10:24:36 OCA PAD AMENDMENT - PROJECT HEADER INFORMATION 03/08/91

Active

Project #: C-36-611 Cost share #: Rev #: 6
Center # : R6688-0A0 Center shr #: OCA file #:
Work type : RES
Contract#: AGR DTD 890216 Mod #: 5 Document : AGR
Prime #: F49620-88-C-0058 Contract entity: GTRC

Subprojects ? : N
Main project #:

Project unit: COMPUTING Unit code: 02.010.300
Project director(s):
KOLODNER J L COMPUTING (404)894-3285

Sponsor/division names: COGNITIVE SYSTEMS INC / NEW HAVEN, CONN.
Sponsor/division codes: 202 / 066

Award period: 880401 to 910831 (performance) 910831 (reports)

Sponsor amount	New this change	Total to date
Contract value	0.00	669,291.00
Funded	0.00	669,291.00
Cost sharing amount		0.00

Does subcontracting plan apply ?: N

Title: CASE-BASED REASONING

PROJECT ADMINISTRATION DATA

OCA contact: E. Faith Gleason 894-4820

Sponsor technical contact

DR. ROGER SCHANK
(203)773-0726

COGNITIVE SYSTEMS

Sponsor issuing office

MS. RUTH A. NELSON
(203)773-0726

COGNITIVE SYSTEMS, INC.
234 CHURCH STREET
NEW HAVEN, CT 06510

Security class (U,C,S,TS) : U ONR resident rep. is ACO (Y/N): N
Defense priority rating : DO-C9 GOVT supplemental sheet
Equipment title vests with: Sponsor X GIT
HOWEVER, NO EQUIPMENT CAN BE PURCHASED WITHOUT SUBCONTRACT MODIFICATION.
Administrative comments -
* MOD DATED FEBRUARY 22, 1991 PROVIDES A NO-COST EXTENSION TO AUGUST 31, 1991.

GEORGIA INSTITUTE OF TECHNOLOGY
OFFICE OF CONTRACT ADMINISTRATION

NOTICE OF PROJECT CLOSEOUT

Closeout Notice Date 05/19/92

Project No. C-36-611 Center No. R6688-0A0

Project Director KOLODNER J L School/Lab COMPUTING

Sponsor COGNITIVE SYSTEMS INC/NEW HAVEN, CONN.

Contract/Grant No. AGR DTD 890216 Contract Entity GTRC

Prime Contract No. F49620-88-C-0058

Title CASE-BASED REASONING

Effective Completion Date 910831 (Performance) 910831 (Reports)

Closeout Actions Required:	Y/N	Date Submitted
Final Invoice or Copy of Final Invoice	Y	920229
Final Report of Inventions and/or Subcontracts	Y	
Government Property Inventory & Related Certificate	Y	
Classified Material Certificate	N	
Release and Assignment	Y	
Other	N	

Comments

Subproject Under Main Project No.

Continues Project No.

Distribution Required:

Project Director	Y
Administrative Network Representative	Y
GTRI Accounting/Grants and Contracts	Y
Procurement/Supply Services	Y
Research Property Managment	Y
Research Security Services	N
Reports Coordinator (OCA)	Y
GTRC	Y
Project File	Y
Other	N
	N

NOTE: Final Patent Questionnaire sent to PDPI.

G-36-611

Case-Based Reasoning at Georgia Tech Annual Report Sept., 1988 - March, 1989

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

April 17, 1989

During the past year, work on this project has been in 6 areas: case selection, index selection, cbr for design problem solving, general purpose adaptation heuristics, cbr for learning a scheduling problem, and development of a cbr shell.

1 Case Selection

The most important support process a case-based reasoner needs is a memory for cases. The memory must make cases accessible when 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. But most importantly, it must be able to select out the most appropriate cases for the case-based reasoner to use at any time.

While much work in the past has gone into organizing cases in a memory and retrieval algorithms for recalling them, little has gone into the problem

of choosing the best case from among the many partial matches that are retrieved. Our work, summarized in the attached paper by Janet Kolodner entitled "Selecting the Best Case for a Case-Based Reasoner," addresses this problem. It is implemented in a program called PARADYME (PARAllel DYnamic MEmory), which is designed to work alongside a case-based reasoner. PARADYME's memory is hierarchical, but there are no indexes in the memory. Instead a parallel retrieval mechanism (implemented on the Connection Machine) retrieves all cases that partially match the retrieval probe. In a second step, selection heuristics choose the best from among those partial matches.

PARADYME's selection method is based on many of the same principles guiding the selection of indexes and retrieval algorithms used in other case memories. However, it differs from those memories in several ways. First, PARADYME's parallel retrieval method allows us to do away with the restrictions indexes put on retrieval in other memory systems. Instead, what would have been indexes in those systems are found as annotations on cases called *salient feature sets.* Cases whose full salient feature set matches a retrieval probe are preferred over others, but cases without full sets of matching salient feature sets can also be retrieved. In this way, indexes are allowed to act as *selectors* rather than restrictors. PARADYME does not get hurt by the inability to predict every important part of a case at the time it happens. PARADYME prefers cases whose salient features (indexes) match the retrieval probe but if no cases with indexed features match, it will recall a case with other matching features.

Second, PARADYME's emphasis when ranking cases is on **usefulness.** Using this criterion for ranking means that PARADYME takes the reasoner's goals into account in selecting out a "best" case. Rather than choosing a most similar case, it chooses the most similar of those cases that are first judged most useful.

PARADYME's selection procedure is based on a set of *preference heuristics.* These heuristics are applied to the set of partially-matching cases to choose a small set of "best" cases. PARADYME uses six different types of preference for this task.

- Goal-Directed Preference
- Salient-Feature Preference

- Specificity Preference
- Frequency Preference
- Recency Preference
- Ease-of-Adaptation Preference

The first preference, *goal-directed preference* is based on the principle of utility. That is, since the memory is working in conjunction with a reasoner that has goals, it makes sense to prefer those cases that can help in achieving the problem solver's goals. Thus, when the problem solver is trying to come up with a main dish, those cases that match on main dish constraints will be preferred over others. When it is trying to evaluate the goodness of a solution, those cases that predict success or failure under similar circumstances are preferred. We state this heuristic as follows:

> **Goal-Directed Preference:** Prefer cases that can help address the reasoner's current reasoning goal, and of these, prefer those that share more constraints over those that share fewer.

The second preference heuristic, *salient-feature preference*, is based on the principle that we should use experience to tell us which features of a new situation are the ones to focus on. If memory has done a good job of recording its experiences, they can be used to tell us which features of previous events led to the choice of particular solutions or solution methods and which features of previous events were responsible for success or failure in those cases. These features are the *salient features* of previous cases, and in indexed memories, they form the indexes. When salient features of previous cases exist in a new situation, they can be used to suggest solutions and predict outcomes for the new case. A case where a friend named Anne didn't eat what was served for dinner, for example, has a salient feature set that predicts failure and includes the following facts: Anne was a guest, fish was served, preparation style of the fish was grilled. When all of these features are present in a probe, we can predict that Anne won't eat. PARADYME prefers cases that share full sets of salient features with the new problem over other cases whose full salient feature sets are not in the probe. We state this preference as follows:

Salient-Feature Preference: Prefer cases that match on salient features over those that match on other features, and prefer those that match on a larger subset of salient features over those matching on a smaller subset.

The third preference heuristic is based on the principle that a more specific match can be more predictive than a less specific match. Thus, all other things being equal, cases that match more specifically are preferred over less specific matches. PARADYME has several ways to judge specificity. First, according to PARADYME's definition of specificity, a case is more specific than another if the features that match in the less specific case are a proper subset of the features that match in the more specific case. Thus, a probe is more specifically matched by a case that matches all of its features than one that matches only a subset. Second, a case matches more specifically than one of its ancestors in memory's generalization hierarchy. For example, a particular Italian meal is more specific than a generic Italian meal. Third, a case matches more specifically if the probe matches features in more of its parts. The specificity preference follows:

Specificity Preference: Prefer cases that match more specifically over less specific matches.

The fourth and fifth heuristics are based on two principles psychologists have discovered – that items that are referenced more frequently are more likely to be recalled than other similar items and that items that have been referenced more recently are more likely to be recalled than other similar items (all else being equal). This gives rise to two preference heuristics:

Frequency Preference: Prefer cases that have been accessed more frequently over less frequently-accessed cases.
Recency Preference: Prefer cases that have been accessed more recently over less recently-accessed cases.

A sixth preference heuristic is also based on the principle of utility, and is specific to case-based reasoning. Some adaptations of previous solutions are easier to make than others. This heuristic says to prefer cases whose solutions are easier to adapt than those whose solutions are harder to adapt.

Ease-of-Adaptation Preference: Cases that match on features that are known to be hard to fix should be preferred over those that match on easy-to-fix features.

The application of preference heuristics is complicated. Each preference heuristic attempts to select out a set of better matches. When a heuristic does this, that set is sent on to the next heuristic for pruning. When no subset of cases is better than the rest using some heuristic, however, the entire set it was selecting from is selected. In this way, the preferences act as *selectors* rather than restrictors. We prefer to recall a case that can address the reasoner's current goal but we don't require it. We prefer to recall a case that matches on salient features, but if there are none, the preference heuristics allow recall of a case that matches on a random set of features.

The heuristics are also ordered. Goal-directed preference is applied first, then salience, then specificity, and then frequency and recency. This way, the set of cases that can be used to achieve the reasoner's current goal is selected out first, then any that match on a full set of salient features (of the right kind) are selected from those, the most specific of those are chosen (if some are more specific than others), and then the more frequently or recently recalled cases are selected from those.

Our current work is addressing the issue of interactions between these heuristics. We are implementing a testbed in which we can apply the heuristics varying their orderings and precedence, in order to judge how they ought to be applied as a group. The test system is working with cases from JULIA, our design problem solver that plans meals.

2 Index Selection

Our analysis of what goes into choosing a best case has led us to several conclusions about what kinds of indexes are useful ones for a case to have. We have found three kinds of indexes useful for problem solving. The first contain features that predict the applicability of some method for achieving a goal (*goal-achievement sets*). Second are those that predict the success of failure of a solution (*solution-evaluation sets*). Third are those that describe unusual outcomes (*outcome-achievement sets*).

