Artificial Intelligence Review **6**, 3—34, 1992.

An Introduction to Case-Based Reasoning*

Janet L. Kolodner**

College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, U.S.A.

Abstract. Case-based reasoning means using old experiences to understand and solve new problems. In case-based reasoning, a reasoner remembers a previous situation similar to the current one and uses that to solve the new problem. 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 (much like lawyers do) or create an equitable solution to a new problem (much like labor mediators do). This paper discusses the processes involved in case-based reasoning and the tasks for which case-based reasoning is useful.

Key Words: Case-based reasoning, problem-solving, experience

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 the following:

> I once served tomato tart (made from mozzerella 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 shouldn't 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 fish 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.

Here she is using examples and counterexamples of a premise (Anne doesn't eat fish) to try to derive an interpretation of the premise that stands up to scrutiny.

The hypothetical host is employing *Case-Based Reasoning* (CBR) (e.g., Hammond 1989c, Kolodner 1988a, Riesbeck and Schank 1989) to plan a meal. In case-based reasoning, a reasoner remembers previous situations similar to the current one and uses them to help solve the new problem. In the example above, remembered cases are used to suggest a means of solving the new problem (e.g., to suggest a main dish), to suggest a means of adapting 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 the fish, will she eat swordfish?).

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 (much like lawyers do) or create an equitable solution to a new problem (much like labor mediators do).

If we watch the way people around us solve problems, we are likely to observe case-based reasoning in use 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 he's seen a patient with similar symptoms previously, he is likely to remember the old case and propose the old diagnosis as a solution to his new problem. If proposing those disorders was time-consuming previously, this is a big savings of time. Of course, the doctor can't assume the old answer is correct. He/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 allows him to generate a plausible answer easily.

Similarly, a car mechanic faced with an unusual mechanical problem is likely to remember other similar problems and to consider whether their solutions explain the new one. Doctors evaluating the appropriateness of a therapeutic procedure or judging which of several are appropriate are also likely to remember instances using each procedure and to make their judgements based on previous experiences. Problem instances of using a procedure are particularly helpful here; they tell the doctor what could go wrong, and when an explanation is available explaining why the old problem occurred, they focus the doctor in finding out the information he needs 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. And many management decisions are made based on previous experience.

Case-based reasoning is also used extensively in day-to-day common-sense reasoning. The meal planning example above is typical of the reasoning we all do from day to day. 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. A childcare provider mediating an argument between two children remembers what worked and didn't work previously in calming such situations, and bases her suggestion on that.

In general, the second time solving some problem or doing some task is easier than the first because we 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.

The quality of a case-based reasoner's solutions depends on four things:

- the experiences it's had,
- its ability to understand new situations in terms of those old experiences,
- its adeptness at adaptation, and
- its adeptness at evaluation.

The less experienced reasoner will always have fewer experiences to work with than the more experienced one. But, as we shall see, the answers given by a less experienced reasoner won't necessarily be worse than those given by the experienced one if he is creative in his understanding and adaptation. Any programs we write to automatically do case-based reasoning will need to be seeded with a representative store of experiences. Those experiences (cases) should cover the goals and subgoals that arise in reasoning and should include both successful and failed attempts at achieving those goals. Successful attempts will be used to propose solutions to new problems. Failed attempts will be used to warn of the potential for failure.

The second, that of understanding a new problem in terms of old experiences has two parts: *recalling* old experiences and *interpreting* the new situation in terms of the recalled experiences. The first we call the **indexing problem**. In broad terms, it means finding in memory the experience closest to a new situation. In narrower terms, we often think of it as the problem of assigning indexes to experiences stored in memory so that they can be recalled under appropriate circumstances. Recalling cases appropriately is at the core of case-based reasoning.

Interpretation is the process of comparing the new situation to recalled experiences. When problem situations are interpreted, they are compared and contrasted to old problem situations. The result is an interpretation of the new situation, the addition of inferred knowledge about the new situation, or a classification of the situation. When new solutions to problems are compared to old solutions, the reasoner gains an understanding of the pros and cons of doing something a particular way. We generally see interpretation processes used when problems are not well understood and when there is a need to criticize a solution. When a problem is well understood, there is little need for interpretive processes.

The third, **adaptation**, is the process of fixing up an old solution to meet the demands of the new situation. Eight methods for adaptation have been identified. They can be used to insert something new into an old solution, to delete

something, ór to make a substitution. Applying adaptation strategies straightforwardly results in competent but often unexciting answers. Creative answers result from applying adaptation strategies in novel ways.

