Reasoning directly from cases in a case-based planner

Robert McCartney
Department of Computer Science and Engineering
University of Connecticut, U-155
Storrs, CT 06269-3155
robert@uconn.edu *

Abstract

A good deal of the reasoning done in a case-based planning system can be done directly from (episodic) cases, as opposed to specialized memory structures. In this paper, we examine the issues involved in such direct reasoning including how this representation can support multiple uses, and what role execution plays in such a framework. We illustrate our points using COOKIE, a direct case-based planner in the food preparation domain.

1 Introduction

In this paper, we examine the issues regarding the direct use of cases by a case-based planner. By direct use, we mean performing the underlying reasoning in a system by manipulating representations of episodes in memory, rather than an intermediate description or specialized structure. While these issues are shared by all case-based reasoners, we focus particularly on case-based planning. We examine case representation, how cases are used for different purposes, and the role of execution; these examinations are based on work with COOKIE, a case-based system that plans and monitors execution in the domain of meal planning and preparation.

What is case-based reasoning?

Case-based reasoning is the solving of problems through the reuse of experience—when faced with a problem situation, the problem solver retrieves a similar situation (with its solution) from memory, then adapts the previous solution to solve the current problem [Riesbeck and Schank, 1989]. This can be distinguished (in theory) from rule-based reasoning, since we reason from cases corresponding to real episodes rather than from rules which are distillations of experience. In fact, this separation is muddied in practice, as case-based systems include cases that are abstractions based on a number of real episodes, and rule-based systems include rules that are in fact grounded in a single experience. Case-based reasoning has been used in a variety of domains for a variety of tasks; it has showed the most promise in situations characterized by uncertainty, lack of a complete domain theory, and/or computational constraints, where more traditional approaches have had little success.

*This research has been supported by Booth Research Center grant BG-6. The author gratefully acknowledges the assistance of Kate Sanders, Mallory Selfridge, and Karl Wurst.

Desiderata for a case-based planner

Case-based planning (CBP) is an attempt to solve planning problems by reusing previous episodes. The planning problem is to find a sequence of primitive actions that leads to some specified results—goals to be accomplished and constraints to be satisfied. A case-based planner does this by remembering experiences. At essence, a case-based planner follows the principles given by Hammond [Hammond, 1988]:

> *If it worked, use it again,* and
>
> *If it didn't work, remember not to do it again.*

To these, we add a principle of our own:

> *If my plan fails, I should figure out why.*

These principles can be translated into the three basic functions that a case-based planning system should be able to accomplish.

First, a CBP system should be able to generate plans given it has succeeded in a similar situation. This process is one of retrieving the similar situation (case), then performing transformations on the case until it matches the description of the current problem. Once it matches, "use it again" becomes a simple reapplication of the sequence of steps. For this to be effective, the transformations applied should be equivalence preserving, at least in regard to the goals and constraints.

Second, a CBP system should be able to recognize that a particular approach failed before, so should not be used again. Simply not retrieving failures for adaptation is insufficient here—if we transform a (successful) plan into one that fails, we would like to avoid doing it in the future.

The third principle relates to the second—to avoid failure in the future, we need to understand why we failed. If we can anticipate failure, then we can either use a different plan or modify our plan to avoid the problem. The principle as given only works in a negative way, but sometimes plans have unexpected good results. A more general principle can be based on expectation failures [Schank, 1982] rather than plan failures:

> If something odd happens, figure out why and remember it.

Overview of COOKIE

COOKIE is a case-based system that plans and monitors the preparation of food. Previous work (notably CHEF [Hammond, 1986] and JULIA [Kolodner, 1987]) has demonstrated the benefits of doing planning research in this domain, which is both rich and unpredictable. COOKIE's input is a set of goals and constraints: it first produces a plan to satisfy the input, then monitors the execution of that plan (performing execution-time repairs of the plan, if necessary), then incorporates the results of that execution into its memory. Execution is done externally to the system by human cooks using real food; case-memory is comprised of these episodes and other real cookings taken from transcripts. There is no causal reasoner for analyzing failure, nor is there a simulator that can be used to execute cookings. Explanations of anomalous behavior are built from the assumptions used to predict the non-anomalous behavior and from adaptations of other explanations found in cooking transcripts. COOKIE is designed to reasoning directly from its cases as much as possible, and relies on few other mechanisms. Specific examples of COOKIE's behavior relating to this paper's topic are given in the next few sections.

