Capitalizing on Failure Through Case-Based Inference*

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

Abstract

In case-based reasoning, previous reasoning experiences are used directly to solve new problems and make inferences, rather than doing those tasks from scratch using generalized methods. One major advantage of a case-based approach is that it can help a reasoner avoid repeating previously-made mistakes. When the case-based reasoner is reminded of a case in which a mistake was made, it provides a warning of the potential for a mistake. If the previous case was finally solved successfully, it can provide a suggestion of what to do instead. In this paper, we describe the process by which a case-based reasoner can take advantage of previous failures. We illustrate with cases from the domains of common-sense mediation and menu planning and show a program called JULIA reasoning in the domain of menu planning.

1. Introduction

Over the past several years, there has begun to be a great deal of interest in case-based and analogical reasoning (e.g., Alterman, 1986, Ashley, 1986, Carbonell, 1983, 1986, Hammond, 1986, Holyoak, 1984, Kolodner, et al., 1984, 1985, Rissland, 1986, Simpson, 1985). Case-based reasoning is a problem solving method in which previous reasoning experiences are used directly to solve a new problem, rather than solving the problem from scratch using generalized methods. The major advantages of a case-based approach are that it can provide shortcuts in problem solving and that it can help a reasoner avoid repeating previously-made mistakes.

We shall see that previous failures serve several purposes during problem solving. They can provide warnings of the potential for failure in the current case, and they may also provide suggestions of what to do instead. Analyzing the potential for failure in a new case, a necessary part of capitalizing on an old failure, may require the problem solver to gather additional information, thus causing the problem solver to change its focus of attention. A previous failed case that was finally solved correctly can help the problem solver to change its point of view in interpreting a situation if that is what is necessary to avoid potential failure.

We shall illustrate the processes involved in capitalizing on failure using examples from two domains: common-sense mediation of everyday disputes and menu planning. Case-based resolution of common-sense disputes is implemented in the MEDIATOR (Kolodner, et al., 1985, Simpson, 1985), an early case-based reasoning program. JULIA (Cullingford & Kolodner, 1986) interactively solves problems in the catering domain. The processes that capitalize on failure are implemented in JULIA.

2. Background

In the simplest case, making a case-based inference involves the following steps:*

* This work is supported in part by NSF under Grant No. IST-8317711 and Grant No. IST-8608362, by ARO under Contract No. DAAG29-85-K-0023, and by ARI under Contract No. MDA-903-86-C-173. Programming of the examples, and much work on analogical reasoning that is incorporated into JULIA's case-based reasoner was provided by Hong Shinn. Discussions with other members of the AI Group, past and present, have also been useful.

* Each of these steps, of course, is a complicated process. For more information about step 1, see Kolodner (1983), Hammond (1986), Holyoak (1984), Schank (1982); about step 2, see Kolodner, et al. (1985), Simpson (1985); for step 3, see Alterman (1986), Ashley (1986), Carbonell (1983, 1986), Hammond (1986), Kolodner (1985, 1986), Kolodner et al., (1985), Rissland (1986), Simpson (1985); for step 4, see Simpson (1985).

1. Recall a relevant case from memory
2. Determine which parts of that case are appropriate to make the necessary problem solving decision for the new case (i.e., focus on appropriate parts of the previous case)
3. Achieve the targetted problem solving goal for the new case by making an inference based on the old case
4. Check the consistency of what is derived in step 3 to the new case

Consider, for example, the following case:

Avocado Dispute 1

> A problem solver is attempting to resolve a dispute over possession of an avocado. Two people want it. The problem solver is attempting to fill in the underlying goals of the disputants (i.e., why does each want the avocado?). It is reminded of a dispute in which two kids wanted the same candy bar. They both wanted to eat the candy bar, and the reasoner compromised by dividing the candy bar equally between them, having one divide it and the other choose his half first.

The problem solver has already been reminded of another case (step 1). Because the problem solver's goal is to infer the underlying goals of the disputants in the avocado case, it focuses on the underlying goals of the disputants in the candy dispute (step 2). They both had the goal of eating the whole candy bar. This goal was inferred through a *default-use* inference. The reasoner makes the case-based inference that the disputants in the avocado dispute also want to eat the disputed object (i.e., the avocado) (step 3). Because this hypothesis is consistent with what is already known about the case (step 4), the representation of the case is updated to include this inferred knowledge.