One of the hallmarks of a case-based reasoner is its ability to learn from its experiences, as a doctor might do when he caches a hard-to-solve problem so that he can solve it easily another time. In order to learn from experience, a reasoner requires feedback so that it can interpret what was right and wrong with its solutions. Without feedback, the reasoner might get faster at solving problems but would repeat its mistakes and never increase its capabilities. Thus, **evaluation** and consequent **repair** are important contributors to the expertise of a case-based reasoner. Evaluation can be done in the context of the outcomes of other similar cases, can be based on feedback or can be based on simulation.

1. REASONING USING CASES

There are two styles of case-based reasoning: problem solving and interpretive. In the problem solving style of case-based reasoning, solutions to new problems are derived using old solutions as a guide. Old solutions can provide almost-right solutions to new problems and they can provide warnings of potential mistakes or failures. In the example above, cases suggest tomato tart as a main dish, a method of adapting tomato tart for those who don't eat cheese, and a type of fish that Anne will eat. A case also warns of the potential for a failure — Anne won't eat certain kinds of fish.

In the interpretive style, new situations are evaluated in the context of old situations. A lawyer, for example, uses interpretive case-based reasoning when he uses a series of old cases to justify an argument in a new case. But interpretive CBR can also be used during problem solving, as we saw the host in our initial example do when trying to justify serving swordfish to a guest known not to like some kinds of fish.

As we shall see, both styles of case-based reasoning depend heavily on a case retrieval mechanism that can recall useful cases at appropriate times, and in both, storage of new situations back into memory allows learning from experience. The problem solving style is characterized by heavy use of adaptation processes to generate solutions and interpretive processes to judge derived solutions. The interpretive style uses cases to provide justifications for solutions, allowing evaluation of solutions when no clear-cut methods are available and interpretation of situations when definitions of the situation's boundaries are open-ended or fuzzy. We will show examples of both kinds of case-based reasoning in this section and discuss the applicability of both.

1.1. *CBR and Problem Solving*

The host in the initial example used problem solving case-based reasoning to propose tomato tart as the main dish and to suggest a means of adapting it to suit the guest allergic to milk products. Also as part of the problem solving

process, she used a remembered case to anticipate that one of the guests would not eat fish, causing her to plan around that problem.

Problem solving case-based reasoning is useful for a wide variety of problem solving tasks, including planning, diagnosis, and design. In each of these, cases are useful in suggesting solutions and in warning of possible problems that might arise.

1.1.1. *CBR for design*

We can view the meal planning example as a kind of design problem. 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 are fulfilled. In that case, solving the problem requires respecifying the problem so that the most important constraints are fulfilled and other are compromised.

Consider, for example, the meal planner in the initial example. She must satisfy the likes and dislikes of her guests, must keep the meal inexpensive, must make the meal hearty, and must use tomatoes. In addition, she must make the main and side dishes compatible with each other, must not repeat major ingredients across dishes, must make the appetizer complement the rest of the meal, etc. Many different meals would do this. For example, vegetarian lasagne would work as a main dish if one tray were made with tofu instead of cheese. And any number of side dishes and appetizers would complement it. Several other pasta dishes would also suffice as main dishes, each with any number of side dishes and appetizers to complement it. A combination of main dishes, one of which would satisfy the meat-and-potatoes people, another the vegetarians, etc., and complementary side dishes and appetizers would also work. With so many options, where should the planner begin?

Suppose now that this meal planner remembers a meal she served to a large group of people. It was easy to make in large quantities, inexpensive, hearty, and used tomatoes. In that meal, she served antipasto, lasagne, a large green salad, and garlic bread. Only this time, she has vegetarians coming for dinner and one guest is allergic to milk products. The lasagne can be adapted to better fit the new situation by taking out the meat. The antipasto can be adapted by substituting tuna for the meat. This will satisfy all constraints except the one specifying that one guest doesn't eat dairy products. The meal can be further adapted such that in one tray of lasagne, tofu cheese substitute is used instead of cheese. This adaptation of the old menu is now suitable for the new situation.

This is an example of an underconstrained problem. The constraints provide guidelines but don't point the reasoner toward a particular answer. In addition, the search space is huge, and while there are many answers that would suffice, they are sparse enough within the search space that standard search methods might spend a long time finding one. Furthermore, the problem is too big to solve in one chunk, but the pieces of the problem interact with each other in strong ways. Solving each of the smaller pieces of the problem in isolation and

putting it all back together again would almost always violate the interactions between the parts.