The rest of this paper is arranged (in order) around the following questions:

- What is an episodic case representation?

- How can an episodic representation scheme support multiple uses in a planner (generation, projection, recognition, explanation, and failure recovery)?
- What is the role of execution in a direct case-based planner?

Finally, in the conclusions, we discuss possible roles for abstraction in a case-based planner.

2 Case representation

Episodic representation—what it is, what it involves

Informally, an episodic representation is one that allows reconstruction of an episode as a story—what happened, when the various things happened relative to each other, what the results were, and so forth. More concretely, an episodic representation is a reasonably complete statement of the facts that is neither simplified nor abstracted. For any domain, the representation includes all of the facts about an episode that are likely to be useful in any future reasoning task. In the food preparation domain, this includes all of the cook-food interactions, as well as seemingly unrelated cook actions that occur during the cooking. For example, if a cook answers the telephone during meal preparation and talks for five minutes to "Dialing for Dollars", that fact is part of the episode, not just that the cook was out of the kitchen for five minutes.

Reasoning directly from cases

In a case-based reasoning system (or more generally, any reasoning system where the rules are ultimately grounded in episodes), we have two options: reasoning by direct manipulation of episodes, or reasoning from abstractions and/or simplifications of episodes. There are advantages and disadvantages to each approach. On the one hand, an episodic representation is likely to contain a good deal of irrelevant information for a particular task; if we want to explain burned biscuits, the cook not monitoring the food for five minutes is enough information, and the facts that it was a phone call, the exact time, that it was from "Dialing for Dollars" and so forth are unnecessary and could have been simplified out of the case. On the other hand, a simplification or abstraction may not have retained the right information. Suppose the above dinner turns out to be poisoned, killing the cook, and his beneficiary happens to work for the television station broadcasting "Dialing for Dollars"—the simplification to "the cook not monitoring the food for five minutes" precludes the obvious explanation of said beneficiary arranging for the phone call so he could poison the meal while the cook was distracted. *The main advantage in reasoning from episodes is that the information used can be extracted when the reasoning task and its context are established, rather than at the time the episode is stored.* This does not rule out a memory where episodes are stored at multiple levels of abstraction, precomputing those that are likely to be useful, but unless we can be certain of precomputing all useful abstractions, the need for direct reasoning cannot be eliminated.

Representation scheme in COOKIE

The representation language used in COOKIE is a temporal propositional logic. For example, (occur ev3 (do chef1 (stir soup5 saucepan1 spoon23)) t1 t2) means that event ev3, chef1 stirring soup5 in saucepan1 with spoon23, occurred beginning at time t1 and ending at time t2. Cases describing cooking episodes are propositional, and contain goals, descriptions of input, output, and (possibly) intermediate states, actions, a critique, and (if applicable) problems encountered and their associated repairs. An example, broiled steak with fried onions, is given in Figure 1. Propositions that are true at the same (specific) times are grouped together here. This is purely a syntactic aid to

```
(cooking steak-and-onions1
  (initial (raw steak23)(raw onion24)(cold frypan1) (quantity steak23 1lb)
           (thickness steak23 1.5 in) (quantity onion24 1 medium) (quantity oil25 1 Tb)
  (goals   (cooked steak23) (cooked onion24) (shape onion24 rings)))
  (final   (cooked steak23) (cooked onion24) (shape onion24 rings)
           (rating steak-and-onions1 success)))
  (facts   (inst steak23 beefsteak) (inst onion24 onion) (inst oil25 corn-oil)
  (events (occur ev0 (do chef1 (slice onion24 thin knife2)) 0 1)
           (occur ev1 (do chef1 (heat burner2 high)) 4 8)
           (occur ev2 (do chef1 (heat broiler1 high)) 0 8)
           (occur ev3 (do chef1 (put oil25 frypan1)) 4 4)
           (occur ev4 (do chef1 (put frypan1 burner2)) 4 4)
           (occur ev5 (do chef1 (put steak23 b-pan1)) 1 1)
           (occur ev6 (do chef1 (put b-pan1 broiler1)) 1 1)
           (occur ev7 (do chef1 (flip steak23 b-pan1 fork8)) 5 5)
           (occur ev8 (do chef1 (put onion24 frypan1)) 5 5)
           (occur ev9 (do chef1 (occasional (stir onion24 frypan1 fork8))) 5 8)
           (occur ev10 (do chef1 (remove onion24 frypan1)) 8 8)
           (occur ev11 (do chef1 (remove steak23 b-pan1)) 8 8)))
```