When a recalled case resulted in failure, however, reasoning is not as straightforward. Consider, for example, the following:

Avocado Dispute 2

> A problem solver is attempting to resolve a dispute over possession of an avocado. Two people both want it. The problem solver is trying to infer the underlying goals of the disputants. This time it is reminded of a case where two sisters both wanted the same orange. The problem solver in that case inferred the sisters' goals by using a *default-use* inference to infer that both disputants wanted to eat the orange. It turned out, however, that the goal of one of the disputants was to use the peel of the orange to bake a cake. The *default-use* inference applied to the orange as a whole led to selection of the wrong plan for resolution of the conflict, and the plan failed. We shall call this part of the case *orange-dispute-f*.
>
> The problem solver reinterpreted the dispute and solved it. The goals of the sisters were amended: one wanted possession of the fruit of the orange, the other its peel. Their underlying goals were also amended: one wanted to satisfy hunger by eating the fruit, the other wanted to bake with the peel. It finally resolved the problem by dividing the orange in a better way. One sister was given the fruit and the other was given the peel. We shall call this part of the case *orange-dispute-s*.
>
> The problem solver also analyzed its failure in *orange-dispute-f*, and added its analysis to its memory of that case: Failure was due to a *wrong-goal inference*. *Default use* applied to the entire disputed object (orange) resulted in failure, while *default use* applied to parts of the orange (the peel and the fruit) would have resulted in success.

Suppose now that the problem solver is reminded of *orange-dispute-f*, the case that resulted in failure. This case acts as a warning to the problem solver of the potential to make a faulty inference in the current case. It must check to see if the inference used previously would also result in error in the current case. The question that must be asked of the avocado dispute based on analysis of the orange dispute is whether an avacado also has parts used for different purposes that might predict the goals of the current disputants better than if they were computed by applying *default-use* to the whole avacado. In other words, based on its reminding of *orange-dispute-f*, which failed, case-based

reasoning alerts the reasoner to the fact that if the disputed object has several parts, the goals of the disputants may have something to do with the parts and not necessarily with the avocado as a whole.* The potential for failure is flagged and two alternative solutions are presented.

Errors in reasoning can happen during any problem solving step. The problem might have been misunderstood initially, resulting in incorrect classification of the problem or incorrect inferences during the problem elaboration phase. Since problem understanding is an early part of the problem solving cycle, such misunderstandings and incorrect inferences propagate through to the planning phase, resulting in a poor plan. A problem might be understood correctly and all the necessary details known about it, but might still be solved incorrectly because poor decisions were made while planning a solution. In general, such errors are due to faulty problem solving knowledge. The problem solver might not have complete knowledge, for example, about under what circumstances a particular planning policy or plan step is appropriate. Finally, a problem might be solved correctly but carried out incorrectly by the agent carrying out the plan, or unexpected circumstances might cause execution to fail. Reminding of a case where any of these things happened warns the the problem solver of the potential for the same type of error in the new case. If the previous case was finally resolved correctly, details of its correct resolution suggest correct decisions for the current case.

3. Some Problem Solving Assumptions

Before presenting the set of processes that capitalizes on previously-failed cases, we briefly present the relevant parts of our problem solving paradigm. First, when we refer to problem solving, we include the entire cycle of understanding a problem and elaborating its features, coming up with a plan for its solution, executing that plan, analyzing the results, and if necessary, going back to the beginning and trying again. Our own previous work (Kolodner, et al., 1985, Simpson, 1985) and that of others (e.g., Hammond, 1986) has shown that case-based inference can be used for a variety of tasks during any of these problem solving phases.

The second important assumption of our paradigm is that memory access and problem solving are happening in parallel (Kolodner, 1985, Kolodner & Cullingford, 1986). The memory's job is to integrate the case that is currently being reasoned about into the memory that already exists (Schank, 1982), resulting in remindings. Memory can return generalized knowledge (e.g., knowledge structures or rules) for the problem solver to use or a previous case that is similar to what the problem solver is currently dealing with. As the problem and its solution are further elaborated, memory is able to recall both more relevant general knowledge and better related cases for the problem solver to use.

Our third important assumption is that case-based reasoning is happening in the context of a set of reasoning goals and that, in addition to the case-based reasoner, other reasoners are also keeping track of those goals and making any suggestions they can. Thus, in addition to the case-based reasoner, a problem reduction problem solver might be available to break the problem into smaller parts, while a constraint propagator might do forward chaining inferences, and a truth maintenance system might be checking for inconsistencies and constraint violations. Something we'll call the overall problem solver keeps track of reasoning goals and subgoals as they come up, and each of the reasoners watches the goal network and attempts to achieve any goal it can.