For these kinds of problems, which I like to call *hardly decomposable*, cases can provide the glue that holds a solution together. Rather than solving the problems by decomposing them into parts, solving for each, and recomposing the parts, as can be done with nearly-decomposable problems, a case suggests an entire solution, and the pieces that don't fit the new situation are adapted. While considerable adaptation might be necessary to make an old solution fit a new situation, this methodology is almost always preferable to generating a solution from scratch when there are many constraints and when solutions to parts of problems cannot be easily recomposed. In fact, engineering and architectural design is almost entirely a process of adapting an old solution to fit a new situation or merging several old solutions to do the same.

Solving a problem by adapting an old solution allows the problem solver to avoid dealing with many constraints, and keeps it from having to break the problem into pieces needing recomposition. For example, the compatibility of the main and side dishes is never considered while solving the problem since the old case provides that. Nor are ease of preparation, expense, or heartiness considered in generating a solution. The old case provides solutions to those constraints also. The problem is never broken into parts that need to be recomposed. Rather, faulty components are corrected in place.

The other major role of cases in design, as for all problem solving tasks, is to point out problems with proposed solutions. When the meal planner remembers the meal where Anne didn't eat fish, it is warned of the potential that its proposed solution will fail.

Several problem solvers have been built to do case-based design. JULIA (Kolodner 1987, Hinrichs 1988, Hinrichs 1989) plans meals, and the examples shown above are all among those JULIA has solved. CYCLOPS (Navinchandra 1988) uses case-based reasoning for landscape design. KRITIK (Goel 1989, Goel and Chandrasekaran 1989) combines case-based with model-based reasoning for design of small mechanical assemblies. It uses case-based reasoning to propose solutions and uses the model to verify its proposed solutions, to point out where adaptation is needed, and to suggest adaptations.

At least one design problem solver is being put to use in the real world. CLAVIER (Barletta and Hennessy 1989) is being used at Lockheed to lay out pieces made of composite materials in an oven to bake. The task is a apparently a black art, i.e., there is no complete causal model of what works and why. Pieces of different sizes need to be in particular parts of the oven, but the size of some pieces and density of a layout might keep other pieces for heating correctly. The person who was in charge of layout kept a card file of his experiences, both those that worked and those that didn't. Based on those experiences, CLAVIER can place pieces in appropriate parts of the oven and avoid putting pieces in the wrong places. It works as well as the expert whose experiences it uses, and is thus useful to Lockheed when the expert is unavailable. CLAVIER almost always uses several cases to do its design. One provides

an overall layout, which is adapted appropriately. The others are used to fill in holes in the layout that adaptation rules by themselves cannot cover.

One can also look at mediation as a kind of design in which the problem specification is overconstrained rather than underconstrained. In mediation, two adversaries have conflicting goals. It is impossible to fulfill the entire set of goals of either side. The role of the mediator is to derive a compromise solution that partially achieves the goals of both adversaries as well as possible.

In solving overconstrained problems, the design specifications must be respecified while solving the problem. When overconstrained problems are solved by constraint methods, many different ways of relaxing constraints must usually be attempted before settling on a set that work. When case-based reasoning is used, a close solution to the constraint relaxation problem is provided by the remembered case, and it is adapted. MEDIATOR (Simpson 1985, Kolodner and Simpson 1989), the earliest case-based problem solver, solved simple resource disputes, e.g., two children wanting the same candy bar or two faculty members wanting to use the copy machine at the same time. PERSUADER (Sycara 1987) solved labor management disputes. In generating solutions to new labor-management disputes, PERSUADER first applied *parameter adjustment* strategies to the best old solution it remembered to make relatively easy changes in an old contract, the kind that must be made all the time, e.g., cost of living adaptations. This resulted in a ballpark solution. It then applied special purpose critics to *evaluate* the ballpark solution in order to identify more specialized problems with an old contract, e.g., to recognize whether or not the company could afford the contract. It then adapted the ballpark solution appropriately either by using an adaptation strategy suggested by another case, or by applying another specialized set of critics. Finally, it used another special purpose set of critics to adapt the solution in order to compensate for any changes that upset the equity of the old solution.

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 whole thing. In general, some case provides a framework for a solution and other cases are used to fill in missing details. In this way, decomposition and recomposition are avoided, as are large constraint satisfaction and relaxation problems.

1.1.2. *CBR for planning*

Planning is the process of coming up with a sequence of steps or schedule for achieving some state of the world. The state that must be achieved may be designated in concrete terms, as in e.g., designating the end state for the Tower of Hanoi problem (configure the game board such that disk1 is on top of disk2 is on top of disk3 and all are on peg3) or describing the end result of making a delivery (the box labeled 'Klein' should be on the Klein driveway). Or it can be designated in terms of constraints that must be satisfied, as in e.g., scheduling the gates at an airport (each flight should be assigned a gate, no two flights should be at a gate at once, no on-time flight should have to wait for a gate, . . .).

