially match its retrieval probe without requiring it to activate all those cases. This is possible because details that appear high in the hierarchy do not get repeated lower in the hierarchy. This will allow memory to either return general knowledge that is activated by a probe or to ask for more specific knowledge to differentiate between the items with a given description. In a memory with many similar cases, this is an advantage.

The memory scheme also imposes a hard problem on the retrieval functions. The problem is that during retrieval, retrieval cues might hit event descriptors in several different structures. There must be a way to put those structures back together again. We shall see that the "concept refinement" step of the retrieval algorithm addresses that issues.

4 Retrieval Probes

Retrieval probes partially describe an event to be retrieved by specifying a subset of the target event's features. Let us consider, for example, some of the ways the "death chili meal" might be partially described. [3]

1. a meal with chili

```
(and (? isa meal) (? dishes chili))
```

2. a mexican meal with very spicy chili

```
(and (? isa meal) (? cuisine mexican) (and (? dishes chili) (? dishes
                         (taste very-spicy))))
```

3. a mexican meal with very spicy food

```
(and (? isa meal) (? cuisine mexican) (? dishes (taste very-spicy)))
```

4. a meal with chili as the main course

```
(and (? isa meal) (? appetizer-scene (dishes guacamole)))
```

5. a mexican meal with avocado

```
(and (? isa meal) (? cuisine mexican) (? dishes (ingredients avocado)))
```

6. a mexican meal with guacamole and very spicy chili

[3] We do not discuss here how this translation happens. A phrase-based analyzer such as DMAP (Riesbeck, 1986) or PHRAN (Arens, 1981) could do it easily. Were DMAP used, it could be easily integrated with what we describe here. A very well integrated system, however, would probably do the language and memory retrieval work at the same time without the need to explicitly create these queries. Their equivalent would have to be created internally, however.

```
(and (? isa meal) (? cuisine mexican) (? dishes guacamole) (and (?
        dishes chili) (? dishes (taste very-spicy))))
```

The important thing to notice in these representations is that they do not distinguish which scene of the specified meal holds the specified descriptors unless that fact is given explicitly in a query. While it is easy to determine that prepared dishes (e.g., chili and guacamole) referred to in a query about a meal refer to its dishes, it requires a lot more knowledge to determine in which scene those dishes were served. In fact, it requires the full extent of knowledge represented about meals in the memory. It would be inefficient to first disambiguate and then find matches since both use the same knowledge. And some of the ambiguity is useful. Instead, disambiguation happens at retrieval time as a byproduct of the retrieval process.

One might ask whether such ambiguous probes will be made by a problem solver that is in control of what gets asked of memory. Sometimes probes to memory made by a problem solver will specifically mention a scene and sometimes they will not. If the problem solver is trying to plan a particular scene, it will be specified. But if the problem solver is trying to deal with a vague statement by a user, the probes may be as above. Suppose, for example, that a user asking JULIA to plan a meal said "Let's serve something with avocado". The problem solver might send a probe to memory that looks like (5) above in order to get ideas about how to use avocado in the meal.

5 The Retrieval Process

During retrieval, each of the features of the memory probe is broadcast into memory. Each item in memory with a broadcast feature is activated. As in DMAP's (Martin & Riesbeck, 1986) memory access process, each time an item is activated, it sends activation to each item above it in the abstraction hierarchy and it sends predictions to items that are normally seen in the context of the activated item. When those messages meet at a node, the concept that sent the prediction gets refined (specialized) to the level of detail of the concept that sent the activation. The algorithm has the following steps:

1. Each item (cue) in the memory probe is transmitted to memory (a serial process) and each is broadcast through the whole memory in parallel. Memory is activated as follows:
    (a) If the probe names a memory concept, (e.g., is of the form (? is-a x)), then the named concept (x) is activated.
    (b) If the probe is descriptive (e.g., is of the form (? property-name property-value)), any item that holds that description is activated.
2. As in DMAP, each node that is activated sends prediction messages to the things it predicts. At present, events predict their sequence of events. This is in keeping with observations of people that show that more concrete descriptions are better for reminding (Kolodner & Cullingford, 1986). By predicting the parts of an event, we are predicting its concrete features. A prediction message in PARADYME has three parts: [4] its source, its target, and the relationship between them.

[4]In DMAP, it has four.

3. **Also as in DMAP, each activated node sends an activation message to each of its antecedents in the abstraction hierarchy. The activation message contains the source of the activation and instructs the nodes it is sent to to activate themselves.**

4. **When predictions and activations meet each other, "concept refinement" happens. During** concept refinement in PARADYME,[5] the concept that sent the prediction gets specialized to the level of detail of the concept that sent the activation. This is done by finding the node that has the same relationship to the concept that sent the activation that the predicting concept has to the predicted one. "Meal-main-sc", for example, is related to "meal" through "meal"'s "sequence of events". If "meal" is activated and predicts "meal-main-sc" and "dcm-main-sc" is activated and activates "meal-main-sc", then "meal" is refined by finding the item whose "sequence of events" "dcm-main-sc" is in ("death chili meal"). Extra activation is then given to those nodes taking part in the concept refinement to distinguish them from other activated nodes in memory.

An example will illustrate. Consider, for example, a probe of the memory shown in Figure 1, using the probe "a meal with chili", represented as follows:

(and (? isa meal) (? dishes chili))[6]

Step 1 will activate the "meal" node and each node with chili specified as a dish. The "death-chili-main-scene" will be activated by the chili probe, as will mexican-main-sc and any other eating scene where chili was a dish. In step 2, "meal" will predict its scenes and "death-chili-main-scene" and other activated scenes will predict their sequence of events. In step 3, "death-chili-main-scene" and other activated scenes will activate "meal-main-scene", which will activate anything above it. "Meal will also activate anything above it. In step 4, the connection between "meal" and "death-chili-main-scene" will be made (as well as connections between "meal" and any other eating scenes with chili). Because "meal" predicts "meal-main-scene" and "death-chili-main-scene" activates it, and because the relationship of "meal" and "meal-main-scene" is through sequence of events, memory activates the item that has "death-chili-main-scene" in its sequence of events, specifically "death-chili-meal". "Death-chili-meal" and the constellation of nodes that contributed to its activation receive extra activation.

Let us go back to the algorithm and examine what it does in each step. At the end of step 1, every item in memory that partially matches the retrieval probe is activated. This step is linear in the size of the retrieval probe. After step 1, all possible candidates are activated, but we do not yet know the connections between them. Some are descriptions of situations (MOPs) and some are descriptions of scenes. We want to retrieve those situations that have had concrete features of their scenes described in the retrieval probe. The next three steps make those connections.

In step 2, situations predict their scenes while scenes predict their events. Each is predicting its more concrete parts. We do not currently do this recursively, so this is a one-step process.

[5]This is somewhat more limited than in DMAP, where an arbitrary function can be executed to refine the concept. We will add additional capabilities of this type as we find we need them.

[6]We ignore the fact that chili is embedded in the representation for now. The program can take care of that, and in terns of complexity, it adds a number of cycles equal to the depth of the embedding.

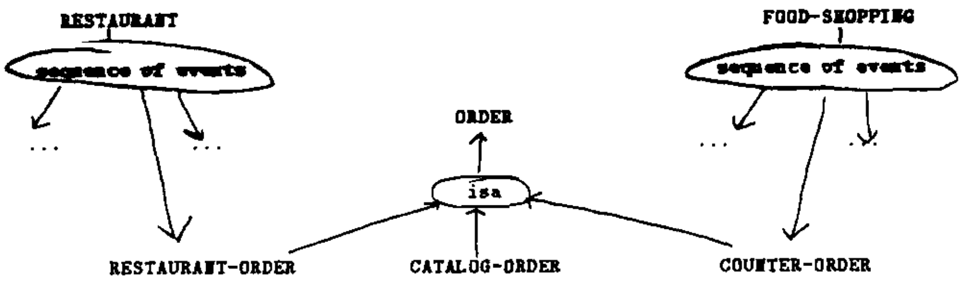

Figure 2

In step 3, each activated item sends activation up its abstraction hierarchy, in essense notifying more abstract nodes that it was described in a probe. This step is linear in the depth of the longest hierarchy being traversed.

Step 4 collects up those predictions that were fulfilled, usually scenes that were described. For each fulfilled prediction, the abstract concept that made the prediction is specialized to the level at which it meshes with the scene that was activated. The full event (case) that is retrieved (given high activation) is the specialized concept that is both of the right type (e.g., meal) and at a level of specificity consistent with the scene descriptions specified in the retrieval probe. This number of cycles required here is the depth of the abstraction hierarchy between the abstract node and the refined one.

While we can see from this small example how connections between different parts of the memory get made, it is hard to appreciate the full power of this algorithm from the examples given. We give one more example from a different domain to show how the concept refinement step narrows down the set of candidate matches to only those that are in the right ballpark. Consider a memory that knows about restaurant visits and buying. The structure of the memory is shown in Figure 2. We can see that the "ordering" scene is shared by both "restaurant visit" and "buying", and that the "ordering scene" holds instances of ordering bluefish in a restaurant and ordering bluefish over the counter in a supermarket. Suppose the query is "remember when we ordered bluefish in that restaurant in Boston". Step one of the query would activate "restaurant visit" and each of the instances of ordering bluefish, among other things. Because "restaurant visit" predicts a particular type of "ordering", namely "restaurant ordering", that ordering scene and the restaurant visit will be hooked up during concept refinement, and the supermarket ordering scene will not get further activated. In a memory with a lot of instances of ordering bluefish at a supermarket and only a small number of instances ordering bluefish in a restaurant, concept refinement will narrow the set of retrieved cases to only the relevant ones. In other words, it confines search to the specified context.

6 What Gets Returned

The result of running this retrieval algorithm on the memory is that several constellations of memory nodes are highly activated. Each constellation represents a case or set of related cases that partially match the retrieval probe. A case in memory is represented by a constellation of nodes spread over several abstraction hierarchies. The cases that are accessed by this method can be found by finding the most specific nodes in the hierarchy whose top is of the type requested in the retrieval probe. Sometimes the most specific active node in a hierarchy will be a generalized description of several cases (e.g., "mexican meal"). If so, memory returns the generalization in lieu of the myriad of cases it organizes. Sometimes there will be several most specific nodes highlighted in a hierarchy. If there are a small number (1 - 3), memory returns them all. If there is a large number, memory has a choice of returning some generalized description that subsumes them all (if one exists), creating and retruning a generalized description that subsumes them (if none exists), returning the entire set, or returning a message saying that more information is needed. Based on our experiences with case-based reasoners, the generalized description that subsumes them all plus the message saying that more information is needed would be most helpful.

7 Choosing the Best Case

While "concept refinement" insures that recalled events are in the right ballpark, it does not by itself choose which is the best match. A fully automated case-based problem solver needs to know which of the many events made available to it is the best to use for problem solving. This could be done by some sort of counting scheme or weighted counting scheme in which the match between the retrieval probe and each activated item gets points for each match to the retrieval probe and loses points for each mismatch. Such a method is problematic, however, for two reasons. First, if the evaluation function is static, it doesn't allow for dealing with the changing importance of features in context. Second, such a method requires a principled way of determining how to weight the features. Although we do not present the choice of a best match as a weighting scheme, one could think of our approach as addressing the problem of how to choose weightings for the features.

There are two major ways people are addressing this problem in the case-based reasoning community. Some people are addressing it by trying to determine how to best choose indices (e.g., Hammond, 1986, Hunter, 1988, Kolodner, 1983) so that only the best cases will be retrieved from the memory. Addressing the problem this way, the work happens at memory update time and retrieval remains a fast process. Others have filtering methods that are used after retrieval (e.g., Koton, 1988, Owens, 1988, Riesbeck, 1988, Stanfill, 1987). Others combine those two methods (e.g., Simpson, 1985, Barletta, 1988).

Our approach to choosing the best case borrows from both methods. In PARADYME, cases are analyzed for their most important features at memory update time, and conjunctions of predictive features are marked as important. At retrieval time, selection processes working after concept refinement prefer those events with full matches in those conjunctions of features. In this way, best events are chosen not merely by counting the number of features that match or even by ranking features with respect to each other, but rather by taking into account which features or combinations of features have been found to be most important in the past. In principle, this allows the importance of features to be judged in context, where context is provided by the retrieval probe and the items that are retrieved by applying a concept refinement retrieval algorithm to it.

Conjunctions of features that are marked as important in PARADYME are those that predict solutions or solution methods. The reason for this is that PARADYME is designed to work along with a problem solver, and these are the kinds of predictions a problem solver needs. There are two kinds of conjunctive feature sets PARADYME uses.

1. Goals, constraints on these goals, and environmental features that went into choosing the method or solution for achieving the goal or goal set are marked.

   A set of features may include one goal or several goals. It includes one if the solution that was chosen for that goal did not involve other goals. It includes several if their solution was integrated. Constraints and descriptors on these goals are also included, as are features of the world or features of the problem that determined which of several possible solutions or solution methods was chosen. If all of the features in one of these conjunctive feature sets is designated in a retrieval probe, the solution or solution method used in the previous case can be predicted.

2. Outcomes that arose using some solution or applying some solution method are marked.

   When outcomes of previous cases match desired outcomes of a current case, the solution or solution method from the previous case can be predicted.

For any particular case, there may be several conjunctive feature sets associated with it. If memory is aware of the goal(s) the problem solver is attempting to achieve, it can choose from among the cases that are retrieved by preferring those where goals and constraints match and full conjunctive feature sets are specified.

While we do not yet have a complete implementation of the choice process, and we do not yet know the priorities of the preference rules we've proposed, PARADYME has several preference heuristics for choosing a best-matching case. Some of the preference heuristics are implemented as part of the retrieval process presented above (e.g., 1 and 2). The others are used to choose between those items retrieved using that algorithm.

1. Prefer predicted pieces of memory over those that are not predicted.

2. Prefer the most specific of those in the same hierarchy.

3. Prefer items that match a retrieval probe completely.

4. If a probe describes specific details, prefer items that have those details.

5. Prefer items that share a major goal or set of goals and constraints on those goal.

6. Prefer those items whose full set of salient features are specified in the retrieval probe.

7. Prefer those items where the goals and constraints of a fully matched conjunctive feature set match current goals and constraints of the problem being solved.
8. Prefer those items with more full sets of salient features specified in the retrieval probe.
9. Prefer those items that match on dimensions that are known to be difficult to fix.

8 Cue Elaboration

The process we have presented is appropriate when the retrieval probe accurately describes an item or several items in the memory. Some retrieval probes, however, are unsuitable for finding matches, either because of the complexities of representational embeddings in memory's structures or because they are too vague or overly-specific. We have identified four circumstances under which a retrieval probe or part of a retrieval probe is unsuitable for retrieval:

1. The retrieval probe might not directly specify a type of situation. A retrieval probe might describe features of a situation without naming the type of situation. We have no examples of this in JULIA's domain. In CYRUS' domain, questions such as "Has Vance ever talked to Woodward or Bernstein?" and "Has Vance's wife ever met Mrs. Begin?" are examples of this. A probe that does not include an "isa" clause, or whose "isa" clause points to a kind of event that happens in many different contexts falls into this category. We will introduce a *condensation heuristic* to deal with this problem.

2. The retrieval probe may describe a situation that is not stored in memory but that is a "near miss" to something stored in memory. Memory, for example, might have a description of a "meal in a particular small Italian restaurant in which eggplant-filled manicotti was served". A probe of "remember the time we had eggplant-filled stuffed-shells for dinner in a little Italian restaurant" would be a near miss to this event. If enough of the rest of the event is describe to make it unique among the other events in memory, the near miss event can be retrieved anyway (e.g., if this was the only visit to a small Italian restaurant where something with eggplant filling was served), but if not, the probe will not retrieve it (e.g., if in may restaurant visit eggplant was eaten as the filling for something). This situation exists when a retrieval probe provides concrete features but memory retrieves only a generalized node that does not mention the concrete features or when memory retrieves many cases that match the retrieval probe, but none match exactly and none are better matches than the rest. A *cue transformation heuristic* that expands a cue into a set of cues conjunctively describing it will solve this problem.

3. The retrieval probe might describe a character or a prop without naming it or its type. The embedding of memory's frame-like structures makes it hard to directly activate events whose features are vaguely described. Memory recognizes this if an event is requested, nodes describing particular characters or props are highly activated, and no such features are highlighted in the events that have been activated. *Condensation heuristics* will deal with this problem too by recognizing a particular character or prop that has been described and then probing memory using the particular character or prop as a replacement for its description.

4. The retrieval probe might describe a relation that is specified more finely in memory's representations than in the retrieval probe. "A meal with a dish with spinach in it" is one example of such a probe. In memory's representations, ingredients of dishes are divided into "mains", "secondaries", and "seasonings", a useful distinction for the problem solver. This fine distinction may not be made in a probe, however. While it is easy to distinguish spices as "seasonings" and sometimes an ingredient is specified or implied to be the "main" one or a "secondary" one, more often this information is not known at the beginning of problem solving and it is memory that must provide this information to the problem solver. Memory knows which of its descriptors are represented this way and recognizes specific situations in which this happens. A *cue transformation heuristic* that expands a cue into its set of disjunctive descriptors will solve this problem.

In each of these cases, heuristics are used to redescribe the retrieval probe and retrieval is attempted again with the set of newly-defined cues. We describe these heuristics below.

1. *Cue Transformation*

   Cue transformation expands a cue to create a larger set of reasonable cues. These new cues might describe the original one conjunctively, provide a disjunction of descriptions equivalent to the original cue, or provide additional information associated with the original cue but not part of it.

   (a) *Replace cue by a conjunct of descriptors*

   As stated above, this type of cue transformation is used when a retrieval probe specifies something quite concrete but the best that can be found in memory is a generalized node that does not mention the concrete feature (e.g., if a search for a meal with stuffed shells returns "Italian meals"). In that case, the specific feature that was not accounted for in the set of retrieved nodes is replaced by its description. "Stuffed shells" in the example would be replaced by a set of cues stating that the food had pasta, ricotta cheese, and tomato sauce in its ingredients, that the structure of it was (shell-shaped) pasta filled with ricotta mixture, topped with tomato sauce and cheese, etc. As a result of replacing an item by its description, "near-miss" matches can be found. For example, replacing stuffed shells by its description might result in retrieval of a meal with manicotti, a close match to stuffed shells.

   (b) *Replace cue by a disjunct of descriptors*

   This type of cue transformation is used when a particular cue is known to have several ways of being described. For example, ingredients can be found as main ingredients, secondary ingredients, and seasonings. If "dishes with tomatoes" are requested in a probe, there is no way to know a priori whether the tomatos are to be main ingredient, a secondary ingredient, or a seasoning of the dish. "Dishes with tomatoes" will be transformed to a disjunct of cues: "dishes with main ingredients tomatoes", "dishes with secondary ingredients tomatoes", "dishes with tomatoes as seasoning". Expanding cues in this way will allow each of these descriptors of a dish to combine with other cues in the retrieval probe so that the best match that takes all of the descriptors into account can be found.

(c) ***Add a closely associated feature to the set of cues***

This type of cue elaboration is equivalent to CYRUS' component-to-component instantiation strategies, and their usefulness is discussed in Kolodner (1983, 1984). In short, a feature that is not yet part of the retrieval probe but that is closely associated with some cue in the retrieval probe is added. An example of this is adding a place associated with an identified person or organization to the retrieval probe. This can help to distinguish between several events that have been equally activated, where each partially matches the retrieval probe, and there is no clear way to distinguish which is the best.

2. *Cue Condensation*

   Cue condensation heuristics condense a set of cues to a single one that describes a larger unit. This process looks for concepts whose marked features are all, or almost all, mentioned in the retrieval probe. It is useful if a type of event has not been specified but has been described, or if features of an event being specified have not been directly named but have been abstractly described. A set of cues describing a dish with shell-shaped pasta filled with ricotta would be replaced by one cue stating that the dish is stuffed shells using cue condensation. An event described as one where people swim in a contest and later get awards would be replaced by one cue stating that the event is a swim-meet using cue condensation.

Cue elaboration is an automatic process done by memory after retrieval. After elaboration, retrieval is attempted again using the newly-defined set of cues. We are still working on probe elaboration methods. While we know many of the heuristics for elaborating a probe, we have not yet experimented with them enough to know exactly how to control their application, nor do we know yet how to fully control their interaction with retrieval processes.

Cue elaboration is similar in spirit to CYRUS' instantiation strategies. CYRUS (Kolodner, 1983, 1984) had two types of elaboration strategies to take care of these problems, each used at a different point in the retrieval process: component-to-context instantiation rules were used prior to memory traversal to infer a context for search, and component-to-component instantiation rules were used after traversal was attempted to elaborate a retrieval probe that did not retrieve a particular event. PARADYME also has two kinds of cue elaboration heuristics, but they are both used after retrieval is attempted and their functions are not exactly the same. PARADYME's *cue transformation* heuristics perform the function of CYRUS' component-to-component instantiation rules in a more expansive way than was done in CYRUS, and PARADYME's *cue condensation* heuristics perform the function of CYRUS' component-to-context instantiation rules and also help with cue transformation rules define a better set of descriptive cues.