Finally, the processes we present below assume that reminding has been of the failed part of a case that might have been resolved correctly later. In the case of the orange dispute, for example, we assume reminding has been of the episode that failed, *orange-dispute-f*. Reminding during problem solving may be of either the successful or the failed version of any case. When reminding is of the successful instance of solving it, the faulty reasoning that preceeded the successful solution is bypassed and a good solution is suggested immediately. The problem solver is never alerted to possible problems. Only when reminding is of a failed attempt at resolving a case is the problem solver alerted and the analysis described below done.

---

* It may be judged in this case that inference based on the parts is inappropriate (since one rarely plants avocado seeds).

4. The Process

Given this set of assumptions, we see that the problem solver might be reminded of a previous case that resulted in failure any time during problem solving. Because of this, the processes that capitalize on previous failures must be applicable during any part of the problem solving cycle. The following set of steps are executed any time during problem solving that a failed case is recalled.

1. Determine whether the failed case was ever followed up on, and, if so, recall the entire reasoning sequence that followed it.

This step makes alternatives that were attempted previously to solve the recalled problem available to the problem solver.

In the representation we are currently using, each full analysis of a problem is kept separately with pointers between them. Thus, the representation for a case that failed and was reanalyzed, such as the orange dispute, is actually represented as two cases. The first is the one that failed (*orange-dispute-f*), where one set of assumptions was made about the goals of the disputants. That one includes the mistaken problem description, the suggested plan (cut it in half), feedback after suggesting or carrying out that plan (after suggesting that the orange be cut in half), and the analysis of what went wrong (a *wrong-goal-inference*). The first (failed episode) also includes a pointer to the next problem solving episode, i.e., the reasoning that is carried out to solve the problem after the failures of the first episode have been diagnosed and repaired. Thus, *orange-dispute-f* points to *orange-dispute-s*, where the problem is described as one where the disputants have the second set of goals, and the solution plan that goes with that (divide agreeably) is recorded.

2. Recall or determine what was responsible for the previous failure.

In some instances, responsibility for failure will already have been attributed during previous reasoning. In that case, this step is an easy step of retrieving the error attribution from the representation of the case. In other instances, there might not have been any analysis of why the previous problem occured. When this happens, it is appropriate for the problem solver to try to figure out why the previous error happened. We do not go into that process in this paper.*

In general, failures happen because some inference was made incorrectly or not made at all. This might be due to faulty or missing information about the problem itself, or faulty or incomplete problem solving knowledge. An analysis of a failure may record only which inference was made incorrectly or was not made, or it may record the reasons why the inference was made incorrectly. As we shall see, the better an analysis of a previous failure is, the more the problem solver will be able to capitalize on the failure. The best analysis of a failure will record reasons for faulty reasoning all the way back to a point in the reasoning where it could have been corrected, i.e., where the missing or faulty information can be obtained or fixed. For example, failure in *orange-dispute-f* can be traced to a *wrong-goal inference*. The goals were inferred incorrectly. The reason for this is that *default-use* was applied to the wrong object (i.e., to orange as a whole rather than the parts of the orange). The reason for this is that the problem solver was viewing the orange in the wrong way: as a whole rather than as a thing with functional parts. If the reasons for this inference error are recorded to this level, then by using this case and following the set of steps to be presented, the problem solver will be able to consider whether some other object might be better viewed as a thing with functional parts. If only the fact that the goal was inferred incorrectly were recorded, it would not have as much to go on, but would only be able to consider if there is another goal associated with the object.

3. Determine the relationship of the decision currently being focussed on to the previous failure and refocus as required:

* If responsibility for failure is not known at the end of this step, it is still possible to capitalize on the failure.

(a) Was the decision analogous to the one the problem solver is currently trying to make responsible for the failure? If so, maintain current problem solving focus.

(b) If not, was the decision analogous to the one the problem solver is currently trying to make dependent on the one responsible for the failure, or alternatively, did the value the problem solver is currently attempting to derive change in the final solution to the problem? If so, refocus the problem solver on the decision analogous to the one that was responsible for the previous failure.

(c) If not, then refocus as in (b) to be careful or maintain current focus to be fast.

