A Tutorial Introduction to Case-Based Reasoning

Janet L. Kolodner and David B. Leake

To appear in Leake, D., ed., 1996, *Case-Based Reasoning: Experiences, Lessons, and Future Directions.* Menlo Park: AAAI Press. In press.

1 Definition and Illustrations

Case-based reasoning means reasoning based on previous cases or experiences. A case-based reasoner uses remembered cases to suggest a means of solving a new problem, to suggest how to adapt a solution that doesn't quite work, to warn of possible failures, to interpret a new situation, to critique a solution in progress, or to focus attention on some part of a situation or problem. An example will illustrate.

A host is planning a meal for a set of people who include, among others, several people who eat no meat or poultry, one of whom is also allergic to milk products, several meat-and-potatoes men, and her friend Anne. Since it is tomato season, she wants to use tomatoes as a major ingredient in the meal. As she is planning the meal, she remembers:

> I once served tomato tart (made from mozzarella cheese, tomatoes, dijon mustard, basil, and pepper, all in a pie crust) as the main dish during the summer when I had vegetarians come for dinner. It was delicious and easy to make. But I can't serve that to Elana (the one allergic to milk).
>
> I have adapted recipes for Elana before by substituting tofu products for cheese. I could do that, but I don't know how good the tomato tart will taste that way.

She decides not to serve tomato tart and continues planning. Since it is summer, she decides that grilled fish would be a good main course. But now she remembers something else:

> Last time I tried to serve Anne grilled fish, she wouldn't eat it. I had to put hotdogs on the grill at the last minute.

This suggests to her that she should not serve fish, but she wants to anyway. She considers whether there is a way to serve fish that Anne will eat.

> I remember seeing Anne eat mahi-mahi in a restaurant. I wonder what kind of fish she will eat. The fish I served her was whole trout with the head on. The fish in the restaurant was a fillet and more like steak than fish. I guess I need to serve a fish that is more like meat than fish. Perhaps swordfish will work. I wonder if Anne will eat swordfish. Swordfish is like chicken, and I know she eats chicken.

This hypothetical host uses remembered cases to suggest a means of solving the new problem (e.g., to suggest a main dish), to suggest how to adapt a solution that doesn't quite fit (e.g., substitute a tofu product for cheese), to warn of possible failures (e.g., Anne won't eat fish), and to interpret a situation (e.g., why didn't Anne eat my fish, will she eat swordfish?).

If we watch the way people solve problems, we are likely to observe case-based reasoning all around us. Attorneys are taught to use cases as precedents for constructing and justifying arguments in new cases. Mediators and arbitrators are taught to do the same. Other professionals are not taught to use case-based reasoning, but often find that it provides a way to solve problems efficiently. Consider, for example, a doctor faced with a patient who has an unusual combination of symptoms. If the doctor has seen a patient with similar symptoms previously, he or she is likely to remember the old case and propose the old diagnosis as a solution to the new problem. If formulating the diagnosis was time-consuming previously, this results in a big savings of time. Of course, the doctor can't assume the old answer is correct. He or she must still validate it for the new case in a way that doesn't prohibit considering other likely diagnoses. Nevertheless, remembering the old case makes it easy to generate a plausible answer. Similarly, a car mechanic faced with an unusual mechanical problem is likely to remember other similar problems and to consider whether their explanations apply to the new one.

Doctors evaluating the appropriateness of a therapeutic procedure or judging which of several options are appropriate are also likely to remember instances of using each procedure and to make their judgments of which to use based on previous experiences. Remembering instances of problems that arose when using a procedure can be particularly helpful here; they tell the doctor what could go wrong, and, when they include an explanation of why the old problem occurred, they focus the doctor in finding out the information needed to make sure the problem won't show up again.

We hear cases being cited time and again by our political leaders in explaining why some action was taken or should be taken. Many management decisions are made based on previous experience. In addition, the "case method" of teaching is used extensively in law, business, and medicine. Cases provide context for discussing more abstract issues and they provide illustrations of those abstract guidelines that students remember and apply in later reasoning.

Case-based reasoning is also used extensively in day-to-day common-sense reasoning. When we order a meal in a restaurant, we often base decisions about what might be good on our other experiences in that restaurant and those like it. As we plan our household activities, we remember what worked and didn't work previously and use that to create our new plans.

In general, when people solve problems, the second time solving the problem is easier than the first because they remember and repeat the previous solution. We are more competent the second time because we remember our mistakes and go out of our way to avoid them.

One of the hallmarks of a case-based reasoner is its ability to *learn* from its experiences, as a doctor might do by committing to memory a hard-to-solve problem in order to solve it easily another time, or by remembering being mislead by a confusing set of symptoms, in order to avoid being mislead in the future.

Case-based reasoning provides a wide range of advantages:

- Case-based reasoning allows the reasoner to propose solutions to problems quickly, avoiding the time necessary to derive those answers from scratch.
- Case-based reasoning allows a reasoner to propose solutions in domains that aren't completely understood.
- Case-based reasoning gives a reasoner a means of evaluating solutions when no algorithmic method is available for evaluation.
- Cases make it possible to interpret open-ended and ill-defined concepts.
- Cases help a reasoner to focus its reasoning on important parts of a problem by pointing out what features of a problem are the crucial ones.
- Cases can warn of the potential for problems that have occurred in the past, alerting a reasoner to take actions to avoid repeating past mistakes.

In summary, reference to old cases is *advantageous* in dealing with situations that recur. Reference to previous similar situations is often *necessary* to deal with the complexities of novel situations. Thus, remembering a previous case to use in later problem solving (and integrating that case with what is already known) is a necessary learning process. Case-based reasoning suggests a model of reasoning that incorporates problem solving, understanding, and learning, and integrates all with memory processes.

2 Case-Based Reasoning Systems

Case-based reasoning, as a paradigm for building intelligent computer systems, is based on the previous observations. Cases serve three sorts of purposes in case-based reasoning systems.

- Cases provide context for understanding or assessing a new situation.
- Cases provide suggestions of solutions to problems.
- Cases provide context for evaluating or criticizing suggested solutions.

As suggestion-providers, cases provide ballpark solutions that are adapted to fit a new situation or merged with each other to provide a novel solution. The reasoner in the initial example used cases in this way to suggest several main dishes and to suggest means of adapting one of them for someone allergic to milk.

As context-providers, cases provide concrete evidence for or against some interpretation of a situation. They also provide means of projecting the results of a suggested solution – one recalls cases with similar solutions and assesses whether their results are relevant to the new situation. In the initial example, the case of Anne eating mahi-mahi in a restaurant provides concrete evidence that Anne does indeed eat some fish, helping the reasoner to decide why Anne didn't eat the trout she was served and whether she might eat some other fish.

Case-based systems have been developed for problem solving tasks, such as design, planning, and diagnosis, and for interpretive tasks, such as legal reasoning and assessment of proposed solutions. Some case-based reasoning systems are autonomous; some work along with a human user to solve problems; and some are embedded in larger autonomous and interactive systems, often for educational purposes.

2.1 Requirements for Successful Case-Based Reasoning Systems

The most important component of a case-based reasoning system is its library of cases – the experiences it refers to as it makes its inferences. This library may be collected by interviewing human experts and collecting their experiences or by collecting the experiences of automated computer systems. As the system processes new problems, its case library grows as new cases from its own experiences are stored.

However, although a less experienced system has fewer experiences to work with than a more experienced reasoner, the answers given by the less experienced system won't necessarily be worse than those given by the experienced one. First, if the novice reasoner has, by chance, had an experience particularly relevant to a novel situation, its answers can be equal to those of an expert. It is important, then, that a case-based reasoner start with a representative set of cases. Second, a less experienced reasoner may do just as well if it is creative in understanding the new situation and adapting solutions it does have available.

The quality of a case-based reasoning system's reasoning depends on five things:

- the experiences it has had or been given,
- its ability to understand the relevance of old experiences to new situations,
- its adeptness at adapting prior solutions to fit new situations
- its adeptness at evaluation of new solutions and repair of flawed solutions, and
- its ability to integrate new experiences into its memory appropriately.

3 The Process of Case-Based Reasoning

Case-based reasoning is a process of "remember and adapt" or "remember and compare." While some composition is necessary to solve large problems, composition of rules, knowledge, or cases is not the primary process. And, rather than remembering abstract operators, concrete instances are recalled.

Cases are used to help in understanding and assessing situations and to help in solving problems. In the general course of everyday reasoning, these processes tend to be interleaved. We can't solve a problem when we don't understand the situation it is embedded in; we often need to solve problems in order to understand the implications of a situation well. We evaluate our solutions by projecting their results using assessment methods; and we may need to solve new problems in the course of evaluation. For simplicity, however, it makes some sense to refer to two different styles of case-based reasoning. *Problem solving* situations call primarily for the use of cases to propose solutions; *interpretive* situations call primarily for using cases for criticism and justification.

This discussion suggests the primary processes required for case-based reasoning, illustrated in Figure 1. First and foremost, partially-matching cases must be retrieved to facilitate reasoning. Thus, **case retrieval** is a primary process, as is its adjunct process, **case storage** (also called **memory update**). The retrieval process depends on choosing

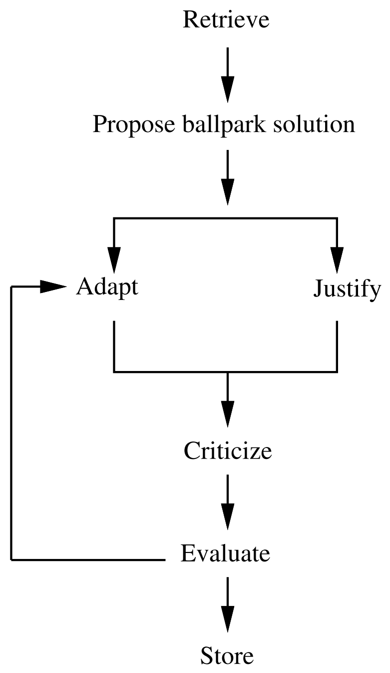

Figure 1: The Case-Based Reasoning Cycle

appropriate indexes to guide search for relevant cases in memory. In order to make sure that poor solutions are not repeated along with the good ones, both styles of case-based reasoning **criticize** candidate solutions to identify potential problems. In order to become more proficient, a reasoner must be able to **evaluate** its performance, based on external feedback. In case-based reasoning, after feedback is analyzed, cases are updated and their outcomes recorded, and cases that were used to solve the problem are re-indexed based on analysis of their usefulness.