9 Discussion

The parallel algorithm presented runs in linear time on a SIMD parallel machine, and its runtime does not vary significantly with the size of the memory as long as memory does not exceed the size of the machine. [7] It works on a hierarchically organized memory where events are stored across several

[7]Specifically, its run-time is AN+2B+1. A is a number designating the overhead of dealing with embedded representations and is 1 plus the depth of an embedding. For the examples we have run, it ranges between 1 and

hierarchies. The concept refinement search method limits retrieval to only reasonable parts of the memory and allows memory probes to describe events by describing features of their substructures. The basic algorithm forms the core of a case retrieval process, but it is not complete. While it finds many fewer events than an intersection search would, it does not address the choice of a best case(s) from those that are retrieved; nor does it include a capability for automatically elaborating a retrieval probe that is poorly specified. To take care of these problems, we have introduced preference heuristics for choosing the best set of cases from those retrieved and we have introduced probe elaboration heuristics for redefining a poorly-specified or near-miss probe.

As an added advantage, we have been able to do away with CYRUS' redundant indexing structure. This means the memory takes up considerably less space in the machine. Were we to run CYRUS (Kolodner, 1983, 1984) or the memory parts of any of our case-based reasoners (e.g., MEDIATOR (Kolodner, et al., 1985, Simpson, 1985), JULIA (Hinrichs, 1988, Kolodner, 1987a,b, Shinn, 1988)) using the new algorithm and memory structures, we would get significant speedup, would use much less memory space, and would retrieve exactly the same items as under the serial scheme.

Problems remain to consider, however. First, due to the architecture of the Connection Machine, we have not done an exact translation from our old retrieval scheme to the new one. CYRUS' retrieval scheme (the old one) was linear in the depth of memory's hierarchies, a much smaller number than the length of a retrieval probe. It would be interesting, from an algorithmic point of view, to attempt implementation of CYRUS' algorithms on a MIMD machine. It would also be interesting from a psychological point of view to have a parallel algorithm whose speed is independent of the length of the retrieval probe.

Second, the algorithm we have implemented requires full connectivity between nodes in the hierarchies of MOPs and scenes. Because generalizations must be made independently in each abstraction hierarchy, however, that connectivity may need to be recomputed during retrieval. The "instruction" portion of the prediction messages in DMAP provide one way that is not very elegant. Some other way to overcome this problem must be found. And, of course, it will add to the complexity of the algorithm.

Third, we have hardly considered memory update procedures. They, of course, must be integrated into the memory scheme so that we can insure that memory's structure and the accessibility of events is maintained as the memory gets large.

10 Bibliography

1. Arens, Y. (1981). Using language and context in analysis of text. *Proceedings of IJCAI-81.*

2. Barletta, R. (1988). Explanation-Based Indexing of Cases. *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

5. N is the length of the retrieval probe. B is the depth of the hierarchy that needs to be traversed in step 3 of the algorithm. We assume a hierarchy of similar size gets traversed in step 4, thus we must add in B two times. B ranges between 1 and 3 in the examples we have looked at, but the memory we have implemented is small. We expect it to remain a small number and to be significantly smaller than N. The constant is the number of cycles necessary for step 2 of the algorith. We expect N to dominate the expression.

3. Hammond, K. J. (1984). *Indexing and Causality: The organization of plans and strategies in memory.* Report No. 351. Dept. of Computer Science. Yale University. New Haven, CT.

4. Hammond, K. J. (1986). *Case-Based Planning: An integrated theory of planning, learning, and memory.* Ph.D. Thesis. Dept. of Computer Science. Yale University.

5. Hinrichs, T. (1988). Towards an architecture for open world problem solving. *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

6. Hunter, L. (1988). *The Use and Discovery of Paradigm Cases.* Ph.D. Thesis. Yale University. Forthcoming.

7. Kolodner, J. L. (1983). Reconstructive Memory: A Computer Model. *Cognitive Science*, vol. 7.

8. Kolodner, J. L. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model.* Hillsdale, NJ: Lawrence Erlbaum Assoc.

9. Kolodner, J. L. (1985). *Experiential Processes in Natural Problem Solving.* Technical Report No. GIT-ICS/85/23. School of Information and Computer Science. Georgia Inst. of Technology. Atlanta, GA.

10. Kolodner, J. L. & Cullingford, R. E. (1986). Towards a Memory Architecture that Supports Reminding. *Proceedings of the 1986 Conference of the Cognitive Science Society.*

11. Kolodner, J. L. (1987a). Extending problem solver capabilities through case-based inference. *Proceedings of the 1987 International Machine Learning Workshop.*

12. Kolodner, J. L. (1987b). Capitalizing on failure through case-based inference. *Proceedings of the 1987 Conference of the Cognitive Science Society.*

13. Kolodner, J. L., Simpson, R. L., & Sycara, E. (1985). A Process Model of Case-Based Reasoning in Problem Solving. *Proceedings of IJCAI-85.*

14. Koton, P. (1988). Reasoning about evidence in causal explanations. *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

15. Lebowitz, M. (1983). Generalization from natural language text. *Cognitive Science*, vol. 7.

16. Lytinen, S. (1984). Frame selection in parsing. *Proceedings of AAAI-84.*

17. Martin, C. & Riesbeck, C. (1986). Uniform parsing and inference for learning. *Proceedings of AAAI-86.*

18. Owens, C. (1988). Domain-Independent Prototype Cases for Planning. *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

19. Reiser, B. & Black, J. (1983). The roles of interference and inference in the retrieval of autobiographical memories. *Proceedings of the 1983 Conference of the Cognitive Science Society.*

20. Riesbeck, C. & Martin, C. (1986). Toward Completely Integrated Parsing and Inference. *Proceedings of the 1986 Conference of the Cognitive Science Society.*

21. Riesbeck, C. (1988). An Interface for Case-Based Knowledge Acquisition. *Proceedings of the DARPA Workshop on Case-Based Reasoning*

22. Rissland, E. & Ashley, K. (1987). *HYPO: A Case-Based Reasoning System.* CPTM #18. Department of Computer and Information Science. University of Massachusetts. Amherst, MA.

23. Schank, R. C. (1982) *Dynamic Memory.* Cambridge: Cambridge University Press.

24. Shinn, H. (1988). Abstractional Analogy: A Model of Analogical Reasoning. *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

25. Simpson, R. L. (1985). *A Computer Model of Case-Based Reasoning in Problem Solving.* Ph.D. Thesis. Technical Report No. GIT-ICS/85/18. School of Information and Computer Science. Georgia Inst. of Technology. Atlanta, GA.

26. Stanfill, C. (1987). Memory-Based Reasoning Applied to English Pronunciation. *Proceedings of AAAI-87.*

27. Sycara, E. (1987). *Resolving Adversarial Conflicts: An approach integrating case-based and analytic methods.* Ph.D. Thesis. Technical Report No. GIT-ICS/87/26. School of Information and Computer Science. Georgia Inst. of Technology. Atlanta, GA.

Abstractional Analogy: A Model of Analogical Reasoning[1]

HONG S. SHINN (shinn@gatech.edu)
School of Information and Computer Science, Georgia Institute of Technology
Atlanta, GA 30332, U.S.A. (404) 894-5550

Length of Paper (in words): **4,000**
Main Topic: **Cognitive Modeling**
Subtopic: **Analogical Reasoning**
Secondary Topic: **Machine Learning**
Subtopic: **Analogical Reasoning**

Abstract

Our model of analogical reasoning is based on the view that it is necessary to grasp the abstraction common to two analogous problems in order to know exactly what can be transferred from one problem to another. The model consists of two major steps. First, create an abstract schema that represents what source and target cases have in common. The abstract schema consists of a problem schema and its solution schema. The problem schema is created by analogically mapping the source problem to the target problem, then its solution schema is created using the source case. Second, apply the solution schema to the target problem. We call such a model of analogical reasoning *abstractional analogy.*

Abstractional analogy provides a way of extracting all knowledge from a source that can be transferred to a target. Transfer can be of reasoning methods and/or of generalized results. Both types of knowledge are learned in the form of solution schemas as a natural byproduct of abstractional analogy. Abstract schemas together with cases can be organized into abstraction hierarchies. Thus, abstractional analogy is a unifying model of three different aspects of cognition: problem solving by analogy, learning of both declarative (generalized results) and procedural (reasoning methods) knowledge, and memory organization.

[1]This research has been supported in part by the Army Research Institute under Contract No. MDA-903-86-C-173, is currently supported in part by NSF under Grant No. IST-8608362, and in part by Lockheed AI Center under Grant No. DTD 09-25-87.

1 Introduction

Experiential reasoning plays a major role in human problem solving and learning. Jardine [Jar74] quotes Francis Bacon:

> *New knowledge is discovered by ingenious adaptation of existing knowledge, rather than by formal inference from fundamental principles.*

Analogical reasoning is one way of adapting existing knowledge to solve a new problem.

Although a number of models of analogical reasoning have been attempted, two contradicting views currently coexist on the process of analogical transfer [Dar83,Ros86]:

1. **Direct transfer of knowledge from source to target**
   Analogy is identified by establishing correspondences between source and target and then interpreting knowledge about the source in the target domain.

2. **Indirect transfer via common abstraction**
   Analogy is identified as a common abstraction and then knowledge transfer is done via the abstraction.

Traditionally, researchers have viewed analogy as direct transfer, and most AI programs that do analogical reasoning employ that method (e.g., [Win80] [Car86]). However, the view of indirect transfer recently has received more attention (e.g., [Pol54,Gen80,GH83,CM85,Der85,And86]). Genesereth [Gen80] states that "the problem of understanding an analogy becomes one of recognizing the shared abstraction."

This is a problem which leads to completely different models of problem solving and learning. That is, does learning by generalization occur during problem solving (i.e., as part of making the analogy) or does it occur afterwards? The direct transfer view implies that generalization occurs after problem solving, while the indirect view suggests that it occurs during problem solving. Ross [Ros86] points out that, while some researchers have seen that generalization is forced by analogical mapping, no one has clearly stated their temporal relationship. We suspect this confusion is caused by failure to understand in detail the process involved in making an analogy. **This paper presents a model called *abstractional analogy* based on the indirect transfer approach and also provides some computational accounts of** this method.

Another important issue on analogical reasoning is that of what knowledge is transferred and how. Polya [Pol45] identifies two types of knowledge transferred during analogical problem solving. These are the method used and the result. Transfer of a previous result — possibly with some minor modification — shortcuts the reasoning involved for a similar problem by reducing its search for a solution. When result transfer is not appropriate, reasoning can be transferred. These two transfer methods are applied in different stages of problem solving. Abstractional analogy integrates these two types of analogy transfer.

The process of abstractional analogy is implemented as the case-based reasoning (CBR) part of the JULIA system [CK86,Kol87b,Kol87a], designed to be a caterer's assistant. JULIA's task is to interactively plan a meal with a client user who provides constraints for the meal. Some constraints are given early on. The need for specification of others is determined during problem solving. JULIA's problem solver includes constraint propagation and satisfaction, a goal-based reduction planner, and CBR modules. The reasoning described in this paper is JULIA's CBR method. JULIA uses CBR [2] whenever a previous similar case is made available to it by its memory. Examples from JULIA will be used throughout this paper.

2 The Process of Abstractional Analogy

Our model of analogical reasoning, abstractional analogy, consists of two major steps, analogy abstraction and then abstraction application. Analogy abstraction is achieved in two substeps. First, an abstract problem schema is created by analogically mapping the source problem to the target problem. Then, a solution schema is created for the problem schema using the source case. The two abstract schemas formed this way uniquely

[2]Although the JULIA system uses the general term "case-based reasoning" to indicate a method of reasoning with previous cases, in this paper the term "analogical reasoning" will be used instead to emphasize the role of analogy in knowledge transfer.

represent the analogy existing between the analogues. In the next step, the solution schema is applied to the target problem. An additional step refines the solution obtained by analogy to fit constraints of the new problem that were not covered.

In later problem solving with a new problem, whenever an existing schema fits the new problem, the schema is applied to the problem: if the generalized result of the schema is available, it is transferred; otherwise the reasoning method is applied. This process will be discussed here in detail.

2.1 Representing Problems and Cases

JULIA uses frame representations [Min75,Wil86] for describing problem and solution structures. In JULIA, a natural language processing (NLP) system [TC88] interprets a problem description into a frame. Suppose a target problem is

> Find a Thanksgiving main dish for 16 vegetarians. The dining room can accommodate only 10 people. What and how should it be served?

The NLP would first identify goals of a problem, and then take everything else, which constrains the goals, and make it into a constraint. Figure 1 shows the result.

Problem:
goals: [g-know(MAIN-DISH) g-know(MEAL-PRESENTATION)]
constraints: [c-season(THANKSGIVING) c-diet(VEGETARIAN)
c-number-of-guests(16) c-dining-space(10)]

(Note: Information such as host, guests, location, and time are omitted.)

Figure 1: Problem C50

Each case in JULIA has problem and solution parts. The problem part describes its problem functionally in terms of goals and constraints while the solution part contains a solution plan and the reasoning history. The representation of a case supports hierarchical structure: a case may be decomposed into subcases which may be again decomposed and so on; thus, cases at all levels have the same structure so that they can be viewed as independent cases. This recursive representation facilitates knowledge transfer at any level.

2.2 Analogical Mapping

Our analogical mapping algorithm accommodates Gentner's systematicity principle ([Gen83]) in that it transfers "a system of connected knowledge, not a mere assortment of independent facts". In other words, during mapping between structures, even the highest order predicates may not be mapped separately from their lower level entities. However, in dealing with similarity, we do not accept Gentner's entire theory of structure mapping. In our mapping scheme, two relations which are functionally similar (i.e., their current partonomic roles in both structures are the same) will not be thrown out.

For example, according to Gentner, two relations *equals[add(a,b),add(b,a)]* and *equals[multiply(a,b),multiply(b,a)]*[3] are not mappable to each other because the highest level predicates are identical, but not the lower level predicates (i.e., add and multiply). On the other hand, in our scheme, these are mappable because their high order predicates are the same while the low level predicates "add" and "multiply" are functionally similar due to their same functional roles in the whole structures. Burstein [Bur86] demonstrates with his system CARL the necessity of mapping between non-identical relations, criticizing Gentner's structure mapping [Gen83] which fails on this kind of similarity.

Another characteristic of our mapping scheme is hierarchical mapping. This is frequently used when problems are represented in hierarchical structure. In fact, analogy between problems usually exists at an abstract level. Thus, mapping starts at the highest level first and proceeds to the next lower level and so on until analogy breaks down. Holyoak [Hol85] also identified hierarchical mapping as a practical necessity.

Thus, in our scheme, the entire mapping process is a recursive application of two-step hierarchical mapping: first identify the next lower level

[3]Polya's analogy example [Pol54]

structures, and then map them systematically under functional similarity.

Let's apply our general mapping scheme to a problem from JULIA's domain. As an example, given the target problem C50 (Figure 1), JULIA[4] is reminded of case C38 (Figure 2): "a vegetarian Thanksgiving main dish for 4 people, all seated and served." Case C38, which had two goals, g-know(menu) and g-know(presentation), was decomposed into two subcases C381 and C382, one for each goal. Since cases are represented in hierarchical structure, JULIA begins mapping with the top level problem structures. Then, it identifies functional similarity. In a frame-based representation, a problem is already analyzed into a problem structure [Wil86]. Thus, it is straightforward to identify the same functional components (e.g., goals and constraints). Next, it starts mapping with goals: if goals fully match, the mapping proceeds to constraints; in case of a partial match, which means some goals match but others do not, only the matched goals will be considered for possible transfer; otherwise, the mapping fails. Mapping then proceeds to constraints on only the matched goals to establish correspondences between them. For example, JULIA identifies correspondence between c-number-of-guests(16) and c-number-of-guests(4) because their functional roles in the problems are the same.

2.3 Problem Abstraction

Problem abstraction is the process of building a problem schema as a common abstraction of two analogous problems. A similarity is a commonality at a higher level of abstraction and an identity is a commonality at the same level. Thus, the commonalities abstracted from similarities together with identities form a problem schema.

Similarities can be identified by using an abstraction hierarchy. One hierarchy, which frame-based representations (as in JULIA) support, is an ISA hierarchy. In this hierarchy, given a pair of objects, a common abstraction is found by simply identifying their immediate common ancestor (e.g., "fruit" for "apple" and "orange"). In JULIA, if the existing hierarchy does not contain a common ancestor for the pair of objects, then an abstract object is created by introducing a new symbol. The new object

[4]'JULIA' usually refers to the entire problem solving system but often is used to refer to only the analogical reasoner, as in this case.

C38:

Problem:

goals: [g-know(MAIN-DISH) g-know(MEAL-PRESENTATION)]

constraints: [c-season(THANKSGIVING) c-diet(VEGETARIAN)
c-number-of-guests(4) c-dining-space(>4)]

Solution: [C381 C382]

Reasoning:

1. OP: plan for each subcase

C381:

Problem:

goal: [g-know(MAIN-DISH)]

constraints: [c-season(THANKSGIVING) c-diet(VEGETARIAN) c-number-of-guests(4)]

Solution: STUFFED-SQUASH

ingredients:

2 squashes with filling:
1/2 cup chopped onion, 1 clove garlic, 1 stalk celery,
1/4 cup walnuts, 1/4 cup sunflower seeds, 1/4 cup raisins,
1/2 tsp. sage, 1/2 tsp. thyme, 1/2 lemon juice,
3 tbs. butter, 1 cup wheat bread, 1/2 cup cheddar cheese

Recipe Source: *Moosewood Cookbook* (by Mollie Katzen, 1977, Ten Speed Press)

C382:

Problem:

goals: [g-know(MEAL-PRESENTATION)]

constraints: [c-number-of-guests(4) c-dining-space(>4)]

Solution: SERVICE

Reasoning:

1. OP: "Since the number of guests was less than the dining space, the eating configuration was SEATED."
   Input: [c-number-of-guests(4) c-dining-space(>4)]
   Output:[c-eating-configuration(SEATED)]
2. OP: "Since the eating configuration was SEATED, the meal presentation was SERVICE."
   Input: [c-eating-configuration(SEATED)]
   Output:[c-meal-presentation(SERVICE)]

Figure 2: Case C38

will be given as its property: the set of common properties (i.e., the union of the set of identical properties and the set of abstractions of pairs of similar properties). Note that the existence of an entity does not require its lexicalization in language. This method of commonality abstraction is also applicable to mathematical objects such as variables [Der85,Der86]. For example, the pair [c-number-of-guests(16) c-number-of-guests(4)] creates a common constraint c-number-of-guests(?X) and ?X will be given as its property "number".

```
A75:
   Problem:
      goals: [g-know(MAIN-DISH) g-know(MEAL-PRESENTATION)]
      constraints: [c-season(THANKSGIVING) c-diet(VEGETARIAN)
            c-number-of-guests(?X) c-dining-space(?Y)]
   Solution: [A751 A752]

A751:
   Problem:
      goal: [g-know(MAIN-DISH)]
      constraints: [c-season(THANKSGIVING) c-diet(VEGETARIAN) c-number-of-guests(?X)]

A752:
   Problem:
      goals: [g-know(MEAL-PRESENTATION)]
      constraints: [c-number-of-guests(?X) c-dining-space(?Y)]
```

Figure 3: Problem schema A75

Now, consider creation of a problem schema for problems C38 (Figure 2) and C50 (Figure 1). JULIA first creates an abstract problem A75 (Figure 3) at the top-level with the commonalities found between C38 and C50. Next, JULIA checks the next lower level of the source schema to see if it was divided into subproblems; if so, it creates subproblems in the same manner recursively. Here, it creates two subproblems A751 and A752 at the lower level, one for each common goal.

2.4 Solution Abstraction

A solution schema is an abstraction of the source solution that is at the same level of abstraction as the problem schema. Figure 4 outlines our solution abstraction algorithm.

Consider, first, transfer of reasoning. JULIA assumes a reasoning history is well maintained in the form of an operator with its preconditions and justifications, input, and output for each step. For each step, JULIA checks to see if the current state of the schema meets the preconditions of the operator of this step. If the preconditions are not met, the schema needs to be transformed. In general, however, there is no domain-independent method for transformation. The transformation problem can be viewed as another separate problem to which analogical reasoning can be applied. If the preconditions are met, the operator is generalized to fit the schema. The generalized operator also needs to be justified using the previous justifications.

However, since the schema includes variables, there may exist more than one reasoning path depending on the value of input data at that step. This could happen, for example, when the operator is "compare two values ?X and ?Y".

If application of the generalized operator to the input of this step always leads to the same reasoning path as that of source case, the applied result will be kept in the schema for the output of that step. If it has more than one alternative reasoning path on this input, JULIA needs to generalize the operator as follows: for the same alternative as that of the source case, JULIA generalizes the operation as in the above case; for the other alternatives, JULIA generalizes the operation in one of the following ways: if an existing schema was retrieved and it has a reasoning path for this alternative, then use it; if the previous justifications similarly fit these other alternatives, then generalize the operator along the similar line; otherwise use domain theory.

Next, consider transfer of result[5]. The source result is generalized to fit the problem schema by considering the generalized requirements in the problem schema and the requirements in the source problem but not in the

[5]There are some applications where, even when a reasoning history is available, transfer of result is desirable [Kol87b].

