Date: 1/13/87

Project No.: T-87-111

Includes Subproject No.: 

Project Director(s): Raymond Miller

GTRC/GIT

Sponsor: U.S. Army Research Office

Title: PROBLEMS IN CASE-BASED REASONING

Effective Completion Date: 8/31/86

(Performance) (Report)

Grant/Contract Closeout Actions Remaining:

☐ None

☒ Final Invoice or Final Fiscal Report

☐ Closing Documents

☒ Final Report of Inventions Questionnaire sent to P.I.

☒ Govt. Property Inventory & Related Certificate

☐ Classified Material Certificate

☐ Other

Continues Project No. 

Continued by Project No.

COPIES TO:

Library

GTRC

ITC

ITC

Research Director

Research Administrative Network

Research Property Management

Accounting

Procurement/GTRI Supply Services

Research Security Services

Reports Coordinator (OCA)

Legal Services

FORM OCA 69-285

Army Research Office

Continued by Project No. 

Other: Ina Lashley

Angel Jones

Russ Embry
Dear Dr. Paris:

Please refer to Funding Document DAAG29-83-G-0016.

According to administrative records, your final report under the referenced funding document for the period July 1, 1983, through August 31, 1986 is now delinquent. We request that the report be submitted immediately.

We are enclosing a packet containing a sample layout and a number of blank document control (DD 1473) forms for your use in preparing the report. This report must be prepared in the exact format as the sample layout. We should be furnished 50 copies of the final report. In addition, we are enclosing a list of reports received at ARO during the period of your support by this office. If this list does not reflect all the reports written under our sponsorship, please send the necessary copies with the final report.

Thank you for your cooperation in this matter and for your interest in our basic research program.

Sincerely,

Richard O. Ulsh
Chief, Information Processing Office

Enclosures
January 8, 1987

Richard O. Ulsh  
Chief, Information Processing  
Office  
U.S. Army Research Office  
P.O. Box 12211  
Research Triangle Park, NC 27709-2211

Dear Mr. Ulsh:

Please find enclosed 50 copies of the report requested as per your letter of October 31, 1986, which is attached. Re: Funding Document DAAG29-83-C-0016.

In addition, 2 copies of a report by Roy Turner entitled, "Using Concurrent Processes for Advice Giving", are also enclosed. At a later time in the next month, Roy Turner will be completing another report entitled, "Issues in the Design of Advisory Systems: The Consumer-Advisor System", which will be sent to you upon its completion.

Thank you.

Sincerely,

Janet L. Kologner  
Associate Professor

JLK:mg
Enclosures
## Two Problems in Case-Based Reasoning

**Janet Kolodner, Roy Turner**

**U. S. Army Research Office**
Post Office Box 12211
Research Triangle Park, NC 27709

During the three years of this fellowship, students have studied the use of experience in problem solving. While problem solving is a much-studied area within Artificial Intelligence, traditional research in problem solving has not considered how repetition can be used to advantage by a problem solver. In the research conducted under this fellowship, we have been studying how a problem solver can take advantage of repetition and can use its experience to better its performance. The processes that capitalize on repetition are called case-based reasoning processes. Through case-based reasoning, a system can make use of reasoning shortcuts, can avoid previously-made mistakes, and can be alerted to the potential for problems. Work on this project has been mostly in the areas of plan modification and integrating memory and problem solving processes. A program, called the Consumer Advisor, or CAS, implements the ideas.
1. Research Summary

During the three years of this fellowship, students have studied the use of experience in problem solving. Problem solving is a much-studied area within Artificial Intelligence. For the most part, research within this area has concerned itself with what we call "from-scratch" reasoning. In from-scratch reasoning, a reasoner or problem solver has some general knowledge, often in a set of rules, that it uses to solve new problems that are presented to it. Means-ends analysis [Newell and Simon, 1960], problem reduction, non-linear problem solving [Sacerdoti, 1977], constraint satisfaction [Stefik, 1981] and backwards and forwards chaining are examples of this type of reasoning.