Goal-Achievement Sets are generally conjunctions of goals, constraints on these goals, and problem and environmental features that predict the method or solution for achieving the goal or goal set. If the features of a goal-achievement set are all present in a new situation, and if the problem solver's current goal matches the goal achieved by the salient feature set, then the method of reaching the goal or the solution to the goal can be predicted from the previous case. Cases that match on the basis of goal-achievement sets are most helpful during problem solving when the problem solver knows what goals it is trying to achieve and knows the environment in which it needs to achieve those goals.

These sets of features may include one or several goals. They include one if the solution that was chosen for that goal did not involve other goals. They include several if solutions to several goals were 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.

Solution-Evaluation Sets are conjunctions of features *predicting* unusual outcomes – in general, failures, unexpected successes, and unexpected side effects. If the features of a goal-achievement set are all present in a new situation, the unexpected result from the previous case can be predicted in the new case. Cases that match on the basis of solution-evaluation sets are most helpful when a reasoner has proposed a solution and needs to evaluate it.

Outcome-Achievement Sets are conjunctions of features *describing* unusual outcomes. If the features of an outcome-achievement set are all present in a new situation, the previous case that is recalled can be used to help explain why the unusual outcome arose. In addition, if these features are all present in a new situation and the reasoner is attempting to figure out how to achieve such an outcome, the method by which it was achieved previously can be suggested by the recalled case. These are thus useful in two situations: when the reasoner is trying to explain an anomolous situation and when the reasoner knows the shape of a solution but not how to achieve it.

Any particular case may have several conjunctive feature sets associated

with it. For example, it could have one for each goal that was achieved in the course of reasoning about the case. It might also have several associated with outcome and several associated with solution evaluation. When attempting to choose best cases, preference heuristics prefer those cases that have one or more salient feature sets of the right kinds that are fully matched by the new situation. That is, if the reasoner is attempting to evaluate the potential for success of a plan, it prefers cases with fully matching solution-evaluation feature sets. If it is trying to achieve a goal, it prefers cases with fully matching goal-achievement feature sets whose goals match its current goal. If it attempting to explain an anomolous situation, or if it is attempting to find out how to achieve a state of affairs, it prefers cases with fully matching outcome-achievement feature sets.

Note that while many case memories index on one feature at a time, our concern here is with conjunctions of features. Any feature of one of these sets might not be a good index by itself. It is the conjunction of features that is important, and the particular conjunctions of features that are important enough to serve as indexes are those that serve useful purposes in reasoning.

Currently, we are testing out these hypotheses by using the set of cases in our JULIA system, hand-coding indexes for those cases, and evaluating as we go along, whether there are any other sets of features we would also like to index on. This enterprise is designed to allow us to test the indexing scheme just presented and to judge whether or not it is complete. Of course, it must also be tested in some other reasoning domain in addition to problem solving for us to be sure about its completeness.

3 Case-Based Design

Much of our work in the past several years at Georgia Tech has gone into the design of case-based reasoning systems for complex real-world domains. In the past several years, we have become particularly interested in design problems. Design is distinguished from other problem solving tasks in several ways:

- Problems are underconstrained. There are in general many ways to solve a problem. There is also, however, a large space to search to find these many possible solutions.

- Problems have many parts, requiring a representational system to manage the interactions between the many parts. Constraints are one way to do this.

- Unlike most problem-solving tasks AI researchers have worked on, in general, design problems are not nearly decomposable. In nearly-decomposable problems, you can work on the parts separately and then with only a little more effort, put the parts together to create a whole. In design, however, solving each of the parts is, in general, easy, but putting the pieces together to form a whole is hard. Problems cannot be solved by simple decomposition techniques.

All of these features of design make it a fully appropriate task for case-based reasoning. Case-based reasoning can be used to provide suggestions in solving underconstrained problems without utilizing search to achieve each of the parts of the problem. And since case-based reasoning can provide fully-solved problems to work with, the problem of dealing with a task that is not nearly-decomposable is made easier. Cases provide a solution with the "glue" that holds the parts together. The parts themselves can then be tweaked to get a solution that fits a new problem well.

On the other hand, using case-based reasoning for design requires us to rethink several of the assumptions made in early case-based reasoning systems, the most important of which is that we can't think of a case-based reasoner working by itself. It must be able to work in tandem with other reasoners. In particular, when problems have parts, some bookkeeping system is necessary to keep track of the relationships between the parts and the constraints put on parts by other decisions that have been made. In our design system, called JULIA, we take care of this problem by integrating the case-based reasoner with a constraint propagator. The constraint propagator has several responsibilities: (1) It propagates constraints from a problem specification into the solution specification. (2) It propagates constraints to the rest of the solution specification after some part of a suggested solution has been adapted.

Constraints allow us to keep track of the relationships between the parts of a problem. This allows later parts of a problem to be solved taking earlier decisions into account. In addition, we also need a system that can notice

inconsistencies. This is necessary for several reasons: (1) So that the inconsistencies between a problem specification and the recalled solution can be identified. (2) So that the side-effects of adaptations can be noticed. (3) So that later decisions can be forced not to violate constraints posted by earlier ones. In JULIA, a reason maintenance system that reasons based on constraints does this.

JULIA's three modules interact with each other in novel ways. JULIA uses its case-based reasoner and adaptation heuristics to suggest means of achieving goals and uses its constraint propagator to propagate the effects of its decisions to the rest of the problem. The reason maintenance system's job is to point out inconsistencies. The adaptation heuristics are used to fix those problems. While a standard reason maintenance system would call a dependency directed backtracker whenever it noticed an inconsistency, JULIA uses its reason maintenance system to point out inconsistencies but not to fix them. It relies on its adaptation strategies to fix problems. The philosophy of dependency directed backtracking is to undo reasoning that led to a faulty answer. Within the case-based reasoning paradigm, the philosophy is to preserve the reasoning, which was sound, and to fix the answer by either transforming it to fit the additional constraints of the problem or substituting something that works. Backtracking is reserved only for those situations in which adaptation won't work.

This work is reported in the attached paper by Tom Hinrichs entitled "Strategies for Problem Solving in Open Worlds." The paper has been submitted to the Cognitive Science Conference. Tom is approximately one year from finishing his Ph.D. thesis.

4 General Purpose Adaptation Heuristics

Another topic we are addressing in the context of JULIA, our design problem solver, is whether or not there is a set of general purpose adaptation heuristics. Most case-based reasoning systems have done adaptation through the use of special-purpose critics. We are attempted to design adaptation heuristics that are general in purpose and can be built in to the architecture of a case-based reasoning system. In that way, the designer of a case-based reasoning system would have to add only the knowledge necessary for the

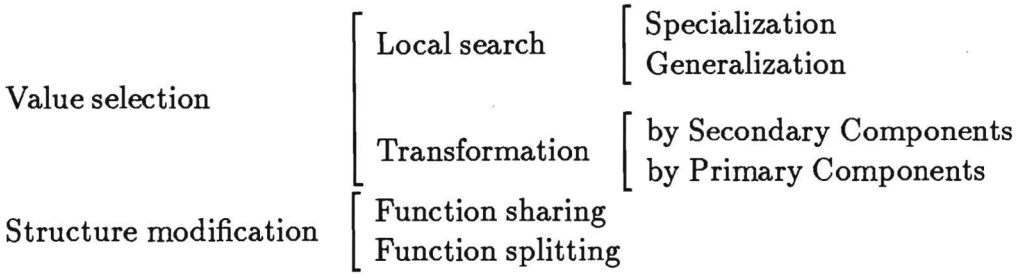

Figure 1: Adaptation strategies

adaptation heuristics to work rather than having to add adaptation heuristics specific to the domain of his case-based reasoner. At this time, we have identified four heuristics, each of which is built into JULIA's architecture.

Figure 1 shows a partial taxonomy of adaptation strategies in JULIA. These are broadly divided into *value selection* strategies, which manipulate the contents of a concept, and *structure modification* strategies, which manipulate the internal structure. The strategies are domain independent because they are directly related to the form of the representation of a concept rather than to higher level domain concepts. This is especially important for design tasks, because the range of designs achievable is determined by the coverage of the strategies.

Local Search

When a value violates constraints, an acceptable alternative is often 'nearby' in the search space. Rather than retracting the value and searching over again, we can tweak it by looking for specializations and generalizations of the value that will satisfy constraints. This is called *local search.* Two benefits accrue from this strategy: 1) the semantic hierarchy provides a limited and well-defined search space in lieu of the problem search space, which may be unlimited and ill-defined, and 2) if many subsequent decisions depend on the previous value, they may not need to be retracted if the new value is sufficiently similar. For example, in a meal planning scenario, a caterer can specialize lasagne to vegetarian-lasagne after learning that there will be vegetarians coming to dinner. The local search strategy saves the problem solver from having to search through the space of Italian dishes, and also

makes it unnecessary to retract any other feature of the meal that might have depended on lasagne.

Transformation

For problems that are less routine, local search may fail. When this happens, it is sometimes possible to use *transformation strategies* to modify the inconsistent value by adding, deleting, or substituting its components. Because there are an unlimited number of concepts that may be created in this way, it is crucial to limit the transformations that are applied. In JULIA, we do this by first determining that it is in fact the components of the value that violate constraints, as opposed to other descriptive features, and second, that the components do not by themselves define the concept to be transformed. To enable this, JULIA's representation of objects distinguishes between components that cannot be changed (eg, `main-ingredients`), and those that are 'easy to change' (eg, `secondary-ingredients`). While our current implementation does not support additions and substitutions, the ability to delete components greatly increases JULIA's flexibility. For instance, in the previous lasagne example, if JULIA had no concept of vegetarian lasagne, it would construct one by eliminating meat as a secondary ingredient. In future problems, the concept of vegetarian lasagne would be available by both locally searching the semantic hierarchy, and being reminded of this case.

When a constraint is violated by the primary components of an object, transformation becomes more difficult. Because the primary components effectively define an object, it is not possible to add or delete them, and substitution must be guided in some fashion. There are basically two ways to do this. First, a substitute may be found that is functionally identical to the original. For instance, a recipe for chicken a la king can be transformed into a vegetarian recipe by substituting Vegetarian Chicken Substitute, a canned substance that is designed to function as chicken in recipes. The second technique for transforming by primary components is to exploit constraints on the internal structure of the object. For instance, a recipe for shish kebab might include the constraint that objects to be skewered must be cohesive solids. In the extreme, such internal constraints would permit simulation of the behavior of the artifact in order to predict failures in much the same way as in CHEF (Hammond, 1986).