When the decision the problem solver is currently trying to make was responsible for the previous failure (i.e, the answer to 3(a) is yes), then more effort must go into making that decision. This is the case in *avocado dispute 2*. The problem solver has the goal of inferring the goals of the disputants, and it was this decision that was responsible for the failure in *orange-dispute-f*.

The more interesting cases, however, is when the answer to 3(b) is yes. In these cases, some decision other than the one currently being attempted was responsible for the previous failure. The problem solver will have to *refocus* itself on that decision, and (re)make it for the current case before continuing. Consider, for example, the following:

Panama Canal Dispute

> Both Panama and the United States want possession of the Panama Canal Zone. The problem solver is attempting to figure out how to classify the dispute. The problem solver is reminded of the dispute between Israel and Egypt over the Sinai. Both wanted the Sinai, and the problem solver had originally classified it as a *physical dispute* over possession of the land. It had therefore suggested that they cut it down the middle and share it. Both Israel and Egypt balked. On further analysis, the failure of this suggestion was tracked down to a set of *missing-goal inferences*. The goals of Israel and Egypt with respect to the Sinai had not been inferred. Israel wanted military control of the area for security reasons, while Egypt wanted possession of the land itself for reasons of national integrety. This interpretation makes the dispute into a *political dispute* rather than a physical one, i.e., one for which political alternatives are suggested rather than alternatives having to do with the physical object itself.

Responsibility for the failure in the previous case (the Sinai Dispute) had already been tracked down to missing goal inferences. The problem solver is currently attempting to decide what kind of dispute it is (e.g., physical or political?). The original classification of the Sinai Dispute as a physical dispute was not per se the reason that solution failed. Rather that decision was based on the goals of the disputants, which had been inferred incorrectly previous to attempting classification. The physical classification, however, changed to political in the final analysis, and was dependent on what was responsible for the failure in reasoning. Remindiing of the Sinai Dispute should *refocus* the problem solver on the set of decisions that were responsible for its failure, namely inference of disputant goals.

If the decision being focussed on at the beginning of this set of steps was a correct one for the previous case and if it did not change when the case was reanalyzed (case c), there is no reason why the problem solver must consider the previous failure at all. However, a careful problem solver will also consider whether that failure is possible in the current environment, thus refocusing itself on whatever caused the failure previously before going on.

In cases where the problem solver changes its focus, it continues by trying to redo the task that could have been made in error, following the set of steps below. If the problem solver changes a decision it had made previously, then it must also remake any decisions that depended on it before going on. After this set of steps is complete, the problem solver must refocus appropriately to finish solving the problem. Processing that happens in the course of recomputing already-made decisions may direct the problem solver in different directions than it had been planning when it was interrupted by the failed case. On the other hand, if there are no other recomputations to be made or if no other problem solving directions are suggested, the problem solver continues after this step as it had been planning originally. That is, it goes back to the goal it was working on when it reached this step and continues from there.

The processing that happens after step 3 depends on whether or not a successful solution was ever found in the previous case and whether or not analysis can be or has been done of the previous failure. If there was neither a solution found to the previous problem nor an explanation of the previous failure, then only an analysis of the potential for failure can be contributed by the the previous case. And, if there is no explanation of the failure, then less can be contributed than if there is an explanation. With an explanation, we know what features of the previous case were responsible for the failure and we can check for the presence of those in the new case. Without that explanation, we can use the justifications for previously made inferences and see if they hold in the new case, but such analysis is in a sense "superstitious" since no causal explanation available.

4. Recall the inference rules and justifying conditions used to infer the focused-on portion of the failed case. IF there was followup, THEN also recall the inference rules and justifying conditions used to infer the focused-on portion of each of the followup cases.*

The inference rules and justifying conditions of any failed cases will be used to check for the potential for failure in the current case. Those from the successfully-resolved case will be used to guide the problem solver to a correct decision.

In the case of *orange-dispute-f*, the inference rule used to infer the goals of the disputants was *default-use* applied to the disputed object. It is justified by its preconditions, i.e., there is an object of current interest (the orange) that has a default use (eating). It might also have been justified by its use previously in the candy dispute, where it worked fine. For *orange-dispute-s*, there were two inference rules used to infer the goals of the disputants. In one case, *default-use* was applied to the fruit of the orange, in the other it was applied to the peel of the orange. The fruit and peel of the orange are its major parts and each are used for different purposes.

5. Check to see if there is the same potential for failure in the new case. This is done by a variety of methods. We list two here.
   (a) Check the reason why the reasoning error was made in the first case. An error can be made because of incomplete information, because of faulty information, because of a faulty inference rule, or because of faulty focus (which might itself be tracked down to one of these causes).
   (b) Determine if the justifying inference rules and conditions from the failed and successful cases also hold in the new case.