In the first case, the end product of the planning process is a set of steps. In the second, the end product is a schedule or state of the world, but a planning process must be used to create it.

An early case-based planner was CHEF (Hammond 1989a). CHEF created new recipes based on those it already knew about. For example, in creating a recipe that combined beef and broccoli, it remembered its recipe for *chicken and snow peas* and adapted that recipe appropriately. First, beef was substituted for chicken and broccoli for snow peas. Then, the set of steps used to create chicken and snow peas was fixed. Since beef has no bone, the deboning step was deleted. Since broccoli takes longer to cook than snow peas, the time designation was changed in the step where the vegetable was cooked.

There are many problems that must be dealt with in planning. First is the problem of protections. Good plans are sequenced, whenever possible, such that late steps in the plan don't undo the results of earlier steps and so that preconditions of late steps in the plan are not violated by the results of earlier steps. (See Charniak and McDermott (1985) for an excellent explanation of these problems.) This requires that the effects of plan steps be projected into the future (the rest of the plan). Second is the problem of preconditions. A planner must make sure that preconditions of any plan step are fulfilled before scheduling that plan step. Thus, planning involves scheduling steps that achieve preconditions in addition to scheduling major steps themselves. These two problems together, when solved by traditional methods, require considerable computational effort. As the number of plan steps increases, the computational complexity of projecting effects and comparing preconditions increases exponentially.

Case-based reasoning deals with these problems by providing plans that have already been used and in which these problems have already been worked out. The planner is required only to make relatively minor fixes in those plans rather than having to plan from scratch. A recipe, for example, provides an ordering of steps that plans for and protects preconditions of each of its steps.

Case-based reasoning also suggests solutions to more complex planning problems. (See Marks, Hammond, and Converse (1989) for a nice explanation of these problems.) For example, in the real world, the number of goals competing for achievement at any time is quite high, and new ones are formed in the normal course of activity. If we try to achieve each one independently of the others, then planning and execution time are at least the sum of achieving each one, and probably more because of interactions. If a planner can notice the possibility of achieving several goals simultaneously or in conjunction with each other, this complexity can be cut significantly. Case-based reasoning provides a method for doing this. Previously-used plans are saved and indexed by the conjunction of goals they achieve. If the conjunct of goals is repeated, the old plan that achieves them together can be recalled and repeated.

Another set of complexities that crop up in planning come from dealing with the relationship between planning and execution in a realistic way. While the traditional view of planning, as suggested above, was that a planner would create a plan (sequence of steps) to be executed later, the newer view of

planning says that planning and execution must be combined. That is, we cannot expect a well-thought-out sequence of steps to work when executed in the real world. There are several reasons for this. First, it is unrealistic to think that all knowledge needed to plan well is known at the start of planning — we often find things out as we go along, and some things we never know for sure. Second, the world is unpredictable. Unexpected interruptions crop up (e.g., a fire alarm rings in the middle of doing some task). Agents who were unknown at the time of planning change things in an unexpected way (e.g., someone buys all the nails of a particular kind that you needed for your plan and they are unavailable). And we can't predict the actions of known agents all the time (e.g., someone else in the family got hungry for chocolate and ate the chocolate chips you were planning to use in your baking). In general, conditions in the world can change between coming up with a plan and carrying it out.

This requires that our planners be able to put off at least some planning until execution time when more is known and that they be able to do execution-time repairs on already-derived plans that cannot be carried out. In addition, if the planner can anticipate execution-time problems, fewer repairs will have to be done at execution time.

Case-based reasoners are addressing many of these issues. PLEXUS (Alterman 1988), a program that knows how to ride a subway, is able to do execution-time repairs by adapting and substituting semantically-similar steps for those that have failed. For example, when riding the New York subway, having ridden BART in the past, PLEXUS expects to find a machine to buy a ticket from. When it doesn't see one, it finds another plan that would achieve the same purpose as buying a ticket from a machine. It does this by finding an appropriate sibling or close cousin of the violated plan step in its semantic network. It attempts to substitute another ticket-buying plan it knows about, buying tickets from a cashier (as is done to get into the movies). It adapts that plan by substituting token for ticket, since it is a token that is necessary in the New York subway.