Function Sharing

When a problem is relatively novel, experience may suggest a solution whose structure is almost, but not quite, adequate. In this case, the problem solver must adapt not just the specific values, but the structure of the problem as well. *Function sharing*, in particular, is a strategy that is applicable when economic or aesthetic considerations encourage using a single mechanism or action to serve multiple functions. It modifies the structure of a problem by combining those variables that are functionally equivalent. For example, in a meal planning scenario, if a caterer planned to serve lasagne as a main course of an Italian meal, it would be appropriate to rule out the traditional pasta course of an Italian meal, since lasagne can serve the function of both main dish of a main course and a pasta course. In JULIA, function sharing is implemented as a constraint that relates two slots or scenes such that one is eliminated if its default function is subsumed by the other.

Function Splitting

When constraints conflict, it is sometimes possible to partition them and solve for them independently. *Function splitting* is a general strategy that increases the number of variables in a problem for the purpose of simplifying constraint satisfaction. For example, if a design must satisfy many people, then it may be advantageous to solve for each person independently, rather than all at once. One way JULIA uses function splitting is to increase choice of dishes when eaters have conflicting dietary constraints. As an example in another domain, function splitting might be used to gain reliability through redundant mechanisms.

Essential to this strategy are the criteria for determining when a variable may be split. These criteria fall into three basic categories:

1. **Nature of the variable.** The variable must represent a group or set of elements so that it can be divided into sub-groups.
2. **Source of conflicting constraints.** The conflicting constraints must derive from a common feature, such as the characters of an episode.
3. **Independence of constraints.** It must be possible to partition the constraints by source such that each partition is satisfiable.

When the criteria are met, the variable can be divided into two new variables which are then re-constrained and solved for independently.

4.1 Remarks

Two main points bear emphasizing: First, because the strategies we have presented are domain independent, the knowledge representation must support the process of adaptation. In JULIA, for example, the representation distinguishes between primary and secondary components of objects, and makes the structure of concepts explicit in terms of variables which can be reasoned about like any other value. Second, adaptation strategies may serve more than one function. In addition to adapting previous cases, they can be used to adapt previous *decisions*, in order to recover from errors. For more detail on these heuristics, see the two attached papers by Tom Hinrichs.

5 CBR Shell Development

Based on our experience with JULIA, we are attempting to design a generic case-based reasoning shell that can be used for design problem solving and other problem solving tasks requiring constraint propagation. It is based on the principles outlined in the two sections above. It is currently in the design stage. We hope to report more about its development in the next report.

6 Learning to Schedule

Our final research activity is in the area of learning. The particular problem we are looking at is scheduling a flexible manufacturing system. Operators of such systems start out doing fairly poorly at scheduling, and, in general, there are no teachers or manuals around to give them instruction. With a small amount of experience, however, they begin to get quite a bit better. Our interpretation is that this is a case-based activity – that they are explaining the failures that arise, figuring out which of their actions are responsible for those failures, figuring out which features of the environment can predict those failures, and indexing their experiences and new knowledge based on all that.

We have been involved in several endeavors in our investigation of these processes. First, we have been working on a representational vocabulary that can be used to describe scheduling problems. Our emphasis has been in two areas necessary for representing problems in this domain. First, there is another agent involved – a "dumb" scheduler that "helps" the operator schedule the system. In reality, the operator monitors and cleans up after this "dumb" scheduler. It turns out to be good for mundane scheduling, allowing the operator to do the tasks that require attention. Second, items in the system move around from container to container. They start out in a start buffer, move to a work-in-progress buffer, move to the input for some machine, back to the work-in-progress buffer, and in some cases, move to an overflow buffer. We have worked on dimensions of a vocabulary that can represent agents and containers. The attached paper entitled "A General Vocabulary for Indexing Cases in Multi-Agent Domains" by Robinson and Wood gives more detail. The next step in this endeavor is to represent scheduling problems themselves.

Our second endeavor has been to understand better how people actually learn this task. We are working along with people in Georgia Tech's School of Industrial and Systems Engineering (ISyE) and School of Psychology to do this. Chris Mitchell and her students in ISyE have developed a simulator for a real flexible manufacturing system, a "dumb" scheduler, and an interface that allows the operator to view the system. We are observing people learning to operate the manufacturing system. Richard Catrambone, a faculty member in Psychology, is aiding us with carrying out these experiments. To now, we have collected five to six hours of protocols (using videotape) on each of six subjects, and we are in the process of analyzing them. A paper submitted to Cognitive Science entitled "Changes in Information Use and Strategy on a Complex Task as a Function of Practice" gives an initial summarization. It is quite abstract rather than getting at the details of what knowledge changed over time and how. We are currently working on extracting from the protocols what knowledge the subjects started with, what strategies they started with, what changed over time, and how that could have happened.

The scheduling activity we are investigating in this project is important for several reasons. First, as pointed out above, we hope to discover how case-based reasoning functions in early learning of a task. This might allow us to build expert systems that start with little skill and through the

analysis of their experience become more skillful. Second, this particular planning activity combines reactive planning, execution, case-based reasoning, and learning. It is a good task in which to study reactive planning, and gives us the opportunity to learn how a reactive planner might get better. Third, this task provides the bottom end of logistics planning. Logistics requires strategies and use of principles in scheduling. At some point, those strategies and principles have to be put to use. That is, the scheduling has to happen. As in other cases of logistical planning, this scheduler has to adhere to principles and strategies and figure out how to deal (within the guidelines of those principles) with the world as it actually is.

7 Other Activity

This summer at IJCAI, Janet Kolodner and Chris Riesbeck will be presenting a tutorial on case-based reasoning. Much time in the past months has been spent in making the slides for that tutorial. Our approach is to present techniques, using examples from particular systems to illustrate those techniques.

Janet Kolodner will be giving the keynote address at the DARPA Case-Based Reasoning Workshop in Pensacola in May. JULIA will be demoed, as will a system called MECH that combines case-based reasoning with EBL and learning from an instructor's explanations. Georgia Tech will also be contributing several papers to the poster sessions.

We have submitted several papers to the Cognitive Science Conference: "Selecting the Best Case for a Case-Based Reasoner" by Janet Kolodner, "Strategies for Problem Solving in Open Worlds" by Tom Hinrichs, and "Changes in Information Use and Strategy on a Complex Task as a Function of Practice" by Steve Robinson, David Wood, and Richard Catrambone. Tom Hinrichs also submitted an abstract to the Case-Based Reasoning Workshop entitled "Strategies for Adaptation and Recovery in a Design Problem Solver." In addition, Steve Robinson and David Wood have had their paper "A General Vocabulary for Indexing Cases in Multi-Agent Domains" published in the *Proceedings of the Southeast ACM Conference.* David Wood presented the paper, which won a student award.

C-36-611

Case-Based Reasoning at Georgia Tech Annual Report March, 1989 - March, 1990

Janet L. Kolodner
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
jlk@ics.gatech.edu

March 21, 1990

During the past year, work on this project has been in 6 areas: case selection, index selection, cbr for design problem solving, cbr for creative problem solving, cbr for learning a scheduling problem, and a framework for case-based decision aiding.

1 Case Selection

The most important support process a case-based reasoner needs is a memory for cases. The memory must make cases accessible when 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. But most importantly, it must be able to select out the most appropriate cases for the case-based reasoner to use at any time.

In the first year of the contract, we worked on methods for selecting the best case from memory. Our work concentrated on heuristics for selection

and the indexing content necessary to make selection work appropriately. We presented 6 heuristics for choosing useful cases: goal-directed preference, salient-feature preference, specificity, frequency, recency, and ease-of-use. We hypothesized that the heuristics were ordered such that goal-directed preference is applied first, then salience, then specificity, and then frequency and recency. This way, the set of cases that can be used to achieve the reasoner's current goal is selected out first, then any that match on a full set of salient features (of the right kind) are selected from those, the most specific of those are chosen (if some are more specific than others), and then the more frequently or recently recalled cases are selected from those.

Our work in the second year on this issue addressed the issue of interactions between these heuristics. We implemented a testbed in which we can apply the heuristics varying their orderings and precedence, in order to judge how they ought to be applied as a group. The testbed is now complete, and we are designing the experiments to be carried out and choosing the issues to address. In addition to looking at the interactions between heuristics, we will also be looking at the effects of differing amounts of information about a case on its selection. Some cases (those a system has reasoned through itself) have considerable information associated with them concerning the reasons why they were solved in certain ways. This means they can be indexed extensively by many different sets of features. Other cases are just examples of things that worked and didn't work. Their indexes are much less embellished. Using this testbed we will be able to look at the relationship between the richness of indexing and retrievability. The test system is working with cases from JULIA, our design problem solver that plans meals.

2 Index Selection

Our analysis of what goes into choosing a best case has led us to several conclusions about what kinds of indexes are useful ones for a case to have. While most efforts look at algorithms for retrieval, this endeavor looks at the **content of indexes**. We reported in papers at the 1989 DARPA Case-Based Reasoning Workshop and the 1989 Cognitive Science Conference that we have found three kinds of indexes useful for problem solving. The first contain features that predict the applicability of some method for achieving a goal (**goal-achievement sets**). Second are those that predict the success

of failure of a solution (**solution-evaluation sets**). Third are those that describe unusual outcomes (**outcome-achievement sets**).

Goal-Achievement Sets are generally conjunctions of goals, constraints on these goals, and problem and environmental features that predict the method or solution for achieving the goal or goal set. If the features of a goal-achievement set are all present in a new situation, and if the problem solver's current goal matches the goal achieved by the salient feature set, then the method of reaching the goal or the solution to the goal can be predicted from the previous case. Cases that match on the basis of goal-achievement sets are most helpful during problem solving when the problem solver knows what goals it is trying to achieve and knows the environment in which it needs to achieve those goals.

These sets of features may include one or several goals. They include one if the solution that was chosen for that goal did not involve other goals. They include several if solutions to several goals were 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.