The two styles of case-based reasoning, however, each require that different reasoning be done once cases are retrieved. In problem solving CBR, a ballpark solution to the new problem is **proposed** by extracting the solution from some retrieved case. This is followed by **adaptation**, the process of revising an old solution to fit a new situation. **Criticism** of the candidate solution often triggers further adaptation before the solution is applied.

In interpretive CBR, a ballpark interpretation or desired result is **proposed**, sometimes based on retrieved cases, sometimes imposed from the outside (as when a lawyer's client requires a certain result). This is followed by **justification**, the process of creating an argument for the proposed solution, done by comparing and contrasting the new situation to prior cases, looking for similarities between the new situation and others that justify the desired result and differences that imply that other factors must be taken into account. Sometimes justification is followed by a **criticism** step in which hypothetical situations are generated and the proposed solution applied to them in order to test the solution, before the argument is actually *evaluated* (e.g., by using it in court and observing the outcome).

These steps and styles are in some sense recursive. The criticize and adapt steps, for example, often require new cases to be retrieved. There are also several loops in the process. As reflected in Figure 1, for example, evaluation of a potential solution may lead to additional adaptation to repair problems. And when reasoning is not progressing well using one case, the whole process may need to be restarted, beginning by choosing a new case to start from.

4 Representing and Indexing Cases

4.1 What is a Case?

> Definition: A case is a contextualized piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner.

Cases come in many shapes and sizes. They may cover a situation that evolves over time (as in designing a building or following a patient through several illnesses), they may represent a snapshot (as in choosing a particular type of window for a building or recording a judge's ruling), or they may cover any size time slice in between those extremes. They may represent a problem solving episode (as do medical and architectural cases), associate a situation description with an outcome (as in legal cases), or some combination.

What is common to all cases is that they represent an experienced situation. That situation, when remembered later, forms a context in which the knowledge embedded in the case is presumed applicable. When a similar situation arises, those decisions and the knowledge that went into making them provide a starting point for interpreting the new situation or solving the problem it poses.

But not every situation that occurs is a useful case. Memorable cases, the ones that it is useful to remember, are different in some way from what is expected – expected by other cases or expected by general knowledge. This changes as normative expectations change.

Not every difference is important, however. There is another criterion for deciding if some experience or situation should usefully be recorded as a case: **Does the case teach a useful lesson?** This criterion requires us to define what a useful lesson is. Perhaps the best guideline we can give on this is that cases recorded in a case library should contribute to achieving the goals of the reasoner. As people, we have a wide variety of goals, and any small difference from a norm might teach a lesson relevant to one of our goals. Thus, from the point of view of human cognition, it is difficult to predict exactly which cases people record when and for what intentions.

For a machine, however, it is somewhat easier. AI systems normally have a limited number of goals. Some are goals for action (move the block), others are goals for reasoning (identify the patient's disorder, evaluate the design of the building, design a heat pump, compare two instances and show the differences). A case has an important lesson to teach if it records a new way of achieving some goal, if it records the results of considerable effort to find ways to achieve a goal, if an unexpected failure occurred when trying to achieve a goal, or if the effect of some situation causes a new goal to arise. There are thus many different kinds of lessons a case can teach:

- how to achieve a goal,
- how to achieve several goals in conjunction,
- the circumstances in which a set of steps for achieving a goal can be carried out successfully,
- how to bring about the state required for achieving some important goal,
- the kinds of problems that might arise in achieving a goal (what might go wrong)
- the effects of an action.

In general, a case teaches a useful lesson when it exemplifies some new way of doing something or some new effect that is likely to be useful in later reasoning. Table 1 summarizes these points, but the bottom line here is this: *If what is different in a new situation teaches something that could not have easily been inferred from the cases already recorded, then it is useful to record it as a new case.*[1]

[1]Of course, now we must consider what *easily inferred* means. This is something a system builder must determine for his/her domain and task.

- A case represents specific knowledge tied to a context. It records knowledge at an *operational* level.
- Cases can come in many different shapes and sizes, covering large or small time slices, associating solutions with problems, outcomes with situations, or both.
- A case records experiences that teach a useful lesson: that have the potential to help a reasoner achieve a goal or set of goals more easily in the future, that warn about the possibility of a failure, or point out an unforeseen problem.

Table 1: The Nature of Cases.

The role of general knowledge

There are several implications of our definition of cases. First, it implies that cases are not the only type of knowledge our intelligent systems need in order to function. If we were building a full cognitive model, we would want to include in it both cases and generalizations of those cases. Generalizations might be the vehicle to use to record what is normal. Second, it implies that the organization of abstractions and cases in a memory changes dynamically over time and with experience. What might start as a new experience, different from the norm, and teaching a new lesson, may eventually become the norm. In a full cognitive model, memory's organizational and representational structures need to change to reflect these changes.

In case-based reasoning systems, general knowledge tends to be recorded in adaptation strategies and in the *models* (Hayes 1985; Forbus 1988; de Kleer & Brown 1984) that some adaptation and matching procedures use. General knowledge, when available, can guide the choice of indices (as in KRITIK (Goel 1989)), the determination of how well a new situation matches a stored case (as in CASEY (Koton 1988)), and the choice of adaptation strategies (as in KRITIK and CASEY).

4.2 The Content of Cases

Our discussion also suggests that there are two major functional parts to a case:

- the lesson(s) it teaches – its content

- the context in which it can teach its lesson(s), described by its *indexes*; designating the circumstances in which it would be appropriately retrieved

We cover case content here, indexes in the next subsection.

There are three major parts to the content of any case (though for any particular case, they may not all be filled in):

- **Problem/situation description**: the state of the world when the episode recorded in the case occurred, and if appropriate, what problem needed solving at that time.
- **Solution**: the stated or derived solution to the problem specified in the problem description.
- **Outcome**: the resulting state of the world when the solution was carried out.

The first component, the *description of the problem* that was being solved or the *description of the situation* being understood, describes relevant previous circumstances. A description includes the goals of the reasoner when the episode recorded in the case was happening, the task the reasoner was engaged in, and any features of the environment that had a bearing on the solution chosen or the outcome that resulted.

The second component, the *case solution*, tells the reasoner how a problem was solved in a particular instance. This component may simply record the solution itself, or it may also include a trace of the way the solution was derived (e.g., Carbonell, 1986; Veloso, Chapter 8) and/or the rationale associated with each subpart of the solution. With these two case components, new problems can be solved in a case-based way by first finding a relevant case and then adapting the solution for that problem to the new situation.

Any reasoner whose cases have just these two components can be used to shortcut reasoning. For example, CASEY (Koton 1988), a program for diagnosing cardiac problems, uses cases with only these two components. When it has a similar prior case to reason from, it is several orders of magnitude more efficient than the model-based program it is based on.

In situations with many unknowns, however, severe inaccuracies can result from reasoning using cases that record only problems and solutions. In particular, taking only the unevaluated solutions to old cases into account in proposing new solutions, it is just as easy to suggest poor solutions as good ones. CASEY avoids this problem by storing only correct solutions and because its domain model, which guides adaptation, is very accurate. But in many situations, accurate models of the world are not available. In such situations, systems are bound to make mistakes. A system that mindlessly uses the knowledge it has to solve problems and stores every new problem and solution becomes more efficient over time, but it also repeats its mistakes as often as its good solutions.

The purpose of the third component, *outcome*, is to allow a reasoner to record feedback from the environment and analysis of that feedback. Outcome records what happened as a result of applying a solution, whether the result was a success or failure, in what ways it succeeded or failed, and when available, an explanation of why. With outcome included in cases, a reasoner can suggest solutions that worked and use cases with failed solutions to warn of potential failures. It can also evaluate whether repeating an old solution will result in the outcome it wants in the new situation. When much is unknown or unpredictable, previous cases that record their outcomes provide clues about the intricacies of a problem situation and can be used to predict the effects of a proposed solution.

4.3 Indexing

The indexes of a case act like indexes to books in a library. A case's indexes are combinations of its important descriptors, the ones that distinguish it from other cases. Just as we use the card catalog in a library to direct us toward books that are likely to fulfill our reading needs, retrieval algorithms use a case's indexes to select cases likely to fulfill the needs of the reasoner. When we add a new case to the case library, we assign indexes to it, just as we assign indexes to a book that gets added to the library.

At retrieval time, retrieval algorithms are given information about the new situation the reasoner is in and the goals and tasks of the reasoner in that situation (e.g., in the situation described at the beginning of this chapter, that the reasoner is trying to come up with a main dish to serve). The new situation is used as a retrieval key. Its important components are used to guide searching and matching functions as they attempt to find and identify the cases that can best address the reasoner's goal.

The **indexing problem** is the problem of making sure that a case is accessed whenever appropriate. This means the case should be indexed so as to be accessible whenever appropriate, and also that retrieval algorithms be able to use those indexes to get the right cases out at the right times.

Assigning indexes to cases

The indexing problem has several parts. First is the problem of assigning labels to cases at the time that they are entered into the case library to ensure that they can be retrieved at appropriate times. A case is appropriately retrieved if it has the potential to help the reasoner fulfill his/her/its goals. **Indexes predict a case's usefulness.** In general, these labels designate circumstances under which the case might have a lesson to teach, and therefore when it is likely to be useful.