Input: Problem schema, source case, and analogy map
Output: Solution schema
Method:
(The analogy map provides the correspondence information between the problem schema and the source case.)

```
if a reasoning history is not available for the source case
then do transfer of result:
   generalize the result to fit the problem schema;
   store the generalized result in the schema
else do transfer of reasoning:
   for each step of the reasoning history of source case do:
   if the current state of the schema meets the preconditions of the operator
   then
      if there is only one alternative on the input of the schema
      then
         generalize the operator up to the abstract level of the schema;
         store the operator in the schema for this step;
         apply the operator to the current state of the schema
         and keep the result in the schema for this step;
      else if there exist more than one alternative
      then
         generalize the operator for each alternative similarly;
         store the generalized operator in the schema for this step;
   else
      transform the schema to make it meet the preconditions of this operator
         if transformation is successful
         then apply the above method
         else abandon this case for another
```

Figure 4: Solution Abstraction Algorithm

problem schema. Space does not permit more discussion of this method [Car83,Tur87]; it will be briefly discussed with a simple example below.

Let's now apply the abstraction algorithm to case C38 (Figure 2) for the problem schema (Figure 3). This is split into two subproblems. Consider, first, subschema A751 with subcase C381. Since subcase C381 does not have a reasoning history, JULIA uses the transfer of result. Using the analogy map, JULIA knows the source result needs to be generalized to fit constraint c-number-of-guests(?X). In this case, generalization is done by using the domain knowledge: "The quantity of food is proportional to the number of guests (say, DK22)." JULIA applies this knowledge to the source result and multiples the quantity of each ingredient of the dish by ?X/4. The generalized result[6] is shown in Figure 5.

Next, apply the abstraction algorithm to subschema A752 and subcase C382. Since the reasoning history is available for subcase C382, JULIA uses it to generalize the subschema. For each step of the reasoning history, JULIA checks if it is applicable to the current state of the schema. JULIA finds that the first step requires the constraints [c-number-of-guests(?X) c-dining-space(?Y)] as input. Next, JULIA checks if the reasoning step is general enough to fit into the problem schema. In the source case, since the number of guests was less than or equal to the dining capacity, the eating-configuration was SEATED. But, in the schema, since the number of guests may or may not be greater than the dining capacity, this step of operation in the source case should be generalized, considering both alternatives for the schema. For the case in which the number of guests is less than or equal to the dining capacity, the schema will use the same reasoning as that of the source case (i.e., if ?NO-GUESTS $\leq$?DINING-CAPACITY, **then** ?EATING-CONFIGURATION is SEATED). But, for the other alternative (i.e., ?NO-GUESTS > ?DINING-CAPACITY), the operation needs to be generalized using one of the above mentioned generalization techniques. One method is to use the domain knowledge: "The eating-configuration is either SEATED or STANDING." After a simple

[6]The generalized result in this case only mediates transfer between source and target cases and may not be interpreted by any means as a solution formula for every case that has the same set of requirements as this case, because it is only one of many possible solutions. There may, however, be times when the generalized result will indeed be a solution formula.

A75:
  Problem:
    goals: [g-know(MAIN-DISH) g-know(MEAL-PRESENTATION)]
    constraints: [c-season(THANKSGIVING) c-diet(VEGETARIAN)
      c-number-of-guests(?X) c-dining-space(?Y)]
  Solution: [A751 A752]
  Reasoning:
    1. OP: plan for each subcase

A751:
  Problem:
    goal: [g-know(MAIN-DISH)]
    constraints: [c-season(THANKSGIVING) c-diet(VEGETARIAN) c-number-of-guests(?X)]
  Solution: STUFFED-SQUASH
    ingredients:
      1/2 ?X squashes with the filling:
      1/8 ?X cup chopped onion, 1/4 ?X clove garlic, 1/4 ?X stalk celery,
      1/16 ?X cup walnuts, 1/16 ?X cup sunflower seeds, 1/16 ?X cup raisins,
      1/8 ?X tsp. sage, 1/8 ?X tsp. thyme, juice from 1/8 ?X lemon,
      3/4 ?X tbs. butter, 1/4 ?X cup wheat bread, 1/8 ?X cup cheddar cheese
  Reasoning: (justifications: case C381 and domain knowledge DK22)

A752:
  Problem:
    goals: [g-know(MEAL-PRESENTATION)]
    constraints: [c-number-of-guests(?X) c-dining-space(?Y)]
  Solution:
  Reasoning:
    1. OP: **If** ?NO-GUESTS $\leq$?DINING-CAPACITY
      then ?EATING-CONFIGURATION is SEATED
      else ?EATING-CONFIGURATION is STANDING
    2. OP: **If** ?EATING-CONFIGURATION is SEATED
      then ?MEAL-PRESENTATION is SERVICE
      else ?MEAL-PRESENTATION is BUFFET

Figure 5: Abstract schema A75

computation, we get "If ?NO-GUESTS > ?DINING-CAPACITY, then ?EATING-CONFIGURATION is STANDING." Similarly, the next step will also be generalized using the domain knowledge: "The meal-presentation is either SERVICE or BUFFET." Figure 5 shows the schema so obtained.

2.5 Abstraction Application

After an abstract schema is created, it is applied to the target problem. The basic idea for schema application is to apply the generalized result if one exists, otherwise apply the reasoning method.

Application of schema A75 (Figure 5) to problem C50 (Figure 1) generates two subproblems. The subproblem for goal g-know(MAIN-DISH) is solved by applying the generalized result of subschema A751 which requires instantiating the variable ?X to 16. On the other hand, the subproblem for goal g-know(MEAL-PRESENTATION) is solved by applying the reasoning method, step by step. The variables ?X and ?Y are bound to 16 and 10, respectively, and each step is applied; the eating configuration is STANDING after the first step; the meal presentation is BUFFET after the second step. The resultant target case is shown in Figure 6.

2.6 Solution Refinement

Application of a schema to the target problem may not lead to a final solution because the instantiated result does not always meet all the requirements. If it is the case, the result of schema application must be refined. In general, the problem of refinement can be viewed as an independent problem where its goal is transforming the current result to make it satisfy the remaining requirements. This means that either an analogical problem solver or other problem solvers can be applied here. In JULIA, refinement with extra constraints is done by using the same reasoning method used for the other constraints.

3 Analogy-Based Learning

Analogical problem solving per se is one form of learning because it learns from previous experience how to solve similar problems. Another form of

C50:

Problem:

goals: [g-know(MAIN-DISH) g-know(MEAL-PRESENTATION)]

constraints: [c-season(THANKSGIVING) c-diet(VEGETARIAN)
c-number-of-guests(16) c-dining-space(10)]

Solution: [C501 C502]

Reasoning: (justification: Schema A75)

1. OP: plan for each subcase

C501:

Problem:

goal: [g-know(MAIN-DISH)]

constraints: [c-season(THANKSGIVING) c-diet(VEGETARIAN) c-number-of-guests(16)]

Solution: STUFFED-SQUASH

ingredients:

8 squashes with the filling:
2 cup chopped onion, 4 clove garlic, 4 stalk celery,
1 cup walnuts, 1 cup sunflower seeds, 1 cup raisins,
2 tsp. sage, 2 tsp. thyme, juice from 2 lemon,
12 tbs. butter, 4 cup wheat bread, 2 cup cheddar cheese

Reasoning: (justification: Schema A751)

C502:

Problem:

goals: [g-know(MEAL-PRESENTATION)]

constraints: [c-number-of-guests(16) c-dining-space(10)]

Solution: BUFFET

Reasoning: (justification: Schema A752)

Figure 6: Case C50

learning occurs as a byproduct of problem solving in the form of schemas. A schema contains two types of general knowledge: procedural (reasoning method) and declarative (generalized result). This section will discuss the latter form of general learning and the related issues: where and how the acquired knowledge is stored and how it is used later.

3.1 Learning During Solution Abstraction

As shown in the previous section, solution abstraction can be viewed as the process of extracting an embedded algorithm out of the reasoning part of a previous case and/or a generalized result out of the solution part. As a result, a solution schema contains a reasoning method and/or a generalized result for a given problem schema. From the learning point of view, a solution schema represents exactly what is learned from a particular analogy.

We should note, however, that the generalized result of a solution schema may not always be interpreted as a solution formula. If the result is proven to be unique it can be used as a solution formula for any case that is an instance of the problem schema. On the other hand, if the result is just one among many possible solutions (e.g., A751), it should be interpreted solely as a mediator of transferring the source solution to the target problem. If this gets used frequently, it may become a prototypical solution, but not a solution formula. Thus, the generalized result is a potential source of either a solution formula or a prototypical solution.

Analogy-based generalization vs Explanation-based generalization

Schema abstraction (i.e., problem abstraction plus solution abstraction) is a kind of analogy-based generalization (ABG). ABG and explanation-based generalization (EBG) (see [DM86,MKK86]) are as different as they are similar.

If a reasoning history is not available for a solved case and needs to be built, it can be done either by using EBG (because a reasoning history for a case in ABG corresponds to an explanation for an example in EBG) or by applying ABG recursively to another analogous case with a reasoning history (see [KL87] for case-based explanation). The problem of constructing an explanation using the domain theory is similar to a state space search problem if we view a problem as an initial state and its solution as a final state. EBG, in this case, has no strategy of controlling the search space. ABG, on the other hand, can significantly constrain the search space using the reasoning history of another similar case.

In addition, since EBG generalizes on a single case, the generalized explanation accommodates only the one possible alternative that the particular case followed. As a result, EBG fails to consider other potential alternatives so that EBG by itself is not capable of learning general reasoning methods. On the other hand, two cases give ABG a chance to explore more than one alternative in problem solving (as we have seen in Section 2.4). This leads ABG to incrementally learn a general method.

3.2 Organizing Memory with Cases

When a target problem is solved, both the target and source cases are stored in memory as specializations of the abstract schema created by abstractional analogy. The schema itself is stored, replacing the previous source case, and both cases will be made children of this schema. In this way, abstractional analogy forms memory into abstraction hierarchies, where each node represents a specific or generalized case.

A retrieval algorithm similar to that described in Kolodner [Kol84] can be used to find the most specific partially matching schema or case, when a new problem is being solved. If a schema has already been created and is recalled from memory, the analogical reasoner uses it to solve the problem directly. If a case more similar than available schemas is recalled, abstractional analogy is applied to it to solve the problem.

4 Summary

Our goal has been to develop a computational model of analogical reasoning based on abstractional analogy. The process of abstractional analogy proceeds as follows. First, given a new problem, an analogous case is retrieved from memory. Second, analogy abstraction creates an abstract schema that represents what source and target cases have in common. An abstract schema consists of a problem schema and its solution schema. In order to

3. Hammond, K. J. (1984). *Indexing and Causality: The organization of plans and strategies in memory.* Report No. 351. Dept. of Computer Science. Yale University. New Haven, CT.

4. Hammond, K. J. (1986). *Case-Based Planning: An integrated theory of planning, learning, and memory.* Ph.D. Thesis. Dept. of Computer Science. Yale University.

5. Hinrichs, T. (1988). Towards an architecture for open world problem solving. *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

6. Hunter, L. (1988). *The Use and Discovery of Paradigm Cases.* Ph.D. Thesis. Yale University. Forthcoming.

7. Kolodner, J. L. (1983). Reconstructive Memory: A Computer Model. *Cognitive Science*, vol. 7.

8. Kolodner, J. L. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model.* Hillsdale, NJ: Lawrence Erlbaum Assoc.

9. Kolodner, J. L. (1985). *Experiential Processes in Natural Problem Solving.* Technical Report No. GIT-ICS/85/23. School of Information and Computer Science. Georgia Inst. of Technology. Atlanta, GA.

10. Kolodner, J. L. & Cullingford, R. E. (1986). Towards a Memory Architecture that Supports Reminding. *Proceedings of the 1986 Conference of the Cognitive Science Society.*

11. Kolodner, J. L. (1987a). Extending problem solver capabilities through case-based inference. *Proceedings of the 1987 International Machine Learning Workshop.*

12. Kolodner, J. L. (1987b). Capitalizing on failure through case-based inference. *Proceedings of the 1987 Conference of the Cognitive Science Society.*

13. Kolodner, J. L., Simpson, R. L., & Sycara, E. (1985). A Process Model of Case-Based Reasoning in Problem Solving. *Proceedings of IJCAI-85.*

14. Koton, P. (1988). Reasoning about evidence in causal explanations. *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

15. Lebowitz, M. (1983). Generalization from natural language text. *Cognitive Science*, vol. 7.

16. Lytinen, S. (1984). Frame selection in parsing. *Proceedings of AAAI-84.*

17. Martin, C. & Riesbeck, C. (1986). Uniform parsing and inference for learning. *Proceedings of AAAI-86.*

18. Owens, C. (1988). Domain-Independent Prototype Cases for Planning. *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

19. Reiser, B. & Black, J. (1983). The roles of interference and inference in the retrieval of autobiographical memories. *Proceedings of the 1983 Conference of the Cognitive Science Society.*

and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Kaufmann, Los Altos, CA, 1986.

[Car83] J.G. Carbonell. Learning by analogy: formulating and generalizing plans from past experience. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Tioga, Palo Alto, CA, 1983.

[Car86] J.G. Carbonell. Derivational analogy: a theory of reconstructive problem solving and expertise acquisition. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Kaufmann, Los Altos, CA, 1986.

[CK86] R.E. Cullingford and J.L. Kolodner. Interactive advice giving. In *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics*, 1986.

[CM85] J.G. Carbonell and S. Minton. Metaphor and commonsense reasoning. In J.R. Hobbs and R.C. Moore, editors, *Formal Theories of the Commonsense World*, Ablex, Norwood, NJ, 1985.

[Dar83] L. Darden. Reasoning by analogy in scientific theory construction. In *Proc. of Int'l Workshop on Machine Learning*, Monticello, IL, 1983.

[Der85] N. Dershowitz. Program abstraction and instantiation. *ACM Transactions on Programming Languages and Systems*, 7(3), July 1985.

[Der86] N. Dershowitz. Programming by analogy. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Kaufmann, Los Altos, CA, 1986.

[DM86] G. DeJong and R. Mooney. Explanation-based learning: an alternative view. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 145–176, Kaufmann, Los Altos, CA, 1986.

[Gen80] M.R. Genesereth. Metaphors and models. In *Proc. AAAI-80*, Menlo Park, CA, 1980.

[Gen83] D. Gentner. Structure-mapping: a theoretical framework for analogy. *Cognitive Science*, 7:155–170, 1983.

[GH83] M.L. Gick and K.J. Holyoak. Schema induction and analogical transfer. *Cognitive Psychology*, 15:1–38, 1983.

[Hol85] K.J. Holyoak. The pragmatics of analogical transfer. *Psychology of Learning and Motivation*, 19:59–87, 1985.

[Jar74] L. Jardine. *FRANCIS BACON: Discovery and the Art of Discourse.* Cambridge University Press, Bentley House, 200 Euston Road, London, 1974.

[KL87] A.M. Kass and D.B. Leake. A case-based approach to building explanations for explanation-based learning. 1987. Working paper.

[Kol84] J.L. Kolodner. *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1984.

[Kol87a] J.L. Kolodner. Capitalizing on failure through cased-based inference. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, Washington, July 1987.

[Kol87b] J.L. Kolodner. Extending problem solver capabilities through case-based inference. In *Proc. of the Fourth Int'l Workshop on Machine Learning*, pages 167–178, Irvine, CA, June 1987.

[Min75] M. Minsky. A framework for representing knowledge. In P.H. Winston, editor, *The Psychology of Computer Vision*, McGraw Hill, New York, 1975.

[MKK86] T.M. Mitchell, R. Keller, and S.T. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning*, 1:47–80, 1986.

[Pol45] G. Polya. *How to Solve It: A New Aspect of Mathematical Method.* Princeton University Press, Princeton, NJ, 1945.

[Pol54] G. Polya. *Mathematics and Plausible Reasoning: Induction and Analogy in Mathematics.* Volume 1, Princeton University Press, Princeton, NJ, 1954.

[Ros86] B.H. Ross. Remindings in learning and instruction. In *Workshop in Similarity and Analogy*, 1986.

[TC88] E. Turner and R.E. Cullingford. Conversation planning using conversational mops. 1988. In Preparation.

[Tur87] R.M. Turner. Modifying previously-used plans to fit new situations. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, Washington, July 1987.

[Wil86] R. Wilensky. *Some Problems and Proposals for Knowledge Representation.* Technical Report UCB/CSD 86/294, University of California, Berkeley, Berkeley, CA, 1986.

[Win80] P.H. Winston. Learning and reasoning by analogy. *Comm. ACM*, 23(12):689–703, 1980.

The Role of Mapping in Analogical Transfer[1]

Hong S. Shinn
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332, U.S.A.

Abstract

This paper aims to provide a view of the role of analogical mapping in the entire process of analogical problem solving. In many models, analogical mapping is responsible for identifying the analogy between two problems by considering structural and semantic similarities. However, given a non-trivial analogy problem, success of mapping does not always guarantee successful transfer of analogy. In fact, there exist many analogy problems, which succeed on analogical mapping but which fail on analogical transfer. While a potential mapping between problems can be generated, that mapping might not be justifiable until transfer from one problem to another is attempted.

We present our analogical mapping method and show how it works for inter-domain and intra-domain analogies. We demonstrate several analogy problems in which a mapping can be generated that cannot be transferred. We also compare our method to Gentner's SME and to Holyoak's ACME, and show that it performs at least as well, and sometimes better than either of those methods.

Key Words: analogical problem solving, analogical mapping.

1 Introduction

This paper aims to provide a view of the role of analogical mapping in the entire process of analogical problem solving. Analogical problem solving contains at least the following components [Shi88,CM85,HT88]: retrieval of a plausibly analogous case, analogical mapping, and analogical transfer. The step of analogical transfer may involve modification of a previous solution and justification of the result obtained before the result is transferred [Shi88].

In many models, analogical mapping is responsible for identifying the analogy between two problems by considering structural and semantic similarities. However, given a non-trivial analogy problem, analogical mapping by itself does not always guarantee that an analogy will be successful. While it can produce a potential mapping between problems, a mapping might not be justifiable until transfer from one problem to another is attempted.

In this paper, we illustrate the role of mapping in analogical problem solving and we present a hierarchical method of analogical mapping that is based primarily on similarity of structures. The method uses relatively little semantic information. Instead, it relies on the analogical transfer step to determine the merit of a potential analogical mapping.

We show how our method works for inter-domain and intra-domain analogies. We demonstrate several analogy problems in which a mapping can be generated that cannot be transferred. We also compare our method to Gentner's SME and to Holyoak's ACME, and show that it performs at least as well, and sometimes better than either of those methods.

[1]This research has been supported in part by the Army Research Institute under Contract No. MDA-903-86-C-173, is currently supported in part by NSF under Grant No. IST-8608362, and in part by Lockheed AI Center under Grant No. DTD 09-25-87.

2 Analogical Mapping Algorithm

Before introducing our algorithm, we need to clarify the problem of analogical mapping. Polya [Pol54] views "analogy" as a systematic correspondence between two systems preserving certain relations. For his basic type of analogy, Polya defines analogy as "similarity of relations", where relations are similar if they are governed by the same laws. He illustrates this with an example: the multiplication of numbers *multiply(x,y)* is analogous to the addition of numbers *add(x,y)* in the sense that both multiplication and addition are commutative. In other words, two relations *multiply(a,b)* and *add(a,b)* are similar because they are governed by the same commutative law: *equals[OP(a,b),OP(b,a)]*. Interpreting Polya's definition of similarity in analogical problem solving, "similarity" in problems is what leads to similar effects on their solutions. The problem here is that, without knowing beforehand what the similarity's effect will be on the solution to the target problem, we must find that similarity which can be used in deriving the solution. Thus, what analogical mapping does is to find the most probable similarity candidates before transfer of knowledge from source to target is attempted.

Our analogical mapping algorithm follows Gentner's systematicity principle ([Gen83], p. 163) in that it transfers "a system of connected knowledge, not a mere assortment of independent facts". In other words, during mapping between structures, even the highest order predicates may not be mapped separately from their lower level entities.

In dealing with similarity, however, we do not accept Gentner's entire theory of structure mapping. In our mapping scheme, two relations which are structurally similar (i.e., the current partonomic roles in both structures are the same) will not be thrown out. For example, according to Gentner, two relations *equals[multiply(a,b),multiply(b,a)]* and *equals[add(a,b),add(b,a)]* are not mappable to each other because the highest level predicates are identical (i.e., *equals*), but not the lower level predicates (i.e., *add* and *multiply*). On the other hand, in our scheme, these are mappable because their high order predicates are the same while the low level predicates "add" and "multiply" are structurally similar due to their similar roles in the whole relations. Burstein [Bur86] demonstrates with his system CARL the necessity of mapping between nonidentical relations, criticizing Gentner's structure mapping which fails on this kind of similarity.

Another characteristic of our mapping scheme is hierarchical mapping. This is frequently used when problems are represented in hierarchical structure. In fact, analogy between problems usually exists at an abstract level. Thus, mapping starts at the highest level first and proceeds to the next lower level and so on until analogy breaks down.