Solution-Evaluation Sets are conjunctions of features *predicting* unusual outcomes – in general, failures, unexpected successes, and unexpected side effects. If the features of a goal-achievement set are all present in a new situation, the unexpected result from the previous case can be predicted in the new case. Cases that match on the basis of solution-evaluation sets are most helpful when a reasoner has proposed a solution and needs to evaluate it.

Outcome-Achievement Sets are conjunctions of features *describing* unusual outcomes. If the features of an outcome-achievement set are all present in a new situation, the previous case that is recalled can be used to help explain why the unusual outcome arose. In addition, if these features are all present in a new situation and the reasoner is attempting to figure out how to achieve such an outcome, the method by which it was achieved previously can be suggested by the recalled case. These are thus useful in two situations: when the reasoner is trying to explain an anomolous situation and when the reasoner knows the shape of a solution but not how to achieve it.

Any particular case may have several conjunctive feature sets associated with it. For example, it could have one for each goal that was achieved in the course of reasoning about the case. It might also have several associated with outcome and several associated with solution evaluation. When attempting to choose best cases, preference heuristics prefer those cases that have one or more salient feature sets of the right kinds that are fully matched by the new situation. That is, if the reasoner is attempting to evaluate the potential for success of a plan, it prefers cases with fully matching solution-evaluation feature sets. If it is trying to achieve a goal, it prefers cases with fully matching goal-achievement feature sets whose goals match its current goal. If it attempting to explain an anomolous situation, or if it is attempting to find out how to achieve a state of affairs, it prefers cases with fully matching outcome-achievement feature sets.

Note that while many case memories index on one feature at a time, our concern here is with conjunctions of features. Any feature of one of these sets might not be a good index by itself. It is the conjunction of features that is important, and the particular conjunctions of features that are important enough to serve as indexes are those that serve useful purposes in reasoning.

Currently, we are testing out these hypotheses by using the set of cases in our JULIA system, hand-coding indexes for those cases, and evaluating as we go along, whether there are any other sets of features we would also like to index on. This enterprise is designed to allow us to test the indexing scheme just presented and to judge whether or not it is complete. Of course, it must also be tested in some other reasoning domain in addition to problem solving for us to be sure about its completeness.

3 Case-Based Design

Much of our work in the past several years at Georgia Tech has gone into the design of case-based reasoning systems for complex real-world domains. In the past several years, we have become particularly interested in design problems. Design is distinguished from other problem solving tasks in several ways:

- Problems are underconstrained. There are in general many ways to solve a problem. There is also, however, a large space to search to find these many possible solutions.

- Problems have many parts, requiring a representational system to manage the interactions between the many parts. Constraints are one way to do this.

- Unlike most problem-solving tasks AI researchers have worked on, in general, design problems are not nearly decomposable. In nearly-decomposable problems, you can work on the parts separately and then with only a little more effort, put the parts together to create a whole. In design, however, solving each of the parts is, in general, easy, but putting the pieces together to form a whole is hard. Problems cannot be solved by simple decomposition techniques. We call these problems **hardly-decomposable.**

All of these features of design make it a fully appropriate task for case-based reasoning. Case-based reasoning can be used to provide suggestions in solving underconstrained problems without utilizing search to achieve each of the parts of the problem. And since case-based reasoning can provide fully-solved problems to work with, the problem of dealing with a task that is hardly-decomposable is made easier. Cases provide a solution with the "glue" that holds the parts together. The parts themselves can then be tweaked to get a solution that fits a new problem well.

Concentration in year 2 of this contract has been in two areas: the types and uses for adaptation strategies and criteria for evaluating the results of design problem solving.

While most work in CBR has been on the use of adaptation strategies for adapting an old solution to fit a new problem, we find that adaptation strategies have at least three uses:

- adapting an old solution to fit a new problem

- patching an almost-right solution to make it better

- adapting a problem specification to fit the possible solutions

Using adaptation these three ways in solving design problems has the potential for solving many design issues.

- Adaptation facilitates making decision in under-constrained situations by allowing the problem solver to re-use an 'almost-right' plan rather than re-solving from scratch.

- Adaptation resolves over-constrained problems by serving as an alternative to retraction.
- Adaptation relaxes preference constraints by minimally weakening them.
- Adaptation extends the vocabulary of a problem solver by creating new components as variations of known ones.

Using adaptation for all of these tasks allows four different kinds of design processes – selection, configuration, parameter fixing, and construction – to be integrated within one design architecture. And it provides a computationally efficient and parsimonious way of dealing with the introduction of new constraints during problem solving, a common design delemma. A paper entitled "The Integrative Role of Adaptation in Design" by Tom Hinrichs and Janet Kolodner give more details of this work. The paper was submitted to AAAI-90.

An important part of any problem solving endeavor is to evaluate the solutions that are created. This is particularly problematic when the problem is under-specified, as is almost always the case in conceptual and preliminary design. Comparison to other cases provides a way of evaluating solutions in these situations, but we must know on what dimensions to do the evaluation. Solutions must be evaluated on consistency and completeness. The enclosed paper by Tom Hinrichs entitled "Some Criteria for Evaluating Designs" give the details of this work. It was submitted to the 1990 Cognitive Science Conference.

4 Creative Problem Solving

In case-based reasoning, and in much of the problem solving people do, solutions are created for new problems by adapting and combining known solutions to similar problems. Sometimes a new solution can be created by merely tweaking or adapting some old one in routine ways. Often, however, problem solving is less routine, requiring exploration of several alternatives, perhaps adapting and merging several possibilities gleaned from experience.

The case-based reasoning projects that are concentrating on problem solving have, for the most part, been looking at routine problem solving. That

is, problems are solved by recalling similar cases and adapting them by well-known procedures to fit the new situation. Our problem solvers come up with good, but boring solutions.

Previous experience seems to play a large role in the creative problem solving people do. It is time to make our case-based problem solvers more creative. Some investigations are doing this by looking at vocabularies that transcend several types of problems on the premise that creativity can emerge by transferring the abstract solution from those far-away problems and specializing it to the new problem (e.g., Kass, Owens, Birnbaum & Collins). In our investigation, we are looking at the creativity that emerges by combining the solutions to several cases in novel ways.

There are four processes we've found that extend basic case-based reasoning methodology to provide creativity in problem solving.

- **Problem specification elaboration**: In almost all AI investigations of problem solving to date, researchers have assumed that the problem specification would define the problem succinctly. Yet interesting problem solving domains have as their hallmarks the fact that problem specifications are rarely well defined. In general, they are incomplete, leaving many different ways to solve a problem. Sometimes, they are unnecessarily constrained. An important part of solving problems in these cases is redefining the problem spec. This includes elaboration, problem restructuring, and negotiating problem space boundaries (Goel & Perolli, 1989).

- **Exploration of solution alternatives**: In general, in problem solving in ill-structured domains, there are many different ways a problem could be solved, some better than others, some with different trade-offs than others. Many possible solutions must often be considered during problem solving. In addition, because problem specifications are often incomplete, the problem solver might need to incrementally converge on a good solution by following a cycle of starting with whatever comes to mind, critiquing that, and based on the critique, adding appropriate details to the problem specification. Alternatively, the problem solver might elaborate the original problem specification several different ways, using each to generate an alternate specification. This process includes both search of the solution space and evaluation of alternatives.

- **Merging of possibilities:** In routine problem solving, parts of several solutions are often merged, but in general, the parts do no overlap (e.g., dessert from one meal might be used with a main dish from another meal). In less routine problem solving, several suggestions for solving the same part of a problem might be merged to come up with a solution (e.g., in deciding to have salmon fettucine and salad for dinner, a meal planner might have remembered three previous cases, a meal with fish, a pasta meal, and a one-dish meal, and merged desirable features from each).

- **Non-routine adaptation:** Previous work has looked at adapting old solutions to fit new problems. In creative problem solving, it sometimes makes sense to adapt one's goals to fit an old solution rather than changing the old solution to fit the new problem. Previous work has looked at routine adaptation strategies (e.g., deletion, addition, substitution) but not at use of "off-the-cuff" ones (i.e., those developed in response to a particular problem). Some of these arise from examining a causal model, some from adapting well-known adaptation strategies, some from applying well-known adaptation strategies in novel ways. There may be other ways non-routine adaptations arise.

Our investigation of these processes has started with implementation of a program that addresses the second problem above: creative problem solving by exploring the case base. Our program is based on a protocol of a person doing creative design. In this case, the person was trying to come up with a dinner dish to use up some leftover white rice. The program, like our subject, retrieves an initial set of instances of using white rice, and evaluates each. Based on the evaluation, it alters the problem specification. Features that it doesn't have but needs, it adds from the case. Those features that it finds unacceptable in any case it remembers, it rules out. It then it takes this new problem specification and probes the memory again. It continues this cycle until it derives a solution that can be easily adapted to make a good solution, or until it retrieves a case that can be adapted to fit the new situation. When it remembers a Chinese dish, for example, the program rules out Chinese as a cuisine since it has just had Chinese food the day before. When it remembers a bread recipe that uses rice, it adds the specification that the recipe it comes up with should be easy to make, since the work involved in making bread is more than it wants to do. When it remembers a breakfast dish that uses

rice, it changes its goals completely and decides that maybe a dinner dish isn't necessary – breakfast would do just as nicely.

The process of creative problem solving through exploration of the case base, as we've defined it so far, has the following steps:

1. Retrieve an initial set of cases, using the initial problem specification
2. For each case:
3. Evaluate the case for its applicability or adaptability to the problem
4. Temporarily alter the set of goals, based on the current source case
5. Use the altered set of goals to retrieve an additional set of cases
6. go back to step 1

The ability to come up with creative, rather than routine, solutions to problems would help many of our case-based problem solvers (e.g., CHEF, JULIA, PERSUADER). The most obvious application of creativity in case-based reasoning is in those domains that are seen as inherently creative: design, for example, or meal planning, or architecture, and so on. But the approach we have introduced here can help case-based reasoners in many domains.

Any problem solver can reach an impasse as it works to solve a problem. Creative case-based reasoning provides techniques to breach such an impasse. By elaborating the original problem specification in more than one way, a creative problem solver can find different paths to solutions.

In a domain where the problem may be poorly defined, perhaps incomplete or overconstrained, the creative process can help elaborate the specification of the problem, and provide a more complete basis for problem solving.