Indexes might represent surface features or more abstract derived features. More important than whether they are surface or derived, specific or abstract, shallow or deep, is that they be those descriptors that are naturally articulated in the process of reasoning or doing some task. Thus, the indexes that are useful in supporting one task might be different than those that are useful for another. This functional and pragmatic approach to indexing suggests a task-related method for choosing a case's indexes:

> Look at the tasks a case might be used for (e.g., suggesting a solution, anticipating a failure) and choose as indexes those sets of its features that describe when it can be useful for each task. A case may have multiple indexes.

This is a fundamentally different approach to indexing than has been taken traditionally in information retrieval and database work. In those conceptions of indexing, features chosen as indexes are the ones that divide a set into partitions of approximately equal size. The ideal is to keep the organizational structures balanced, so a feature is a good discriminator if it helps balance the organizational structure. In the CBR conception of the problem, on the other hand, we require indexes to distinguish cases from each other for some purpose. We are not concerned with balancing the organizational structure. Rather, we are concerned with cutting the case library into conceptually useful pieces.

In addition to needing to be relevant to a task, good indexes for case-based reasoning satisfy two other properties:

- Indexes should be abstract enough to retrieve a relevant case in a variety of future situations.
- Indexes should be concrete enough to be easily recognizable in future situations.

An example will illustrate the process of choosing indexes for a case and what these abstract principles mean. Consider the following case.

> **Problem:** Twenty people were coming to dinner, it was summer and tomatoes were in season, we wanted a vegetarian meal, and one person was allergic to milk products.
>
> **Solution:** We served tomato tart (a cheese and tomato pie). To accommodate the person allergic to milk, we used tofu cheese substitute instead of cheese in one of the tarts.

A step by step process allows us to choose indexes based on the principles above:

Step 1: Determine what the case could be useful for by designating its points of relevance to the set of tasks the reasoner is being asked to carry out.

Step 2: Determine under what circumstances it would be useful for each of these tasks.

Step 3: Translate the circumstances into the vocabulary of the reasoner, making them recognizable.

Step 4: Massage the circumstances to make them as generally applicable as possible.

The first two steps designate the lessons the case teaches, or the points it makes, and for each, the circumstances in which those lessons can usefully be taught. In Step 3, identified circumstances are put into the right form and represented appropriately. In Step 4, they are generalized appropriately. We will demonstrate this process for our sample case.

Step 1: There are two uses for this case, or two lessons it teaches:

1. It provides conditions for success when choosing a vegetarian main dish with tomatoes: *Choose tomato tart to feed vegetarians in the summer.*
2. It provides conditions for success when trying to accommodate a person allergic to milk when a main dish with cheese is being served: *When trying to adapt a dish with cheese in it for someone who eats no milk products, use tofu cheese substitute.*

Determining how a case could be useful in future reasoning is an introspective process on the part of the indexer. People generally find it easy to look at a case and recognize some of the lessons it teaches, but the goal is to determine the full range of lessons a case teaches and to index it for all of those. One way to get at the full range of lessons a case can teach is to consider how cases are used by the system. Do they suggest solutions? Do they point out the potential for failure? Do they help in projecting results? For any particular system, the useful lessons a case can teach are those that are of a type the system knows how to use. The indexer thus asks of each case, "For which tasks that case-based reasoning supports in this system can this case provide guidelines?" In our example, the case provides guidelines for the tasks of constructing a solution and choosing an adaptation method.

Step 2: The next step in choosing indexes is to determine under what circumstances this case would be useful for each of the possible uses designated in Step 1. Sometimes this question will have been answered by the time Step 1 is done. Sometimes, additional reasoning is needed. In the first use for our example—constructing a solution—there are two circumstances when the case would be helpful, and in the second use—adapting a solution—there is one.

1. When the case is useful for constructing a solution:
    - When the goal is to choose a main dish, dish is to be vegetarian, dish is to include tomatoes.
    - When the goal is to chose a main dish, dish is vegetarian, time is summer.
2. When the case is useful for adapting a solution:
    - When main dish has cheese as an ingredient, one or a few guests are allergic to milk products, goal is to accommodate those guests.

Note that different people will have different ways of specifying these circumstances. For example, one person may see the circumstances as pertaining to a main dish with cheese, others to a dish with cheese. The next two steps normalize the specification.

Step 3: Descriptions are translated into the vocabulary of the reasoner, making them *recognizable* to the reasoner. The indexer needs to consider how indexes are structured and the set of legal values for each role. We address the choice of indexing vocabulary (dimensions recorded in indices, and their possible values) in the next subsection. For now, let us suppose a simple set of dimensions (slots) for representation of possible indices: *guests, host, cuisine, ingredients, preparation method, dishes* (subdivided into *salad, main dish, sides, beverage, dessert,* and *dessert-beverage), reasoning goal, constraints, season,* and *results.* Let us further suppose that some of these slots have further substructure. Each dish, for example, has several optional descriptors: *cuisine, taste, texture, ingredients, preparation method, constraints.* We represent an index by filling in the slots for relevant features of the case. The process results in the following three indexes for this case, one corresponding to each situation in which the case is expected to be useful:

```
1.  reasoning goal: choose main dish
     main dish:
        constraints: vegetarian
        ingredients: tomatoes
2.  reasoning goal:  choose main dish
     main dish:
        constraints: vegetarian
     season:  summer
3.  reasoning goal:  adapt dish
     main dish:
        constraints: no milk
        ingredients: cheese
```

Step 4: We massage the designated indexes to make them as *generally applicable* as possible, either by generalizing a vocabulary item that fills a slot (e.g., *tomato* is a kind of *salad vegetable*, so it will be useful to retrieve the case whenever the goal is to choose a vegetarian main dish including salad vegetables), or by generalizing a slot itself (e.g., *main dish* is one of the more general set of *dishes*, and it may be useful to retrieve the case when attempting to find any dish involving tomatoes). One can use the suggestions made by this case not only when choosing a main dish, but also when designing a meal with vegetarian dishes and tomatoes or for vegetarians in the summer. This is designated by generalizing *choose main dish* to *design meal*, and then moving the constraint designations to their appropriate place with respect to the new goal (e.g., *main dish* is one of the set of *dishes* in a meal).

The new indexes cover the whole meal rather than only selection of a main dish, suggesting that tomato tart be served. The third index is made more generally applicable by generalizing *main dish* to *dish*: the adaptation the case teaches is useful not only for adapting main dishes, but also for adapting any dish with cheese in it for the needs of someone who can't eat milk products. These three more general indexes can be designated as follows:

```
1.  reasoning goal: design meal
     dishes:
        constraints: vegetarian
        ingredients: tomatoes
2.  reasoning goal:  design meal
     dishes:
        constraints: vegetarian
     season:  summer
3.  reasoning goal:  adapt dish
     dishes:
        constraints: no milk
        ingredients: cheese
```

Properties of a Good Indexing vocabulary

The previous example assumes that we start from a given indexing vocabulary. Some efforts have been made to establish general indexing vocabularies for particular classes of reasoning tasks (e.g., Domeshek, 1992; Leake, 1992; Schank and Osgood, 1990), but the developer of a case-based reasoning system must usually derive an indexing vocabulary. While indexes are chosen for particular cases, indexing vocabulary needs to cover the domain or set of domains to be handled by the CBR system.

In general, vocabularies have two parts: a set of *descriptive dimensions* and a set of *values* along each dimension. The bottom line in choosing an indexing vocabulary is this: *The indexing vocabulary must capture those dimensions of the domain that need to be captured for useful reminding. The level of detail required in the symbols used for representation depends on how specifically-similar cases must be to provide credible advice.*

Because retrieval cues must be based on the representational structures built in the normal course of processing, it is also important that the indexing vocabulary match the vocabulary of the reasoner:

- Indexing should be by concepts that are normally used by the reasoner to describe the items being indexed.
- Indexing vocabulary should anticipate the vocabulary a retriever might use.
- Indexing should anticipate the circumstances in which a retriever is likely to want to retrieve something (i.e., the task context in which it will be retrieved) and the descriptors the retriever is likely to have available to describe the item he/she/it wants to retrieve.

This pragmatic advice follows from the qualities of indexes presented above: Indexes should allow useful remindings. If we can determine, for any domain and task, the dimensions that are useful for judging similarity, then we know what dimensions the vocabulary must cover. When we determine how detailed and how specifically-applicable advice from cases needs to be to adequately cover the assigned tasks, we can determine the level of detail that must be captured in the symbols representing values for each dimension.

A warning is in order, however. Coming up with a complete and correct set of dimensions and vocabulary terms that is efficient in terms of matching, covers everything, and is adequately expressive is a monstrous—if not impossible—task. Just when a domain has been analyzed almost in its entirety, a counter-example might be discovered that shows a fault in the vocabulary. There is no way to insure against that and no way to prove that the vocabulary is right. Nevertheless, there is a need in any system for a vocabulary that works. The spirit of vocabulary-definition must be to propose a vocabulary that *works most of the time.* It should make intuitive sense, provide reasonable coverage, be reasonably expressive, and support a large number of remindings.

Defining an Indexing vocabulary

Two approaches can be helpful for gathering the information needed to set up an indexing vocabulary for a domain or set of domains:

- **The functional approach:** Examine a corpus of available cases and the tasks that must be supported, looking at what each case can be used for and the ways it needs to be described to make it available.

- **The reminding approach:** Examine the kinds of remindings that are natural among human experts who do the designated task, looking for relevant similarities between new situations the experts are put in and the cases they are reminded of to find out which kinds of descriptors are the important ones for judging similarity and in what circumstances.