Let us consider (a) first. This is the way we determine potential for failure in a new case if we know why the previously-made decision failed. If a previous reasoning error was made because of lack of knowledge, the appropriate knowledge is now sought for the current case. If it was because of faulty information, this step will require clarification of the analogous knowledge in the new case. If it was because of a faulty inference rule, that rule will be ruled out in this case. And if it was because of faulty focus (probably due to one of the other types of error), a suggestion will be made from the previous case of where to focus in the new case. Analyzing the orange dispute using this step, we find that the reason for the *wrong-goal-inference* was faulty focus. Focus had been on the orange as a whole while it should have been on its functional parts. The suggestion is thus made to focus on the functional parts of the avacado, rather than the avacado as a whole in inferring the goals of the disputants with respect to the avacado. As in the analysis of the orange dispute from above, in the next steps, the reasoner will either ask the disputants which parts of the avacado they are interested in or will decide that the only functional part that is worth considering is the fruit.

When there is no knowledge about why a previously-made decision was in error, the best that can be done is to evaluate whether conditions that led to that decision are also present in the current case. This is case (b). These

* Recall that the problem solver might have refocused its goals in the last step, so the portion of the case being focused on now might not be the one originally considered.

conditions can be found in the justifications for the value that was computed previously. If justifications of both the failed and the successful decision are applicable in the new case, an evaluation must be done of which is best. In *orange-dispute-f*, for example, the goals of each disputant were computed using a *default-use* inference applied to the disputed object. Justification for the *default-use* inference comes from its antecedent clause, which asks whether there is some major default use for the object in question that has an "obvious" goal associated with it. An orange and an avocado, of course, both have the same default use (eating) and "obvious" goal (satisfy hunger). In *orange-dispute-s*, the goals of each disputant were computed using a *default-use* inference applied to the functional parts of the disputed object. Justification for this application of this inference rule is a combination of the justification for choosing the objects to be focussed on (the disputed object has functional parts) and the antecedent clause of *default-use* applied to each of those parts.

Using the orange dispute as a model for the avocado dispute, justifications for each of the goal decisions made in resolving that dispute are evaluated with respect to the avocado dispute. Since the avacado has a default use (eating), the inference from *orange-dispute-f* can be made. Since it also has parts with default uses (the fruit is eaten while the seed can be planted), the inferences from *orange-dispute-s* can also be made. In this case, further evaluation is needed to determine which way to make the inference. While case-based reasoning, in this case, does not provide an answer, it does warn of the potential for misinterpreting the case and it also provides suggestions of alternate interpretations. It thus acts as a *preventive measure* to aid in avoiding failure.

It is interesting to note that the knowledge necessary to do the computations just described may not yet have been considered (e.g., the problem solver may not have considered if an avacado has parts used for different purposes). Sometimes, gathering appropriate knowledge consists of just an easy question to the user. In some cases, however, answering the questions posed in this set of steps may require significant reasoning. This extra computation, while significant, is done only when a previous case points to the need to look out for a problem. As we stated previously, it is a preventive aid in avoiding failure.

The output of this step is an evaluation of whether the previous failure could happen in the new case, and if the previous case was solved successfully, an evaluation of whether the previous successful solution is applicable to the new case. Based on these two evaluations, the reasoning continues.

6.

(a) If the previous failure will not repeat itself in the new case, go on with the problem solving. The (failed) suggestion from the previous case can be transferred to the new case if there is some independent reason that it can be supported or or a decision can be made independent of the recalled case.

(b) If the previous failure could repeat itself in the new case, rule out the inference rule or value used previously for the new case.

(c) If the previous successful solution is judged applicable to the new case, use it and apply case-based reasoning methods to derive a value for the new case based on it.

(d) If the previous successful solution or any of the interim solutions from the previous problem are judged inapplicable to the new problem, rule them out for the new case.

(e) If both the failed and successful solutions to the previous problem are judged applicable to the new one, use some decision-making procedure to decide between them.