Figure 1: Broiled steak and onion rings.

the user; internally, a case is a conjunction of propositions with individual temporal characteristics. This case is a relatively simple one; other cases include (among other things) termination tests for events, underlying assumptions used in planning the episode, and expectation failures with their associated real-time repairs.

3 Multiple purposes from a single representation scheme

One benefit claimed for an episodic representation is that it is *task-neutral*: the same representation should be useful for a variety of reasoning tasks [McCartney and Sanders, 1990]. In COOKIE, the episodic representation directly supports plan generation and projection, explanation, and failure recovery during execution, and interacts with low-level abstractions during plan recognition.

Plan generation and projection

Plan generation (coming up with a sequence of actions to perform a task) and projection (predicting the effects of a proposed set of actions) in COOKIE are done by performing transformations on cooking episodes. The transformed episode corresponds to the generated plan (actions and expectations) and is processed directly by the execution module (see Section 4).

Projection is based on the assumption that the objects in the transformed episode will behave as their counterparts did in the original episode, so any observed facts in the original episode become expected facts (that is, the projections) in the transformed episode. For a given situation, there is a set of permissible transformations whose application should lead to behavioral equivalence in the transformed case. Suppose, for example, we want a plan for broiled hamburger and onions, and can use the following transformations:

(x→y: if (and (inst x z) (inst y z)(food z)))

```
(x→y: if (and (inst y ground-beef)
              (inst x beefsteak)
              (shape y patty)))
```

These correspond to the assumptions that food is fungible and we can substitute hamburger for steak if the hamburger is in a patty shape. We generate the new plan by performing the constant-for-constant substitutions steak23→hamburger26, onion24→onion27, and oil27→oil25 in steak-and-onions1. Executing the new plan involves executing the events of the transformed episode at the prescribed times, corresponding to this "recipe":

> Slice an onion while heating the broiler for one minute. Put burger under broiler for 4 minutes. When burger has been in for 3 minutes, put frying pan on high heat, adding 1 T oil. Add onions to pan, and flip burger under broiler. Stir onions occasionally for 3 minutes, then remove from pan and remove burger from broiler. Serve together.

We project the facts that the hamburger and onions will be cooked, the onion will be in ring shapes, and the cooking will be a success.

Plan recognition

Part of case-based planning is the assimilation of experience into a usable memory structure. In the steak-cooking case, the relationships between the actions and goals are not explicit; ascribing actions to goals (making the relationships explicit) is the function of plan recognition— given a set of actions, determine the plan (and goals) that they serve. In COOKIE, this is done to a very limited degree, as this conflicts with the philosophy of reasoning from complete episodes. We allow the recognition of low-level action aggregates corresponding to subplans that express action groupings common to a variety of plans. These are expressed as subplan-schemas (corresponding to MOPs) which allow us to relate actions and goals within an episode. A subplan schema consists of goals, steps, and type and temporal constraints on the steps. We have, for example, a subplan schema for frying things; this schema has the goal of cooking some food item and these steps: preparing the food, heating a pan, oiling the pan, adding the food to the pan, interacting with the food in the pan, and removing the food from the pan. When COOKIE processes the steak-and-onions episode, the actions having to do with the onion and pan are used to recognize an instance of a general frying by mapping the actions in the episode to corresponding steps in the schema (and the "cooked onion" goal in the episode to the corresponding schema goal). Once we have recognized the frying subplan here, we allow these parts to be used as a separate episode (for subsequent reasoning) with the added assumption that using this episode is based on the assumption that the onion cooking does not interact with the rest of the cooking (which is recognized as a broiling). Since schemas are only used for recognition (not generation), we can afford to be fairly non-restrictive with the steps—describing them in a general way and allowing most to be optional. This recognition allows the frying and broiling to be used either together or separately; furthermore it gives us an instance of two subplans that can be coordinated in a single meal by one cook.