The results of these analyses tell us important dimensions to focus on, the range of values each can take on, and the level of detail that is advantageous in a representation. This analysis is done keeping three things in mind.

- the range of tasks the case-based reasoner is responsible for

  This allows us to constrain the vocabulary to only what is needed for the designated tasks.

- the range of cases available to support those tasks

  This acts to both constrain and broaden. If only cases from a particular domain are available, it tells us to constrain the vocabulary to cover only the domain. If cases that are available span domains, it tells us that dimensions and vocabulary must be chosen to be more generally applicable.

- the degree and directions in which the system will be expanded in the future

  This warns us that if we make the choice of index vocabulary too narrow to begin with then we may not be able to easily expand the system later.

More generally, the process of choosing an indexing vocabulary is as follows:

1. Collect a representative set of cases. Overall, cases should be representative of the problems that arise in the domain, the contexts in which problems arise, solutions, and solution outcomes.

2. Identify the points each case can make, or the lessons it can teach.

3. Characterize the situations in which each case can appropriately make each of its points.

4. For each, describe indexes that would allow the case to be recalled in each designated situation, making sure that descriptions of indexes are both abstract enough to be generally-applicable and concrete enough to be recognizable.

5. Design the vocabulary to cover these needs, extracting dimensions first, values along each dimension second.

5 Reasoning with Cases

5.1 Retrieval and Memory Update

There are several retrieval issues that need to be addressed. First, the CBR system must have some means of identifying the cases that have the potential to be most useful. This is done by *matching and ranking* procedures. The pattern matching community and many in the analogical reasoning community find it natural to talk about identifying items that are most similar to a new item. They define mapping and matching algorithms that know how to compare two items to each other and score their degree of match. Indeed, degree of match is an important component of determining usefulness of a case. But determining the degree of usefulness of a case also requires consideration of what purpose the case will be put to after retrieval and what aspects of cases were relevant to outcomes in the past. These considerations allow matching procedures to determine which dimensions of a case are important to focus on in judging similarity.

Since the indexes of a case mark the important dimensions to focus on in judging similarity, matching algorithms can use indexes as a guide in determining which features to focus on for similarity judgments. But since any case might be indexed in multiple ways, matching algorithms must be able to distinguish which indexed features to focus on at any time.

For this reason, input to retrieval algorithms includes both a description of the new situation and also an indication of what the reasoner will use the case for. Usually this description is in terms of reasoning tasks to be accomplished or reasoning goals to be achieved. Matching and ranking procedures use this description to determine which features of a case are the most important to focus on in judging similarity.

Second, retrieval depends on the *retrieval algorithms* themselves. These are the processes that know how to search the case library to find cases with the potential to match the new situation well. Matching procedures are applied to those cases identified by retrieval algorithms as potentially useful.

As we know from studies of data structures and algorithms, one cannot discuss algorithms for retrieval without also discussing the structures those algorithms must search. A list is searched differently, for example, than a complex tree structure. It is the same in discussing retrieval of cases. Different organizations of cases give rise to different algorithms for retrieving them. A list of cases requires different kinds of retrieval algorithms, for example, than does a complex tree or graph structure. Each has its advantages and disadvantages. One can be sure, for example, in a flat memory (a list) that the best case will be retrieved, but if the memory is large, it may take too much time. Complex tree structures, on the other hand, require less computation to get to a useful case, but they do not guarantee that the best-matching case will be retrieved.

The CBR community has identified a number of strategies and algorithms for organizing and retrieving cases – some are serial, some parallel; some use flat or shallowly-indexed organizing structures, others use hierarchical, more deeply-indexed structures. Some use indexes to build structures that discriminate cases from each other at a very fine grain-size; other structures discriminate more coarsely; others hardly discriminate at all (Kolodner 1993). The most widely-used method is similar to inverted indexes – a shallowly-indexed structure that can be searched serially or in parallel. Such methods have been efficient even for very large case libraries (25,000 or more records; see Kitano and Shimazu, Chapter 13).

Retrieval algorithms and matching functions are, indeed, the major pieces of case retrieval. There is, however, one additional piece to the process. That piece is called *situation assessment*. Situation assessment is the process of analyzing a situation and elaborating it such that its description is in the same vocabulary as cases already in the case library. For example, consider a judge's task of determining sentences for crimes, which was modeled by the CBR system JUDGE (Bain 1989). The law gives judges considerable latitude in determining sentences, but appropriate sentences can be determined by retrieving cases of similar previous crimes and adjusting their sentences depending on the degree to which the new crime is more or less severe. Unfortunately, the descriptions of a crime that are established in court may not be directly useful for retrieving cases of similar crimes, requiring analyzing the situation to determine relevant features. For example, a description of the facts of a fight, as processed by JUDGE, might be:

> John punched Joe in the nose. Joe punched John back, knocking him down. John got back up again, this time with a knife in his hand. He tried to stab Joe. Joe pulled out his gun and shot John, killing him.

This description gives the sequence of events in the fight, but there is much unstated. We can see, for example, that the violence escalated, starting with punching, moving on to stabbing, and finally to shooting. We can see that Joe was acting, at least partially, in self-defense, i.e., that his motive was saving himself from John, who had started the fight. We can guess that Joe was scared and that John was angry. We might wonder what Joe did to provoke John, if anything. While none of these descriptors are in the sequence of events itself, they can all be descriptors of the case. And some are descriptors that, if known, help a reasoner to draw conclusions about the situation.

Noticing that the violence escalated, that it was initiated by John, that Joe had to defend himself, and perhaps even that Joe was scared are all important to determining the sentence. They are also the features that are useful in comparing this situation to others to make sure that the proposed sentence is consistent with other similar crimes. It is these descriptors, rather than those in the raw description of the case, that are useful as indexes to cases. It makes little sense to index by who the fighters were or what weapons they used – neither of those is predictive of how to sentence the crime, and neither is generally-applicable enough to be worth indexing on. It makes more sense to index cases by whether the killer was defending himself, whether the fight was initiated by the killer or the person who was killed, and so on. If cases are indexed in the case library by those dimensions, then retrieval of useful cases can happen only after those dimensions are inferred for a new situation.

Situation assessment is tricky, however. There are always innumerable features of new situations that could be elaborated, some of which are quite expensive to infer. And often, old cases need to be retrieved to help with elaborating a new situation. Thus, while situation assessment is an important component of retrieval, it must be controlled.

Putting all these aspects of retrieval together, the (idealized) retrieval process is as illustrated on the left side of Figure 2. The reasoner asks the retriever for cases by describing the new situation and its current reasoning goal. The new situation is analyzed. Its description is elaborated by situation assessment procedures, which determine where similar cases are likely to be in memory, i.e., what the indexes for the new situation would be if it were stored in the case library. Retrieval algorithms use the case and the computed indexes to search the case library. As they search, they call on matching procedures, either to assess the degree of match between the new situation and cases that are encountered, or to assess the degree of match along individual dimensions. Retrieval algorithms return a list of partially-matching cases, each of which has at least some potential to be useful. Ranking procedures analyze the set of cases to determine which have the most potential to be useful, and those are returned to the reasoner.

An analogous process is carried out when cases are inserted into the case library (the

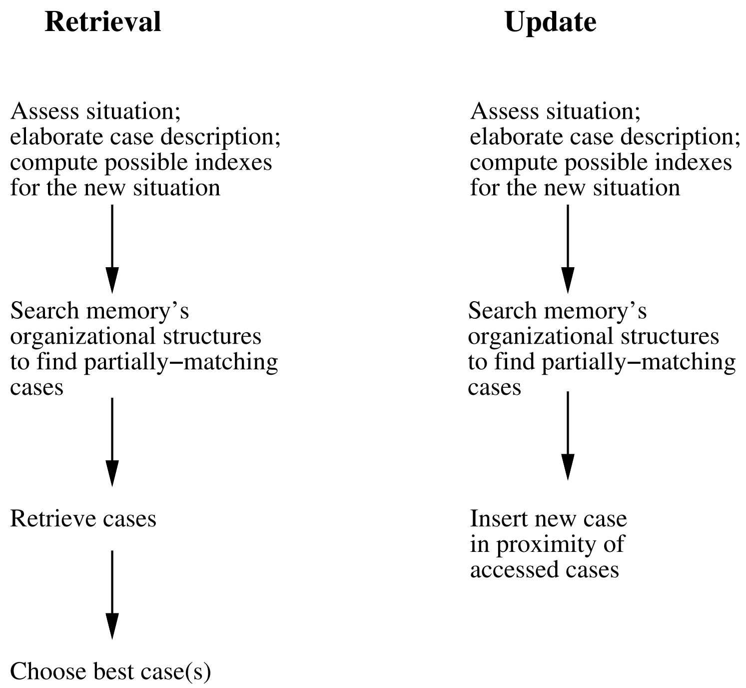

Figure 2: Retrieval and Update Processes

right side of Figure 2). Index-selection procedures determine the way the case should be indexed. Insertion algorithms use those indexes to insert the case appropriately into the case library's organizational structures. In general, they do the same sort of search done by retrieval algorithms. They look for a place to insert the case rather than for a place to retrieve a similar case from, but the two searches are exactly the same. One would expect to find a similar case in the place where the new case would be inserted. At this point, the two processes diverge. Where retrieval algorithms rank the usefulness of retrieved cases at this point, insertion algorithms invoke insertion procedures to insert the case in the case library and reorganize its organizational structures as necessary.

5.2 Adaptation

Because no old situation is ever exactly the same as a new one, a problem-solving CBR system must usually adapt old solutions to apply them to new situations. In adaptation, one takes a problem description and a solution that isn't quite right and manipulates the solution to make it better fit the problem description.

Adaptation might be as simple as substituting one component of a solution for another or as complex as modifying the overall structure of a solution. Substituting vegetarian lasagne for a more traditional lasagne in a meal is a simple substitution. Adding a new step to a plan modifies the overall structure of a solution.