Artificial Intelligence researchers have shown that a great deal can be done by a computer using these methods. What traditional research in problem solving has not considered, however, is how repetition can be used to advantage by a problem solver. A from-scratch problem solver applies the same (possibly long and complex) methods the second or third time it sees the same problem or a similar one. From-scratch problem solvers have no way to recognize that they have seen the same or a similar problem previously, and therefore have no way to incorporate shortcuts into their processing.

In the research conducted under this fellowship, we have been studying how a problem solver can take advantage of repetition and can use its experience to better its performance. There are several requirements on a problem solver that can reuse experience. First, it must store its experiences in a memory so that they are accessible at appropriate times. Second, it must be able to recognize and evaluate the similarity of a current case to a previous one. Third, it must be able to focus its attention on potentially applicable parts of any previous cases it finds and extract appropriate guidelines from previous cases. Fourth, it must be able to transform what it finds in previous cases to fit its current case. Fifth, it must be able to judge the reliability for a new case of anything it borrows from an old case. Sixth, it must know when it is appropriate to use advice from a previous case and when from-scratch methods might be more appropriate.

The sum of these processes is a method of reasoning we call case-based reasoning [Kolodner et al, 1985; Kolodner and Simpson, in press]. Through case-based reasoning, a system can make use of reasoning shortcuts, can avoid previously-made mistakes, and can be alerted to the potential for problems. Within our research group, much effort has been spent on examining processes of case-based reasoning (see, e.g., the papers referenced above and Kolodner, 1985, 1986). Along with the work done under this fellowship, research on these topics is supported by grants and contracts from ARO and NSF. Much of the basic research done to determine how to give the computer these capabilities has been supported under those two grants, including work on memory organization, criteria for similarity, and integrating problem solving and learning processes. Under the first two years of this fellowship, work was directed at implementing several systems and parts of systems that employ case-based reasoning. In the third year of the fellowship, we had a student who was sufficiently advanced to contribute both an implementation and ideas of his own about case-based reasoning processes.

There have been three students supported by this fellowship, each working on issues in case-based reasoning, but each working on different issues and each in a different domain. The three students and their topics were as follows*:

Year one: Dana Eckart worked in the domain of diagnosis, in particular diagnosis of affective disorders (depressions) of various kinds. His major concern was in representing cases and features of cases (in this case, symptoms of depression) and in designing a memory that could organize cases in a useful way and use them to drive the diagnostic process.

Year two: Kirt Pulaski worked on a common sense domain: planning business trips. His major concern was the design of a frame system tool that could both store generalized knowledge (as frame systems traditionally do) and also organize cases.

Year three: Roy Turner's work is in another common sense domain: giving advice about acquisition of household products. He has had two research concerns: integrating concurrent problem solving and memory processes, and transforming a previous plan to fit a new situation. In the process of addressing these issues, he has also concretized many of the things that had been worked on by the previous two students supported under this fellowship, and he has created a frame system (program) that has been useful to our entire research group.

* While we know it was your intention that this fellowship go to one exceptional student for a full three years, we were unable to do that. Unfortunately, the first student chosen decided after a year that he would rather do systems than AI, and the second decided to leave the Ph.D. program and go into industry. The third student, however, can be considered a success story, and has done quite a bit with your support.
Roy Turner can be considered the success story of this fellowship. He has both created a substantial program that implements many of our ideas about case-based reasoning, and he has done substantial work in the areas of plan modification [Turner, 1986] and integrating memory and problem solving processes [Turner, in press]. His program, the Consumer Advisor, or CAS [Klodner, 1986; Turner, 1986, in press], and the papers he has written about it, comprise the research results of this contract, which we summarize below. For more information, see the referenced papers.