5. Case-Based Inference in JULIA

In the following problem solving session, we see JULIA following the set of steps above to capitalize on a previous problem solving failure. JULIA (Cullingford & Kolodner, 1986) is designed to be an automated colleague whose task is to help a caterer design a meal. JULIA's problem solving components include a case-based reasoner, a problem reduction planner, a constraint propagator, and a reason-maintenance system. It also has a memory for events. Each decision JULIA makes when it is solving a problem is recorded along with the justifications for the decisions. Thus, in later problem solving, those justifications are available to use in case-based reasoning.* In the first case-

* See Kolodner (1986) for a description of the representational support for these processes, especially the content and structure of the justifications JULIA maintains.

based reasoning example, we see JULIA using a previous case to avoid serving a spicy Mexican meal to people who don't eat spicy food. The previously-failed case, which failed because of a lack of this information, causes JULIA to gather the appropriate information in the new case, thus letting it plan for those people immediately. This makes JULIA retract two previously-made decisions (the entree and the cuisine). JULIA refocuses itself on the cuisine, retracts and remakes that decision, and continues with its problem solving.

The dialog begins with JULIA introducing itself and the user stating her problem. We state the problem in English here. Actual input for this problem is a conceptual representation.

```
<JULIA> Hello. I'm JULIA, a Caterer's Advisor. May I help you?
USER =>
 I'm having a dinner next week for my research group.
 There is no room for all of them in the dining room.
<JULIA>
*** The initial problem is:
***** Frame #<FRAME 34502032> *****
Name: M-MEAL774
   Isa: (<M-MEAL>)
   Category: INDIVIDUAL
   Slots:
      ACTUAL-RESULT: NIL
      CHARACTERS: (?HOST ?GUESTS ?PARTICIPANTS)
      CONSTRAINTS: (<C-LIMITED-SPACE778>)
      DEFINED-SLOTS: NIL
      DESCRIPTOR: NIL
      EXPECTED-RESULT: NIL
      FOLLOW-UP: NIL
      GOALS: (<E-EAT776> <S-HUNGER777>)
      GUESTS: <*JLK*S-GROUP>
      HOST: <*JLK*>
      ORDER: NIL
      PARTICIPANTS: (?HOST ?GUESTS)
      SETTING: <*JLK*S-HOUSE>
      STEPS: NIL
      TIME: NIL
```

JULIA's problem reduction problem solver starts out. It immediately reduces the instantiate-a-meal task to its component parts and begins working on the first of them, the meal descriptor task. Meal descriptors include, among other things, the meal's cuisine.

```
***** Planning for the task MEAL-TASK775 .....
 TASK-TREE = ((<MEAL-TASK775>))
 The active goal chain is  ( M-MEAL )
 Reducing the task MEAL-TASK775 to subtasks (<MEAL-DESCRIPTOR-TASK>
   <MAIN-COURSE-TASK> <APPETIZER-TASK> <SALAD-TASK> <DESSERT-TASK>)
***** Planning for the task MEAL-DESCRIPTOR-TASK .....
 TASK-TREE = ((<MEAL-DESCRIPTOR-TASK> <MAIN-COURSE-TASK> <APPETIZER-TASK>
         <SALAD-TASK> <DESSERT-TASK>) (<MEAL-TASK775>))
 The active goal chain is  ( MEAL-DESCRIPTOR  <=  M-MEAL )
```

As part of this task, JULIA attempts to formulate a cuisine, but can't do it with the information supplied so far by the user, so it asks.

```
Constraint Satisfaction .....
  Checking constraints for slot CUISINE of MEAL-DESCRIPTOR783 ...
  Instantiating slot CUISINE of frame MEAL-DESCRIPTOR783 ...
<JULIA> What cuisine would you like?
USER => mexican
<JULIA>
Constraint Satisfaction .....
  Checking constraints for slot CUISINE of MEAL-DESCRIPTOR783 ...
  Instantiating slot CUISINE of frame MEAL-DESCRIPTOR783 ...
  --> The value is  MEXICAN-CUISINE
```

JULIA goes on to its next task, choosing a main course, and suggests chili for the main course (we leave out the way it did that). At that point, it is reminded of a meal in which chili was served. If this meal had been a successful one, the reminding would provide suggestions about other parts of the meal. In this case, however, some guests, who did not eat hot food, were left unsatisfied.

```
***** Planning for the task MAIN-COURSE-TASK .....
 TASK-TREE = ((<MAIN-COURSE-TASK> <APPETIZER-TASK> <SALAD-TASK>
          <DESSERT-TASK>) (<MEAL-TASK775>))
 The active goal chain is  ( SC-MAIN-COURSE  <=  M-MEAL )
<JULIA> What about CHILI791 for the main course?
USER => ok
<JULIA>
***** Reminded of MEAL80, where chili was the main course
***** Case-based reasoning with the case MEAL80
*** Trying to do analogy-mapping with the case MEAL80 ...
Checking if the previous plan for goals S-HUNGER80 E-EAT80 was successful .....
Previous plan execution failure found
The set of goals failed was S-HUNGER80  E-EAT80
It was because ((NOT EVERY ONE ATE SPICY DISH))
```