Adaptation can happen during solution formulation or after a problem has been pointed out by feedback (either from projecting the results of applying a solution, or actually applying the solution in the world). Adaptation can take several forms: something new might be inserted into the old solution, something might be deleted from it, some item might be substituted for another, or some part of the old solution might be transformed.

Kolodner's (1993) case-based reasoning textbook describes ten methods by which adaptation can be done:

- Substitution methods
  - Reinstantiation
  - Parameter Adjustment
  - Local Search
  - Query Memory
  - Specialized Search
  - Case-based substitution
- Transformation methods
  - Common-Sense Transformation
  - Model-Guided Repair
- Special-purpose adaptation and repair
- Derivational Replay

The six substitution methods substitute values appropriate for the new situation for values in the old solution. *Reinstantiation* is used to instantiate an old solution with new objects. For example, consider CHEF (Hammond 1989), a case-based reasoning system that generates new recipes by retrieving and adapting recipes for similar previous dishes. CHEF uses reinstantiation to create a chicken and snow peas recipe from a recipe for beef and broccoli.

Chicken is substituted for beef everywhere in the recipe, and snow peas are substituted for broccoli.

Parameter adjustment is a heuristic for adjusting numerical parameters of an old solution. It relies on specialized heuristics that relate differences in input specifications to differences in output. JUDGE, for example, uses parameter adjustment to sentence a criminal to a shorter sentence than another criminal because the crime was less violent.

Local search provides a way of searching an auxiliary knowledge structure for a replacement for some old value or structure that is inappropriate for the new situation. If in an old menu, one served oranges for dessert but they are not currently available, local search allows the reasoner to search the semantic network of foods to find a close relative of oranges (say apples, another winter fruit) that can serve as a substitute.

Query memory means asking either auxiliary knowledge structures or the case memory to return something with a given description. In *specialized search*, both auxiliary knowledge structures and case memory are also queried, but specialized search heuristics are used to guide memory search.

Case-based substitution uses other cases to suggest substitutions. For example, a meal planner that has determined that the lasagne he or she is planning for the main dish is inappropriate may choose a substitute by recalling other Italian meals with pasta as a main dish.

Transformation methods are used to transform an old solution into one that will work in a new situation. In *common-sense transformation*, common-sense heuristics are used to replace, delete, or add components to a solution. A common-sense heuristic called "delete secondary component", for example, can be used to transform lasagne into vegetarian lasagne by deleting the meat. *Model-guided repair* is transformation guided by a causal model. Reasoning about devices, whether for diagnosis or design, can often be guided by a model.

Special-purpose adaptation and repair heuristics are used to carry out domain-specific and structure-modifying adaptations not covered by the other methods. These heuristics are indexed by the situations in which they are applicable. CHEF, for example, uses a specialized adaptation heuristic to insert a "defatting" step into any recipe that includes duck. The meal planner JULIA (Kolodner 1987; Hinrichs 1992) uses specialized adaptation heuristics to modify the structure of its solutions. One such heuristic knows how to remove redundancies from a structure, for example, while another recognizes when two functionally similar components must both be included in a solution. Such heuristics can also implement strategies for repairing solutions after they have been carried out and failed. CHEF uses specialized repair heuristics for this purpose. Each is a rather abstract strategy for repairing a faulty solution, indexed by the type of failure it repairs (e.g., the problem caused when a side effect of one plan step disables a precondition for another plan step).

Special-purpose adaptation heuristics are often implemented as *critics* and controlled through use of a rule-based production system. Critic application also provides a way of implementing parameter adjustment, common-sense transformation, and model-guided repair. CASEY's model-guided repair heuristics are implemented as critics.

Derivational replay reuses the method for deriving an old solution or solution piece to derive a solution in a new situation. For example, a menu planner that knows that it previously chose oranges for dessert by a process of choosing a fruit appropriate to the season (winter) can adapt its old menu for a summer meal by using the same process to find a fruit appropriate to the summer season. Like case-based substitution, derivational replay provides a case-based means of doing adaptation.

5.3 Justification

The task of an interpretive CBR system is to decide whether a concept fits some open-ended or fuzzy-bordered classification, to demonstrate the rightness of an argument or position, or to predict outcomes, in context of previous examples. This evaluation in context is something we do every day. Members of university admissions committees evaluate the potential of applicants to succeed at their school by comparing and contrasting them to other similar students who have and have not done well. Doctors distinguish between patients having one illness and another similar one by comparing and contrasting the current case to previous examples. Parents use comparisons to prior examples to answer questions like "My daughter wants to see a scary movie. Is this one appropriate for her or will she cry and run out like the last time?" and "My son says he has a sore throat. Should I keep him home from school or send him and see what develops?" In the example at the beginning of this chapter, the process is used to answer the question "Is swordfish a fish Anne will eat or isn't it?"

For these tasks there are often no hard-and-fast rules that can be applied, or knowledge to apply them; a case-based reasoner makes up for these lacks by assuming that the world is consistent. Justifying a classification by prior cases depends on comparing the new situation and prior cases, asking questions such as "how similar are swordfish and mahi-mahi?", "Is my son's complaint like other ones he's made when he's just tired, or is it more like those he makes when he is really sick?" or "In what ways is this new applicant similar to those who have done well in graduate school? In what ways is he similar to those who have done less well?"

In addition to looking at similarities, justification looks at what differentiates the new situation from old similar ones to ascertain whether or not the old interpretation is likely to hold. Perhaps an applicant is similar in some respects to students who have not done well (e.g., his grades and GRE scores are low like theirs), but he has extensive experience, maturity, and drive that most of those students didn't have. Maybe that means we shouldn't pay attention to the similarities. Perhaps the movie the daughter wants to see is like others in which she walked out scared and crying, but she has changed since then (e.g, has become more mature), rendering the interpretation based on the old situations unsuitable. To determine if she will get scared and cry at this one, the parent has to consider how this situation is like others the daughter has been in, in what ways it is different, and whether those differences are enough to justify the prediction that she won't be too scared this time. This requires considering similarities and differences between this movie and others she's seen and also considering similarities and differences between her when she saw those movies and now, since those are the two major factors determining whether she will get scared or not. Section 7.2 sketches how some particular interpretive CBR systems perform justification.

6 Learning

A case-based reasoner gets more efficient over time as a result of collecting more experiences and indexing them properly. It can get more effective over time if, in addition, it analyzes the effects of its solutions and keeps track of what worked and didn't work and why. There are few special processes needed to make learning happen in a case-based system. Effective learning can happen as a byproduct of the reasoning and memory update.

A case-based reasoner's performance becomes *more efficient* by remembering old solutions and adapting them rather than having to derive answers from scratch each time. If a case was adapted in a novel way, if it was solved using some novel method, or if it was solved by combining the solutions to several cases, then when it is recalled during later reasoning, the steps required to solve it won't need to be repeated for the new problem.

Case-based reasoners also become *more competent* over time, deriving better answers than they could with less experience. One of case-based reasoning's fortes is in helping a reasoner to anticipate and thus avoid mistakes it has made in the past. This is possible because it caches problem situations, indexing them by features that predict its old mistakes. Remembering such cases during later reasoning provides a warning to the reasoner of problems that might come up, enabling the reasoner to work to avoid them.

Within AI, when one talks of learning, it usually means the learning of generalizations, either through inductive or explanation-based means. While the memory of a case-based reasoner notices similarities between cases and can therefore notice when generalizations should be formed, inductive formation of generalizations is responsible for only some of the learning in a case-based reasoner. Case-based reasoning achieves most of its learning in two other ways:

- through the accumulation of new cases, and
- through the assignment and unassignment of indexes.

New cases give the reasoner additional familiar contexts for solving problems or evaluating situations. A reasoner whose cases cover more of the domain will be a better reasoner than one whose cases cover less of the domain. One whose cases cover failure as well as successful instances will be better than one whose cases cover only successful situations. New indexes allow a reasoner to fine tune its recall apparatus so that it remembers cases at more appropriate times.

That is not to say that generalization is not important. Indeed, the cases a case-based reasoner encounters give it direction in the creation of appropriate generalizations, i.e., those that can be useful to its task. How can that work? When several cases are indexed the same way and all predict the same solution or all can be classified the same way, the reasoner knows that a useful generalization can be formed. In addition, the combination of indexes and a predicted solution or classification also give the reasoner guidance in choosing the level of abstraction of its generalizations. Some case-based reasoners use only cases, others use a combination of cases and generalized cases. Even when a case-based reasoner does not use generalizations to reason, they are useful in helping the reasoner organize its cases.

Case-based reasoning is not the first method that combines reasoning and learning, but it is unique in making learning little more than a byproduct of reasoning. A case-based reasoner that remembers its experiences learns as it reasons. Feedback from early experiences gives it insight for solving later problems. Reports of fielded case-based reasoners bear this out. CLAVIER, for example, configures composite parts for loading in an autoclave (Barletta & Hennessy, 1989, Hennessy & Hinkle, 1992, Mark et al., Chapter 14). In over a year of use at Lockheed, its case library grew from 20 to over 150 cases, it became a more efficient problem solver, and its solutions became more accurate. At the same time, it served as a corporate memory for those who work on the shop floor. It accumulated the experiences of all those workers in its memory, learning not only what one of them had been able to learn but rather learning the things all of them had learned.

In addition to the learning that happens as a natural consequence of reasoning, one can think about enhancing a case-based system with learning capabilities designed to make its components work more efficiently. For example, one could put controls on which cases are added to a case library so that retrieval can be kept as efficient as possible. One could have a reasoner learn which solutions are costly to carry out and which are inexpensive (to enhance its selection of cases), which elaborations are costly to carry out and which are inexpensive (to make situation assessment more efficient), and so on. One can also use cases to enhance general knowledge.