In our scheme, the entire mapping process is a recursive application of a two-step hierarchical mapping: first map the two problem structures systematically under structural similarity and then decompose them into the next lower level structures (see Figure 1). Structural similarity is found not only in physical structures but also in functional structures. Functional structures are described by functional objects and relations such as functions, purposes, goals, constraints, conditions, and states. For example, *an air conditioner is like an electric fan* because their top level functions are the same (i.e., excite-air). Another example of analogy is found between *society* and *organism* because they are similar in their functional organizations.

As a result of analogical mapping, an analogy map is generated for two cases showing correspondences between both relations and their objects. An analogy map represents a common structure between source and target structures with a binding list between source and target elements. The common structure represents a common problem schema which is used as a medium of transfer in analogical problem solving [Shi88,CM85]. For example, analogical mapping between *multiply(a,b)* and *add(a,b)* generates the analogy map as a common structure *OP(a,b)* with the binding list [(OP multiply add)] meaning that there is one binding *OP* and it binds to *multiply* in the source and to

```
Input: A source case and a target problem
Output: An analogy map (AMAP)
Algorithm:
  Recursive application of two-step hierarchical mapping:
  given two problem structures,
    1. Map them systematically under structural similarity:
       identify components whose partonomic roles in both structures are the same;
       map components as specifically as possible under the current AMAP;
       if mappable
       then add correspondences between components to AMAP
       else return AMAP
    2. Hierarchical refinement:
       decompose the current level into the next lower level structures;
       pair them in the same partonomic roles
```

Figure 1: Analogical Mapping Algorithm

add in the target.

3 Related Work

Gentner's structure mapping theory with its implementation, Structure-Mapping Engine (SME) [FFG86], demonstrates the importance of systematicity in interpreting an analogy. But, it is often criticized because of its syntactic approach.

Many recent models consider semantic and pragmatic characteristics of analogy as well as syntactic information to guide analogical mapping [FFG86]. For instance, Burstein [Bur86] introduces some top-down constraints on relations and primarily relates objects in terms of their functional roles in analogical mapping. Winston's mapping is driven by importance-dominated matching [Win80,Win82]; importance is mainly determined by causal relations in the situations.

Holyoak and Thagard's mapping theory [HT88] attempts to take into account all three dimensions of analogy: syntax, semantics and pragmatics. Their program called ACME computes an analogical map by means of constraint-satisfaction based on five heuristic constraints: logical compatibility, uniqueness, relational consistency, semantic similarity, and role identity. Semantic and pragmatic information help to constrain the search for the most plausible mapping. But, the problem with this approach is that there exist many analogy problems on which such heuristics do not work (an example will be shown in Section 5.1).

Our mapping algorithm is similar to SME in that both enforce systematicity (as shown in the previous section), but different in that ours maps predicates under similarity by functional roles while SME maps under identity. Ours is also similar to Burstein's and ACME in that it maps components by considering part-whole relationships. However, unlike ACME and Winston's, much of the semantic information is not explored during mapping. Rather, it will be checked when the knowledge to be transferred is justified in the transfer step.

4 Applications of Analogical Mapping

Analogical mapping is a step of predicting a plausible analogy, which will be tried for transfer. During the actual transfer attempt, mapping results are filtered considering semantic similarity. **The following applications show how these processes are performed.**

The first two applications of our mapping algorithm rely on similarity in physical structures: Section 5.1 shows analogy examples between different domains, while Section 5.2 compares analogies within the same domain. These examples are also used to compare our algorithm to two general analogical mapping mechanisms, SME and ACME. In Section 5.3, an application from the JULIA project shows an example in functional structures.

4.1 Inter-Domain Examples

Applying our analogical mapping algorithm, let's solve the problem $\frac{d}{dx}[\sin x - \ln x]$ using the following case:

Problem: $\int[e^x + 1]\,dx$
Solution: $e^x + x + C$
Reasoning steps: $\int[e^x + 1]\,dx \Longrightarrow \int e^x\,dx + \int 1\,dx \Longrightarrow e^x + x + C$

When the mapping algorithm, in the first cycle, is applied to the top level structures (i.e., $\frac{d}{dx}[\sin x - \ln x]$ and $\int[e^x + 1]\,dx$), it successfully produces the analogy map

$$\mathcal{F}[f(x)\;OP\;g(x)]$$

with bindings $[(\mathcal{F}\ \frac{d}{dx}\ \int)\ (\mathrm{OP}\ -\ +)\ (f(x)\ \sin x\ e^x)\ (g(x)\ \ln x\ 1)]$. Since the first reasoning step of the source case predicts the following analogy (in an abstract form):

$$\mathcal{F}[f(x)\;OP\;g(x)] = \mathcal{F}[f(x)]\;OP\;\mathcal{F}[g(x)]$$

the target problem reduces as follows:

$$\frac{d}{dx}[\sin x - \ln x] = \frac{d}{dx}\sin x - \frac{d}{dx}\ln x$$

In the next cycle, the mapping between the next lower level structures $\int e^x\,dx$ and $\frac{d}{dx}\sin x$ succeeds, but analogical transfer between these two fails. This is the level where the analogy breaks down and the mapping process halts. Thus, the analogy between the above two cases resides only at the top level. This example shows the utility of hierarchical mapping in identifying analogy, since the analogy at higher levels of abstraction can be used even though there does not exist a complete analogy.

Consider another problem

$$e^{2x+3}$$

using the same source case. It is similar to the first example in that mapping predicts

$$e^{2x+3} = e^{2x} + e^3$$

However, this hypothesis is not correct; the correct transformation is $e^{2x+3} = e^{2x} * e^3$. This example shows that successful analogical mapping may not guarantee the existence of analogy when semantic similarity is missing. The semantic similarity is checked using reasonings similar to those of the source case. This is done during the process of analogical transfer to justify the hypothesized analogy. (See [Shi88] for more discussion of the justification problem.)

Let us apply SME and ACME to the first analogy problem:

Target: $\frac{d}{dx}[\sin x - \ln x]$
Source: $\int[e^x + 1]\,dx$

SME fails to recognize this analogy because the two high level predicates $\frac{d}{dx}$ and $\int$ are not identical. This example shows that structural mapping under predicate identity is too strong. In case of ACME, the logical compatibility requires the second arguments $\ln x$ and 1 to be the same logical kind (e.g., constants to constants) so that ACME also fails on this analogy. This case suggests that semantic and pragmatic information should be used cautiously because of their heuristic nature.

4.2 Intra-Domain Examples

Given a problem

$$\int \frac{1}{\sqrt{3 - y^2}}\,dy$$

consider analogical mapping problems with each of the following three cases.

Case 1:

Problem: $\int \frac{1}{\sqrt{3+y^2}}\,dy$
Solution: $\sinh^{-1}\frac{y}{\sqrt{3}} + C$

Case 2:

Problem: $\int \frac{1}{\sqrt{1-x^2}}\,dx$
Solution: $\sin^{-1} x + C$

Case 3:

Problem: $\int \frac{1}{\sqrt{5-z^2}}\,dz$
Solution: $\sin^{-1}\frac{z}{\sqrt{5}} + C$

All three mappings succeed with our mapping algorithm because the three cases are all structurally similar to the target problem.

In the first case, analogy transfer from case 1 to the target problem is not possible (because the previous reasoning of case 1 is not applicable to the target problem). In the second case, transfer from the source case is not possible until some modification is performed. That is, in order to apply the solution of case 2 to the target problem, the form $\sqrt{a - z^2}$ embedded in the target problem needs be transformed to the form $\sqrt{1 - x^2}$ in case 2. In the third case, the source solution can be transferred to the target domain so the target solution will be $\sin^{-1}\frac{y}{\sqrt{3}} + C$.

The success of analogical mapping leads directly to analogy transfer in the third example. The first example shows, however, that the success of mapping may not guarantee successful analogy transfer. (It only predicts a possibility of transfer which should subsequently be verified.) Furthermore, the second example shows that even when analogical mapping eventually leads to analogy transfer, successful analogical mapping may not directly dictate what is to be transferred from the source case to the target problem. (It may only hint at what is to be transformed in order to reach a transferable state.) So, the role of analogical mapping is to identify a plausible analogy based on known similarity before transfer of analogical knowledge from the source case to the new problem is attempted [Shi88].

Note that SME and ACME are similar to our mapping algorithm in that they will come up with successful mappings with all three cases. This shows that, even when ACME considers semantic and pragmatic accounts, it is not able to distinguish analogies which lead to analogical transfer (i.e., case 3) from analogies which do not (i.e., cases 1 and 2). In other words, ACME is not more powerful than SME and ours in dealing with these three.

4.3 An Application in JULIA

Our analogical mapping mechanism is part of the case-based reasoner [Shi88] in JULIA, an intelligent caterer's advisory system [CK86,Kol87]. Each problem case in JULIA has problem and solution parts. The problem part describes its problem functionally in terms of goals and constraints while the solution part contains a solution plan and the reasoning history.

Since problem cases are represented in a hierarchical structure, JULIA maps the top level problem structures first. It tries to identify similarity between two functional structures. In a frame-based representation, it is straightforward to identify the same functional components (e.g., goals and constraints). JULIA starts mapping with goals between problems: if the goals fully match, the mapping proceeds to constraints; in case of a partial match, which means some goals match but others do not, only the matched goals will be considered for possible transfer; otherwise, the mapping fails. Mapping then proceeds to constraints on only the matched goals to establish correspondences between them. For example, JULIA would consider two cost constraints LOW-COST and INEXPENSIVE[2] similar, because they are functionally the same in that they both constrain the cost. Then, should LOW-COST and EXPENSIVE be considered similar, too? JULIA views that they, too, are functionally similar due to the same reason. However, these do not have as much in common semantically as LOW-COST and INEXPENSIVE.

This problem will be resolved during actual transfer and there the degree of semantic similarity determines the degree of learning involved. Suppose the source case made the following inference during its problem solving:

> **If** c-cost(LOW-COST)
> **then** c-ingredient-cost(LOW-COST) and c-cooking(LOW-COST)
> **because** cost of dish is cost of ingredients plus cooking cost

Then, during analogical transfer, JULIA will try to transfer the previous inference rule with the similar concept INEXPENSIVE using its justification ("because") clause. In other words, JULIA hypothesizes a rule substituting LOW-COST in the rule for INEXPENSIVE, seeing if the justification previously used is similarly applicable. Since the justification also holds for the target case, the new rule will be transferred:

> **If** c-cost(INEXPENSIVE)
> **then** c-ingredient-cost(INEXPENSIVE) and c-cooking(INEXPENSIVE)
> **because** cost of dish is cost of ingredients plus cooking cost

However, if it were EXPENSIVE, the similar inference may not be true because not every ingredient needs to be expensive to make a dish expensive.

5 Summary and Conclusions

We have shown that successful mapping may not guarantee successful transfer of analogy. Analogical mapping only predicts a possibility of transfer which should subsequently be verified. Even

[2]INEXPENSIVE ranges from low cost to moderate cost so that its meaning is slightly broader than that of LOW-COST.

when analogical mapping eventually leads to analogy transfer, successful analogical mapping may not directly dictate what is to be transferred from the source case to the target problem.

An analogical mapping algorithm has been introduced as a recursive application of two-step hierarchical mapping: first map the two problem structures systematically under structural similarity and then decompose them into the next lower level structures. Structural similarity is identified during this mapping process, while semantic similarity is checked during analogical transfer. These two processes together guarantee the correctness of analogy transfer.

Acknowledgments

This work could not have been done without the support and guidance of Janet L. Kolodner. I would like to thank Patsy L. Holmes, David Wood, Mark A. Graves, Joel Martin, and Mike Redmond for useful comments and discussion on earlier versions of this paper.

References

[Bur86] M.H. Burstein. Concept formation by incremental analogical reasoning and debugging. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Kaufmann, Los Altos, CA, 1986.

[CK86] R.E. Cullingford and J.L. Kolodner. Interactive advice giving. In *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics*, 1986.

[CM85] J.G. Carbonell and S. Minton. Metaphor and commonsense reasoning. In J.R. Hobbs and R.C. Moore, editors, *Formal Theories of the Commonsense World*, Ablex, Norwood, NJ, 1985.

[FFG86] B. Falkenhainer, K.D. Forbus, and D. Gentner. The structure-mapping engine. In *Proc. AAAI-86*, 1986.

[Gen83] D. Gentner. Structure-mapping: a theoretical framework for analogy. *Cognitive Science*, 7:155–170, 1983.

[HT88] K.J. Holyoak and P. Thagard. Analogical mapping by constraint satisfaction: a computational theory. 1988. Accepted at Cognitive Science.

[Kol87] J.L. Kolodner. Capitalizing on failure through cased-based inference. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, Washington, July 1987.

[Pol54] G. Polya. *Mathematics and Plausible Reasoning: Induction and Analogy in Mathematics.* Volume 1, Princeton University Press, Princeton, NJ, 1954.

[Shi88] H.S. Shinn. Abstractional analogy: a model of analogical reasoning. 1988. Submitted for publication.

[Win80] P.H. Winston. Learning and reasoning by analogy. *Comm. ACM*, 23(12):689–703, 1980.

[Win82] P.H. Winston. Learning new principles from precedents and exercises. *Artificial Intelligence*, 19:321–350, 1982.

Organizing and Using Schematic Knowledge for Medical Diagnosis*

Roy M. Turner
School of ICS
Georgia Institute of Technology

Abstract

A major problem for both human and computer diagnosticians is representing and organizing problem-solving knowledge in such a manner that the knowledge can be quickly accessed and easily used. Some researchers (e.g., [Lesgold *et al.*, 1981] and [Feltovich *et al.*, 1984] feel that expert medical diagnosticians have at least some of their problem-solving knowledge in a schematic form—procedures, or plans and scripts—that can efficiently be brought to bear on diagnostic problems.

In this research, we present an approach to diagnostic reasoning, called *schema-based reasoning*, that allows a reasoner to access and use the most specific *procedural* information available for the problem at hand. Our approach represents the problem solver's knoweldge as *schemata*: packets of procedural knowledge about how to achieve a goal or set of goals. Schemata are organized in memory by the category of diagnostic problem they are useful for and along hierarchies defined by features of the schemata. When presented with a new problem, the reasoner retrieves schemata based on features of and goals present in the problem; the schemata are then applied by the reasoner to achieve its goals.

The process of schema-based reasoning was designed with an eye towards learning from experience; we discuss some initial ideas along these lines in this paper, specifically comparing our approach to *case-based reasoning* [e.g., Ashley, 1986; Kolodner, 1987; Kolodner *et al.*, 1985; Simpson, 1985].

Our approach is implemented in the MEDIC program, a schema-based diagnostic reasoner whose domain is pulmonology.

1 Introduction

One of the major differences between a novice and an expert is that the expert has his or her knowledge in a readily accessible and usable form. The novice at medical diagnosis—a medical student—does not necessarily suffer from a lack of facts; rather, the novice does not have the facts organized in a fashion that allows them to be brought to bear quickly and efficiently on a problem. In addition, the novice does not have available the procedural knowledge necessary to allow him or her to quickly and easily solve diagnostic problems. The novice's knowledge, in other words, is not operational.

In order to operationalize a diagnostician's knowledge, he or she is given cases to solve, either practice or real. As cases are solved, the student learns what is important in the problem-solving environment, and learns what knowledge to use to solve which kinds of problems. Part of the learning process consists of converting "book knowledge" of signs, symptoms, and diseases into procedural knowledge: schemata for solving diagnosis problems (cf. [Lesgold *et al.*, 1981]). Another part of the learning process is organizing the information learned—new facts as well as new schemata—in a form that allows it to be brought to bear efficiently on future problems.

The problem of how to represent and organize diagnostic knowledge has been studied in several artificial intelligence projects. In CENTAUR [Aikins, 1980], for instance, problem-solving knowledge is represented in the form of *prototypes* (frames) with associated rules; this has the effect of clustering the rules used around the contexts in which they are useful. MDX [Gomez and Chandrasekaran, 1982] is similar, in that its

*This research has been funded in part by NSF Grants IST-831771 and IST-8608362 and a grant DTD 09-25-87 from the Lockheed AI Center.

knowledge is represented as rules stored in *specialists* in a specialization (or taxonomic) hierarchy. Kolodner and Kolodner [1987] proposed a scheme in which knowledge used in diagnosis is stored in episodes and *generalized episodes* in a *dynamic memory* [Schank, 1982]. This approach has several benefits: (1) due to the properties of the dynamic memory, the most specific knowledge possible for a particular problem can be retrieved; (2) previous cases of problem solving are available for use in similar situations; and (3) the general knowledge, and the organization of the memory, is changed as new cases are added to the memory.

Though all these approaches organize the reasoner's knowledge in a form that is readily accessible, they tend to ignore the procedural knowledge necessary to perform diagnosis. That is left in the program itself and does not reside in the reasoner's knowledge structures. This assumes that one general method of performing diagnosis, using many specialized pieces of knowledge, can allow the reasoner to effectively diagnose problems. However, many researchers (e.g., [Lesgold *et al.*, 1981] and [Feltovich *et al.*, 1984]) believe that as diagnosticians become more expert, they increasingly use schema-like information—procedures, or plans and scripts—to perform diagnosis, and that this procedural knowledge is gained from experience. By using schema-like information, the reasoner can bring specialized problem-solving procedures to bear on diagnostic problems.

In this paper, we discuss a means of memory organization and retrieval that allows a reasoner to find the most specific problem-solving procedures available for a particular problem, then to use that information to solve the problem. Our work is based, to some extent, on preliminary work which was done on the SHRINK project some years ago [Kolodner, 1983]. In our approach, we represent the problem solver's knowledge as *schemata*, which are packets of procedural knowledge much like SHRINK's "process MOPs". Schemata are organized in memory by many different hierarchies of features present in the schemata, and are retrieved by indexes composed from features present in the problem being solved. When retrieved, a schema guides the reasoner in selecting actions to perform to solve a problem. Our approach, which we call *schema-based reasoning*, is implemented in the MEDIC program, a diagnostic reasoner whose domain is pulmonology.

Our research was begun with the idea that the results should lend themselves to making use of problem-solving experience. We will also discuss in this paper some initial ideas along these lines, as well as the relationship of our work to a form of reasoning from experience called *case-based reasoning* [e.g., Ashley, 1986; Kolodner, 1987; Kolodner *et al.*, 1985; Simpson, 1985].

2 Schemata

According to Bartlett, a schema is:

> ...an active organization of past reactions, or of past experiences, which must always be supposed to be operating in any well-adapted organic response [Bartlett, 1932].

This is very similar to how we view schemata: a schema is knowledge that tells a reasoner how to respond to a particular situation.[1] A schema is basically a packet of procedural knowledge, represented in a declarative form, that can achieve a goal or set of goals; it is somewhat like a program with procedures. A diagnostic reasoner should have many schemata, for the many different situations and goals it will encounter. When the reasoner is confronted with a problem, it retrieves a schema for some portion of the problem, then interprets, or *applies*, the schema by taking the actions it describes.

Figure 1 shows a simplified picture of what one of MEDIC's schemata looks like, in this case a schema that can be used to interpret a finding of dyspnea. In addition to the actions to be taken in following the schema (the schema's *steps*), the schema contains information about goals it can be used to achieve, features of situations in which it is useful, preconditions or restrictions on its use, and the expected results of using it.

Each of the steps of a schema is composed of three parts: an action, a goal, and information used to choose the next step. The action is either a primitive action the reasoner can perform or another schema. The goal is a goal that the step is meant to achieve. The "next step" information consists of tests to perform against the state of the world and steps to be selected if the tests are true. This information is used to order the steps of a schema; there can be several different orderings, depending on the state of the world at the time the schema is used, including optional

[1] Though this research does not address how schemata come to exist, they can be thought of as being the result of past experiences, whether of the program or of the human expert who gave them to the program.

```
Goal: interpret a finding of dyspnea
Patient: any patient
Findings: dyspnea
Preconditions: there is a finding of dyspnea
Actions:
     A1: action: ask how many stairs patient can climb
         goal: determine severity of dyspnea
         next:
              if pt. can climb flight of stairs ⟹ A3
              else ⟹ A2
     A2: action: ask how far patient can walk
         goal: determine severity of dyspnea
         next: A3
     A3: action: estimate the severity of the dyspnea
         goal: determine severity of dyspnea
                    ⋮
     A10: action: postulate hypotheses of pulmonary disease,
         cardiac disease
         goal: explain dyspnea
         next: done
Indices:
     patient/PATIENT1 → SCENE2
               ⋮
```

Figure 1: sc-dyspnea—a schema for interpreting a finding of dyspnea.

steps. If the action portion of a step is omitted, then the step suggests a goal that should be satisfied at that particular point in schema application.

In many ways, a schema is similar to traditional reasoning knowledge structures. For example, if each step in the schema has an action that is a primitive action, then the schema is equivalent to a *script* [Schank and Abelson, 1977] with *tracks* [Cullingford, 1981]. If, on the other hand, all of the steps of a schema have actions that are either schemata or primitive actions, the schema can be viewed as a abstract plan or as equivalent to one of NASL's [McDermott, 1978] tasks. Finally, we can view a schema as a packet of rules, perhaps similar to MDX's *concepts* or *specialists*. In this view, the rules' "antecedents" would consist of the information in the schema—findings, characteristics of the patient, etc.—(excluding the schema's steps); the "consequents" would be the actions that are specified by the schema.