Creative techniques can help a case-based reasoner produce variety in its solutions. Rather than using disjoint parts of different cases as the basis for a new solution, for example, a creative approach would merge corresponding parts of different cases to come up with a novel solution.

An understanding of creative problem solving processes has several important implications. If we understand creative processes, which parts are hard and which are easy, we will be able to create the right kinds of tools to

help problem solvers with their tasks. We will find out which processes can be relegated to a machine and which will need to continue to be done by people. This understanding will also help us in building the kinds of tools and developing the kinds of curriculums that can best be used to train creative problem solvers of the future.

The paper entitled "A Case-Based Approach to Creativity in Problem Solving" by Janet Kolodner and Louise Penberthy describes more about this project. It was submitted to the 1990 Cognitive Science Conference.

5 Learning to Schedule

In the first year of this contract, we spent considerable time investigating scheduling in the domain of flexible manufacturing. Operators of such systems start out doing fairly poorly at scheduling, and, in general, there are no teachers or manuals around to give them instruction. With a small amount of experience, however, they begin to get quite a bit better. Our interpretation is that this is a case-based activity – that they are explaining the failures that arise, figuring out which of their actions are responsible for those failures, figuring out which features of the environment can predict those failures, and indexing their experiences and new knowledge based on all that. Related to this project, we began work on a representational vocabulary for describing scheduling, and we observed people learning the task. We found that we couldn't learn enough from these observations for an interesting project.

As a result, we have switched to a different task domain, that of scheduling the time of a busy single working parent. As in the flexible manufacturing domain, the planner must be reactive and opportunistic. Also as in that domain, we can assume that the planner begins by knowing how to achieve goals in isolation and learns over time how to integrate the achievement of several goals in tandem. Our work thus far has centered on a representation for plans that allows fast execution, supports noticing and acting upon expectation failures, evolves easily, and supports partial planning, reactivity, and opportunistic action. The paper entitled "Representing Routine Plans: Issues Important to Acting, Planning, and Learning in the Real World" by Stephen Robinson and Janet Kolodner addresses these issues.

In the next year, we expect to create a program that begins as a novice scheduler and becomes able to interleave tasks with ease. Currently the

program, called SUPERMOM, contains cases, plans, scripts and goals pertaining to its weekday morning routine. It can be interrupted by events such as running out of milk or discovering that one of the kids is sick. It has mechanisms for devising alternative plans, executing those plans, and storing and indexing the experiences int its memory, and it can follow those indexes in later situations when it runs into similar problems.

The scheduling activity we are investigating in this project is important for several reasons. First, as pointed out above, we hope to discover how case-based reasoning functions in early learning of a task. This might allow us to build expert systems that start with little skill and through the analysis of their experience become more skillful. Second, this activity combines reactive planning, execution, case-based reasoning, and learning. It is a good task in which to study reactive planning, and gives us the opportunity to learn how a reactive planner might get better. Third, this task provides the bottom end of logistics planning. Logistics requires strategies and use of principles in scheduling. At some point, those strategies and principles have to be put to use. That is, the scheduling has to happen. As in other cases of logistical planning, this scheduler has to adhere to principles and strategies and figure out how to deal (within the guidelines of those principles) with the world as it actually is.

Our plan is to do our investigation in the common-sense scheduling domain and to reimplement it in the flexible manufacturing domain as a later step.

6 Case-Based Decision Aiding

While much of the research work on this contract has been addressing issues in fully automating case-based reasoning, this part of the endeavor looks at the use of a case memory to aid people in decision making. Psychologists tell us that if people have the right cases available, then they are able to make case-based inferences accurately and easily. However, people often don't have the right cases available to them, either because they have not experienced them or because they did not remember them at the right times (due to indexing them wrong, bias of various kinds, etc.). A case-based decision aiding tool helps a person solve problems by augmenting his/her memory.

We are at the very beginning of a project in this area, joint with faculty

from Georgia Tech's College of Architecture. We have also discussed such systems with faculty at Georgia State's Management School, and we are hopeful about starting several projects in this area. We have already given several talks about such systems: at Georgia State, NRL, Bellcore, and as a keynote address to the Judgment and Decision Making Society.

7 Other Activity

We added two new faculty members to the AI group at Georgia Tech who do research in case-based reasoning. Ashok Goel, from Ohio State, studies the integration of model-based and case-based methods for design problem solving. Ashwin Ram works on learning from cases.

During IJCAI-89, Janet Kolodner and Chris Riesbeck presented a tutorial on case-based reasoning. Our approach is to present techniques, using examples from particular systems to illustrate those techniques. Our materials have been updated for the 1990 AAAI, where we will be presenting another tutorial. The tutorial is also being given as part of Georgia Tech's Education Extension's two week course on expert systems building.

Janet Kolodner gave the keynote address at the DARPA Case-Based Reasoning Workshop in Pensacola in May, 1989. JULIA was be demoed, as well as a system called MECH that combines case-based reasoning with EBL and learning from an instructor's explanations. Georgia Tech contributed several papers to the poster sessions. Georgia Tech is also well-represented at the 1990 AAAI Spring Symposium on Case-Based Reasoning. Janet Kolodner will be on two panels, Tom Hinrichs will give a talk and be on a panel, and Ashok Goel will present his work.

C 36-611

Case-Based Reasoning at Georgia Tech Final Technical Report

Janet L. Kolodner
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
jlk@cc.gatech.edu

May 7, 1992

In the three years of this DARPA-funded project, case-based reasoning work has been in a wide variety of areas: case selection, index selection, cbr for design problem solving, general purpose adaptation heuristics, cbr for learning to schedule, case representation, and case-based tools for designers. We briefly describe each effort here and refer to published theses and conference proceedings papers for more in-depth descriptions of the work and its contributions.

1 Case Selection

The most important support process a case-based reasoner needs is a memory for cases. The memory must make cases accessible when 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. But most importantly, it must be able to select out the most appropriate cases for the case-based reasoner to use at any time.

While much work in the past has gone into organizing cases in a memory and retrieval algorithms for recalling them, little has gone into the problem of choosing the best case from among the many partial matches that are retrieved. Our work, summarized in the attached paper by Janet Kolodner entitled "Selecting the Best Case for a Case-Based Reasoner," addresses this problem. It is implemented in a program called PARADYME (PARAllel DYnamic MEmory), which is designed to work alongside a case-based reasoner. PARADYME's memory is hierarchical, but there are no physical indexes in the memory. Rather, cases are annotated or labeled by their important features. Retrieval is a two step process. First, a parallel retrieval mechanism (implemented on the Connection Machine) retrieves all cases that partially match the retrieval probe. In a second step, selection heuristics use the labels associated with cases to choose the best from among those partial matches.

PARADYME's selection method is based on many of the same principles guiding the selection of indexes and retrieval algorithms used in other case memories. However, it differs from those memories in several ways. First, PARADYME's parallel retrieval method allows us to do away with the restrictions indexes put on retrieval in other memory systems. Instead, what would have been indexes in those systems are found as annotations on cases called *salient feature sets*. Cases whose full salient feature set matches a retrieval probe are preferred over others, but cases without full sets of matching salient feature sets can also be retrieved. In this way, indexes are allowed to act as *selectors* rather than restrictors. PARADYME does not get hurt by the inability to predict every important part of a case at the time it happens. PARADYME prefers cases whose salient features (indexes) match the retrieval probe but if no cases with indexed features match, it will recall a case with other matching features.

Second, PARADYME's emphasis when ranking cases is on **usefulness**. Using this criterion for ranking means that PARADYME takes the reasoner's goals into account in selecting out a "best" case. Rather than choosing a most similar case, it chooses the most similar of those cases that are first judged most useful.

PARADYME's selection procedure is based on a set of *preference heuristics*. These heuristics are applied to the set of partially-matching cases to choose a small set of "best" cases. PARADYME uses six different types of preference for this task.

- Goal-Directed Preference
- Salient-Feature Preference
- Specificity Preference
- Frequency Preference
- Recency Preference
- Ease-of-Adaptation Preference

The first preference, *goal-directed preference* is based on the principle of utility. That is, since the memory is working in conjunction with a reasoner that has goals, it makes sense to prefer those cases that can help in achieving the problem solver's goals. Thus, when the problem solver is trying to come up with a main dish, those cases that match on main dish constraints will be preferred over others. When it is trying to evaluate the goodness of a solution, those cases that predict success or failure under similar circumstances are preferred. We state this heuristic as follows:

> **Goal-Directed Preference:** Prefer cases that can help address the reasoner's current reasoning goal, and of these, prefer those that share more constraints over those that share fewer.

The second preference heuristic, *salient-feature preference*, is based on the principle that we should use experience to tell us which features of a new situation are the ones to focus on. If memory has done a good job of recording its experiences, they can be used to tell us which features of previous events led to the choice of particular solutions or solution methods and which features of previous events were responsible for success or failure in those cases. These features are the *salient features* of previous cases, and in indexed memories, they form the indexes. When salient features of previous cases exist in a new situation, they can be used to suggest solutions and predict outcomes for the new case. A case where a friend named Anne didn't eat what was served for dinner, for example, has a salient feature set that predicts failure and includes the following facts: Anne was a guest, fish was served, preparation style of the fish was grilled. When all of these features are present in a probe, we can predict that Anne won't eat. PARADYME

prefers cases that share full sets of salient features with the new problem over other cases whose full salient feature sets are not in the probe. We state this preference as follows:

> **Salient-Feature Preference:** Prefer cases that match on salient features over those that match on other features, and prefer those that match on a larger subset of salient features over those matching on a smaller subset.

The third preference heuristic is based on the principle that a more specific match can be more predictive than a less specific match. Thus, all other things being equal, cases that match more specifically are preferred over less specific matches. PARADYME has several ways to judge specificity. First, according to PARADYME's definition of specificity, a case is more specific than another if the features that match in the less specific case are a proper subset of the features that match in the more specific case. Thus, a probe is more specifically matched by a case that matches all of its features than one that matches only a subset. Second, a case matches more specifically than one of its ancestors in memory's generalization hierarchy. For example, a particular Italian meal is more specific than a generic Italian meal. Third, a case matches more specifically if the probe matches features in more of its parts. The specificity preference follows:

> **Specificity Preference:** Prefer cases that match more specifically over less specific matches.