7 Some Uses of CBR

7.1 Problem Solving CBR

Case-based reasoning has been applied to a wide variety of problem solving tasks, including planning, diagnosis, and design. In each of these tasks, cases are useful in both suggesting solutions and in warning of possible problems that might arise. There are additional specific advantages for each of these problem solving tasks.

7.1.1 CBR for Design

In design, problems are defined as a set of constraints, and the problem solver is required to provide a concrete artifact that solves the constraint problem. Usually the given constraints underspecify the problem (i.e., there are many possible solutions). Sometimes, however, the constraints overconstrain the problem (i.e, there is no solution if all constraints must be fulfilled). In addition, in design, the solutions of individual pieces of a design problem are tightly coupled to solutions of other pieces. While constraints can be used to maintain the connections between pieces, methodologies that require backtracking are too tedious for complex problems. Case-based reasoning addresses all of these issues.

- Cases suggest solutions to underconstrained problems. The solutions might not be exactly right, but because many different solutions might be appropriate, adaptation heuristics can generally create a satisfactory solution easily.
- When problems are over-constrained, cases suggest an alternative set of constraints that has worked in combination in the past. While some adaptation might still have to be done, the full application of constraint relaxation can be avoided.
- When problem subparts are tightly coupled, cases can provide the glue that holds a solution together. Rather than solving the subparts by decomposing, recomposing, and fixing discrepancies, as is done in solving nearly-decomposable problems, a case suggests an entire solution, and the pieces that don't fit the new situation are adapted in place.

Several problem solvers have been built to do case-based design. JULIA designs meals, and the meal planning examples shown above are all among those JULIA has solved. CYCLOPS (Navinchandra 1988) uses case-based reasoning for landscape design. KRITIK and KRITIK-2 (Goel 1989; Goel & Chandrasekaran 1989; Stroulia *et al.* 1992) combine case-based with model-based reasoning for design of small mechanical and electrical devices. They use case-based reasoning to propose solutions and use model-based causal knowledge to verify their proposed solutions, to point out where adaptation is needed, and to suggest adaptations. CADET (Sycara & Navinchandra 1989) also uses a combination of case-based and causal knowledge to do design.

An active current area focuses on case-based tools to aid human designers. As mentioned previously, CLAVIER is in production use at Lockheed to support design of autoclave layouts. ARCHIE and its successor ARCHIE-2 (Domeshek & Kolodner 1992; Griffith & Domeshek, Chapter 3) are retrieval-only systems built to help architects do conceptual design, MIDAS (Domeshek *et al.* 1994) helps aircraft designers with the conceptual design of aircraft subsystems; while AskJef (Barber *et al.* 1992) aids human-computer interface designers with their task. A tool for creating case-based advisory systems for design, Design-MUSE (Domeshek, Kolodner, & Zimring 1994) has been created based on these projects.

In almost all design problems, more than one case is necessary to solve the problem. Design problems tend to be large, and while one case can be used to solve some of it, it is usually not sufficient for solving the entire problem. In general, some case can provide a framework for a solution, while other cases can be used to fill in missing details. In this way, decomposition and recomposition are avoided, as are large constraint satisfaction and relaxation problems.

7.1.2 CBR for Planning

Planning involves a number of complexities (Charniak & McDermott, 1987, provide an excellent overview). Good plans must be sequenced appropriately so that late steps in a plan don't undo the intended results of earlier steps, so that preconditions of late steps in a plan are not violated by the results of earlier ones, and so that preconditions of later plan steps are fulfilled before the steps are scheduled. As the number of plan steps increases, the computational complexity of projecting effects and comparing preconditions increases exponentially. In addition, a planner dealing with the real world must face the real world's complexity, including the fact that it is in many ways unpredictable and that time is not limitless. Streams of goals might need to be achieved almost simultaneously. Time used for planning can take away from the time available for execution. Because conditions in the world can change between coming up with a plan and carrying it out, a plan might fail at execution time and require replanning, recovery, or repair. A planner with little time might miss opportunities during planning, and opportunities may need to be noticed and taken advantage of during execution (Marks, Hammond, & Converse 1989).

Case-based reasoning can address many of these planning issues:

- Cases provide already-worked-out plans in which sequencing, protection maintenance, and scheduling of preconditions have already been worked out. Rather than reasoning from scratch, the planner is required only to make fixes in old plans.
- If cases are indexed by the conjunctions of goals they achieve, they can be used to suggest ways of achieving several goals simultaneously or in conjunction with each other.
- Warnings provided by cases can help a planner anticipate and avoid problems, decreasing the likelihood of failure at execution time.

- Suggestions made by cases can shortcut the planning process, providing relatively more time for execution.
- Suggestions made by cases allow the reasoner to notice some opportunities (e.g., to achieve goals simultaneously) easily during planning.
- Adaptation strategies used to adapt old plans to new situations can also be used for execution-time recovery and repair, and to update plans during execution to take advantage of opportunities.

CBR planning research addresses many of these issues. PLEXUS (Alterman 1986; 1988), a program that adapts knowledge about riding a subway to other tasks, is able to do execution-time repairs by adapting and substituting semantically-similar steps for those that have failed.

CHEF addresses the problem of anticipating difficulties before execution time by learning from its problematic experiences. When problems happen at execution time, CHEF attempts to explain them and then to figure out how they could be repaired. It stores the repair in memory, and indexes the case by features that are likely to predict that the problem will recur. Before it begins to derive future plans, it looks for potential failure situations, anticipates the problems they point out, and uses the previous repair to suggest a plan that will avoid the problem it has anticipated.

Other case-based planners address others of these problems. TRUCKER (Marks, Hammond, & Converse 1989) is an errand running program that keeps track of its pending goals and is able to take advantage of opportunities that arise that allow it to achieve goals earlier than expected. MEDIC (Turner 1994) is a diagnosis program that is able to reuse previous plans for diagnosis but is flexible enough in its reuse to be able to follow up on unexpected turns of events. EXPEDITOR (Robinson & Kolodner 1991), plans the events in the life of a single parent who must deal with children and work. It caches its experiences at achieving multiple goals by interleaving them. While it is slow in its initial planning, it gains competence over time as it is able to reuse its plans.

7.1.3 CBR for Diagnosis

In diagnosis, a problem solver is given a set of symptoms and asked to explain them. A case-based diagnostician can use cases to suggest explanations for symptoms and to warn of explanations that were found to be inappropriate in the past.

Of course, one cannot expect a previous diagnosis to apply intact to the new case. Just as in planning and design, it is often necessary to adapt an old case to fit a new situation. CASEY diagnoses heart problems by adapting the diagnoses of previous heart patients to new patients. CASEY's adaptations are based on a valid causal model, so its diagnoses are as accurate as those made from scratch using the same causal model.

Cases are also useful in diagnosis for pointing the way out of previously-experienced reasoning quagmires. PROTOS (Bareiss 1989), which diagnoses hearing disorders, is designed to ensure that this happens in an efficient way.[2]

In PROTOS' domain, many of the diagnoses manifest themselves in similar ways, and only subtle differences differentiate them. A novice is not aware of the subtle differences; experts are. PROTOS begins as a novice, and when it makes mistakes, a "teacher" explains its mistakes to it. As a result, PROTOS learns these subtle differences. As it does, it leaves *difference* pointers in its memory that allow it to move easily from the obvious candidate diagnosis to the correct one.

Generating a diagnosis from scratch is a time-consuming task. In almost all diagnostic domains, however, there is sufficient regularity for a case-based approach to diagnosis generation to provide efficiency. Of course, no person or program can assume that a case-based suggestion is correct. The case-based suggestion must be validated. Often, however, validation is much easier than generation. In those kinds of domains, case-based reasoning can provide big wins.

7.2 Interpretive CBR

Interpretive case-based reasoning is used for tasks such as classifying a new situation in context, showing cause or demonstration of rightness of an argument, position, or solution, or predicting the effects of a solution.

7.2.1 Classification

One way a case-based classifier works is to ask whether the new concept is enough like another one known to have a given classification. PROTOS provides an example. Rather

[2]PROTOS' task includes aspects of both problem-solving and interpretive CBR. The next section discusses how PROTOS uses cases to interpret open-ended classifications.

than classifying new hearing disorders using necessary and sufficient conditions, PROTOS does classification by trying to find the closest matching case in its case base.

Its first step is to guess what category the new problem fits into by looking at how the important features of the new case overlap with important features of categories of hearing disorders it knows about. This guess is PROTOS' first hypothesis about what category the new case fits into. PROTOS verifies its hypothesis by comparing the new case to exemplars in the hypothesized category to see if it can find a good match. If it finds a match, it chooses that category for the new case. However, PROTOS is also aware of the kinds of classification mistakes that are commonly made in its domain. In particular, it knows instances of examples that have been misclassified because of strong similarity to cases in the wrong category. If it finds that the new case matches one of these *failure cases,* it makes a new hypothesis – that the new case is an instance of the category this failure case was an instance of. It repeats the matching process in that category, continuing until it finds a good match.

The case-based classification procedure PROTOS implements models the largely unconscious automatic processing we do as we recognize objects in the world around us. PROTOS has no strategies for weighing evidence for and against a series of conclusions. However, weighing competing evidence is an essential part of another interpretive task, adversarial reasoning.

7.2.2 Adversarial Reasoning

Adversarial reasoning means making persuasive arguments to convince others that our positions are right. A persuasive argument states a position and supports it, sometimes with hard facts and sometimes with valid inferences. But often the only way to justify a position is by citing relevant previous experiences or cases. Law thus provides a good domain for the study of adversarial reasoning and case-based justification, and much research in this area uses the legal domain (Ashley & Rissland 1987; Ashley 1990; Bain 1989; Branting 1989; Rissland 1983). In general, cases are useful in constructing arguments and justifying positions when there are no concrete principles or only a few of them, if principles are inconsistent, or if their meanings are not well-specified.