There are several differences between schemata and these other knowledge structures, however. First, schemata are more general, in effect subsuming the functionality of the others. Second, new schemata can, in principle, be created by applying old schemata to new situations, then generalizing the result. This would allow schemata to be created for situations not anticipated when the reasoner was given its knowledge; the reasoner could adapt to its environment by operationalizing its problem-solving knowledge. And third, schemata are active participants in memory organization, as will be discussed below. This organization allows the most specialized schemata for a portion of a problem to be retrieved and applied to that problem.[2]

3 Retrieving Schemata

Since a reasoner will encounter many different goals in many different situations, it should have a wealth of schemata at its disposal. The reasoner's memory must organize these schemata in such a way that the most specific schema available to achieve a particular goal can be found when the reasoner needs it. In order to do this, the schemata must be linked to one another in memory so that retrieval is facilitated.

MEDIC's memory organizes its schemata in two general ways: by the category of diagnostic situation in which they are useful, and by the use of specialization hierarchies of schemata. Figure 2 shows a portion of MEDIC's memory. There are four types of memory structure present in the figure (and in MEDIC's memory):

1. *diagnostic memory organization packets* (dxMOPs) (cf. the memory organization packets (MOPs) of [Schank, 1982])—representing generalized sessions of diagnosis;
2. *cases*—representing individual (presumably unusual) cases of diagnosis that the reasoner knows about;
3. schemata; and
4. *scenes*—representing portions of diagnostic sessions in which a schema was used; in other words, a scene represents an instantiation of a schema in a particular case.

The dxMOPs and cases provide contexts against which a current problem can be matched (cf. the diagnostic categories of [Kolodner and Kolodner, 1987])—i.e., they allow the reasoner to categorize the current problem. These memory structures contain information about the findings which occur in particular types of problems, the hypotheses that are usually considered, patient characteristics, and, most importantly, schemata that can be used to achieve goals arising in consultations of this type. For example, the dxMOP "dx-consult" in Figure 2 represents a generalized consultation. It contains the information that consultations involve a patient, a doctor,

[2]MDX's specialists also participate in its memory organization; however, schemata are different from specialists both in the type of information they contain and in the way they are used by the reasoner.

and the program; findings will generally be discovered; and hypotheses will be considered. A dxMOP also contains pointers to schemata that can be used to achieve goals expected in the type of consultation the dxMOP represents: e.g., "sc-finding" to handle goals to interpret findings, "sc-hypothesis" to handle hypotheses, and so forth. A case, though much more specific than a dxMOP, contains basically the same type of information; instead of representing a category or prototype of a diagnostic situation, however, it represents an exemplar of a particular kind of diagnostic session. Its scenes represent instances of schemata having been applied in the past to solve particular goals in the situation represented by the case.

MEDIC's memory is organized in a manner similar to that described by [Kolodner, 1984] for episodic memory structures. The memory is basically an interconnected set of discrimination nets, or hierarchies, in which the leaf nodes are cases and scenes, and the interior nodes are dxMOPs or schemata. Memory nodes are linked by *indices*, each of which is a feature/value pair, where the "feature" is drawn from the more abstract memory structure of the two, and the "value" is the value, in the more specific structure, of the feature. Features are selected based on their predictiveness, or ability to point to useful specializations. For instance, one of the indices between "dx-consult" and "dx-cc-dyspnea" (representing consultations in which the chief complaint is dyspnea) is "chief complaint/dyspnea."

DxMOPs and schemata both serve to organize portions of memory. A dxMOP organizes other (less abstract) dxMOPs and cases along dimensions defined by the features of the consultation described by the dxMOP. Schemata organize more specific schemata and instances of schemata having been applied; this organization is along features of the schemata: goals it achieves, characteristics of the situations in which it is useful, etc. The two types of organization hierarchies—by dxMOP and by schemata—are connected by links between dxMOPs and the schemata useful in the situations they describe, and between cases and the scenes (instances of schema application) that occurred in them.

Retrieval consists of using features present in the current problem (characteristics of the patient, findings present, and problem-solving goals) to *traverse* [Kolodner, 1984] the indexing structure of the memory to find the most specific schemata available that fit the current situation. The actual process of retrieval is beyond the scope of this paper; it is basically the same as that described in [Kolodner, 1984].

As an example of retrieval, suppose the reasoner is working on a problem in which there is a finding of dyspnea on exertion. One of the reasoner's goals would be to interpret the finding: to flesh it out and to explain its occurrence. The schema that contains the information necessary to do this is labeled "sc-DOE" in Figure 2. The reasoner can find this schema by two paths. It can traverse the indices of "dx-consult", using information about the finding present in the problem, to first find "dx-cc-dyspnea", then "dx-cc-DOE". From here, "sc-DOE" can be found by using information about how to satisfy the goal of interpreting DOE. Or the reasoner can begin by looking in "sc-consult" for a schema which achieves the general goal: in this case, "interpret a finding." The indices of this schema can then be traversed, through "sc-dyspnea" to "sc-DOE".

Multiple paths to the same schema serve the purpose of helping ensure that a good schema can be found for specific situations. Since there are redundant paths to each schema, the reasoner can still find a schema even if there is not enough information present in the current problem to allow it to traverse some paths.

4 Applying Schemata

Schema application is somewhat analogous to program interpretation. First, a step is selected. If the step's action is a primitive action, the reasoner executes it directly; if the action is a schema, then the reasoner recursively applies it. If the action fails, or if no action is specified, the reasoner attempts to find and apply another schema to achieve the goal of the step. A new step is then selected by using information contained in the "next step" portion of the step, and the process continues.

There are several control problems that are beyond the scope of this paper. For instance, the reasoner should react to new information as it is discovered by retrieving schemata to handle it: e.g., when a finding is discovered, the reasoner should find a schema that can interpret the finding, and when a hypothesis is proposed, the a schema to evaluate the hypothesis should be found. At any particular time, there may be several active schemata. The problem of selecting the schema to apply at a particular time is one of focusing the reasoner's attention; it depends on (among other things) the reasoner's current goal and the importance of the findings and hypotheses under consideration.

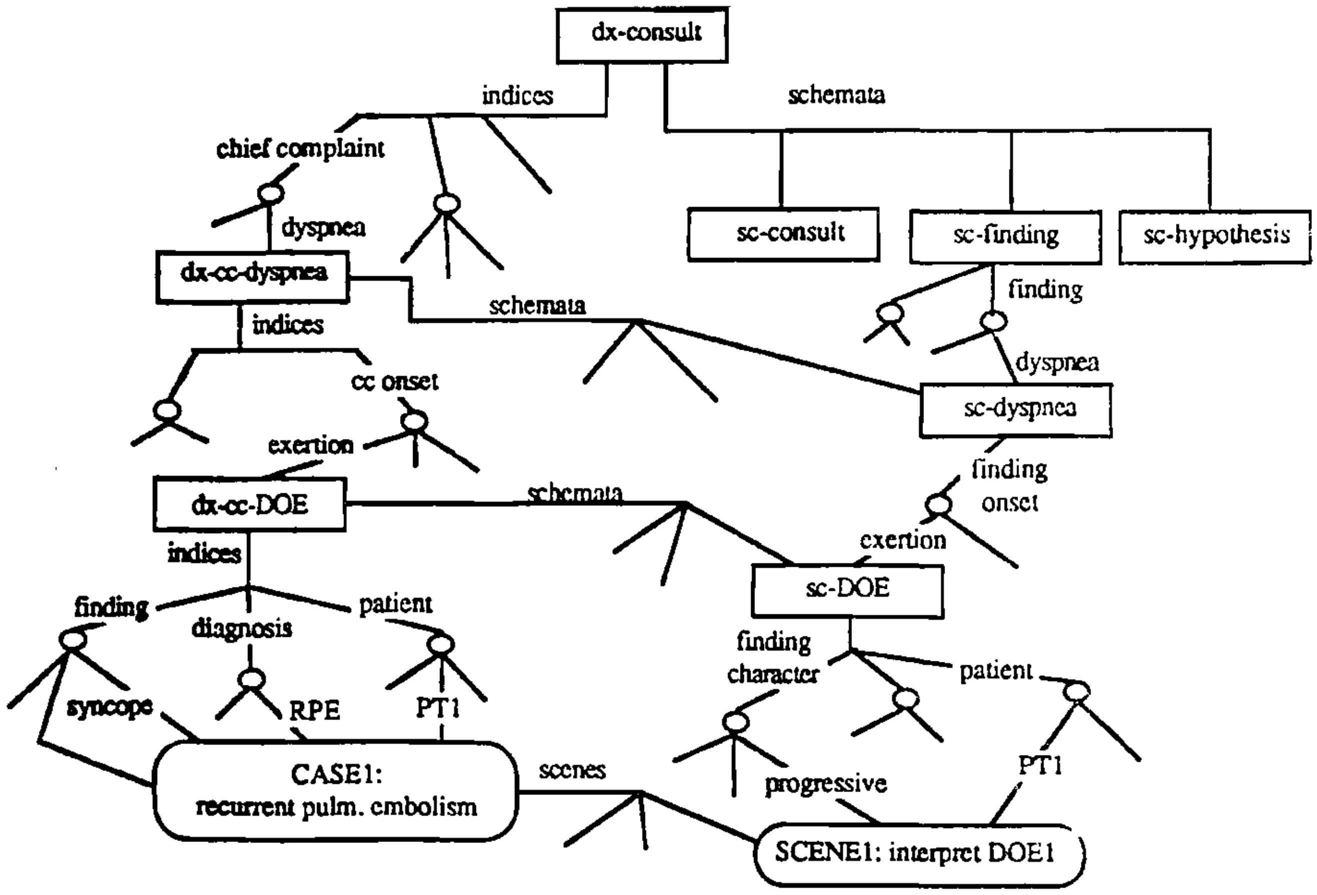

Figure 2: A portion of MEDIC's memory.

5 Medic

Our approach to diagnostic reasoning is implemented in the MEDIC program, a schema-based reasoner working in the domain of pulmonology. MEDIC consists of a memory, a reasoner, and a short-term memory (STM). The memory is organized as we have described in this paper. The reasoner is a bit more complex, in that it is able to respond (to a limited extent) to new information as it occurs. The reasoner operates at all times under the direction of a schema.

Conceptually, MEDIC has two types of schemata in its memory: global schemata and local schemata. A *global schema* is one that can direct the reasoner in performing large segments of a consultation. Examples are: "sc-consult", which directs the entire course of a consultation; "sc-getInfo", which gathers information from the user; and "sc-formDx", which forms a diagnosis from the hypotheses present in STM. A *local schema* is one that can direct the reasoner to achieve very specific goals, such as interpreting a finding or evaluating a hypothesis. Examples are: "sc-finding", an abstract schema which interprets any finding; "sc-dyspnea", which interprets a finding of dyspnea; "sc-dzHypothesis", which can evaluate any disease hypothesis; "sc-pulmDz", a specialization of sc-dzHypohthesis for pulmonary disease; and "sc-RPE", a further specialization for evaluating recurrent pulmonary embolism.

Having very specialized schemata allows the reasoner to act in a more focused manner than if it had only more abstract schemata. For example, a schema for interpreting dyspnea can be used to directly ask the user questions aimed at eliciting specific information to evaluate the severity: "How far can the patient walk?" or "How many stairs can the patient climb?" If a more abstract schema were used, a way would have to be found to gather the information and fill in the severity. Further specializations can be used to make even finer interpretations, or interpretations in rare but important contexts: e.g., a schema for interpreting dyspnea in someone who is restricted to a wheelchair should cause the reasoner to ask different questions than a schema for interpreting dyspnea in someone who is ambulatory.

Currently, MEDIC can diagnose very simple cases of pulmonary disease. Its basic algorithm is described in Figure 3. When the user asks for a consultation with MEDIC, the program attempts to find a dxMOP which describes the situation. This dxMOP is then the source of schemata to achieve goals arising during the consultation. In addition, the reasoner retrieves a *strategy* from the dxMOP; the strategy is used by the reasoner to select from among its active schemata. An example of a strategy can be seen in Figure 4; this strategy provides a goal ordering to the reasoner which causes the reasoner to perform a crude form of hypothetico-deductive reasoning: select goals (schemata) related to hypotheses first, then select those that relate to findings (with the hope of generating hypotheses), etc.

```
begin
    loop forever:
        Wait until user requests a consultation;
        Add goal of diagnosing patient to short-term
                memory;
        Retrieve dxMOP using goal;
        Use strategy from dxMOP, if possible;
        Select a schema from the dxMOP to satisfy
                goal, add it to agenda;
        loop until done:
            Select a schema from agenda using strategies,
                local information in the dxMOP;
            Apply one action of the schema;
            if there was an interruption then:
                Handle interruption;
            fi;
            Specialize current dxMOP;
            if specialization succeeded then:
                Set current dxMOP to be the
                        specialization;
            fi;
        end loop;
        Accept and process feedback;
        Update memory;
    end loop;
end.
```

Figure 3: Basic schema-based reasoning algorithm.

```
Situation: any
Goal ordering:
    select goals related to hypotheses
    select goals related to findings
    select goals for gathering information
    select goal for forming diagnosis
```

Figure 4: Strategy "st-HD-Reas" for hypothetico-deductive reasoning style.

Let's look at an example of a consultation with MEDIC, a portion of which is shown in Figure 5. Suppose a user requests a consultation. The reasoner looks in memory for a way of satisfying the goal of diagnosing a patient and finds a dxMOP, "dx-consult", representing how consultations are generally done. This is made the current dxMOP, and it is used as a source both of a strategy and of schemata to satisfy active goals. The strategy it contains is "st-HD-reasoning", the strategic schema mentioned above which provides a goal ordering to induce hypothetico-

```
Please describe the patient.
: (patient (sex female) (weight (value 204)) (height
            (value 64)) (race white))
Adding information about patient to STM.
What is the chief complaint?
: (finding (entity (dyspnea (duration (years 2))
            (character progressive))))
Adding chief complaint to STM...adding finding of
            <DYSPNEA0> to STM.
How many flights of stairs can the patient climb?
: (less-than 1)
How far can the patient walk on level ground?
: (yards 20)
I judge the qualitative value of SEVERITY of <DYSPNEA0>
            to be SEVERE.
            ...(same for cardiac disease)...
...explaining dyspnea...
Processing <HYPOTHESIS0> [pulmonary disease];
            relating to other hypotheses...
...generating expectations given <HYPOTHESIS0>...
...I'm scoring hypothesis <HYPOTHESIS0> (<PULM-DZ0>)
...hypothesis explains: (<FINDING0>) [dyspnea]...
...failed predictions for hypothesis: —
...hypothesis doesn't explain: —
...trying to specialize the hypothesis of
            <HYPOTHESIS0> (<PULM-DZ0>)...
...specialized <HYPOTHESIS0> to <HYPOTHESIS1>
            (<RPE>) [recurrent pulmonary embolism]
...generating expectations given <HYPOTHESIS1>...
            ...
Is there a finding of <SYNCOPE>?
: Yes
            ...
Enter information (<return> if no more).
:
            ...
My diagnosis is: Recurrent pulmonary embolism.
```

Figure 5: Part of a consultation with MEDIC.

deductive reasoning. The only goal active is one to diagnose the patient; the schema to achieve this in dx-consult is "sc-consult". This is added to the reasoner's agenda.

The reasoner now selects a schema from its agenda, using the goal ordering provided by the current strategy; the use of specific information from the dxMOP is not currently implemented. The only schema to select is sc-consult, so the reasoner selects that and begins to apply it. The user is asked for some initial information about the patient, including a description of the patient (a white female who is overweight)[3] and the chief complaint (progressive dyspnea). The information is added to STM. Adding the chief complaint causes the reasoner to be interrupted, and it searches memory for a schema to interpret the finding. Schema "sc-dyspnea" is found and activated.

Since the strategy in use dictates that goals related to findings have precedence over goals for gathering information or forming a diagnosis, sc-dyspnea is selected and used. This schema is a specialized version of a general schema to interpret findings; instead of asking general questions, however, the schema can ask very specific things related to dyspnea (e.g., asking how many stairs the patient can climb as a measure of the severity). The last step of this schema is to explain the finding by postulating diseases that could cause it; using this step, the reasoner postulates hypotheses of pulmonary disease and cardiac disease. Adding these hypotheses to STM again interrupts the reasoner, which finds and adds to the agenda schemata to evaluate the hypotheses: "sc-pulmDz" and "sc-cardiacDz".

The strategy orders goals related to hypotheses before any others; hence, one of the two schemata just added is selected, in this case, sc-pulmDz. The reasoner uses this schema to score the hypothesis of pulmonary disease,[4] and then tries to specialize the hypothesis using information that is in STM. One possible specialization, based on the fact that the patient is overweight, is recurrent pulmonary embolism (RPE); this is hypothesized, resulting in a schema ("sc-RPE") being activated to evaluate it.

The reasoner then selects sc-RPE and begins to evaluate the hypothesis of pulmonary embolism. Eventually, it will have evaluated all the hypotheses it can and will have exhausted the information the use can give it. The main schema, sc-consult, will then suggest the step of forming a diagnosis, which will be attempted.[5] In this case, the best hypothesis is recurrent pulmonary embolism, and that will be proposed to the user.

The current implementation of MEDIC is incomplete in several ways. For example, MEDIC does not have a principled way of choosing from among several active schemata the one to follow at any particular point in diagnosis; i.e., there is currently no theory addressing the control of the reasoner's attention. In addition, there is currently neither the domain knowledge nor the variety and number of schemata present in memory to allow very sophisticated diagnoses to be made; gathering this knowledge from our domain expert is one of the next steps in this project.

Learning is currently not addressed, either. We cannot expect to give a diagnostic reasoner operating in a sophisticated domain all of the knowledge that it will need to solve all of the problems presented it [Kolodner and Kolodner, 1987]; instead, if the program's knowledge is to be made as operational as possible for the domain, the program will need to be able

[3]Input to MEDIC is in a version of Conceptual Dependency [Schank & Abelson, 1977]; there is currently no natural language interface.

[4]Using a scoring scheme very similar to that of INTERNIST-1 [Miller *et al*, 1982].

[5]Again, using a method similar to that of INTERNIST-1.

to adapt its own knowledge, both factual and procedural, to its problem-solving environment. Though learning is not a focus of our work, we have tried to formulate the schema-based reasoning approach (and design MEDIC) in such a way that learning could be added at a later time. We discuss this in the next section.

6 Relationship to Case-based Reasoning

Though we do not explicitly address learning in our work, this research was begun with the idea that the style of reasoning which would evolve from it would be amenable to learning from past problem-solving experience. In this section, we discuss one approach to this type of learning, called *case-based reasoning*.

Case-based reasoning [e.g., Kolodner, 1987; Kolodner *et al.*, 1985; Simpson, 1985] involves reusing information from past problem-solving episodes to help solve a new problem. Case-based reasoning (CBR) basically provides reasoning short-cuts: problem-solving that went on in similar previous cases is retrieved and reused in a new problem. CBR has been successfully applied to several different planning tasks [Simpson, 1985; Sycara, 1987; Hammond, 1986], to advice-giving [Kolodner, 1986; Turner, 1987a, 1987b; Cullingford and Kolodner, 1986], and, in one instance (and in a somewhat limited way), to diagnosis [Kolodner, 1983; Kolodner and Kolodner, 1987].

In our approach, "cases" correspond to consultations that have been performed by the reasoner.[6] When a consultation is finished, the reasoner would represent the result as a case and store it in memory. It would be indexed from the dxMOP or dxMOPs that were used in the consultation it represents, since it is a specialization of those dxMOPs. For example, a consultation involving a young alcoholic man with lung cancer might be solved using information from a dxMOP representing consultations involving alcoholic patients and from a dxMOP representing consultations involving patients with lung cancer. The new case representing the consultation would then be indexed by both of these dxMOPs, using features that differentiate it from them.

A case has information about the patient involved in the consultation, findings that occurred, hypotheses that were considered, the diagnosis, and any feedback about the consultation that was obtained from the user. Most importantly, however, a case contains the actions that were performed to diagnose the patient in that consultation; these actions, as mentioned, comprise the *scenes* of the case.

There are four ways that case-based reasoning can be used in our approach:

1. the results of reasoning done in the past can be reused;
2. a case can suggest schemata to use in a similar situation;
3. a case can provide information to allow the creation of new schemata; and
4. the process of storing cases in memory can produce useful changes in the reasoner's general knowledge structures, including the specialization of existing schemata.

When a reasoner is faced with a new problem, it may be *reminded* [Schank, 1982; Kolodner, 1984] of a previous consultation—i.e., a case representing the previous consultation may be retrieved from memory using the features of the new problem. In this situation, the old case can be used as a source of reasoning short-cuts in the new problem; this is how case-based reasoners usually use cases.[7] For example, considerable reasoning effort may have been expended in the old consultation to interpret a particular finding; if the same finding occurs in the new problem, the reasoner can reuse the interpretation from the old consultation instead of repeating the reasoning that was done. One advantage of this was mentioned in [Kolodner & Kolodner, 1987]: an old case can be used as a source of hypotheses about a new problem.