Explanation

Explanation in COOKIE is closely related to projection and generation, as explanations are only generated when an expected state fails to occur—a generated plan fails or a projection proves false

on execution. The generated explanations can come from two sources. One is the failure of any transformation or separability assumptions made. Suppose we generate a plan for cooking onions based on the onion-frying subplan of steak-and-onions. We explicitly know that this plan is based on underlying assumptions of ingredient fungibility and subplan independence, so if the plan fails, we can assign ingredient difference and subplan dependence as candidate explanations. The other source of explanations is the modification of explanations from other cases (as in SWALE [Kass and Leake, 1988]), which can be used to choose among or make more specific the candidate explanations. The explanation mechanism (*i.e.*, attributing unexpected behavior to causes) in COOKIE is quite simple, but it allows explanation in the absence of a detailed causal theory. Although these explanations do not individually provide the predictive power of a causal explanation, combining evidence from multiple cases could lead to an effective set of predictive features without the need of deep understanding.

Failure recovery

Repair in COOKIE is case-based—execution-time repair is based on remembering and adapting repairs to similar problems problems in previous cases. Such repairs are necessary in planning unless we have either complete prescience or the luxury of being able to "undo" back to a choice point when something goes wrong. In COOKIE's domain, we have neither, and often are faced with problems. The steak-and-onions case had no repairs, but we can use two other cases to illustrate how repair works. The first case is fried-burger-and-onions; it is similar to the previous steak-and-onions, but both the burger and onions are fried in the same pan, the onions being added when the burger is flipped. Due to high fat content, the pan is not oiled for the burger, and the fat released in cooking the meat is enough to keep the onions from sticking. The other cooking is one of curried cauliflower, prepared by stir-frying the cauliflower in 1 T oil, then adding the curry spices for the last couple of minutes of cooking. When it was prepared, the addition of the spices soaked up all of the oil in the pan, causing everything to stick. This was repaired by adding 1 T oil and mixing.

Suppose we have frying onions as a goal; furthermore, we generate this recipe from the onion-frying subplan in fried-burger-and-onions— as in the other burger-and-onions, we separate this meal into a frying of a hamburger and a frying of onions (based on a non-interfering subgoals) with a number of shared steps that could be separated. The generated recipe is to slice onions, heat pan, add onions, stir, remove. When executed, however, the stir fails as the onions stick to the pan. We use the cauliflower case to repair the plan by adding oil, and note two possible explanations for the failure; some subgoal interaction in the original cooking (which is true here), and variation among onions. It doesn't matter that we cannot tell which is which; whenever we fry onions in the absence of meat, we will be prepared to add oil.

4 The role of execution in a CB planner

The purpose of planning is to produce instructions to be executed by some actor. The instructions are a sequence of primitive actions, including information about timing and how progress should be monitored. The information used in this process by a case-based planner is experiential; we get plans for new situations based on what we did in other situations. The subsequent execution of that plan adds an experience to our knowledge base, as well as providing information for evaluating and improving our ability to plan. In COOKIE, the plans given to the execution monitor are episodic representations of "doing it again"; the execution monitor controls execution, monitors expectations, and provides feedback useful in subsequent planning. In this context, plan generation and execution can be considered as mappings: from an episode to expected behavior for generation,

and from expected behavior to a real episode for execution.

From episode to expected behavior

Plans are generated in a direct case-based planner by adapting episodes— transformations are performed on episodes until they match the goals and constraints specified. Basic assumptions in case-based reasoning are that performing the same actions on the same objects would produce identical results, and there exist transformations that are equivalence preserving in terms of those results—if we perform such transformations on these actions, objects, and results, then perform the actions on the objects, we get these results. The adapted episodes can be seen as "expected episodes", and describe the expected behavior of reproducing the described episode.

From expected behavior to episode

Knowing the expected behavior simplifies execution monitoring to a great degree. The actions and conditions in the expected behavior become observed actions and conditions in the episode when execution is done, as long as all expectations are met. If expectations are not met, the anomalies become part of the new episode, and the planner may be invoked to repair the plan. The role of the execution monitor, then, is to cause actions to be initiated at the expected times (if possible), monitor all of the conditions given in the episode, and signal expectation failures back to the planner for possible repair. Expectation failures are labeled as such in the new episode, which will also include any response to that failure.