A program called HYPO (Ashley & Rissland 1987; Ashley 1990) models the argumentation lawyers do. HYPO not only determines which cases are most similar to its new situation, but it also uses its cases to create cogent and coherent arguments in support of some position or other. HYPO's method for creating an argument and justifying a solution or position has several steps. The new situation is first analyzed for relevant factors. Based on these factors, similar cases are retrieved. They are positioned with respect to the new situation. Some support the new position and some are against it. The most on-point cases of both sets are selected. The most on-point case supporting the new situation is used to create an argument for the proposed solution. Those in the non-support set are used to pose counter-arguments. Cases in the support set are then used to counter the counter-arguments. The result of this is a set of three-ply arguments in support of the solution, each of which is justified with cases. An important side effect of creating such arguments is that potential problem areas get highlighted.

Argumentation procedures can provide the basis for critiquing solutions before using them in the world. Strategic planners do such critiquing as they create plans. Teachers do this sort of critiquing as they create lesson plans. Generals do this sort of critiquing as they create battle plans. All problem solvers that plan for contingencies must do at least some critiquing to ascertain which contingencies are the ones to plan for. Comparing and contrasting with previous cases provides a means of dealing with the complexities inherent in dealing with situations in which much is unknown.

When no case matches well enough, it is sometimes necessary to consider hypothetical situations. HYPO uses hypotheticals for a variety of tasks necessary for good interpretation: to redefine old situations in terms of new dimensions, to create new standard cases when a necessary one doesn't exist, to explore and test the limits of reasonableness of a concept, to refocus a case by excluding some issues, to tease out hidden assumptions, and to organize or cluster cases. HYPO creates hypotheticals by making "copies" of a current situation that are stronger or weaker than the real situation for one side or the other. This work is guided by a set of modification heuristics that propose useful directions for hypothetical case creation based on current reasoning needs. HYPO's strategies for argumentation guide selection of modification heuristics.

When a concept is being created on the fly, interpretation requires an additional step: derivation of a set of relevant features for a category. For example, to decide if Anne will eat swordfish, one must first attempt to characterize what makes a fish that Anne will eat acceptable to her. Otherwise, there is no basis for comparing swordfish to fish that she will eat. Mahi-mahi, a fish she has eaten, is meat-like. Trout, which she won't eat, looks like fish. These are the dimensions we need to use in determining if swordfish is in the category of fish she will eat. It is easy for us (as people) to determine what those dimensions are, but getting a computer to do it automatically is a challenge equivalent to the credit assignment problem.

7.2.3 Projecting Effects

Projection, the process of predicting the effects of a decision or plan, is an important part of the evaluative component of any planning or decision making scheme. When everything about a situation is known, projection is merely a process of running known inferences forward from a solution to see where they lead. More often, however, in real-world problems, not everything is known and effects cannot be predicted with accuracy based on any simple set of inference rules.

Cases provide a way of projecting effects based on what has been true in the past. Cases with similar plans that were failures can point to potential plan problems. Cases with similar plans that were successes give credence to the current plan.

Automated use of cases for projection has not been a focus of case-based reasoning research, but aid to a person doing projection is being addressed. CSI's Battle Planner (Goodman 1989) is a case-retrieval system whose interface is set up to allow a person to use cases to project effects. A student commander can propose a solution plan to the system. The Battle Planner retrieves the best-matching cases that use a similar plan and divides them into success and failure situations. The person can examine the cases, use them to fix his plan, and then attempt a similar evaluation of the repaired plan. Or, the person can use the system to do a sensitivity analysis. By manipulating the details of his situation and looking at the changes in numbers of wins and losses (in effect, asking a series of "what-if" questions), he can determine which factors of the current situation are the crucial ones to repair. SCIED (Chandler 1994; Chandler & Kolodner 1993) uses similar methods to help teachers made science activities work for their classrooms.

7.3 Case-Based Decision Aiding and Teaching

Psychologists have found that people are comfortable using cases to make decisions (Ross 1986, 1989, Klein & Calderwood 1988, Read & Cesa 1990) but don't always remember the right ones (Holyoak 1985, Gentner 1989). To alleviate this problem, the computer can be used as a retrieval tool to augment people's memories. Several systems mentioned earlier are designed with this in mind: ARCHIE-2, SCIED, the CSI Battle Planner, and AskJef.

We can also create teaching strategies and build teaching tools that teach based on good examples. If people are comfortable using examples to solve problems and know how to do it well, then one of our responsibilities as teachers might be to teach them the right examples and effective ways to index them (see Burke & Kass, Chapter 5, and Schank, Chapter 15). Indeed, the teaching methodology called problem-based learning holds much in common with CBR and is based on this notion (Kolodner, Hmelo, & Narayanan 1996). Third, if we understand which parts of this natural process are difficult to do well, we can teach people how to do better case-based reasoning. One criticism of using cases to make decisions, for example, is that it puts unsound bias into the reasoning system, because people tend to assume an answer from a previous case is right without justifying it in the new case. This tells us that we should be teaching people how to justify case-based suggestions and that justification or evaluation is crucial to good decision making. If we can isolate other problems people have in solving problems in a case-based way, then we can similarly teach people to do those things better.

8 Building Case-Based Reasoners

Applying CBR depends on selecting the most useful type of process for a given task and how to gather the needed cases. The above discussion suggests three types of case-based systems, all three of which have been implemented:

- **Autonomous** systems solve problems by themselves. They carry out retrieval, adaptation, and evaluation steps by themselves, as the typical expert system does, but using case-based methods. CHEF, JULIA, PLEXUS, CASEY, PROTOS, and HYPO are examples of autonomous CBR systems.

- **Human-machine** systems work along with people to solve problems or interpret situations. The system and person together comprise the case-based reasoning system. The system does what tends to be difficult for people – retrieving and presenting cases. People do what tends to be more difficult for machines – adapting cases to fit new situations and comparing and contrasting cases to interpret new situations. The simplest of these systems are retrieval-only systems, but a system might also help a person with adaptation or evaluation. Help with adaptation generally involves carrying out adaptations when asked. Help with evaluation generally means showing the user the factors that should be considered during evaluation and clustering cases to make comparison easy. The CSI Battle Planner, ARCHIE-2, CLAVIER, and SCIED are all examples of retrieval-only human-machine CBR systems.

- **Embedded** systems are those in which a retrieval-only or autonomous case-based system is embedded as a component in a system with a larger purpose than problem solving or situation interpretation (see Riesbeck, Chapter 17). For example, several interactive learning environments and training systems have the larger purpose of helping users to learn some task or domain. A case library associated with each system holds stories that help the system to illustrate its points to students (e.g., Burke & Kass, Chapter 5).

Determining which type of system to build requires analyzing the task that needs to be carried out, which requires answering questions such as:

- How much creativity is required to derive solutions?
- How much complexity is involved in evaluating solutions and effecting repairs?
- Are esthetic judgments or value judgments necessary?
- Where does a person working on this task have trouble? Which parts are easy?

Straightforward parts of the task can be done by the machine. Those requiring esthetic or value judgments might be done by the machine, with approval given by the user, or might be done by the user with retrieval and/or adaptation help provided by the system. When much creativity is involved in deriving solutions, or when there is much complexity involved in evaluating solutions and effecting repairs, a human-machine system is best.

What is required for the simplest of systems is a library of cases that coarsely cover the set of problems that come up in a domain. Both success stories and failures must be included. And, the cases must be appropriately indexed. This library, along with a friendly and useful interface, provides augmentation for human memory. And automated processes can be built on top of it incrementally.

Several case-based reasoning shells are available on the market (see Chapter 1). When using CBR shells, system builders are responsible for collecting cases for the case library. In some shells the system builders assign their indexes; other shells guide retrieval with decision trees or use keyword retrieval. Some shells require system builders to know knowledge engineering (e.g., Cognitive Systems' ReMind). Others are designed to require only expertise in a domain (e.g., Design-MUSE is aimed toward design professionals).

Case Collection

Case collection has two phases – initial collection of cases, or *seeding* of the case library, and testing of the case library to make sure it provides the coverage and reliability that are required. The result of testing and subsequently refining the case library is that the library is *trained* for the tasks it will support. Training, itself, is a life-long process – one that, in general, needs to continue throughout the life cycle of a system.

Three general principles guide case collection.

- Cases should cover the range of reasoning tasks the system will be responsible for doing or supporting.
- Over this range of reasoning tasks, cases should cover the range of well-known solutions and well-known mistakes.
- Collecting cases is an incremental process – we do the best we can to provide coverage initially, and then we find out what's missing by trying to use the cases that have already been collected.

In general, cases need to be collected according to the needs of the reasoner, that is, based on the reasoning tasks and goals it will do or support. Cases in the case library should provide as much coverage as possible about both achieving reasoning goals and the obstacles that are likely to be encountered in achieving them. But a case-based system, like an expert, needs to be given time to grow. No matter how careful we are at initially seeding the case library, there will be some problems that are beyond its capabilities. As those are encountered and the limits of the case library are found, new cases need to be added to fill the gaps. Interestingly enough, the very cases that the system can't solve, when added to the case library (with the help of an expert), can generally play that role.

While the first pass at building a case library is unlikely to provide full coverage, an organized approach to determining the kinds of cases that need to be collected can result in good initial coverage. Subsequent training using those cases will expose three kinds of problems:

- Gaps in the library's coverage
- Inadequacies of the indexing scheme

- Inadequacies in the contents of cases in the case library

Maintenance of a case library after the system is fielded is also an issue. It will generally be necessary to add cases and refine indexes in the case library as the system is used. Kitano & Shimazu (Chapter 13) present a case study of a long-term case acquisition and refinement process.