JULIA will try to avoid making this mistake again. It finds (through looking at the representation of the previous case) that the previous failure was because of a missing constraint about spices and seeks to find out if this constraint should be taken into account in the current case. After asking, it finds that to be so and creates a a "non-spicy-food" constraint for the current case. It propagates that constraint and checks it against what it has already decided. It finds out that chili and Mexican food are spicy, and rules both out. Because choosing a main course is dependent on having a value for cuisine, it deletes the choose-a-main-course task from the task network, reschedules the meal-descriptor task and the choose-a-main-course task, and attempts the meal-descriptor task again in an effort to choose a cuisine.

```
*** Attempting to avoid the previous plan failure.....
The assigned blame was that C-NON-SPICY-PREF80 had not been considered.
To avoid previous plan failure ...
Asking the user of a missing constraint C-NON-SPICY-PREF
<JULIA> Is there anyone who doesn't like spicy food? (How many?)
USER => 3
<JULIA>
```

```
Trying to propagate the constraint C-NON-SPICY-PREF793 .....
  --> Generating a new constraint C-NON-SPICY-CUISINE794
  --> Generating a new constraint C-NON-SPICY-DISH795
Applying constraint C-NON-SPICY-DISH795 to CHILI791
    --> Aborting CHILI791
Applying constraint C-NON-SPICY-CUISINE794 to MEXICAN-CUISINE
    --> Aborting MEXICAN-CUISINE
  --> Killing the current task MAIN-COURSE-TASK .....
  --> Rescheduling MEAL-DESCRIPTOR-TASK MAIN-COURSE-TASK into the task network .....
***** Planning for the task MEAL-DESCRIPTOR-TASK .....
 TASK-TREE = ((<MEAL-DESCRIPTOR-TASK> <MAIN-COURSE-TASK> <APPETIZER-TASK>
          <SALAD-TASK> <DESSERT-TASK>) (<MEAL-TASK775>))
 The active goal chain is  ( MEAL-DESCRIPTOR  <=  M-MEAL )
Constraint Satisfaction .....
  Checking constraints for slot CUISINE of MEAL-DESCRIPTOR783 ...
  --> Applying constraint C-NON-SPICY-CUISINE794 to slot CUISINE
    --> The slot CUISINE is not yet filled in
  instantiating slot CUISINE of frame MEAL-DESCRIPTOR783 ...
```

Because there has been little in the way of preferences offered by the user up to now, JULIA cannot suggest a new cuisine by itself at this point. It asks the user again for a cuisine preference, this time telling the user constraints on the preference. The user suggests Italian, and JULIA goes on. To complete the menu, JULIA continues its reasoning, choosing lasagne for the main course and is reminded of a case in which vegetarians were at a lasagne dinner and could not eat. JULIA knows that in the previous case, they could have eaten if the meatless version of the dish had been served, and proposes the same in this case. The meal JULIA finally comes up with includes vegetarian antipasto as the appetizer, veggie lasagne and italian bread for the main course, mixed green salad as the salad, and ice cream for dessert.

6. Discussion

In our scheme, potential failures can be encountered and thus need to be dealt with during any step of the problem solving. Any time the problem solver encounters a case with a previous problem, it considers whether there is the potential for that problem in the new case. This may cause it to refocus itself until the potential for failure is determined, and if such potential is determined and the problem solver has to retract decisions made previous to the current one, then it must remake any decisions dependent on those decisions. Such processing, of course, requires that the problem solver be integrated with a reason-maintenance system that keeps track of the dependencies among its decisions. Other steps require that the reasoner record justifications for each of the decisions it makes. We have not done a great deal of work in these areas, but our experience so far leads us to believe that a standard truth maintenance system (Doyle, 1979, McAllester, 1980, DeKleer, 1986) is not adequate to do all of the work we need such a system to do. In particular, in addition to its standard bookkeeping functions, such a system will need strategies or policies to follow in making decisions about how to make the world consistent when a condition check fails, or will need to interact with a reasoner that can make such decisions. While it is standard for a truth maintenance system to retract decisions that are inconsistent and to propagate those retractions as far as it needs to, in the problem solving situation we are looking at, it is often more advantageous to try to satisfy constraints in a different way (e.g., to replace a retracted value with another that satisfies the necessary constraints).