The fourth and fifth heuristics are based on two principles psychologists have discovered – that items that are referenced more frequently are more likely to be recalled than other similar items and that items that have been referenced more recently are more likely to be recalled than other similar items (all else being equal). This gives rise to two preference heuristics:

> **Frequency Preference:** Prefer cases that have been accessed more frequently over less frequently-accessed cases.
> **Recency Preference:** Prefer cases that have been accessed more recently over less recently-accessed cases.

A sixth preference heuristic is also based on the principle of utility, and is specific to case-based reasoning. Some adaptations of previous solutions

are easier to make than others. This heuristic says to prefer cases whose solutions are easier to adapt than those whose solutions are harder to adapt.

> **Ease-of-Adaptation Preference:** Cases that match on features that are known to be hard to fix should be preferred over those that match on easy-to-fix features.

The application of preference heuristics is complicated. Each preference heuristic attempts to select out a set of better matches. When a heuristic does this, that set is sent on to the next heuristic for pruning. When no subset of cases is better than the rest using some heuristic, however, the entire set it was selecting from is selected. In this way, the preferences act as *selectors* rather than restrictors. We prefer to recall a case that can address the reasoner's current goal but we don't require it. We prefer to recall a case that matches on salient features, but if there are none, the preference heuristics allow recall of a case that matches on a random set of features.

The heuristics are also ordered. Goal-directed preference is applied first, then salience, then specificity, and then frequency and recency. This way, the set of cases that can be used to achieve the reasoner's current goal is selected out first, then any that match on a full set of salient features (of the right kind) are selected from those, the most specific of those are chosen (if some are more specific than others), and then the more frequently or recently recalled cases are selected from those.

Appendix A holds papers related to this effort.

2 Index Selection

Our analysis of what goes into choosing a best case has led us to several conclusions about what kinds of indexes are useful ones for a case to have. We have found three kinds of indexes useful for problem solving. The first contain features that predict the applicability of some method for achieving a goal (*goal-achievement sets*). Second are those that predict the success of failure of a solution (*solution-evaluation sets*). Third are those that describe unusual outcomes (*outcome-achievement sets*).

Goal-Achievement Sets are generally conjunctions of goals, constraints on these goals, and problem and environmental features that predict the method or solution for achieving the goal or goal set. If the features of a

goal-achievement set are all present in a new situation, and if the problem solver's current goal matches the goal achieved by the salient feature set, then the method of reaching the goal or the solution to the goal can be predicted from the previous case. Cases that match on the basis of goal-achievement sets are most helpful during problem solving when the problem solver knows what goals it is trying to achieve and knows the environment in which it needs to achieve those goals.

These sets of features may include one or several goals. They include one if the solution that was chosen for that goal did not involve other goals. They include several if solutions to several goals were 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.

Solution-Evaluation Sets are conjunctions of features *predicting* unusual outcomes – in general, failures, unexpected successes, and unexpected side effects. If the features of a goal-achievement set are all present in a new situation, the unexpected result from the previous case can be predicted in the new case. Cases that match on the basis of solution-evaluation sets are most helpful when a reasoner has proposed a solution and needs to evaluate it.

Outcome-Achievement Sets are conjunctions of features *describing* unusual outcomes. If the features of an outcome-achievement set are all present in a new situation, the previous case that is recalled can be used to help explain why the unusual outcome arose. In addition, if these features are all present in a new situation and the reasoner is attempting to figure out how to achieve such an outcome, the method by which it was achieved previously can be suggested by the recalled case. These are thus useful in two situations: when the reasoner is trying to explain an anomolous situation and when the reasoner knows the shape of a solution but not how to achieve it.

Any particular case may have several conjunctive feature sets associated with it. For example, it could have one for each goal that was achieved in the course of reasoning about the case. It might also have several associated with outcome and several associated with solution evaluation. When attempting to choose best cases, preference heuristics prefer those cases that have one

or more salient feature sets of the right kinds that are fully matched by the new situation. That is, if the reasoner is attempting to evaluate the potential for success of a plan, it prefers cases with fully matching solution-evaluation feature sets. If it is trying to achieve a goal, it prefers cases with fully matching goal-achievement feature sets whose goals match its current goal. If it attempting to explain an anomolous situation, or if it is attempting to find out how to achieve a state of affairs, it prefers cases with fully matching outcome-achievement feature sets.

Note that while many case memories index on one feature at a time, our concern here is with conjunctions of features. Any feature of one of these sets might not be a good index by itself. It is the conjunction of features that is important, and the particular conjunctions of features that are important enough to serve as indexes are those that serve useful purposes in reasoning.

Appendix A holds papers related to this effort.

3 Case-Based Design

Much of our work in the past several years at Georgia Tech has gone into the design of case-based reasoning systems for complex real-world domains. In the past several years, we have become particularly interested in design problems. Design is distinguished from other problem solving tasks in several ways:

- Problems are underconstrained. There are in general many ways to solve a problem. There is also, however, a large space to search to find these many possible solutions.
- Problems have many parts, requiring a representational system to manage the interactions between the many parts. Constraints are one way to do this.
- Unlike most problem-solving tasks AI researchers have worked on, in general, design problems are not nearly decomposable. In nearly-decomposable problems, you can work on the parts separately and then with only a little more effort, put the parts together to create a whole. In design, however, solving each of the parts is, in general, easy, but putting the pieces together to form a whole is hard. Problems cannot be solved by simple decomposition techniques.

All of these features of design make it a fully appropriate task for case-based reasoning. Case-based reasoning can be used to provide suggestions in solving underconstrained problems without utilizing search to achieve each of the parts of the problem. And since case-based reasoning can provide fully-solved problems to work with, the problem of dealing with a task that is not nearly-decomposable is made easier. Cases provide a solution with the "glue" that holds the parts together. The parts themselves can then be tweaked to get a solution that fits a new problem well.

On the other hand, using case-based reasoning for design requires us to rethink several of the assumptions made in early case-based reasoning systems, the most important of which is that we can't think of a case-based reasoner working by itself. It must be able to work in tandem with other reasoners. In particular, when problems have parts, some bookkeeping system is necessary to keep track of the relationships between the parts and the constraints put on parts by other decisions that have been made. In our design system, called JULIA, we take care of this problem by integrating the case-based reasoner with a constraint propagator. The constraint propagator has several responsibilities: (1) It propagates constraints from a problem specification into the solution specification. (2) It propagates constraints to the rest of the solution specification after some part of a suggested solution has been adapted.

Constraints allow us to keep track of the relationships between the parts of a problem. This allows later parts of a problem to be solved taking earlier decisions into account. In addition, we also need a system that can notice inconsistencies. This is necessary for several reasons: (1) So that the inconsistencies between a problem specification and the recalled solution can be identified. (2) So that the side-effects of adaptations can be noticed. (3) So that later decisions can be forced not to violate constraints posted by earlier ones. In JULIA, a reason maintenance system that reasons based on constraints does this.

JULIA's three modules interact with each other in novel ways. JULIA uses its case-based reasoner and adaptation heuristics to suggest means of achieving goals and uses its constraint propagator to propagate the effects of its decisions to the rest of the problem. The reason maintenance system's job is to point out inconsistencies. The adaptation heuristics are used to fix those problems. While a standard reason maintenance system would call a dependency directed backtracker whenever it noticed an inconsistency, JU-

LIA uses its reason maintenance system to point out inconsistencies but not to fix them. It relies on its adaptation strategies to fix problems. The philosophy of dependency directed backtracking is to undo reasoning that led to a faulty answer. Within the case-based reasoning paradigm, the philosophy is to preserve the reasoning, which was sound, and to fix the answer by either transforming it to fit the additional constraints of the problem or substituting something that works. Backtracking is reserved only for those situations in which adaptation won't work.

3.1 Adaptation

Within the context of design, a particular topic we have addressed is the compilation of a set of general purpose adaptation heuristics. Most case-based reasoning systems have done adaptation through the use of special-purpose critics. We are attempted to design adaptation heuristics that are general in purpose and can be built in to the architecture of a case-based reasoning system. In that way, the designer of a case-based reasoning system would have to add only the knowledge necessary for the adaptation heuristics to work rather than having to add adaptation heuristics specific to the domain of his case-based reasoner.

The list below shows a taxonomy of adaptation strategies in JULIA. Some (the first 4 sets) manipulate the contents of concepts, some manipulate the internal structure of solutions, and some modify the constraints defining a problem specification. The strategies are domain independent because they are directly related to the form of the representation of a concept rather than to higher level domain concepts. This is especially important for design tasks, because the range of designs achievable is determined by the coverage of the strategies.

- Concept substitution strategies; including specialize, generalize, substitute sibling, and substitute by function
- Set transformation strategies; including insert and delete
- Quantity substitution strategies; including increase quantity and decrease quantity
- Logical substitution strategies; including invert

- Structure modification strategies; including share function, split function, and transpose
- Constraint modification strategies; including enlarge domain, reduce domain, generalize range, and diminish range

While most work in CBR has been on the use of adaptation strategies for adapting an old solution to fit a new problem, we find that adaptation strategies have at least three uses:

- adapting an old solution to fit a new problem
- patching an almost-right solution to make it better
- adapting a problem specification to fit the possible solutions

Using adaptation these three ways in solving design problems has the potential for solving many design issues.

- Adaptation facilitates making decision in under-constrained situations by allowing the problem solver to re-use an 'almost-right' plan rather than re-solving from scratch.
- Adaptation resolves over-constrained problems by serving as an alternative to retraction.
- Adaptation relaxes preference constraints by minimally weakening them.
- Adaptation extends the vocabulary of a problem solver by creating new components as variations of known ones.

Using adaptation for all of these tasks allows four different kinds of design processes – selection, configuration, parameter fixing, and construction – to be integrated within one design architecture. And it provides a computationally efficient and parsimonious way of dealing with the introduction of new constraints during problem solving, a common design delemma. A paper entitled "The Integrative Role of Adaptation in Design" by Tom Hinrichs and Janet Kolodner give more details of this work. It was printed in the Proceedings of AAAI-91.