9 Summary

Case-based reasoning can mean adapting old solutions to meet new demands, using old cases to explain new situations, using old cases to critique new solutions, or reasoning from precedents to interpret a new situation or create an equitable solution to a new problem. A case-based reasoner learns as a byproduct of its reasoning activity. It becomes more efficient and more competent as a result of storing its experiences and referring to them in later reasoning.

In a departure from traditional methods of reasoning investigated by AI researchers and psychologists, case-based reasoning views reasoning as a process of remembering one or a small set of concrete instances or cases and basing decisions on comparisons between the new situation and the old one. Decomposition and recomposition are, as a result, de-emphasized, as is the use of general knowledge. Instead, emphasis is on the manipulation of knowledge in the form of concrete specific instances. Large chunks of composed knowledge are seen as the starting point for reasoning.

A case, however, is more than a large chunk of composed knowledge. It is a *contextualized* piece of knowledge representing an experience, and any case worth recording in a case library (whether human or machine) teaches a lesson fundamental to achieving the goals of the reasoner who will use it. Cases are *indexed* by combinations of their descriptors that predict the situations in which they will be useful. Cases represent knowledge at an operational level. They come in many different shapes and sizes, covering large or small time slices, associating solutions with problems, outcomes with situations, or both.

Because case-based reasoning integrates reasoning and learning, it is not enough for a case-based reasoner to stop reasoning after it derives a solution. Rather, it must continue by collecting feedback about its solution and evaluating that feedback. Without evaluation processes based on feedback, learning could not happen reliably, and case-based reasoning itself would be too unreliable to depend on.

The quality of a case-based reasoner's reasoning depends on the experiences it has had, its ability to understand new situations in terms of those old experiences, its adeptness at adaptation, and its adeptness at evaluation and repair. The major processes employed by a case-based reasoner are case storage and retrieval, adaptation, and criticism.

Case-based reasoning is applicable to a wide range of real world situations, ranging from knowledge-rich situations in which construction of solutions is complex to knowledge-poor situations in which cases provide the only available knowledge. It has many advantages, allowing a reasoner to propose solutions to problems quickly, to reason in domains that are not well understood, to evaluate solutions when algorithmic methods are not available, to avoid previous problems, and to focus on important parts of a situation. Its major disadvantages are linked to using cases poorly, for example, relying too heavily on their proposals without evaluating them. Yet case-based reasoning is a natural way of reasoning for people and one that shows considerable promise for machines.

There are several kinds of case-based systems that can be built. Automated systems solve problems from start to finish. If we take case-based reasoning seriously as a cognitive model, however, we can think about building systems that can interact with people in a natural way to solve problems. The simplest of these uses the machine's case library to augment the memory of a person, leaving the human user to reason based on retrieved cases. More sophisticated systems embed case-based reasoners or case retrievers in more broad-purpose systems, e.g., interactive learning environments, to provide more active assistance.

10 Acknowledgments

This article is adapted, with the permission of the publisher, from Kolodner, Janet L., 1993, *Case-Based Reasoning*, Morgan Kaufmann Publishers, Inc., San Mateo, CA. More detail about each of the issues discussed in this article can be found in that book. The work reported in this article was partially funded by DARPA under Contract No. N00014-91-J-4092, monitored by ONR; by ONR under Contract No. N00014-92-J-1234; by NSF under Contract No. IRI-8921256; and by ARI under Contract No. MDA903-90-K-0112.

References

Alterman, R. 1986. An adaptive planner. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 65–69. Philadelphia, PA: AAAI.

Alterman, R. 1988. Adaptive planning. *Cognitive Science* 12:393–422.

Ashley, K. 1990. *Modeling legal argument: reasoning with cases and hypotheticals*. Cambridge: MIT Press.

Ashley, K., and Rissland, E. 1987. Compare and contrast, a test of expertise. In *Proceedings of the Sixth Annual National Conference on Artificial Intelligence*, 273–284. San Mateo, CA: AAAI.

Bain, W. 1989. Judge. In Riesbeck, C., and Schank, R., eds., *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates. chapter 4, 93–140.

Barber, J.; Bhatta, S.; Goel, A.; Jacobsen, M.; Pearce, M.; Penberthy, L.; Shankar, M. and Stroulia, E. 1992. AskJeff: Integrating case-based reasoning and multimedia technologies for interface design support. In Gero, J., ed., *Artificial Intelligence in Design*, 457–476. Boston: Kluwer.

Bareiss, R. 1989. *Exemplar-Based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning*. San Diego: Academic Press.

Barletta, R., and Hennessy, D. 1989. Case adaptation in autoclave layout design. In Hammond, K., ed., *Proceedings of the DARPA Case-Based Reasoning Workshop*, 203–207. San Mateo: DARPA.

Branting, K. 1989. Integrating generalizations with exemplar-based reasoning. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, 139–146. Ann Arbor, MI: Cognitive Science Society.

Carbonell, J. 1986. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In Michalski, R.; Carbonell, J.; and Mitchell, T., eds., *Machine Learning: An Artificial Intelligence Approach*, volume 2. Los Altos, CA: Morgan Kaufmann. 371–392.

Chandler, T. 1994. The science education advisor: Applying a user centered design approach to the development of an interactive case-based advising system. *Journal of Artificial Intelligence in Education* 5(2):283–319.

Chandler, T., and Kolodner, J. 1993. The science education advisor: A case-based advising system for lesson planning. In *Proceedings of the World Conference on Artificial Intelligence in Education.*

Charniak, E., and McDermott, D. 1987. *Introduction to Artificial Intelligence.* Reading, MA: Addison-Wesley.

de Kleer, J., and Brown, J. 1984. A qualitative physics based on confluences. *Artificial Intelligence* 24:7–83.

Domeshek, E. 1992. *Do the Right Things: A Component Theory for Indexing Stories as Social Advice.* Ph.D. Dissertation, The Institute for the Learning Sciences, Northwestern University.

Domeshek, E.; Herndon, M.; Bennett, A.; and Kolodner, J. 1994. A case-based design aid for conceptual design of aircraft subsystems. In *Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications*, 63–69. Washington: IEEE Computer Society Press.

Domeshek, E., and Kolodner, J. 1992. Towards a case-based aid for conceptual design. *International Journal of Expert Systems* 4(2):201–220.

Domeshek, E.; Kolodner, J.; and Zimring, C. 1994. The design of a tool kit for case-based design aids. In Gero, J., ed., *Artificial Intelligence in Design*, 109–126. Boston: Kluwer.

Forbus, K. 1988. Qualitative physics: Past, present, and future. In Shrobe, H., ed., *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence.* Palo Alto: Morgan Kaufmann. 239–296.

Goel, A. 1989. *Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving.* Ph.D. Dissertation, The Ohio State University.

Goel, A., and Chandrasekaran, B. 1989. Case-based design: A task analysis. In Hammond, K., ed., *Proceedings of the DARPA Case-Based Reasoning Workshop*, 100–109. San Mateo: DARPA.

Goodman, M. 1989. CBR in battle planning. In Hammond, K., ed., *Proceedings of the DARPA Case-Based Reasoning Workshop*, 264–269. San Mateo: DARPA.

Hammond, K. 1989. *Case-Based Planning: Viewing Planning as a Memory Task.* San Diego: Academic Press.

Hayes, P. 1985. Naive physics 1: Ontology for liquids. In Hobbs, J., and Moore, R., eds., *Formal theories of the commonsense world.* Norwood, NJ: Ablex. 71–107.

Hennessy, D., and Hinkle, D. 1992. Applying case-based reasoning to autoclave loading. *IEEE Expert* 7(5):21–26.

Hinrichs, T. 1992. *Problem Solving in Open Worlds: A Case Study in Design.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Kolodner, J. 1987. Extending problem solving capabilities through case-based inference. In *Proceedings of the Fourth International Workshop on Machine Learning*, 167–178. Irvine, CA: Machine Learning.

Kolodner, J. 1993. *Case-Based Reasoning.* San Mateo, CA: Morgan Kaufmann.

Kolodner, J.; Hmelo, C.; and Narayanan, N. 1996. Problem-based learning meets case-based reasoning. In *Proceedings of the Second International Conference on the Learning Sciences.* Submitted.

Koton, P. 1988. Reasoning about evidence in causal explanations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 256–261. San Mateo, CA: AAAI.

Leake, D. 1992. *Evaluating Explanations: A Content Theory.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Marks, M.; Hammond, K.; and Converse, T. 1989. Planning in an open world: A pluralistic approach. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, 749–756. Ann Arbor, MI: Cognitive Science Society.

Navinchandra, D. 1988. Case-based reasoning in CYCLOPS, a design problem solver. In Kolodner, J., ed., *Proceedings of the DARPA Case-Based Reasoning Workshop*, 286–301. Palo Alto: DARPA.

Rissland, E. 1983. Examples in legal reasoning: Legal hypotheticals. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.* IJCAI.

Robinson, S., and Kolodner, J. 1991. Indexing cases for planning and acting in dynamic environments: Exploiting hierarchical goal structures. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, 882–886. Chicago, IL: Cognitive Science Society.

Schank, R., and Osgood, R. 1990. A content theory of memory indexing. Technical Report 2, Northwestern University Institute for the Learning Sciences.

Stroulia, E.; Shankar, M.; Goel, A.; and Penberthy, L. 1992. A model-based approach to blame assignment in design. In Gero, J., ed., *Artificial Intelligence in Design*, 519–537. Boston: Kluwer.

Sycara, K., and Navinchandra, D. 1989. Index transformation and generation for case retrieval. In Hammond, K., ed., *Proceedings of the DARPA Case-Based Reasoning Workshop*, 324–328. San Mateo: DARPA.

Turner, R. M. 1994. *Adaptive Reasoning for Real-World Problems: A Schema-Based Approach.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Book table of contents for resolving chapter references