Similarly, a previous case may suggest actions the reasoner can use to achieve goals in a new problem. In our approach, the scenes of a case generally represent instantiatiations of schemata; thus, the scenes of an old case can suggest schemata to use in a new problem. In addition, the reasoner may choose to instantiate a schema in a particular way, based on how it was instantiated in the old consultation.

The reasoner may choose to create new schemata from the scenes of a case, rather than using the schemata that were instantiated in the case. One reason for this to occur would be if the old case and the new problem are quite similar, but not identical,

[6]The cases we mentioned earlier in the paper are given to the program by a human. However, they can be used in the same way as the cases described in this section.

[7]We have not addressed in any detail how these short-cuts are transferred to the new problem. However, a reasonable approach, given the nature of schema-based reasoning, would be to give the reasoner schemata which it can use to perform case-based reasoning.

with respect to features related to a schema that was used in the prior case to achieve some goal. For example, suppose a scene of the old case was concerned with interpreting dyspnea, and the patient was in a wheelchair. In this scene, the usual dyspnea schema would have been modified to take into account that the patient could not walk: i.e., questions such as "How far can the patient walk on level ground?", that would normally be asked, may have been changed to questions that are more useful for that patient. If the new problem involves a patient who walks with crutches, then it is likely that the goal of interpreting dyspnea can be achieved in the new problem using a slight generalization of the actions performed in the old scene. One method of performing this type of generalization is *abstractional analogy*, described in [Shinn & Kolodner, 1988]; alternatively, some sort of explanation-based learning [DeJong, 1983; DeJong & Mooney, 1986] could be used. Once a new schema is formed, it would be indexed from the schema it is a specialization of. It could then be used to solve similar problems in future.

An old case may have scenes that are not instantiations of schemata, but represent instead the end result of some sort of "from-scratch" problem-solving that was carried out in lieu of an appropriate schema to achieve a goal. When the reasoner recalls such a case and has a similar goal, it may choose to create a new schema by generalizing the actions that were performed in the previous scene. Abstractional analogy or explanation-based learning could be used here, too, and the new schema would be stored in memory for future use.

New generalized knowledge, including new schemata, would also be created during the process of storing cases in memory. It is a relatively short jump from the memory described in this paper to a full-fledged *dynamic memory* [Schank, 1982] of the kind implemented in CYRUS [Kolodner, 1984] and used in several case-based reasoning approaches [e.g., Hammond, 1986; Kolodner, 1983; Simpson, 1985; Sycara, 1987]. The differences are that in a dynamic memory: (1) new information can be added; and (2) as information is added, both the existing memory structures as well as the organization changes to facilitate future retrieval.

As mentioned, a new case would be indexed from those dxMOPs it represents a specialization of, using features that differentiate it from those dxMOPs. One of three things can happen for each index in each dxMOP. The index may not currently be used in that dxMOP; in this situation, the case will simply be stored using that index. The index might, however, already point to another dxMOP; in this situation, the new case will be indexed from the dxMOP residing at that index. Finally, there may already be a case at index: in this situation, a *collision* is said to have occurred. Following [Kolodner, 1984], the procedure in this situation would be first to create a new dxMOP by generalizing both cases, then to store the new dxMOP using the index. The two cases—the one that was stored at the index and the new one being added—are then indexed from the new dxMOP by their differences from *it*.[8] The new dxMOP represents a generalization of the two cases and a specialization of the parent dxMOP from which the old case was indexed.

The process of storing cases in memory effectively causes a reasoner to learn new specializations of existing diagnostic categories, based on consultations it has seen. As cases are being added to memory, causing new specialized dxMOPs to be created, we would like for the reasoner to learn specializations of its problem-solving knowledge, too. Each case, recall, represents a consultation the program has had with a user; as such, it contains scenes that represents the actions taken to achieve one goal or set of goals—i.e., instantiations of schemata for the situation faced in that consultation. It would make sense to store the scenes of a consultation in memory by indexing them beneath the schemata they are instantiations of. In this way, new schemata would be created in the same way dxMOPs are created during memory update. These new schemata would represent specializations of existing schemata which were created based on experience using those schemata to solve problems.

The results of creating new dxMOPs and schemata are twofold. First, by creating new dxMOPs, the reasoner learns about new categories of consultations. This allows it to know how to apply its schemata in different situations, since each dxMOP has useful schemata associated with it. By doing this, the reasoner would learn in which types of consultation specific schemata are useful: in other words, the reasoner would be learning the conditions under which its procedural knowledge is applicable. Second, new schemata are produced, allowing the reasoner to solve problems which are similar to the new schemata more quickly, better, or both. In effect, the reasoner is adapting its problem-solving knowledge to better

[8]In [Kolodner, 1984], similarity-based learning was used; for a medical domain such as this, however, SBL should probably be augmented with some sort of explanation-based techniques.

fit its domain: it is operationalizing its procedural knowledge.

In summary, then, case-based reasoning techniques would add much to a schema-based reasoner. Cases provide a way of finding schemata for goals present in new situations that are similar to previously-seen consultations. A case can also suggest new a schema, which can be formed by generalizing the instantiation of a schema or a sequence of actions used to achieve a goal in that case. Finally, storing cases in memory affects the schemata the reasoner has available to it: as a case is stored, new schemata and dxMOPs are formed. The new schemata can be used to solve new problems; information from the new dxMOPs can be used to decide when schemata are applicable.

7 Related Work

As we mentioned, our work is based on preliminary research along the same lines done on the SHRINK program some years ago [Kolodner, 1983; Kolodner & Kolodner, 1987]. In its most specific and mature form, the approach taken by that research was to perform diagnosis using information from an experientially-modified memory consisting of two kinds of structures: DIAGNOSTIC MOPs and PROCESS MOPs.

The DIAGNOSTIC MOPs in SHRINK are similar to our dxMOPs, in that they are memory structures that represent generalized information used during diagnosis. However, there are some major differences. Although DIAGNOSTIC MOPs are meant to be derived from experience, using a process similar to that outlined above for dxMOPs, they are not truly episodic structures: that is, they contain no references to actions. Instead, they represent disease categories, in much the same way as do MDX's [Gomez & Chandrasekaran, 1982] specialists; the difference is that DIAGNOSTIC MOPs are dynamic structures that are updated from experience, whereas specialists are static. In contrast, dxMOPs represent categories of *consultations* rather than of diseases; though a dxMOP may refer to diseases as hypotheses, it contains other information relating to the consultation as a whole. An example of this would be a dxMOP representing consultations involving alcoholic patients. Though not representing a disease, this dxMOP holds information which helps the reasoner to diagnose this type of patient. For example, anemia is generally a fairly important finding; however, the dxMOP for alcoholic patients would contain information which would allow the reasoner to ascribe the finding of anemia in such a patient to alcoholism rather than searching for some other cause. In addition, dxMOPs *do* refer to procedural knowledge: a dxMOP refers to a set of schemata that are useful for achieving goals that are likely to arise during consultations of the type described by the dxMOP.

PROCESS MOPs are similar in some ways to our schemata, though more closely related to scripts. As new cases use a PROCESS MOP, "compiled paths" [Kolodner, 1983] are added to it, analogous to the tracks in a script. Schemata, on the other hand, are more general than scripts, as we have mentioned. A schema allows *some* "compiled paths" to be represented as variations in the ordering of the steps due to its steps' "next" information. However, the degree of freedom here is limited by the constraint that the schema, no matter what path is taken, should satisfy a particular goal or set of goals; which path is taken depends on the environment. We allow specialized versions of schemata to exist to handle related or specialized goals, and these are organized in a manner that facilitates their retrieval in particular situations. PROCESS MOPs were meant to be specializable and to participate in memory organization [Kolodner, personal communication]; however, they were not a major focus of the research on SHRINK, and this aspect of their use was not fully implemented.

The manner in which the reasoner uses each kind of procedural information also differs. PROCESS MOPs are recalled and followed by the reasoner from start to finish; only one is active at once. Many schemata, each being used to satisfy a goal, can be active at once, and the reasoner need not completely apply any schema before switching to another. This allows a degree of flexibility that SHRINK does not have to respond to changes in the task during diagnosis.

Our approach also differs from work relying on a strict interpretation of the term "case-based reasoning:" that the reasoner uses information *only* from cases and not from any generalized structures. The MEDIATOR [Simpson, 1985] is an example of such a case-based reasoner. Though it has Generalized Episodes (GEs) present in its memory as a result of storing cases of problem-solving, it uses this information only for the organization and retrieval of cases. In contrast, the use of generalized knowledge structures is central to our approach: schemata and dxMOPs are used preferentially to cases, representing as they do compiled problem-solving information.

Our knowledge structures and memory organization superficially resemble those of MDX. However, our schemata are more general than MDX's specialists in being able to represent scripts and plans as well as rules. This allows MEDIC to represent procedural knowledge of how to perform diagnosis, in addition to domain knowledge, directly and declaratively.

8 Conclusion

The schema-based reasoning process outlined in this paper is one approach towards providing a diagnostic reasoner with information that is operational for its problem-solving environment. Schemata are retrieved by the reasoner and used to achieve goals present in diagnostic problems: interpret a finding, evaluate a hypothesis, etc. Schemata are stored in a memory that organizes them in two ways: by the situations for which they are appropriate, and along hierarchies defined by their features. The reasoner retrieves schemata by using goals in and feature of the current problem to traverse memory, then applies the schemata to achieve its goals.

Our approach was developed keeping in mind the eventual need for learning in a problem solver for real-world problems. If we were to allow cases of problem solving to be added to our memory in the manner described above, several benefits would accrue: (1) the reasoner's knowledge of consultations would increase; (2) it would learn about situations in which its procedural knowledge is applicable; and (3) its store of schemata would increase, with the overall effect that the program's procedural knowledge would adapt to the task environment as problem solving is done.

Our ideas are being tested in MEDIC, a schema-based diagnostic reasoner whose domain is pulmonology. There is no reason to believe, however, that diagnosis is the only domain in which schema-based reasoning is worthwhile. The basic idea is quite general, and should be applicable to other domains and other types of problem-solving tasks.

9 Acknowledgments

Many thanks to our domain expert, Eric Honig of Emory University and Grady Memorial Hospital, and to Janet Kolodner, Elise Turner, Joel Martin, Mike Redmond, and Phyllis Koton for their comments on drafts leading to this paper.

10 References

Aikins, J.S. (1980). *Prototypes and Production Rules: A Knowledge Representation for Computer Consultations*. Doctoral dissertation, Stanford University.

Ashley, K. (1986). Knowing what to ask next and why: Asking pertinent questions using cases and hypotheticals. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*.

Bartlett, F.C. (1932). *Remebering, a Study in Experimental and Social Psychology*, Cambridge University Press, Cambridge. Quoted in E. Rich, *Artificial Intelligence*, New York: McGraw–Hill Book Company, 1983.

Cullingford, R.E. (1981). SAM. In *Inside Computer Understanding*, R.C. Schank and C.K. Riesbeck (Eds.), Hillsdale, NJ: Lawrence Erlbaum Associates.

Cullingford, R.E., and Kolodner, J.L. (1986). Interactive advice giving. In *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics*.

DeJong, G. (1983) Acquiring schemata through understanding and generalizing plans. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*.

DeJong, G.F., and Mooney, R.J. (1986). Explanation-based learning: An alternative view, *Machine Learning* 1:2, (April, 1986).

Feltovich, P.J., Johnson, P.E., Moller, J.A., and Swanson, D.B. (1984). LCS: The role and development of medical knowledge in diagnostic expertise. In W.J. Clancey and E.H. Shortliffe (Eds.), *Readings in Medical Artificial Intelligence*, Reading, Massachusetts: Addison-Wesley Publishing Company, pp.l 275–319.

Gomez, F., and Chandrasekaran, B. (1982). Knowledge organization and distribution for medical diagnosis. In W.J. Clancey and E.H. Shortliffe (Eds.), *Readings in Medical Artificial Intelligence*, pp. 320–338. Reading, Massachusetts: Addison-Wesley Publishing Company, 1984. (Originally published in *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 1, pp. 34–42 (1981).)

Hammond, K.J. (1986c). Case-based planning: An integrated theory of planning, learning, and memory. (PhD dissertation) Yale University Department of Computer Science technical report YALE/CSD/ RR#448.

Kolodner, J.L. (1982). The role of experience in development of expertise. In *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, PA, pp. 273–277.

Kolodner, J.L. (1983). Towards an understanding of the role of experience in the evolution from novice to expert, *International Journal of Man–Machine Studies*, vol. 19, pp. 497–518.

Kolodner, J.L. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey.

Kolodner, J.L. (1985). Experiential processes in natural problem solving. Technical Report #GIT-ICS-85/123, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia.

Kolodner, J.L. (1987). Capitalizing on failure through case-based inference. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 715–726.

Kolodner, J.L., and Kolodner, R.M. (1987). Using experience in clinical problem solving: Introduction and framework. In *Proceedings of the 1987 IEEE International Conference on Systems, Man, and Cybernetics.*

Kolodner, J.L., Simpson, R.L., and Sycara, K. (1985). A process model of case-based reasoning in problem solving. In *Proceedings of IJCAI-85*.

Lesgold, A.M., Feltovich, P.J., Glaser, R., and Wang, Y. (1981). The acquisition of perceptual diagnostic skill in radiology. Technical report No. PDS-1, University of Pittsburgh Learning Research and Development Center.

McDermott, D. (1978). Planning and acting, *Cognitive Science*, vol. 2, pp. 71–109.

Schank, R.C. (1982). *Dynamic Memory*, Cambridge University Press, New York.

Miller, R.A., Pople, H.E., Jr., and Myers, J.D. (1982). INTERNIST-1, an experimental computer-based diagnostic consultant for general internal medicine, *New England Journal of Medicine*, vol. 307, pp. 468–476.

Schank, R.C., and Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Shinn, H., and Kolodner, J.L. (1988). Abstractional analogy: A general paradigm of analogical transfer, submitted to the Seventh National Conference on Artificial Intelligence.

Simpson, R.L Jr. (1985). *A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation.* Doctoral dissertation, Technical Report #GIT-ICS-85/23, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.

Sycara K., (1987). "Adversarial Reasoning in Conflict Resolution", Ph.D. dissertation, Georgia Institute of Technology, Atlanta, Georgia, 30332.

Turner, R.M. (1987a). *Issues in the Design of Advisory Systems: the Consumer-Advisor System*, Technical Report #GIT-ICS-87/19, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.

Turner, R.M. (1987b). Modifying previously-used plans to fit new situations. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, Washington.

Opportunistic Use of Schemata for Medical Diagnosis[1]

Roy M. Turner
(royt@gatech.edu)

School of ICS
Georgia Institute of Technology
Atlanta, Georgia 30332

Abstract

Medical diagnosis can be considered to be a kind of planning task in which the goals are such things as "interpret a finding" and "evaluate a hypothesis," and in which the operators are such things as asking for information and drawing inferences. However, diagnosis is unlike many typical planning tasks, planning and plan execution proceed simultaneously. New information, arising as a result of an action taken by the reasoner, may impact the reasoner's future behavior. To cope with this, the diagnostician must be able to respond to changes in its environment as they occur: it must be *opportunistic.*

In this paper, we describe an approach to opportunistic reasoning in medical diagnosis. Our approach, called *schema-based reasoning*, uses packets of procedural knowledge—schemata—to direct the reasoner to solve goals as they arise during problem solving. Several schemata can be active at once, and the reasoner can switch between using them as the situation demands. The reasoner selects which schema to follow at any given time by using information about the type of consultation it is performing, and by using strategies represented as *strategic schemata.* Our approach is implemented in the MEDIC program, a schema-based diagnostic reasoner whose domain is pulmonology.

[1]This research has been funded in part by NSF Grants IST-831771 and IST-8608362 and grant DTD 09-25-87 from the Lockheed AI Center.

Opportunistic Use of Schemata for Medical Diagnosis*

Roy M. Turner
School of ICS
Georgia Institute of Technology

Medical diagnosis can be considered a planning task. This is not the traditional view, however. For example, Gomez and Chandrasekaran [Gomez & Chandrasekaran, 1982] and others view diagnosis as a classification task: a problem, consisting of a set of signs and symptoms, is classified as being an instance of a disease or set of diseases. However, this viewpoint overlooks the fact that actions are performed in order to classify a disease: in other words, planning and plan execution must be done as part of the classification process. When viewed as planning, goals in diagnosis are such things as "diagnose the patient," "interpret a finding," and "evaluate a hypothesis." Operators, at the lowest level, are such things as asking questions, requesting tests be performed, and making inferences based on information known about the patient and the reasoner's general knowledge of the domain.

Medical diagnosis is unlike many traditional planning tasks in that an initial, complete statement of the problem is generally impossible. Instead, the diagnostician must gather information about the problem as part of the process of performing diagnosis. The result of this is that the diagnostician cannot formulate a plan for diagnosis, then carry it out: the problem statement would change as the plan for performing diagnosis is executed. The effect of executing one step (e.g., asking a question) would likely alter the assumptions upon which later steps are based (e.g., a new finding might radically alter the diseases considered as diagnoses, or might suggest specialized methods for interpreting the finding). The problem for a diagnostician, then, is to be able to interleave planning and execution (cf. [McDermott, 1978]) so as to make use of new information as it becomes available. In other words, a diagnostician should be *opportunistic*.

Our approach to this problem makes use of packets of procedural information called *schemata*, which are retrieved from memory in response to goals arising from changes in the problem solver's environment: e.g., new findings, new hypotheses, etc. Most schemata can achieve very specific goals, such as "interpret a finding" or "evaluate a hypothesis"; others control larger parts of the reasoner's processing, such as directing the reasoner in the overall consultation. Schemata are flexible enough to encode several variations of how to achieve their goal; in addition, specializations of schemata provide the reasoner with information about how to satisfy specific goals or goals arising in specific contexts.

When a goal arises that can be achieved by a schema, that schema is retrieved from memory and made active. As the reasoner may have many goals simultaneously, there may be many active schemata at any time. The reasoner must decide which goal to focus on, and hence, which schema to *apply*. In our approach, the reasoner uses information from two sources to help it focus its attention. One source is from memory structures representing generalized consultations similar to the current problem. Information from these generalized consultations, such as information about which findings are generally important in this context, can be used by the reasoner to help it select a goal to achieve. The second source is from packets of procedural knowledge, called *strategic schemata*, which contain generally useful strategies in the form of goal orderings: e.g., a medical reasoner would have strategies for performing hypothetico-deductive reasoning, reasoning under time pressure, etc.

In this paper, we discuss our approach to opportunism in medical diagnosis. Our approach is called *schema-based reasoning*. Our ideas are being tested in MEDIC [Turner, 1988], a schema-based diagnostic reasoner whose domain is pulmonology.

*This research has been funded in part by NSF Grants IST-831771 and IST-8608362 and grant DTD 09-25-87 from the Lockheed AI Center.

OPPORTUNISM USING SCHEMATA

Opportunism involves responding to changes in the task environment as they arise during problem solving. There are at least three capabilities a reasoner must have in order for it to respond to changes; it must be able:

1. to interleave planning to achieve a goal and execution of that plan;
2. to respond immediately to new information appearing in the environment; and
3. to select the appropriate goal to pursue at each point in problem solving—that is, it must be able to focus its attention.

Traditional planners do not interleave planning and execution. Instead, the planner formulates a plan, then applies it. On the other hand, rule-based problem solvers and purely reactive planners such as PENGI [Agre & Chapman, 1987] do not really perform planning *per se*. The problem with these approaches is that there is very little coherence in their actions; consequently, their behavior may seem strange and unintuitive to a user. This presents a problem, especially in a medical domain, since a user is unlikely to accept a system if he or she cannot understand its reasoning.

A middle ground is needed between traditional planners and purely reactive planning. In our approach, problem solving is carried out by retrieving packets of procedural knowledge from memory, then applying them. These packets, or *schemata*, can be thought of as small plans or pieces of plans that achieve a goal; for instance, a reasoner may have a schema which can interpret a finding or one that can evaluate the likelihood of a hypothesis that a particular disease is present. Figure 1 shows a simplified view of a schema for interpreting a finding of dyspnea.[1] A schema contains *steps* to be performed by the reasoner in order to satisfy a particular goal.

Our approach is more flexible than traditional planning for three reasons. First, the order of the steps of a schema is not completely fixed ahead of time, but rather depends, to some extent, on the situation at the time of schema execution. For example, the step labeled "S1" contains information that allows the reasoner to select the next step based on the answer to the question asked in S1.

[1]Shortness of breath.

```
Goal: interpret a finding of dyspnea
Patient: any patient
Findings: dyspnea
Preconditions: there is a finding of dyspnea
Steps:
    S1:  action: ask how many stairs patient can climb
         goal: determine severity of dyspnea
         next:
              if pt. can climb flight of stairs ⟹ S3
              else ⟹ S2
    S2:  action: ask how far patient can walk
         goal: determine severity of dyspnea
         next: S3
    S3:  action: estimate the severity of the dyspnea
         goal: determine severity of dyspnea
                  ⋮
    S10: action: postulate hypotheses of pulmonary disease,
         cardiac disease
         goal: explain dyspnea
         next: done
Indices:
    patient/PATIENT1 → SCENE2
              ⋮
```

Figure 1: sc-dyspnea—a schema for interpreting a finding of dyspnea.