3.2 Evaluating results of case-based design

An important part of any problem solving endeavor is to evaluate the solutions that are created. This is particularly problematic when the problem is under-specified, as is almost always the case in conceptual and preliminary design. Comparison to other cases provides a way of evaluating solutions in these situations, but we must know on what dimensions to do the evaluation. Solutions must be evaluated on consistency and completeness.

The attached paper by Tom Hinrichs entitled "Some Criteria for Evaluating Designs," (in Appendix B), reports on this work.

Additional detail about all of these design topics can be found in the papers in Appendix B and in Tom Hinrich's Ph.D. thesis, "Problem Solving in Open Worlds: A Case Study in Design," included with this report.

4 Learning to Schedule

Another research activity was in the area of learning, particularly learning to schedule plan steps. We began by looking at scheduling a flexible manufacturing system. Operators of such systems start out doing fairly poorly at scheduling, and, in general, there are no teachers or manuals around to give them instruction. With a small amount of experience, however, they begin to get quite a bit better. Our interpretation is that this is a case-based activity – that they are explaining the failures that arise, figuring out which of their actions are responsible for those failures, figuring out which features of the environment can predict those failures, and indexing their experiences and new knowledge based on all that.

We chose to continue by working in an analogous domain, but one that was more accessible: scheduling the time of a busy single working parent. As in the flexible manufacturing domain, the planner must be reactive and opportunistic. Also as in that domain, we can assume that the planner begins by knowing how to achieve goals in isolation and learns over time how to integrate the achievement of several goals in tandem. Our work thus far has centered on a representation for plans that allows fast execution, supports noticing and acting upon expectation failures, evolves easily, and supports partial planning, reactivity, and opportunistic action. The papers in Appendix C address these issues.

Our program, called EXPEDITOR, begins as a novice scheduler and increases its performance in three ways. It becomes able to interleave tasks with ease, it becomes able to deal with common failures as they arise, and it learns new plans that help it carry out the activities that are necessary so that those failures will occur less often. The program contains cases, plans, scripts and goals pertaining to its weekday morning routine. It can be interrupted by events such as running out of milk or discovering that one of the kids is sick. It has mechanisms for devising alternative plans, executing those plans, and storing and indexing the experiences int its memory, and it can follow those indexes in later situations when it runs into similar problems.

The scheduling activity we are investigating in this project is important for several reasons. First, it will help us to discover how case-based reasoning functions in early learning of a task. This might allow us to build expert systems that start with little skill and through the analysis of their experience become more skillful. Second, this particular planning activity combines reactive planning, execution, case-based reasoning, and learning. It is a good task in which to study reactive planning, and gives us the opportunity to learn how a reactive planner might get better. Third, this task provides the bottom end of logistics planning. Logistics requires strategies and use of principles in scheduling. At some point, those strategies and principles have to be put to use. That is, the scheduling has to happen. As in other cases of logistical planning, this scheduler has to adhere to principles and strategies and figure out how to deal (within the guidelines of those principles) with the world as it actually is.

5 Creative Problem Solving

In case-based reasoning, and in much of the problem solving people do, solutions are created for new problems by adapting and combining known solutions to similar problems. Sometimes a new solution can be created by merely tweaking or adapting some old one in routine ways. Often, however, problem solving is less routine, requiring exploration of several alternatives, perhaps adapting and merging several possibilities gleaned from experience.

The case-based reasoning projects that are concentrating on problem solving have, for the most part, been looking at routine problem solving. That is, problems are solved by recalling similar cases and adapting them by well-

known procedures to fit the new situation. Our problem solvers come up with good, but boring solutions.

Previous experience seems to play a large role in the creative problem solving people do. It is time to make our case-based problem solvers more creative. Some investigations are doing this by looking at vocabularies that transcend several types of problems on the premise that creativity can emerge by transferring the abstract solution from those far-away problems and specializing it to the new problem (e.g., Kass, Owens, Birnbaum & Collins). In our investigation, we are looking at the creativity that emerges by combining the solutions to several cases in novel ways.

There are four processes we've found that extend basic case-based reasoning methodology to provide creativity in problem solving.

- **Problem specification elaboration**: In almost all AI investigations of problem solving to date, researchers have assumed that the problem specification would define the problem succinctly. Yet interesting problem solving domains have as their hallmarks the fact that problem specifications are rarely well defined. In general, they are incomplete, leaving many different ways to solve a problem. Sometimes, they are unnecessarily constrained. An important part of solving problems in these cases is redefining the problem spec. This includes elaboration, problem restructuring, and negotiating problem space boundaries (Goel & Perolli, 1989).

- **Exploration of solution alternatives**: In general, in problem solving in ill-structured domains, there are many different ways a problem could be solved, some better than others, some with different trade-offs than others. Many possible solutions must often be considered during problem solving. In addition, because problem specifications are often incomplete, the problem solver might need to incrementally converge on a good solution by following a cycle of starting with whatever comes to mind, critiquing that, and based on the critique, adding appropriate details to the problem specification. Alternatively, the problem solver might elaborate the original problem specification several different ways, using each to generate an alternate specification. This process includes both search of the solution space and evaluation of alternatives.

- **Merging of possibilities**: In routine problem solving, parts of several solutions are often merged, but in general, the parts do no overlap (e.g., dessert from one meal might be used with a main dish from another meal). In less routine problem solving, several suggestions for solving the same part of a problem might be merged to come up with a solution (e.g., in deciding to have salmon fettucine and salad for dinner, a meal planner might have remembered three previous cases, a meal with fish, a pasta meal, and a one-dish meal, and merged desirable features from each).

- **Non-routine adaptation**: Previous work has looked at adapting old solutions to fit new problems. In creative problem solving, it sometimes makes sense to adapt one's goals to fit an old solution rather than changing the old solution to fit the new problem. Previous work has looked at routine adaptation strategies (e.g., deletion, addition, substitution) but not at use of "off-the-cuff" ones (i.e., those developed in response to a particular problem). Some of these arise from examining a causal model, some from adapting well-known adaptation strategies, some from applying well-known adaptation strategies in novel ways. There may be other ways non-routine adaptations arise.

Our investigation of these processes has started with implementation of a program that addresses the second problem above: creative problem solving by exploring the case base. Our program is based on a protocol of a person doing creative design. In this case, the person was trying to come up with a dinner dish to use up some leftover white rice. The program, like our subject, retrieves an initial set of instances of using white rice, and evaluates each. Based on the evaluation, it alters the problem specification. Features that it doesn't have but needs, it adds from the case. Those features that it finds unacceptable in any case it remembers, it rules out. It then it takes this new problem specification and probes the memory again. It continues this cycle until it derives a solution that can be easily adapted to make a good solution, or until it retrieves a case that can be adapted to fit the new situation. When it remembers a Chinese dish, for example, the program rules out Chinese as a cuisine since it has just had Chinese food the day before. When it remembers a bread recipe that uses rice, it adds the specification that the recipe it comes up with should be easy to make, since the work involved in making bread is more than it wants to do. When it remembers a breakfast dish that uses

rice, it changes its goals completely and decides that maybe a dinner dish isn't necessary – breakfast would do just as nicely.

The process of creative problem solving through exploration of the case base, as we've defined it so far, has the following steps:

1. Retrieve an initial set of cases, using the initial problem specification
2. For each case:
    - Evaluate the case for its applicability or adaptability to the problem
    - Temporarily alter the set of goals, based on the current source case
    - Use the altered set of goals to retrieve an additional set of cases
3. go back to step 1

The ability to come up with creative, rather than routine, solutions to problems would help many of our case-based problem solvers (e.g., CHEF, JULIA, PERSUADER). The most obvious application of creativity in case-based reasoning is in those domains that are seen as inherently creative: design, for example, or meal planning, or architecture, and so on. But the approach we have introduced here can help case-based reasoners in many domains.

Any problem solver can reach an impasse as it works to solve a problem. Creative case-based reasoning provides techniques to breach such an impasse. By elaborating the original problem specification in more than one way, a creative problem solver can find different paths to solutions.

In a domain where the problem may be poorly defined, perhaps incomplete or overconstrained, the creative process can help elaborate the specification of the problem, and provide a more complete basis for problem solving.

Creative techniques can help a case-based reasoner produce variety in its solutions. Rather than using disjoint parts of different cases as the basis for a new solution, for example, a creative approach would merge corresponding parts of different cases to come up with a novel solution.

An understanding of creative problem solving processes has several important implications. If we understand creative processes, which parts are

hard and which are easy, we will be able to create the right kinds of tools to help problem solvers with their tasks. We will find out which processes can be relegated to a machine and which will need to continue to be done by people. This understanding will also help us in building the kinds of tools and developing the kinds of curriculums that can best be used to train creative problem solvers of the future.

The paper entitled "A Case-Based Approach to Creativity in Problem Solving" by Janet Kolodner and Louise Penberthy, in Appendix D, describes more about this project.

6 Case Representation

Cases can be large, and it is clear, from talking to people about how they use cases to solve problems, that people access and use pieces of cases even when a case as a whole seems far from the new case. For example, one physician interviewed in a protocol study remembered six different case pieces in the course of slving one complex problem: one helped him evaluate the ambiguous results of a test, another helped him evaluate the potential for growth of an aneurism, another warned of the potential for problems with a particular medical procedure he was considering, and three helped him determine if a set of symptoms that were present were important to take into account in solving the major problem. Each of these was only part of a larger case, and none of the larger caes were very good matches themselves to the new situation when taken as a whole. Designers also use pieces of cases as they solve problems. As they break large design cases into smaller parts to solve them, they recall similar smaller parts of old cses they help with the design.

There are two requirements for enabling recall of case parts.

- The representational scheme must make parts of cases easily accessible.
- The indexing scheme must index case parts as if they are independent units.

If we think of pieces of cases as cases themselves, then the representational problem boils down to a means of organizing related cases in such a way that they can be accessed in clusters as needed. That is, a whole case (consisting

of smaller pieces) must be as accessible as any of the parts of the case. There are two ways to do this:

- Represent cases monolithically with large cases containing their pieces as parts. This requires a scheme for locating appropriate case pieces within a larger case and an indexing scheme by which the whole case has as indexes both its own indexes and those of its pieces.
- Represent the pieces of large cases as cases and provide links allowing full cases to be reconstructed.

Most current case-based reasoners use the first method. One of our aims in CELIA, a case-based troubleshooter, has been to explore the second. Reports in Appendix E describe CELIA's scheme. In short, it adheres to the principles below.