Hammond (1986) takes the complexity out of this issue by having the reasoner explicitly try to avoid mistakes in one of its early planning steps. The advantange of this, of course, is that after potential mistakes are discovered, the problem solver need only keep them in mind during the remainder of problem solving rather than having to deal with new issues and possible change of focus part of the way through. There is thus no need for the complexity of a truth maintenance system. On the other hand, the reasoner can only avoid those mistakes that can be foreseen at the onset of problem solving, but cannot avoid mistakes that the problem solver might not be able to anticipate until late in

the problem solving.

Carbonell (1986) deals with this issue in yet another way. His work assumes that each old problem is stored as a sequence of reasoning steps, and that any time two problems are similar in their set of steps, the second is stored on with the first, branching from it at the place they begin to be different. Thus, once the case-based reasoner is reminded of a previous case, it has available to it all of the cases that have been solved by the same initial set of steps as the one it is currently trying to solve. This means that at each decision point in the problem solving, each of the previous decisions that have been made are available along with their justifications. Reasoning similar to that described in this paper happens to evaluate which of the possibilities is appropriate for the new case. The advantages of this method are similar to the advantages in Hammond's method: the problem solver, in general, never needs to refocus itself, and there is no need for a truth maintenance system. The major disadvantage, however, is that once Carbonell's problem solver finds a set of previous cases that are similar to its current one, it is wedded to that set, and no other cases that might be similar along a different set of dimensions can contribute to the problem solving.

7. Summary

Previous problem solving failures can be a powerful aid in helping a problem solver to become better over time. When a previous case in which an error was made is recalled, it flags the potential for a similar mistake and the reasoner considers whether the same potential for error exists in the new case. The direct result of this is that reasoning is directed to that part of the current problem that was responsible for the previous error, sometimes changing the problem solver's focus. Evaluation of the potential for error in the current case may require the problem solver to gather knowledge it doesn't already have, another way focus might be redirected. A case with an error may also suggest a correct solution for the new case. The combination of these helps the problem solver to avoid repeating mistakes and suggests shortcuts in reasoning that avoid the trial and error of previous cases.

8. References

Alterman, R. (1986). An Adaptive Planner. *Proceedings of AAAI-86*, pp. 65-69.

Ashley, K. (1986). Knowing What to Ask Next and Why: Asking Pertinent Questions Using Cases and Hypotheticals. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society.*

Carbonell, J. G. (1983). Learning by Analogy: Formulating and Generalizing Plans from Past Experience. In Michalski, R. S., Carbonell, J. G, & Mitchell, T. M., *Machine Learning*, Tioga.

Carbonell, J. G. (1986). Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M., *Machine Learning II*, Morgan Kaufmann Publishers, Inc.

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

de Kleer, J. (1986). An Assumption-Based TMS. *Artificial Intelligence*, vol 28, pp. 127-162.

Doyle, J. A truth maintenance system, *Artificial Intelligence*, vol 12, pp. 231-272.

Hammond, K. (1986). Learning to Anticipate and Avoid Planning Problems through the Explanation of Failures. *Proceedings of IJCAI-86.*

Holyoak, K. J. (1984). The Pragmatics of Analogical Transfer. In Bower, G. (Ed.), *The Psychology of Learning and Motivation*, Academic Press.

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

Kolodner, J. L. (1985). Experiential Processes in Natural Problem Solving. Technical Report No. GIT-ICS-85/23. School of Information and Computer Science. Georgia Institute of Technology. Atlanta, GA 30332.

Kolodner, J. L. (1986). Some Little-Known Complexities of Case-Based Inference. *Proceedings of TICIP-86.*

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

Kolodner, J. L. & Simpson. R. L. (1984). Experience and Problem Solving: A Framework. *Proceedings of the Sixth Annual Conference of the Cognitive Science Society.*

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

McAllister, D. (1980). An Outlook on Truth Maintenance. Artificial Intelligence Laboratory, AIM-551, MIT, Cambridge, MA.

Rissland, E. L. & Collins, R. T. (1986). The Law as a Learning System. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society.*

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

Simpson, R. L. (1985). A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation. Ph.D. Thesis. Technical Report No. GIT-ICS-85/18. School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.