CHEF (Hammond 1989a) addresses the problem of anticipating problems 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 its hypothesized repair in memory, and indexes the case by features that are likely to predict that the problem will recur. Before it begins plan derivation, it looks for failure situations and uses any it finds to anticipate the problems they point out. Later, it uses the repaired failure situations to suggest a plan that will avoid the problem it has anticipated. Once, for example, CHEF created a strawberry souffle by modifying a receipt for a vanilla souffle. When it made the strawberry souffle, the souffle did not rise. It explained the failure to rise as an imbalance in the amount of leavening and liquid with too much liquid. Since one way to get rid of extra liquid is to pour it off, it hypothesized that had it poured off the extra liquid created when the strawberries were pulped, the souffle would have risen. It indexed the whole case as a souffle with fruit. The next time it attempted to make a souffle with fruit, it remembered this case before attempting planning, warning it of the potential for an imbalance between liquid and leavening.

During planning, the repaired strawberry souffle recipt suggested pouring off the liquid created by pulping the fruit.

Other case-based planners address other of these problems. TRUCKER (Hammond 1989b) 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 1989) is a diagnosis program. It 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. Thus, if it is tracking down a pulmonary complaint and the patient offers information about a heart condition that it had not known about previously, MEDIC analyzes which is more important to follow up. If it turns out the heart problem is more important, MEDIC will interrupt its current diagnosis plan and move to the more appropriate one, and will return to the original one if appropriate later. EXPEDITOR plans the events in the life of a single parent who must deal with kids and work. It caches its experiences 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. Finally, the CSI BATTLE PLANNER (Goodman 1989) shows how cases can be used to criticize and repair plans before they are executed.

1.1.3. *CBR for diagnosis*

In diagnosis, a problem solver is given a set of symptoms and asked to explain them. When there are a small number of possible explanations, one can view diagnosis as a classification problem. When the set of explanations cannot be enumerated easily, we can view diagnosis as the problem of creating an explanation. A case-based diagnostician can use cases to suggest explanations for symptoms and to warn of explanations that have been found to be inappropriate in the past. The following example is from an early case-based reasoning project called SHRINK (Kolodner and Kolodner 1987), designed to be a psychiatric diagnostician.

> A psychiatrist sees a patient who exhibits clear signs of Major Depression. The patient also reports, among other things, that she recently had a stomach problem that doctors could find no organic cause for. While random complaints are not usually given a great deal of attention in psychiatry, this time the doctor is reminded of a previous case in which he diagnosed a patient for Major Depression who also complained of a set of physical problems that could not be explained organically: Only later did he realize that he should also have taken those complaints into account; he then made the diagnosis of Somatization Disorder with Secondary Major Depression. Because he is reminded of the previous case, the psychiatrist hypothesizes that this patient too might have Somatization Disorder with Depression secondary to that and follows up that hypothesis with the appropriate diagnostic investigation.

Here the doctor uses the diagnosis from the previous case to generate a hypothesis about the diagnosis in the new case. This hypothesis provides the doctor with a reasoning shortcut and also allows him to avoid the mistake made previously. In addition, the hypothesis from the previous case causes him to focus his attention on aspects of the case that he would not have considered

otherwise: the unexplainable physical symptoms associated with Somatization Disorder.

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 diagnosis to fit a new situation. CASEY (Koton 1988) was able to diagnosis heart problems by adapting the diagnoses of previous heart patients to new patients.

For example, when CASEY was trying to diagnose a patient Newman, it was reminded of another patient David. While both shared many symptoms, there were also differences between them. Newman, for example, had calcified heart valves and syncope on exertion, while David did not. CASEY adapted the diagnosis for David to account for these differences. Newman's aortic valve calcification was inserted as a additional evidence for aortic valve disease, his syncope was inserted as additional evidence for limited cardiac output, and mitral valve disease was added to the diagnosis.

CASEY is a relatively simple program built on top of an existing model-based diagnostic program. When a new case is similar to one it has seen previously, it is several orders of magnitude more efficient at generating a diagnosis than is the model-based program (Koton 1988). And, because its adaptations are based on a valid causal model, its diagnoses are as accurate as those made from scratch based on the same causal model.

Cases are also useful in diagnosis in pointing the way out of previously-experienced reasoning quagmires. While this normally happens as a side effect of SHRINK's and CASEY's reasoning, PROTOS (Bariess 1989) is designed to ensure that this happens in an efficient way. PROTOS diagnoses hearing disorders. In this 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 diagnosis to the correct one. In one problem it worked on, for example PROTOS thought that its new case was an instance of cochlear-age. When it examined the most prototypical case of cochlear-age, however, it was not a very good match. However, it there was a difference link labeled notch-at-4k connecting that case to another case whose diagnosis was cochlear-age-and-noise. Since the new case had the feature notch-at-4k, it followed that link, found a case very similar to its new one, and was able to make a valid diagnosis first time around.

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, the diagnostician cannot assume that a case-based suggestion is the answer. 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.