- Cases are divided into subparts, called *snippets.*
- Each snippet represents pursual of some reasoning goal (or set of goals pursued in conjunction with each other).
- Each snippet contains information pertaining to pursuit of its goal(s). This includes the snippet's problem description, actions taken in pursuit of its goal(s) (real actions or mental actions), and pointers to related snippets.
- Links between snippets preserve the structure of the reasoning. Each snippet is linked to the snippet for the goal that suggested it and to the snippets for the goals it suggests.
- A full case is represented by a header that holds global information about eh case and a set of causally-connected case snippets.
- Each snippet is independently indexed by a combination of *global* information (i.e., pertenant features of the larger case it is embedded in) and *local* information (i.e., pertenant features of the snippet itself). Full cases are also indexed.

A snippet's problem description includes a pointer to the case header, the goal being pursued, and the problem solving context surrounding pursual of

the goal. The problem solving context describes the problem solving that has gone on previous to pursuing the snippet's subgoal, including relevant subgoals that have been pursued and the results of actions taken so far.

This representational scheme supports generalizations of episode pieces across different kinds of episodes. When cases are represented in individual parts, a memory can notice similarities between parts of cases and has the potential to make generalizations about case parts. For example, a system that represents meals in pieces has the potential to make generalizations about autumn desserts, vegetarian main course, or appetizers with cheese. It is much harder to make generalizations about similar pieces of dissimilar cases ina system that represents cases as monolithic wholes. Indeed, this is the rationale behind Schank's MOPs, a predecessor of case-based reasoning.

An issue that arises is how we break cases into their snippets. CELIA divides its cases according to reasoning goals because it is a reasoner whose purpose is to learn the reasoning goals associated with diagnosis. More recently, as builders of case-based reasoning systems have begun to attempt to represent cases of considerable size, a consensus has been developing that cases should be divided according to the lessons they teach. Above I mentioned a physician who remembered six different case pieces as he was solving a problem. Each was associated with some constellation of subgoals associated with an old problem and also present in the new situation, and each had a lesson to teach relevant to the subgoals it addressed.

Appendix E holds papers about case representation. Papers in Appendix F by Domeshek and Kolodner also address this issue.

7 Case-Based Design Tools

While much of the research work on this contract has been addressing issues in fully automating case-based reasoning, this part of the endeavor looks at the use of a case memory to aid people in decision making. Psychologists tell us that if people have the right cases available, then they are able to make case-based inferences accurately and easily. However, people often don't have the right cases available to them, either because they have not experienced them or because they did not remember them at the right times (due to indexing them wrong, bias of various kinds, etc.). A case-based decision aiding tool helps a person solve problems by augmenting his/her memory.

The ARCHIE project addresses the creation of case-based decision aids for designers. ARCHIE is an architect's associate. Work on this project is based on our experience with JULIA and on Ashok Goel's experience with KRITIK, a case-based designer of small mechanical devices that uses a causal model to aid with adaptation and evaluation.

The first version of ARCHIE is implemented using Cognitive Systems CBR Shell. It holds a handful of architectural cases. The tool provided a framework for us in setting up the case representations, and provided a vehicle for representing some of the causal models needed to analyze how closely two cases match each other.

The major issues we grappled with in designing ARCHIE were case representations for complex design and guidelines for dividing a case into useful parts. Because design cases are so complex, designers work on part of a case at one time keeping other parts of the case in mind but not concentrating on them. The representations we devise need to support that kind of reasoning. In addition, we looked at the ways causal models need to be integrated with cases to support design and we tried to make case representations support that integration. Just as cases need to be divided into parts, causal models also exist at several levels of detail that need to be connected to each other appropriately and also connected to the general case framework in a useful way. We also looked at the knowledge that is necessary in order to focus a presentation of a case appropriately for a designer, and at ways to get that knowledge from the designer without too much intrusion.

The papers in Appendix F discuss ARCHIE, both what it has shaped up to be and the problems we discovered as we continued building it. In short, ARCHIE was failure in terms of being a general purpose framework for a case-based advisory system. Analysis of its failures, however, has been most enlightening, and is pointing the way toward more tools. ARCHIE-2 deals with knowledge representation and knowledge presentation issues that our experience with ARCHIE pointed out as important.

Appendix A
The Indexing Problem

1. Kolodner, Janet L. (1988). Retrieving Events from a Case Memory: A Parallel Implementation. *Proceedings of the DARPA Workshop on Case-Based Reasoning*, May, 1988, Morgan Kaufmann Publishers, San Mateo, CA.

2. Kolodner, Janet L. (1989). Judging Which is the "Best" Case for a Case-Based Reasoner. *Proceedings of the DARPA Workshop on Case-Based Reasoning*, May, 1989, Morgan Kaufmann Publishers, San Mateo, CA.

3. Kolodner, Janet L. (1989). Selecting the Best Case for a Case-Based Reasoner. *Proceedings: 11th Annual Conference of the Cognitive Science Society*, August, 1989, Lawrence Erlbaum Assoc., Hillsdale, N.J.

Appendix B
Design

1. Hinrichs, Thomas R., (1988). Towards and Architecture for Open World Problem Solving. *Proceedings of the DARPA Workshop on Case-Based Reasoning,* May, 1988, Morgan Kaufmann Publishers, San Mateo, CA.

2. Hinrichs, Thomas R., (1989). Strategies for Adaptation and Recovery in a Design Problem Solver. *Proceedings of the DARPA Workshop on Case-Based Reasoning,* May, 1989, Morgan Kaufmann Publishers, San Mateo, CA.

3. Hinrichs, Thomas R., (1990). Green Eggs and Kosher Ham, Some Neglected Issues in Adaptation. *Proceedings: AAAI Spring Symposium on Case-Based Reasoning,* 1990.

4. Hinrichs, Thomas R., (1990). Some Criteria for Evaluating Designs, *Proceedings: 12th Annual Conference of the Cognitive Science Society*, July, 1990, Lawrence Erlbaum Assoc., Hillsdale, NJ.

5. Hinrichs, Thomas R. and Janet L. Kolodner, (1991). The Roles of Adaptation in Case-Based Design, *Proceedings: Ninth National Conference on Artificial Intelligence,* July, 1991 (Also printed in *Proceedings of the Case-Based Reasoning Workshop*, May, 1989).

6. Goel, Ashok K. and Eleni Stroulia, (1991). Model-Based Repair in Experience-Based Design, *Proceedings of the Model-Based Reasoning Workshop, Ninth National Conference on Artificial Intelligence, AAAI-91.*

Enclosed separately:

Hinrichs, Thomas R., (1991) *Problem Solving in Open Worlds: A Case Study in Design.* Ph.D Thesis, Report # GIT-CC-91-36. College of Computing, Georgia Institute of Technology, Atlanta, GA.

Appendix C
Learning to Schedule

1. Robinson, Stephen M., and David W. Wood, (1989). A General Vocabulary for Indexing Cases in Multi-Agent Domains. *Proceedings: 27th Annual Southeast Region ACM Conference*, 1989.

2. Robinson, Stephen M., and Janet L. Kolodner, (1991). Learning to Schedule Tasks in Complex, Everyday Environments. Presented at *the Symposium on Learning Methods for Planning and Scheduling,* Stanford, January, 1991.

3. Robinson, Stephen M. and Janet L. Kolodner, (1991). Indexing Cases for Planning and Acting in Dynamic Environments: Exploiting Hierarchical Goal Structures. *Proceedings of the 13th Annual Conference of the Cognitive Science Society,* July, 1991, Lawrence Erlbaum Associates, Hillsdale, NJ.

Appendix D
Creative Problem Solving

1. Kolodner, Janet L. and Theresa L. Penberthy, (1990). A Case-Based Approach to Creativity in Problem Solving. *Proceedings : 12th Annual Conference of the Cognitive Science Society,* July, 1990, Lawrence Erlbaum Associates, Hillsdale, NJ.

Appendix E
Case Representation

1. Redmond, Michael, (1990). What Should I Do Now? Using Goal Sequitur Knowledge to Choose the Next Problem Solving Step. *Proceedings: 12th Annual Cognitive Science Conference* , July, 1990, Lawrence Erlbaum Associates, Hillsdale, NJ.

2. Redmond, Michael, (1990). Distributed Cases for Case-Based Reasoning; Facilitating Use of Multiple Cases. *Proceedings of AAAI 90.*

3. Redmond, Michael, (1991). Improving Case Retrieval Through Observing Expert Problem Solving. *Proceedings: 13th Annual Cognitive Science Conference,* 1991, Lawrence Erlbaum Associates, Hillsdale, NJ.

4. Redmond, Michael, (1991). What Should I Do Now? Using Goal Sequitur Knowledge to Choose the Next Problem Solving Step. *Proceedings of AAAI-91.*

Appendix F
Case-Based Design Aiding

1. Kolodner, Janet L. (1991). Improving Human Decision Making Through Case-Based Decision Aiding. *AI Magazine,* Volume 12, No. 2., pp. 52-68.

2. Goel, Ashok K., Janet L. Kolodner, Michael Pearce, Richard Billington and Craig Zimring (1991). Towards a Case-Based Tool for Aiding Conceptual Design Problem Solving. *Proceedings of the DARPA Case-Based Reasoning Workshop,* 1991, Morgan Kaufmann Publishers, San Mateo, CA.

3. Goel, Ashok K., Janet L. Kolodner, Michael Pearce, Richard Billington and Craig Zimring (1991). An Experience-Based Approach to Cooperative Design. *Proceedings: AAAI Spring Symposium on Design of Composite Systems*, March, 1991.

4. Pearce, Michael, Ashok K Goel, Janet L. Kolodner, Craig Zimring, Lucas Sentosa and Richard Billington (1992). Case-Based Decision Support: A Case Study in Architectural Design. Submitted to *IEEE Expert, Special Track on Case-Based Reasoning*.

5. Domeshek, Eric A. and Janet L. Kolodner (1991). Toward a Case-Based Aid for Conceptual Design. To appear in *The International Journal of Expert Systems*, JAI Press.

6. Domeshek, Eric A. and Janet L. Kolodner (1992). A Case-Based Design Aid for Architecture. To appear in *Proceedings: AI and Design Conference*, 1992.