Execution in COOKIE

Execution monitoring in COOKIE is provided by DEFARGE[1], a dedicated execution monitor that provides the mechanisms for executing plans, adds episodes to COOKIE's knowledge base, interacts with the real-time repair capabilities of COOKIE, and provides feedback to be used in further planning. DEFARGE provides the mapping from expected to real episodes; it causes plans to be executed by side-effect. Its input is the episodic representation of the plan—an adaptation of a real episode. It converts the propositions in the episode into a set of external actions to be directed; starting events, terminating events, and testing conditions, which are assigned expected times based on the times in the episode. In execution, DEFARGE gives instructions and receives information from the external cook. As events terminate and condition tests are reported, the appropriate propositions (with the actual time information) are added to the episode description. If a test results in an anomaly (any expectation failure), the information is forwarded to the planner which can prescribe repairs to be executed and associated with the anomaly.

Since DEFARGE can extract the appropriate tests and expectation from what it is given by the generator, the amount of reasoning done by the monitor is limited. This is by design—the role of the monitor is to extract the necessary information from the plan, interact with the user and planner, do the necessary accounting (keeping track of time, propagating constraints), and convert the execution trace to a usable episode. The relative simplicity is enabled by the detailed information in the episodic representation of the generated plan.

[1] Named for the notorious Mme. Defarge, who monitored quite a few executions [Dickens, 1859].

5 Conclusions

We do not claim that direct use is the only way to use experiential knowledge; it is, however, one that can be used with minimal domain knowledge and computational overhead. It provides a computational framework for reasoning in domains where we don't have detailed, well structured information and causal models. There is strong evidence that human memory is at least partially organized around episodes, and that experience can be used in the future in ways unimagined at the time.

We have only discussed one role for abstraction—in plan recognition we use subplan schemas to ascribe actions to particular goals and to separate out parts of episodes for later use. Most CBR systems, by contrast, have abstraction hierarchies as a central feature of the work. In HYPO, for example, legal cases are viewed at a number of different levels of abstraction [Ashley, 1988]. In CHEF (as in most CBR systems), allowable transformations are implicit in the abstraction hierarchy for objects (allowing transformation between objects that share an ancestor in the isa hierarchy). We do not dispute the usefulness of such abstractions; we simply claim that they are inadequate. Unless we have precomputed all useful abstractions, direct case manipulation is still necessary. Similarly, unless we can somehow encode transformations as a hierarchy that reflects the context-sensitivity of their applicability, we will need to deal with explicit transformation sets.

References

[Ashley, 1988] Ashley, Kevin D. (1988), Modeling legal argument: reasoning with cases and hypotheticals. Tech. Rept. 88-01, University of Massachusetts Department of Computer Science.

[Dickens, 1859] Dickens, Charles (1859). *A Tale of Two Cities*. Oxford University Press.

[Kolodner, 1987] Kolodner, Janet L. (1987). Capitalizing on failure through case-based inference. In *Proc. of the 9th Annual Conference of the Cognitive Science Society*, pp 691–696.

[McCartney and Sanders, 1990] McCartney, Robert, and Kathryn E. Sanders (1990). The case for cases: a call for purity in case-based reasoning. In *Proc. of the AAAI Spring Symposium on Case-based Reasoning*.

[Hammond, 1988] Hammond, Kristian J. (1988), Case-based planning. In *Proc. of a Workshop on Case-based Reasoning*, pp. 17–20.

[Hammond, 1986] Hammond, Kristian J. (1986), Case-based planning: and integrated theory of planning, learning, and memory. Tech. Rept. YALEU/CSD/RR 488, Yale University Department of Computer Science.

[Kass and Leake, 1988] Kass, Alex M. and David B. Leake (1988). Case-based reasoning applied to constructing explanations. In *Proc. of a Workshop on Case-based Reasoning*, pp. 190–208.

[Riesbeck and Schank, 1989] Riesbeck, Christopher K., and Roger C. Schank (1989). *Inside Case-based Reasoning*. Lawrence Erlbaum Associates, Hillsdale, NJ.

[Schank, 1982] Schank, Roger C. (1982). *Dynamic Memory: a Theory of Learning in Computers and People*. Cambridge University Press.