1.1.4. *CBR for explanation*

Explaining anomalies is prevalent in all of our problem solving and understanding activities as people. If we read in the paper about a plane crash, we try to explain why it happened. If we fail at something we are doing, we try to explain what went wrong so we won't repeat it. If we hear of someone doing something unexpected, we try to explain it. Explanation has been called the credit assignment problem and the blame assignment problem, depending on whether one is explaining a success or a failure. In general, it is the problem of zeroing in on or identifying what was responsible for something that happened. It is a problem that the AI world has been grappling with for some time.

A case-based approach to explanation (Schank, 1986) says that one can explain a phenomenon by remembering a similar phenomenon, borrowing its explanation, and adapting it to fit. The SWALE (Kass and Leake 1988) program does just that. When it hears that Swale, a racehorse in the prime of its life, died in its stall, it remembers several similar situations, each of which allows it to derive an explanation for the Swale case. When it remembers Jim Fixx dying of a heart attack after a run, it considers whether Swale had just been out for a run and if he had a heart condition that his owners had been unaware of. When it remembers Janis Joplin, it considers whether Swale was taking illicit drugs. Some explanations it can derive in this way are more plausible than others and some require more adaptation than others. For example, in considering the Janis Joplin explanation, it must also come up with someone who wanted Swale to have the drugs (Joplin wanted them herself, but a horse can't) and a reason why that person would want Swale to have the drugs. This requires quite a bit more inference than an explanation based on the Jim Fixx experience.

Thus, case-based explanation requires a retrieval mechanism that can retrieve similar cases, an adaptation mechanism that must be quite creative, and a validation mechanism that can decide if a proposed explanation has any merit.

1.2 *Interpretive CBR*

Interpretive case-based reasoning is a process of evaluating situations or solutions in the context of previous experience. It takes a situation or solution as input, and its output is a classification of the situation, an argument supporting the classification or solution, and/or justifications supporting the argument or solution. It is useful for situation classification, evaluation of a solution, argumentation, justification of a solution, interpretation, or plan, and projection of effects of a decision or plan.

Supreme Court justices use interpretive case-based reasoning when they make their decisions. They interpret a new case in light of previous cases. Is this case like the old one? How is it the same? How is it different? Suppose we interpret it in some way, what are its implications. Lawyers use interpretive case-based reasoning when they use cases to justify arguments.

People on the street use interpretive case-based reasoning every day. The child who says, 'But you let sister do it', is using a case to justify his/her argument. Managers making strategic decisions base their reasoning on what's been

true in the past. And we often use interpretive case-based reasoning to evaluate the pros and cons of a problem solution. An arbitrator, for example, who has just come up with a salary for a football player, might look at other players with similar salaries to judge if this new salary is consistent with other salary decisions. A battle planner might look at battles where strategies similar to the one he has just chosen were used to project the effects of his chosen strategy.

Interpretive case-based reasoning is most useful for evaluation when there are no computational methods available to evaluate a solution or position. Often, in these situations, there are so many unknowns that even if computational methods were available, the knowledge necessary to run them would usually be absent. A reasoner who uses cases to help evaluate and justify decisions or interpretations is making up for his lack of knowledge by assuming that the world is consistent.

We briefly discuss three tasks where interpretive case-based reasoning is useful: justification, interpretation, and projection. In justification, one shows cause or proof of the rightness of an argument, position, or solution. In interpretation, one tries to place a new situation in context. Projection means predicting the effects of a solution. All of these tasks share the common thread of argumentation. Some cases will support one interpretation or effect. Others will support a different one. The reasoner must compare and contrast the cases to each other to finally come up with a solution.

1.2.1. *Justification and adversarial reasoning*

Adversarial reasoning means making persuasive arguments to convince others that we or our positions are right. Lawyers argue adversarially on a day to day basis. So do the rest of us as we try to convince others of our position on some issue. We also argue adversarily with ourselves when trying to convince ourselves of the quality or utility of some solution we have just derived. In making a persuasive argument, we must state a position and support 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.

The American legal system is a case-based system. There are many rules, but each has terms that are underspecified and many contradict each other. The definitive way of interpreting laws is to argue based on cases. Law thus provides a good domain for the study of adversarial reasoning and case-based justification. Much of the work in this area is in the legal domain (Rissland 1983, Bain 1986, Ashley 1987, 1988, Branting 1989,).

HYPO (Ashley, 1988, Rissland and Ashley 1987) is the earliest and most sophisticated of the case-based legal reasoners. HYPO's methods for creating an argument and justifying a solution or position has several steps. First, the new situation is analyzed for relevant factors. Based on these factors, similar cases are retrieved. They are positioned with respect to the new situation. Some support it 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.