The second source of flexibility in our approach is also due to the nature of a schema's steps. In addition to specifying actions that should be taken—either primitive actions or other schemata—a step in a schema usually specifies the goal that the step is to satisfy. If the step fails, the reasoner can attempt to find another way of satisfying the step at run time. In addition, a step does not necessarily specify an action. Instead, it can specify *only* a goal, thus forcing the reasoner to attempt to satisfy the goal at run time.

The third reason our approach is flexible is that a single plan is not formulated for all of the goals in a problem, then executed. Instead, individual schemata are retrieved and applied to satisfy goals. As the situation changes, which schemata are active will also change. For example, as new goals arise during problem solving, new schemata can be found and activated to satisfy them.

In order to exhibit opportunism, a reasoner must be able to notice and respond to new information as it becomes available. In the context of a diagnosis program, new information comes from the user; information may be volunteered, or it may come from answers to questions asked by the system. In either case, when new information becomes available, the reasoner should interrupt what it is doing and incorporate the information into what it knows; the new information may also cause the reasoner to alter the course of its problem-solving behavior.

In our approach, all actions are governed by information contained in schemata; thus, to handle a piece of information, a schema must be found and activated. When a new item of information is encountered, the reasoner interrupts what it is doing and looks for a schema that can satisfy the goal created by the occurrence of the information: e.g., if the information is a finding, the goal will be to interpret it. When a schema is found, it is activated. The reasoner can then either return to what it was doing (i.e., to the schema it was applying), or it can choose, based on the altered state of the environment, to pursue a different goal (i.e., to apply a different schema—perhaps the one just activated).

There may be many goals present for a given diagnostic problem: e.g., goals to interpret findings, evaluate hypotheses, and to produce a diagnosis. Each of these will lead to one or more schemata being activated. In addition, information learned during diagnosis will result in more schemata being activated, as discussed above. Once we allow the reasoner to have more than one schema active at a time, we face the need for ability (3) above: the reasoner must be able to select the appropriate schema to apply at each point in problem solving;[2] i.e., the reasoner must decide on which goal to focus its attention.

This is not an easy task, since the importance of a goal varies with context. For example, in one situation, a goal to explain a severe rash may be quite important; however, in another situation involving both a severe rash and hemoptysis,[3] the goal of explaining the hemoptysis should take precedence over the goal of explaining the rash.

One approach to this problem is to use knowledge about the type of consultation the reasoner is performing. This information, in our approach, is present in memory structures called *diagnostic memory organization packets*, or dxMOPs (cf. [Schank, 1982], [Kolodner, 1985], and the diagnostic categories of Kolodner and Kolodner [1987]). These structures participate in memory organization, and provide one way for the reasoner to retrieve schemata from memory (see [Turner, 1988] for details). Each dxMOP represents a particular class of consultations, and provides information about goals, actions, etc., that can be expected in such a consultation. Information from a dxMOP can be used to decide which goal—and hence, which schema—to pursue. For example, suppose the dxMOP is the one shown in Figure 2; this dxMOP represents consultations involving alcoholics whose chief complaint is dyspnea. Among the expected findings are anemia and ataxia,[4] which are both explained by alcoholism. If the reasoner discovers that the patient indeed has anemia, it would not need to follow it up, since the finding is anticipated by the dxMOP and marked with a low importance.

```
Goal: diagnose the patient
Patient: an alcoholic patient
Chief complaint: dyspnea
Findings:
    anemia: low importance, explained by
            alcoholism
    ataxia: low importance, explained by
            alcoholism
        :
Hypotheses: TB, sarcoid, generalized pulmonary or heart
            disease
Schemata:
    sc-consult: for goal of diagnosing patient
    sc-finding: for generic findings
    sc-TB: for evaluating TB
        :
Indices:
    finding/mass-on-X-ray → dxMOP3
        :
    patient/PATIENT4 → dxMOP2
        :
```

Figure 2: A dxMOP for consultations involving alcoholics with dyspnea.

Information contained in a dxMOP can help the reasoner order goals; but what if there is no such information present in the dxMOP retrieved from memory? Or what if there are constraints associated with the current problem that are not anticipated in the dxMOP, such as time being limited?

In order to focus the reasoner's attention in this type of situation, we use the idea of meta-reasoning [e.g., Davis & Buchanan, 1984]: reasoning that takes place to guide planning. In our approach, meta-reasoning information is present in the form of strategic knowledge structures called *strategic schemata*. A strategic schema is a packet of information which represents a strategy for the reasoner's behavior; since the reasoner's behavior is determined by which schemata it chooses to apply, a strategy is equivalent

[2]We do not allow schemata to be applied in parallel. This is because we are trying, as far as possible, to model the behavior of human diagnosticians, who generally act as though they are thinking of one thing at a time. There are two reasons for modeling humans: (1) if the reasoner behaves similarly to a human, then the user is more likely to understand its reasoning, and hence, accept it; and (2) reasoning in a manner similar to that of a human should make explanations easier (though we do not currently address explanation).

[3]Blood in the sputum.

[4]Poor motor coordination.

```
Situation: any
Goal ordering:
      select goals related to hypotheses
      select goals related to findings
      select goals for gathering information
      select goal for forming diagnosis
```

Figure 3: A simple strategy for hypothetico-deductive reasoning style.

```
Situation: time is short
Goal ordering:
      select goal related to chief complaint
      select goals related to hypotheses
      select finding goal only if very important
      select goals of forming diagnosis
```

Figure 4: A simple strategy for reasoning under time pressure.

to an ordering of the schemata applied. Strategic schemata are useful for representing general strategies such as hypothetico-deductive reasoning or reasoning under time pressure. The reasoner can then use information, if available, from a dxMOP to modify its use of these strategies for a specific situation.

A simple example of a strategic schema is shown in Figure 3. This schema provides a goal ordering for the reasoner that induces a crude form of hypothetico-deductive reasoning: select any goals (i.e., select their corresponding schemata) that relate to hypotheses first; if there are none, then select goals related to findings in the hope of producing hypotheses; if none, then select goal of gathering information from the user; and finally, if that cannot be done, select the goal of forming a diagnosis. Figure 4 shows a simple strategic schema for reasoning under time pressure: follow up the chief complaint, if possible; evaluate hypotheses; only select goals related to findings if the findings are very important; and finally, when all else fails, form a diagnosis.

The overall algorithm for schema-based reasoning, including opportunism, is shown in Figure 5. Note that the reasoner can be interrupted; these interruptions can include interruptions both by the user and by schema application, as new information is added to STM.

```
begin
    loop forever:
        Wait until user requests a consultation;
        Add goal of diagnosing patient to short-term
                memory;
        Retrieve dxMOP using goal;
        Use strategy from dxMOP, if possible;
        Select a schema from the dxMOP to satisfy
                goal, add it to agenda;
        loop until done:
            Select a schema from agenda using strategies,
                local information in the dxMOP;
            Apply one action;
            if there was an interruption then:
                Handle interruption;
            fi;
            Specialize current dxMOP;
            if specialization succeeded then:
                Set current dxMOP to be the
                        specialization;
            fi;
        end loop;
        Accept and process feedback;
        Update memory;
    end loop;
end.
```

Figure 5: Basic schema-based reasoning algorithm.

MEDIC

Our approach to diagnostic reasoning is being implemented in the MEDIC program, a schema-based reasoner which performs diagnosis in the domain of pulmonology. MEDIC consists of three major modules: a long-term memory, which is organized as described in [Turner, 1988]; a short-term memory (STM); and a schema-based reasoner, which is directed at all times by schemata.

Conceptually, there are three types of schemata in MEDIC's memory: global, local, and strategic schemata. A *global schema* is one that directs a major portion of a consultation; an example is the schema which contains information that the reasoner can use to conduct the consultation: ask for a patient description, ask about the chief complaint, gather information, then form a diagnosis. Gathering information and forming a diagnosis are also directed by global schemata. A *local schema* is one which directs the reasoner in achieving very specific goals: e.g., interpret a finding of dyspnea or evaluate the hypothesis of lung cancer. Strategic schemata represent general reasoning strategies and are described above.

Currently, MEDIC can diagnose very simple cases of pulmonary disease. MEDIC follows an algorithm very similar to that in Figure 5. Let's look at an example of a consultation with MEDIC, a portion of which is shown in Figure 6. Suppose a user requests a consultation. The reasoner looks in memory for a way of satisfying the goal of diagnosing a patient and finds a dxMOP, "dx-consult", representing how consultations are generally conducted. The reasoner then uses this dxMOP as a context for diagnosis: it is used as

```
Please describe the patient.
: (patient (sex female) (weight (value 204)) (height
            (value 64)) (race white))
Adding information about patient to STM.
What is the chief complaint?
: (finding (entity (dyspnea (duration (years 2))
            (character progressive))))
Adding chief complaint to STM...adding finding of
            <DYSPNEA0> to STM.
How many flights of stairs can the patient climb?
: (less-than 1)
How far can the patient walk on level ground?
: (yards 20)
I judge the qualitative value of SEVERITY of <DYSPNEA0>
            to be SEVERE.
            ...(same for cardiac disease)...
...explaining dyspnea...
Processing <HYPOTHESIS0> [pulmonary disease];
            relating to other hypotheses...
...generating expectations given <HYPOTHESIS0>...
...I'm scoring hypothesis <HYPOTHESIS0> (<PULM-DZ0>)
...hypothesis explains: (<FINDING0>) [dyspnea]...
...failed predictions for hypothesis: —
...hypothesis doesn't explain: —
...trying to specialize the hypothesis of
            <HYPOTHESIS0> (<PULM-DZ0>)...
...specialized <HYPOTHESIS0> to <HYPOTHESIS1>
            (<RPE>) [recurrent pulmonary embolism]
...generating expectations given <HYPOTHESIS1>...
            ...
Is there a finding of <SYNCOPE>?
: Yes
            ...
Enter information (<return> if no more).
:
            ...
My diagnosis is: Recurrent pulmonary embolism.
```

Figure 6: Part of a consultation with MEDIC.

a source both of a strategy and of schemata to satisfy active goals. The strategy it contains is "st-HD-reasoning", the strategic schema mentioned above which provides a goal ordering to induce hypothetico-deductive reasoning. The only goal active is one to diagnose the patient; the schema to achieve this in dx-consult is "sc-consult". This is added to the reasoner's agenda of active schemata.

The reasoner now selects a schema from its agenda, using the goal ordering provided by the current strategy; the use of specific information from the dxMOP is not currently implemented. The only schema to select is sc-consult, so the reasoner selects that and begins to apply it. The user is asked for some initial information about the patient, including a description of the patient (a white female who is overweight)[5] and the chief complaint (progressive dyspnea). The information is added to STM. Adding the chief complaint causes the reasoner to be interrupted, and it searches memory for a schema to interpret the finding. Schema "sc-dyspnea" is found and activated.

[5]Input to MEDIC is in a version of Conceptual Dependency [Schank and Abelson, 1977]; there is currently no natural language interface.

Since the strategy in use dictates that goals related to findings have precedence over goals for gathering information or forming a diagnosis, sc-dyspnea is selected and used. This schema is a specialized version of a general schema to interpret findings; instead of asking general questions, the schema can ask very specific things related to dyspnea (e.g., asking how many stairs the patient can climb as a measure of the severity). The last step of this schema is to explain the finding by postulating diseases that could cause it; using this step, the reasoner postulates hypotheses of pulmonary disease and cardiac disease. Adding these hypotheses to STM again interrupts the reasoner, which finds and adds to the agenda schemata to evaluate the hypotheses: "sc-pulmDz" and "sc-cardiacDz".

The strategy orders goals related to hypotheses before any others; hence, one of the two schemata just added is selected, in this case, sc-pulmDz. The reasoner uses this schema to score the hypothesis of pulmonary disease,[6] and then tries to specialize the hypothesis using information that is in STM. One possible specialization, based on the fact that the patient is overweight, is recurrent pulmonary embolism[7] (RPE); this is hypothesized, resulting in a schema ("sc-RPE") being activated to evaluate it.

The reasoner then selects sc-RPE and begins to evaluate the hypothesis of pulmonary embolism. Eventually, it will have evaluated all the hypotheses that it can and will have exhausted the information the user can give it. The main schema, sc-consult, will then suggest the step of forming a diagnosis, which will be attempted.[8] In this case, the best hypothesis is recurrent pulmonary embolism, and that will be proposed to the user.

There is still much work to be done on MEDIC. At the present, the program has very little domain knowledge, and relatively few schemata. Additional thought must also be given to the form and content of the strategic schemata, which are currently quite simple; eventually, we would like for them to specify actions for the reasoner to perform in order to select a goal to pursue, making them more like the reasoner's other schemata. MEDIC also does not make use of context-specific information in dxMOPs to focus its attention.

[6]Using a scoring scheme very similar to that of INTERNIST-1 [Miller *et al.*, 1982].

[7]Blood clots occurring in the lungs.

[8]Again, using a method similar to that of INTERNIST-1.

RELATED WORK

Opportunistic reasoning has been addressed in the blackboard approach to problem solving of HEARSAY-II [Erman *et al.*, 1980] and OPM [Hayes-Roth, 1985]. These program's *Knowledge Sources* (KS's) are atomic, rule-like specialists that are invoked in response to some arbitrary condition occurring. They correspond only loosely to schemata. Schemata are larger-grained than KS's, and capable of being interrupted. Schemata also serve to cluster actions to be taken to achieve a goal; many KS's, on the other hand, may be needed to achieve a single goal. The use of schemata should allow the reasoner to behave in a manner that a reasoner can understand: e.g., question-asking should be more focused. Schemata should also facilitate explanation, since the actions taken to achieve a goal, though possible temporally disjoint, can still be explained in relation to one another.

The VISIONS Schema System [Weymouth, 1986] uses an approach similar to ours for interpreting visual scenes. Their schemata are specialists in particular vision tasks and can work in parallel to interpret a scene. Unlike our schemata, theirs are largely represented using procedures written in a programming language; it is therefore not possible for their program to reason about or modify their schemata, as is potentially possible using our representation of schemata. In addition, parallel execution of schemata is not feasible in our domain, given the goals of focused question-asking and modeling a human diagnostician's behavior.

The NASL [McDermott, 1978] program concentrated on the interaction of planning and execution. In many respects, our schemata are similar to NASL's tasks: both are hierarchical, bottoming out at the primitive action or primitive task level. However, our schemata are somewhat more flexible than NASL's tasks, and we make explicit use of goals; the latter allows the potential of specifying the goal of a task without necessarily specifying the steps to achieve it, thus allowing the reasoner to make such decisions at run-time.

We face some of the same problems as did NASL, too, in the chore of selecting which schemata to pursue at each point in problem solving. NASL made use of choice rules, which contained the strategic knowledge of that system. Our strategic schemata can be viewed in the current implementation as packages of such choice rules. However, the ultimate goal is to make them less rule-like and more schema-like, specifying steps for the reasoner to perform in order to select schemata to apply.

Firby [1987] is also concerned with interleaving planning and execution in environments that change during planning. The behavior of his RAP planner is quite similar to that of our reasoner. However, RAPs would seem to be somewhat more simple than our schemata, and oriented towards real-time control rather than diagnosis. In addition, we extend the idea of using packets of control knowledge to the meta-level by using strategic schemata to direct the reasoner's attention.

CONCLUSION

Medical diagnosis can be fruitfully viewed as a planning task in which planning is interleaved with diagnosis. New information may be discovered during diagnosis which should impact the future problem-solving behavior of the diagnostician. The diagnostician must be *opportunistic* in order to take notice of and respond to this new information as it becomes available.

Schema-based reasoning is one approach to this problem. By representing problem-solving knowledge as packets of procedural information designed to achieve a goal, the reasoner can activate schemata as goals for dealing with changes in the environment arise. By storing schemata in a memory based on the goals they achieve and the situations in which they are useful, the reasoner can find the appropriate schemata for goals as they arise.

Schemata are flexible, and enhance the reasoner's ability to respond to changes in the environment in two ways: (1) the order of their steps need not be completely determined—this allows the reasoner to select the next step of a schema based on the state of the world resulting from the application of the previous step; and (2) steps may specify goals, which the reasoner can attempt to satisfy at run-time by retrieving schema specific to the current situation.

Schemata are selected for application based on the reasoner's focus of attention—i.e., the goal the reasoner is trying to achieve. Goals are selected by the reasoner based on information from two sources: general goal-ordering information, stored in strategic schemata; and specific goal-ordering information, stored in the dxMOP representing consultations similar to the current one.

Though this research addresses medical diagnosis, we believe that schema-based reasoning can be usefully applied to other tasks. Our approach should be useful for any task in which planning and execution must be interleaved, or in which all features of the problem cannot be known at the start of the problem.

ACKNOWLEDGMENTS

Many thanks to Janet Kolodner, Hong Shinn, and Elise Turner for their comments on earlier drafts of this paper, and to our domain expert, Eric Honig, of Emory University and Grady Memorial Hospital.

REFERENCES

Agre, P.E., and Chapman, D. (1987). Pengi: An implementation of a theory of activity, in *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 268–272.

Cullingford, R.E., and Kolodner, J.L. (1986). Interactive advice giving. In *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics.*

Davis, R., and Buchanan, B.G. (1984). Meta-leval knowledge. In B.G. Buchanan and E.H. Shortliffe (eds.), *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley Publishing Company, Reading, Massachusetts, pp. 507–530.

Erman, L.D., Hayes-Roth, F., Lesser, V.R., & Reddy, D.R. (1980). Then HEARSAY-II speech-understanding system: Integrating knowledge to resolve uncertainty, *Computing Surveys*, Vol. 12, No. 2.

Firby, R.J. (1987). An investigation into reactive planning in complex domains, in *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 202–206.

Gomez, F., and Chandrasekaran, B. (1982). Knowledge organization and distribution for medical diagnosis. In W.J. Clancey and E.H. Shortliffe (Eds.), *Readings in Medical Artificial Intelligence*, pp. 320–338. Reading, Massachusetts: Addison-Wesley Publishing Company, 1984. (Originally published in *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 1, pp. 34–42 (1981).)

Hayes-Roth, B. (1985). A blackboard architecture for control, *Artificial Intelligence*, Vol. 26, No. 3, pp. 251–321.

Kolodner, J.L. (1985). Experiential processes in natural problem solving. Technical Report #GIT-ICS-85/123, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia.

Kolodner, J.L., and Kolodner, R.M. (1987). Using experience in clinical problem solving: Introduction and framework. In *Proceedings of the 1987 IEEE International Conference on Systems, Man, and Cybernetics.*

McDermott, D. (1978). Planning and acting, *Cognitive Science*, vol. 2, pp. 71–109.

Miller, R.A., Pople, H.E., Jr., and Myers, J.D. (1982). INTERNIST-1, an experimental computer-based diagnostic consultant for general internal medicine, *New England Journal of Medicine*, vol. 307, pp. 468–476.

Schank, R.C. (1982). *Dynamic Memory*, Cambridge University Press, New York.

Schank, R.C., and Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, Hillsdale, NJ.

Turner, R.M. (1988). Organizing and using schematic knowledge for medical diagnosis. Submitted to the Seventh National Conference on Artificial Intelligence.

Weymouth, T.E. (1986). *Using Object Descriptions in a Schema Network for Machine Vision*, Technical Report 86–24 (Ph.D. thesis) Dept. of Computer and Information Science, Univ. of Massachusetts.

Towards an Architecture for Open World Problem Solving*

Thomas R. Hinrichs
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

Abstract

Problem solving in open worlds involves the management of inconsistency, imprecision, and lack of knowledge. In this paper we examine some specific problems that arise in open worlds and describe inferences that are needed to deal with them. We present a problem solving architecture that integrates case-based reasoning and constraint propagation to achieve flexibility in open domains. This architecture is implemented in a computer program called JULIA.

1 Introduction

Problem solvers must often work with incomplete or inconsistent knowledge, either because the domain is unbounded, continually changing, or not well understood. Two general techniques for dealing with such situations are: 1) to postpone commitment until more information is available, and 2) to make an assumption, or educated guess, about the missing information. Individually, these techniques may be insufficiently flexible: If a problem is under-constrained, delayed commitment by itself may never solve it. On the other hand, a problem solver that always leaps to conclusions may fail to exploit information that arrives late. To remedy these problems, we are exploring an approach that integrates both methods in the form of constraint propagation and case-based reasoning.

*This research was funded in part by NSF Grant No. IST-8608362, in part by ARO contract No. DAAG-29-85-K-0023, and in part by Lockheed Grant No. DTD09-25-87

Case-based reasoning involves recalling previous problem solving episodes, or *cases*, that are similar to the current situation and adapting parts of those cases to fit the new problem. In this paper, we describe a problem solving architecture which integrates case-based reasoning with constraint propagation in order to support reasoning in open worlds. We have implemented this architecture in a program called JULIA [Cullingford *et al.* 1986], which is an interactive catering advisor that helps users to plan meals.