1.1. Plan Modification

CAS' means of planning is by plan instantiation, and it treats general plans (e.g., a plan for building a bookshelf) and plans used in previous episodes (e.g., the particular set of steps Pete used in building his bookshelf) in exactly the same way. Any plan, whether a general one or a specific one, is made up of steps, and each of those steps may have sub-steps. Plans for a new situation are constructed by either placing together plan pieces or modifying a composite plan (whether general or specific) to fit the new situation. When a new situation is encountered, the problem solver is given the most specific composite plan (general or specific) available, and if possible, modifies it to fit the new situation. If no composite plan is available, it pieces together plan steps.

Because new situations almost never match previous ones exactly, a problem solver that makes use of previously-used plans must be able to flexibly adapt and modify those plans [Carbonell, 1986]. A past plan that is recalled during problem solving and that is not completely applicable to the current situation shouldn't be rejected without first trying to patch it so that it can be applied.

Our solution to this problem is novel in two ways. First, preconditions are not all treated as conditions that must be fulfilled to attempt a plan. Some, marked as relative preconditions, are treated rather as conditions for good fulfillment of the plan. If they are violated, the solution won't be as nice as if they are fulfilled, but solution is still possible. A relative precondition has associated with it knowledge (compiled from experience using the plan in the past) that predicts the likely outcome should the violation be ignored. This allows the planner to decide if it is worthwhile modifying a plan to eliminate a violation, or if the plan should be applied as is. This also allows the planner to recommend less than optimal plans to the user when those are the only plans available.

The second novelty in our solution is that all preconditions, whether relative or absolute, are not treated just as conditions that must be fulfilled, but rather have a great deal of planning knowledge associated with them. Each precondition to a plan has the following information stored with it:

1. the state that must exist for the precondition to be satisfied;
2. the plan step or steps that the precondition is required for;
3. the likely result of applying the plan if the precondition is violated; and
4. directives for modifying the plan so that the precondition must no longer be fulfilled.

Each of these items plays a role in the planning process. Item 1 supplies criteria for recognizing whether or not a precondition is satisfied. Items 2 and 3 are used in evaluating whether to modify the plan or ignore the precondition problem. Item 4 is used to fix the plan if the planner decides that the precondition violation is unacceptable.

Directives for plan modification (Item 4) provide heuristics for changing the plan so that the offending precondition is either no longer violated or no longer necessary. There are two ways a plan can be modified: Strategic modifications change the overall strategy of a plan. Tactical modifications change local plan steps. Examples of strategic changes include (1) adding steps to the plan (i.e., planning to achieve the precondition); (2) deleting steps from the plan (e.g., changing the plan so that the offending step is no longer there); and (3) re-ordering steps of the plan. Examples of tactical changes are (1) replacing a step with another plan or action; and (2) changing a step within some step of the plan.

Consider, for example, a plan for building from wood. In the general plan, one decides on a design, buys the necessary materials, cuts them to size, assembles them, and does the finishing work. One precondition for using this plan is that the builder have carpentry experience. This condition is required for the cutting and assembly parts of the building (Item 2). The likely result of building if this plan is violated is a shoddy-looking item (Item 3). One way to get around this problem is to ask an agent with carpentry skill to do the assembly instead. This is represented in Item 4 as: IF there is someone available who can do carpentry, THEN delete the cutting step, delete the assembly step, insert a step where that
somebody does the cutting, and insert a step where that somebody does the assembly.

Thus, if a problem solver is attempting to plan for bookshelf building, but the user has no knowledge of carpentry, it might ask the user if somebody with carpentry skill is available to help, and then follow the set of steps above to omit the offending steps from the plan (cutting and assembling the bookshelves) and replacing it with the alternate plan. By making this change, the precondition is no longer applicable, and the new plan can be used to solve the problem.

In the same way, a previously-used plan can be reused and modified appropriately at a later time. Suppose, for example, that a later user wants to build bookshelves but has no carpentry skill. CAS is reminded of the plan it created above and attempts to apply it to the new situation. That is, it asks the new user if someone with carpentry skill is available to help, and if so, attempts to use the previous plan. If, in this case, another precondition is also in danger of being violated (perhaps, for example, this person doesn’t know how to choose a design), then the plan will be modified taking that problem into account. Replanning for lack of carpentry skill, however, is not necessary.