Consider, for example, a non-disclosure case that HYPO argued.[1] John Smith had developed a structural analysis program called NIESA while an employee of SDRC. He had generated the idea for the program and had been completely responsible for its development. When joining SDRC, he had signed an Employee Confidential Information Agreement in which he agreed not to divulge or use any confidential information. Upon leaving SDRC, he went to work for EMRC as VP for engineering. Eleven months later, EMRC began marketing a structural analysis program called NIESA. Smith had used his development notes from SDRC's NIESA program in developing the new program. SDRC had disclosed parts of the NIESA code to some fifty customers. Now, SDRC is suing John Smith for violations of the non-disclosure agreement.

HYPO is arguing for the defendant (SDRC). It uses the factors present in this case to pose a set of questions that must be answered: Did disclosure of NIESA code to 50 customers annul John Smith's non-disclosure agreement? Did the fact that Smith was the sole developer annul the non-disclosure agreement? Several cases are recalled, and an argument is made based on Midland-Ross v. Sunbeam that, since the plaintiff disclosed its product information to outsiders, the defendant should win a claim for trade secrets misappropriation. However, a counter-argument can be created based on differences between the two cases. In the Midland-Ross case, there was disclosure to many more outsiders, and the defendant received something of value for entering into the agreement. This point can be supported by Data General v. Digital Computer, where the plaintiff won, even though it had disclosed to more outsiders than in the Midland-Ross case. However, a rebuttal is made based on that case. In that case the defendant won because the disclosures were restricted. In this case, disclosure was to the sole developer of the new product.

We see similar argumentation and justification in day-to-day argumentation. For example,[2] we would expect an almost-thirteen year old who is told by his parents that he cannot to go see a midnight showing of *Little Shop of Horrors* to use all the cases at his disposal to plead his case. If his friend, who is already 13, is allowed to see it, he will cite that case in his favor. If his sister has been allowed to go at midnight to see it, he will cite that. And as his parents give their arguments, he will use other cases to counter those.

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.

1.2.2. *Classification and interpretation*

Is swordfish a fish Anne will eat or isn't it? This is the interpretive question the reasoner in the example at the beginning of this chapter must face. In general, interpretation in the context of case-based reasoning means deciding whether a concept fits some open-ended or fuzzy-bordered classification. The classification might be derived on the fly (e.g., types of fish Anne will eat) or it might be

well-known but not well-defined in terms of necessary and sufficient conditions. Many of the classifications we assume are defined are classifications of the open-ended variety. For example, we assume that a vehicle means a thing with wheels used for transportation, but when a sign says 'No vehicles in the park', it is probably not referring to a wheelchair or a baby stroller, both of which fit our simple definition. Making such distinctions is much of what lawyers are asked to do everyday. Similarly, when a physician must determine if someone is schizophrenic or an audiologist must determine if someone has a particular hearing disorder, the recognition process is fraught with ambiguity. There are many ways schizophrenia can show itself, and necessary and sufficient conditions don't say how to deal with the borderline cases. And, as we see in the swordfish example, we are called on to do interpretive reasoning quite often as we do our everyday common-sense tasks.

One way a case-based classifier works is to ask whether the new concept is enough like another one known to have the target classification. PROTOS (Bareiss, 1989), which diagnoses hearing disorders, works like this.[3] Rather than classifying new cases using necessary and sufficient conditions, PROTOS does classification by trying to find the closest matching case in its case base to the new situation. It classifies the new situation by that case's classification. To do this, PROTOS keeps track of how prototypical each of its cases is and what differentiates cases within one classification from each other. It first chooses a most likely classification, then chooses a most likely matching case in that class. Based on differences between the case it is attempting to match and the new situation, it eventually zeros in on a case that matches its new one well.

When no case matches well enough, it is sometimes necessary to consider hypothetical situations. Much of the work on this type of interpretation comes from the study of legal reasoning. Suppose, for example, that a lawyer must argue that his client was not guilty of violating a nondisclosure agreement because the only one he disclosed to was not technically competent to copy it. There may be several cases that justify this argument. To test it, one might create hypothetical cases that go beyond the real cases in testing boundaries. One might propose a situation where the person disclosed to was not technical but was president of a company with technical personnel. Another hypothetical might be one in which the person disclosed to was not technical, but his wife, who he disclosed to, was. Interpretation based on hypothetical cases helps in fine-tuning an argument for or against a particular interpretation.

This is what HYPO (Rissland 1986, Ashley 1987) does. HYPO is the earliest and most sophisticated of the interpretive case-based reasoners. 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. For example, to counter a counterexample, one might propose variations on a new situation that make it more like the counterexample.