In the following section, we use the meal planning domain to illustrate some problems that arise in open worlds and techniques that address them. Section 3 describes the problem solving process with an extended example and section 4 presents a layered architecture which implements this process.

2 Open Worlds

An open world is any domain for which a problem solver has incomplete or inconsistent knowledge. Typical situations in which open worlds arise are: 1) interactive problem solving, 2) under-constrained problems, 3) incomplete domain theories, and 4) problems of ill-defined scope. In this section, we will discuss each of these situations and show how they involve reasoning with incomplete knowledge.

2.1 Interactive Problem Solving

Problem specifications are often incomplete. Many problem solvers cope with this by asking questions and accepting advice from a user. To be effective, such interactive problem solvers must meet two criteria: they must be reactive and they must reason opportunistically. A *reactive* problem solver is one which responds dynamically to changes in its environment [Kaelbling 1986]. An *opportunistic* problem solver exploits serendipitous features of the environment to satisfy multiple goals or constraints [Hayes-Roth *et al.* 1979]. These criteria are illustrated in the following hypothetical dialog:

Caterer:	How much do you want to spend?
Client:	Let's have something cheap like Mexican food. I'm on a diet though, so it should be a light meal.
Caterer:	How about a taco salad?

In this exchange, the client answers the original question, changes the focus of the conversation, and volunteers additional information. The problem solver, in turn, must react to the shift in focus and assimilate the new information opportunistically to derive a solution which satisfies the goals and constraints.

In addition, the requirements of interactive problem solving prohibit chronological backtracking. First, because the focus shifts dynamically, the search space cannot be explored hierarchically. Second, responsibility for decisions is shared between the system and the user, and the system must not unilaterally revoke the user's decisions. Therefore each decision must be individually justified to permit dependency-directed backtracking, and the problem solving architecture must include truth maintenance.

2.2 Under-Constrained Problems

Sometimes a problem has no single right answer, or even an optimal one. There may be many solutions that satisfy the given constraints; for example, there are usually many possible menus that will be acceptable for a given meal. How can a problem solver generate these potential solutions, and how should it choose among them?

Often, it is not practical to generate the complete search space. For instance, the class of all dishes is both too large and too poorly defined to enumerate. A better strategy is to generate a small subset of possibilities, use constraints to filter out unacceptable values, and choose among the remaining satisficing candidates.

One way to generate candidates is to recall them from previous cases [Kolodner *et al.* 1985]. If the problem solver is reminded of cases which were similar to the current one, this provides a limited set of candidate values among which to choose. Using case-based reasoning in this way amounts to a kind of early commitment which complements the delayed commitment of constraint propagation.

2.3 Incomplete Domain Theories

When a problem solver does not know all of the relationships that hold in a domain, it may make incorrect inferences or fail to make any inferences at all. Sometimes the only recourse is to make an assumption about what is probably true. For instance, a caterer can usually assume that money is important to a client and should be conserved. This may not be true, however, if the goal of the meal is to impress a guest. Determining when such assumptions apply is difficult when the domain theory is incomplete.

To complicate matters, these assumptions may change over time. In the short term, a problem solver must revise its assumptions as new information becomes available. For example, if a problem solver plans an inexpensive meal and later learns that the boss is coming to dinner, then it must retract the 'conserve money' assumption and its consequences, and make new assumptions about cost, ease of preparation, and formality. In other words, the problem solver must ensure that the plan is consistent with its assumptions. This consistency can be enforced by constraint propagation and truth maintenance.

Assumptions may also change over the long term, as a problem solver learns when an assumption does or does not apply. Such learning is simplified if the problem solver can recall previous cases, and if those cases contain feedback indicating why they succeeded or failed. For instance, a meal may fail because the planner neglected to provide for vegetarian guests. In the future, the planner should be reminded of this failure and try to determine whether or not there will be vegetarians present. Over time, this should be generalized to accomodate any special case eaters. Case-based reasoning of this sort can offset an incomplete domain theory by helping a problem solver to recognize implicit assumptions, thereby allowing it to anticipate and avoid failure [Hammond 1986,Kolodner 1987].

2.4 Ill-defined Problem Scope

Real world problems are seldom stated in terms of an initial state, a goal state and a set of operators. Usually, a problem solver must infer the nature and scope of the problem and the desired specificity of the solution. For instance, if a client says: "I'm having a party for my research group..." a catering advisor must first assume the existence of a meal based on its own role as caterer. Next, it must look at the context of this meal in order to propose relevant constraints, such as cost and formality. As the solution evolves, the problem solver must also decide how specific to be. For example, a dish might be specified as a salad or more particularly as a waldorf salad. The locale of a meal might be sufficiently specified with an address, or it might be necessary to indicate a particular room. The required level of specificity is seldom given explicitly, and must often be inferred from previous cases and constraints.

In addition to determining the specificity of the solution, the problem solver may also have to infer how specific its own operators should be. In particular, a case-based reasoner may be able to adapt and modify previous solutions at different levels of granularity. For example, situations such as Thanksgiving Dinner are sufficiently traditional that menus from previous cases may be adapted almost intact. More often, bits and pieces of cases can be mixed and matched to derive a satisficing solution. Sometimes, however, the repertoire of known dishes isn't quite sufficient. In this situation, it may be possible to modify recipes by substituting one ingredient for another to satisfy a constraint. Thus, the granularity of problem solving operators never really bottoms out. The adaptation of previous cases just becomes progressively more like reasoning 'from-scratch'.

3 Problem Solving in JULIA

JULIA solves open world problems by reasoning from previous cases and propagating constraints to refine a problem statement (see figure 1). We illustrate this process with a portion of an execution trace in which the problem is to plan a party for about 20 guests. First, the problem solver posts a goal to refine the problem and retrieves a plan to achieve it:

```
GOAL = REFINE (PROTOCOL3)
PLAN = REFINE-SOCIAL-OCCASION
```

The scope of this problem is ill-defined because it is not explicitly stated whether JULIA should plan the entire party, the meal by itself, or just the menu. The problem solver hedges initially by assuming that the problem is to plan a meal in the context of the party:

```
Assuming problem is to refine meal.
GOAL = REFINE (MEAL-1)
PLAN = REFINE-MEAL
```

This problem is under-constrained because there are an infinite number of possible solutions. Therefore, rather than immediately working on the menu, JULIA asks for information which could constrain the search:

How much do you want to spend?
-> *(cheap-meal Mexican-cuisine p-diet)*

The user's reply is a list of desired features: cheap-meal is a range of costs per person,

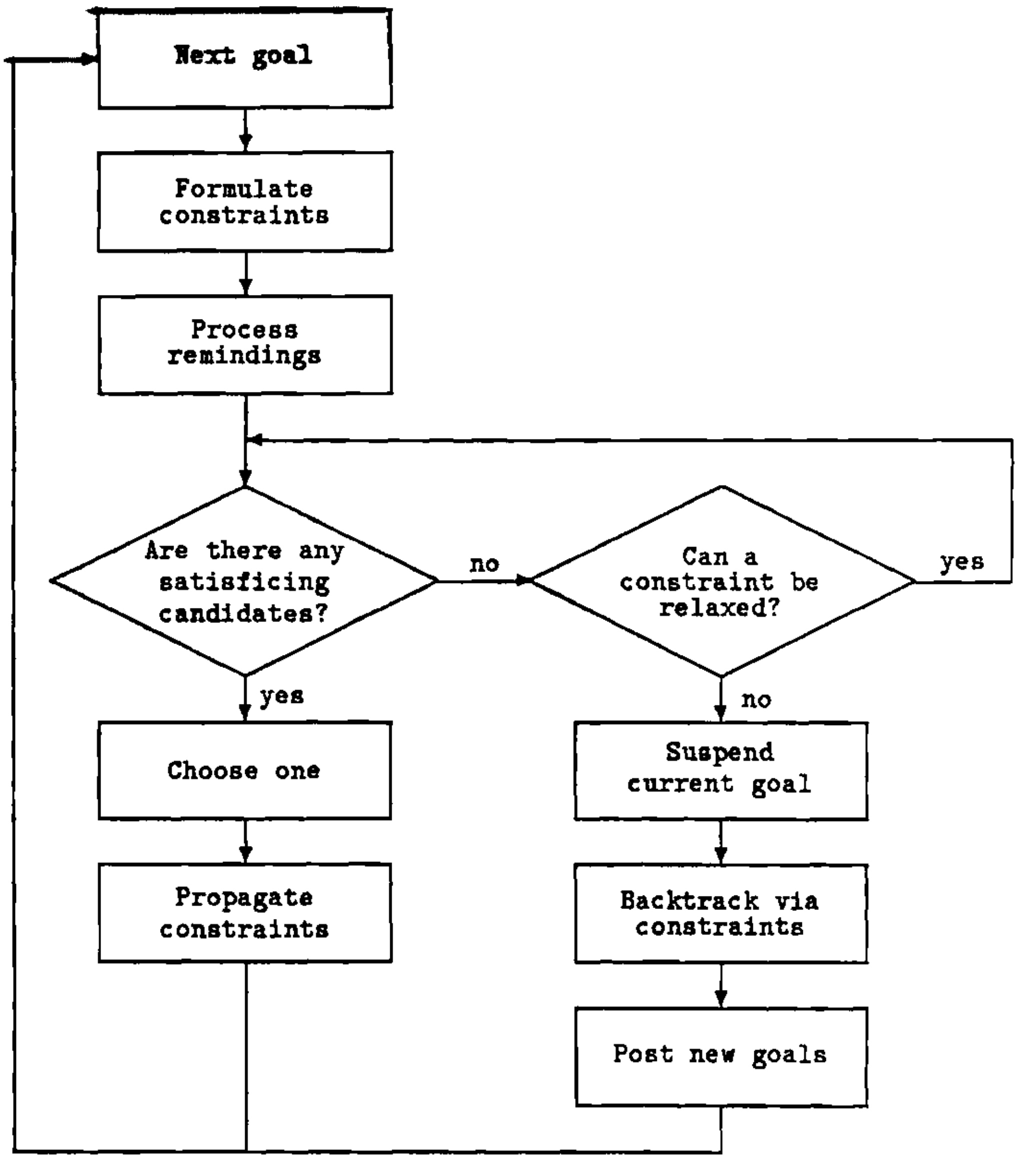

Figure 1: Abbreviated Problem Solving Algorithm

Mexican-cuisine is a description of a typical Mexican meal, and p-diet is a preservation goal. Therefore, the English translation would be "I would like a cheap, Mexican meal which is also low-calorie." Not only has the user answered the question, but he has also volunteered new information. In order to opportunistically use this information, the problem solver formulates and propagates constraints on the new features. When it does, a contradiction is found between the default meal structure (appetizer, salad, main course, and dessert), the structure of a typical Mexican meal (omit salad), and the structure of a typical diet meal (a single main course). JULIA assumes that p-diet, an explicit goal, is more important than Mexican-cuisine, a descriptor, and therefore chooses the single course:

```
Withdrawing plan step ?REFINE-APPETIZER
Withdrawing plan step ?REFINE-SALAD
Withdrawing plan step ?REFINE-DESSERT
```

Thus, JULIA reacts to the user's input by revising its own problem-solving plan as well as the meal plan. At this point the problem solver starts to look for a main dish. Although the problem is still under-constrained, there is enough information available for the case-based reasoner to retrieve similar cases:

```
GOAL = FIND-VALUE (MAIN-DISHES)
Reminded of case:  DECEMBER-MEAL
Reminded of case:  LO-CAL-DINNER
Reminded of case:  DEATH-CHILI-MEAL
```

```
I remember the DEATH-CHILI-MEAL which
failed because Tom didn't eat spicy
food.  Will this be a problem in the
current situation?
-> yes
```

To avoid repeating a previous mistake, the problem solver adds a constraint to reject any dish whose taste is spicy:

```
adding constraint:
(DIFFERENT (?DISHES TASTE) SPICY)
```

The case-based reasoner ranks the meals it remembers by similarity and suggests values from the most similar case. When it finds a value that satisfies the constraints, it suggests it to the user:

```
Would you like a taco-salad?
-> yes
```

If a taco-salad were unacceptable, the problem solver would try to relax constraints on other candidates, and failing that it would postpone this goal in the hope that other features would generate new remindings and suggest other candidates. The problem solver continues on from here to fill in other descriptors and dishes until the meal plan is complete.

4 Architecture

The preceeding example suggests some of the functions that are needed to reason with incomplete knowledge. Specifically, there must be some means of: 1) selecting a goal or focus of attention, 2) formulating, propagating, and relaxing constraints, 3) retrieving cases from memory, analyzing failures, and transferring values and constraints, and 4) modifying the problem structure and justifying individual decisions.

Ideally, these functions should be integrated as much as possible in order to minimize redundancy and maximize constructive interaction. At the same time, good software engineering practice dictates information hiding and modularity. These requirements are reconciled in JULIA with a layered architecture which consists of four modules, as shown in Figure 2.

The goal scheduler and TMS are fairly traditional. The goal scheduler is a problem reduction problem solver, and the TMS is a justification-based truth maintenance system based on [Doyle 1979]. In the next sections, we discuss the constraint propagator and the case-based reasoner, which form the core of the problem solving architecture.

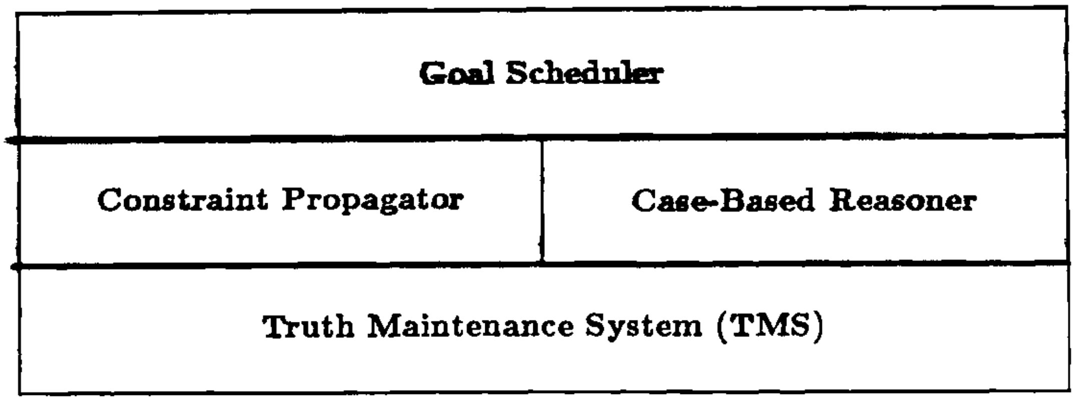

Figure 2: The Problem Solving Architecture

4.1 Constraint Propagator

The constraint propagator has two main functions. First, it evaluates and filters suggested values in a manner similar to Molgen [Stefik 1981]. Second, it propagates values and constraints through a network as in the constraint propagators of [Waltz 1972] and [Steele 1980].

Constraints in JULIA consist of a type, arguments, and an importance. The constraint type is a frame with slots containing a predicate and a generator function, as in ISIS [Fox 1983] and PRIDE [Mittal *et al.* 1986]. An argument may be either a constant or a path to a slot. The importance indicates whether or not the constraint may be relaxed. For example, the constraint to rule out spicy dishes looks like:

```
(different (?main-dishes taste) spicy
required)
```

where `different` is the constraint type, the first argument is a path, the second argument is a constant, and the importance is `required`.

Constraints reside under slots. This permits a constraint to be triggered when its slot receives a value, and it also determines the scope of the constraint. For instance, if the not-spicy constraint is stored under the `ACTIVITIES` slot of the meal, then all courses inherit it. If, on the other hand, it were stored under a subslot of this such as `MAIN-COURSE`, then only the main course would be required to be non-spicy.

4.2 Case-Based Reasoner

Two important elements of case-based reasoning are: 1) recalling similar cases in previous situations and 2) adapting parts of those cases to fit the new situation. In JULIA, the case-based reasoner retrieves previous cases from a dynamic memory [Schank 1982,Kolodner 1984]. It then ranks them in order of similarity to the current problem, weighing similar goals more heavily than similar descriptors. The reasoner suggests values from the most similar cases by constructing TMS nodes that package the values along with the reasons for and against them. The Constraint Propagator checks the suggested values and rules out those that violate constraints.

Another function of the case-based reasoner is failure avoidance. Previous cases contain feedback from the user in a slot called `Actual-Events`. The feedback is a sketchy causal chain which indicates 1) a *goal* which either succeeded or failed, 2) an *event* which resulted in the success or failure of the goal, and 3) a *reason* which is either a theme or a constraint which enabled (or disabled) the event. When the case-based reasoner detects a previous goal failure, it tries to determine whether or not it is relevant by comparing the default function of the object of the event with the current focus of attention. Thus, in the example in section 3 the failure event was that Tom didn't eat chili. The default function of chili is to serve as a main dish, so when the problem solver is looking for a main dish, this failure will be considered potentially relevant.

At this point, the current implementation of JULIA simply asks the user whether the failure is in fact relevant, and if so it transfers the constraint in the *reason* slot. A more sophisticated approach would be to analyze the reason itself to determine how the failure is relevant and how to best avoid it in the current situation.

5 Discussion

Although constraint propagation and case-based reasoning are not new in and of themselves, their integration provides a novel approach to problem solving in open worlds. In particular, it allows a problem solver to deal with incomplete information in five ways:

- The problem solver can react to a user or to a dynamic environment by triggering constraints when new information arrives.
- The problem solver can opportunistically satisfy multiple goals and constraints by combining the bottom-up inferences of constraint propagation with the top-down expectations of case-based reasoning and goal scheduling.
- The case-based reasoner can help reduce the search space when a problem is underconstrained by suggesting values from previous cases.
- The problem solver can deal with an incomplete domain theory by making plausible assumptions based on previous successes and failures.
- The case-based reasoner can help the problem solver to infer the scope of a problem by referring to previous similar cases.

These capabilities suggest several ways in which constraint propagation and case-based reasoning are complementary:

- **Commitment.** Constraint propagation is a form of delayed commitment; inferences are made as information arrives. Alternatively, case-based reasoning can be viewed as a kind of early commitment because it provides a way to make plausible assumptions about missing information.
- **Rate of adaptation.** Constraint propagation permits a problem solver to react immediately to new information. Reacting within a problem-solving session like this can be thought of as short-term adaptation. Case-based reasoning, on the other hand, reacts by altering behavior between sessions, thus adaptation is long-term.
- **Source of information** Information provided by one technique is used by the other: constraints index cases, and in turn, cases suggest additional constraints.

Because constraint propagation and case-based reasoning are complementary, their integration is a first step towards an architecture for open world problem solving.

Acknowledgements

I would like to thank Janet Kolodner for her guidance and encouragement during this research, and Craig Stanfill and Reid Simmons for their comments on an earlier draft of this paper.

References

[Cullingford *et al.* 1986] R.E. Cullingford and J.L. Kolodner. Interactive advice giving. In *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics*, 1986.

[Doyle 1979] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3), 1979.

[Fox 1983] M.S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. PhD thesis, CMU, 1983. CMU-RI-TR-83-22.

[Hammond 1986] K.J. Hammond. *Case-based Planning: An Integrated Theory of Planning, Learning and Memory.* PhD thesis, Yale, 1986. YALE/CSD/RR-488.

[Hayes-Roth *et al.* 1979] B. Hayes-Roth, and F. Hayes-Roth. A Cognitive Model of Planning. *Cognitive Science*, (3):275–310,1979.

[Kaelbling 1986] L.P. Kaelbling. An architecture for intelligent reactive systems. In M.P. Georgeff and A.L. Lansky, editors, *Proceedings of the Workshop on Reasoning about Actions and Plans*, pages 345–410, Morgan Kaufman, Los Altos, CA, July 1986.

[Kolodner 1984] J.L. Kolodner. *Retrieval and Organization Strategies in Conceptual Memory: A Computer Model.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1984.

[Kolodner *et al.* 1985] J.L. Kolodner, R.L. Simpson, and K. Sycara. A process model of case-based reasoning in problem solving. In *Proceedings of IJCAI-85*, pages 284–290, Los Angeles, 1985.

[Kolodner 1987] J.L. Kolodner. Capitalizing on failure through cased-based inference. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, Washington, July 1987.

[Mittal *et al.* 1986] S. Mittal and A. Araya. A knowledge based framework for design. In *Proceedings of AAAI-86*, pages 856–865, Philadelphia, PA, August 1986.

[Schank 1982] R.C. Schank. *Dynamic Memory: A theory of reminding and learning in computers and people.* Cambridge University Press, London, 1982.

[Steele 1980] G.L. Steele Jr. *The Definition and Implementation of a Computer Language based on Constraints.* PhD thesis, MIT, 1980. AI-TR-595.

[Stefik 1981] M.J. Stefik. Planning with constraints (molgen: part 1). *Artificial Intelligence*, 16(2):141–169, 1981.

[Waltz 1972] D. Waltz. *Generating Semantic Descriptions from Drawings of Scenes with Shadows.* Technical Report, MIT Artificial Intelligence Laboratory, 1972. AI-TR-271.