1.2. Integrating Memory and Problem Solving Processes

In protocols we have taken of human problem solvers, we have observed that planners often interrupt themselves to turn their planning in a different direction [Kolodner, 1988; Turner, in press]. Planners seem to interrupt themselves for two reasons: (1) their current planning isn’t getting them anywhere for some reason, or (2) they are reminded of something that makes them think the planning will work better in a different direction. In one of our examples, for instance, a person was reasoning about acquiring bookshelves for very little money. He thought about buying them and got to a dead end because he didn’t know what they cost. He then started thinking about building them, and was reminded of a specific building episode and began making suggestions based on that. Our interpretation of this type of processing is that at least two separate processes are running in parallel: a memory process and a planning process. When memory is reminded of something pertinent to the planner, it interrupts the planner, and the planner might decide to use the episode the memory is reminded of to redirect its planning. At the same time, memory uses the problem the planner is working on guide its search.

CAS implements this architecture using three modules that run concurrently. Its memory process is responsible for searching CAS’ memory to provide pertinent plans and experiences for the planner to use. The planner, a separate process, can be interrupted by memory whenever memory is reminded of something. Communication between these two processes is controlled by another process, the referee, which acts as an intermediary between them. When the memory is reminded of a plan or a previous episode, it sends the memory structure to the referee, which posts it on the system’s blackboard. The referee then notifies the planner that a reminding has occurred. Communication with the user is provided via a fourth process, the user interface. The user’s utterances are processed by the interface, then passed along to the referee for posting to the blackboard. Similarly, should the planner decided to report to or question the user, it sends its request to the interface via the referee.

1.2.1. Memory

The memory used in CAS is patterned after Kolodner’s (1984) CYRUS program, which is an instantiation of a dynamic memory, as described by Schank (1982). When searching itself, it creates a search key from the problem the planner is working on that then uses any information available in that search key to direct its search. This includes the problem statement (in as much detail as is available), the plan that is the current focus of the planner’s attention, any feedback provided by the user, etc. As the planner modifies the problem statement or changes its focus, the memory reformulates its search key and continues its search with the reformulated key. It is thus always searching based on the planner’s goals and problems. When it finds an episode or generalized knowledge pertinent to the current problem, it passes it along to the planner. CAS searches its memory in parallel. It spawn a separate memory daemon for each of the searchable knowledge structures referred to in its search key. There can thus be several of these daemons running simultaneously.

1.2.2. Planning

As stated previously, CAS’ planner uses a plan instantiation strategy. If the memory returns a plan, the planner modifies it as described above and uses it for the current situation. If memory returns an episode, the planner creates a plan from it and modifies it by the same procedures if necessary.
The planner consists of a scheduler and an agenda of planning demons, coroutines that can each process information from a single reminding. There can be several different types of information contained in remindings: plans, episodes, etc. The planner makes opportunistic use of remindings as they occur by creating a demon appropriate for the type of information contained in the reminding and adding it to the agenda. The highest-ranked demon is then run. Demons are ranked according to the worth of the memory structure they are processing. Should a new reminding occur that results in the creation of a higher-ranked demon, work on the currently active demon is suspended and the new demon is allowed to run. At a later time, the new demon may lower its own rank, or it may finish; in either case, the old demon can then run again.

The demons act as experts about the memory structures they process, and they are responsible for setting their own priorities as processing unfolds. A demon can change its priority for various reasons: a problem goal is not satisfied by the potential solution the demon is considering, a preservation goal is violated, etc. When a demon lowers its priority, attention is directed to other, higher-priority demons. Without explicitly specifying a control strategy, nonetheless the scheduler and the demons conspire to perform a best-first processing of remindings from memory.

2. Bibliography