When a concept is being created on the fly, interpretation requires an additional step: derivation of a set of defining features for a category. For example, to decide if Anne will eat swordfish, one must first attempt to characterize what makes a fish Anne will eat acceptable. Otherwise, the basis for comparing swordfish to other fish she will eat cannot be known. 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. It is easy for us to determine what those dimensions are, but getting a computer to do it automatically is a challenge equivalent to the credit assignment problem.

1.2.3. *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 it leads. More often, however, in real-world problems, everything is not known and effects cannot be predicted with accuracy based on any simple set of inference rules.

Consider, for example, a battlefield commander who must derive a strategy for an upcoming battle. Doctrine provides a set of rules for doing battle, and they can be used to create a first approximation of a battle plan. But doctrine gives rules for situations in general, not for particular situations in which the troops are tired, the strategies of the enemy commander are well-known, it's been raining for a week, or the mountains are shaped in a way that allows a trap to be laid. In battlefield planning, as in many other situations, the little details of a situation are important to the worthiness of a plan. There are many details that could be attended to, and only some are important. And, as in other adversarial situations, it is impossible to know everything about the other side, predict all of their strategies, or predict all of their reactions or counterplans. Yet a good plan must be created, and it must be evaluated based on projected effects.

Cases provide a way to projecting effects based on what has been true in the past. Cases with similar plans that were failures can point to potential plan problems. If a previous plan similar to the currently proposed one failed because the troops were too tired, for example, the commander is warned to evaluate whether his troops are too tired, and if so, knows to fix his strategy to take that into account. He might change the plan so that tiredness will not be a factor, give his troops a rest, or get fresh troops in for the battle. Cases with similar plans that were successes give credence to the current plan. In addition, when parts of a plan are targeted for evaluation, cases can help with that. The effects of using a particular kind of trap, for example, can be evaluated by recalling another case where a similar trap was used.

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 successful 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.

Projection is one of the most important bottlenecks facing the planning community today. A real-world planner must be able to project effects of plan steps to interleave tasks with each other, to plan late steps in a plan before earlier ones are executed, and to set up contingencies. Case-based reasoning has much to contribute to solving this problem, but effort so far has gone into helping a person use cases to do projection, and little effort has gone into automating the projection process to date.

1.2.4. *Interpretive case-based reasoning and problem solving*

Much work on interpretation has centered on the law domain and has looked at justifying an argument for or against some interpretation of the law. One should not walk away from this discussion, however, with the impression that case-based interpretation is merely for interpretive problems. On the contrary, it has much usefulness as part of the evaluative or critical component of problem solving and decision making whenever strong causal models are missing. Though there has been little work in the area, the processes involved in interpretative case-based reasoning have the potential to play several important roles for a problem solver. First, if the framework for a solution is known, or if constraints governing it are known, these methods could be used to choose cases that would provide such a solution. Second, argument creation and justification result in knowledge of what features are the important ones to focus on. Knowing where to focus is important in problem solving also. Third, a side effect of HYPO's methods is that it can point out which features, if they were present, would yield a better solution. It does this by keeping track of near-miss dimensions and creation of hypothetical cases. A problem solver could use such information to inform its adaptation processes. Finally, interpretive methods can be used to predict the usefulness, quality, or results of a solution.

2. CBR AND LEARNING

Case-based reasoning is a methodology for both reasoning and learning. A case-based reasoner learns in two ways. First, it can become *a more efficient reasoner* 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. Second, a case-based reason becomes *more competent* over time, deriving better answers than it could with less experience. One of case-based reasoning's fortés is in helping a reasoner to anticipate and thus avoid mistakes it has made in the past. This is possible because it catches 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, and the reasoner can 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 much of its learning in two other ways: through the accumulation of new cases and through the assignment of indexes. New cases give the reasoner more 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 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. And, one can think of the generalization process as a way of evolving useful rules from cases that a rule-based reasoner could use. Rules could be used when they matched cases exactly, while cases would be used when rules were not immediately applicable.

3. PROCESSES AND ISSUES

The traditional view of reasoning in both artificial intelligence and cognitive psychology has been that the reasoner is handed a problem, and by composing and instantiating abstract operators, he is able to solve it. As the examples have

shown, the case-based reasoning paradigm takes a different approach. Rather than viewing reasoning as primarily a composition process, we view it 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 instance.

As we have seen, cases are used in two very different ways during reasoning. They provide ballpark solutions that are adapted to fit a new situation, and they provide concrete evidence for or against some solution that drives a criticism or evaluation procedure. Case-based reasoning is thus a process of 'remember a case and adapt its solution' or 'remember a case and evaluate the new one based on its outcome'.

The two styles of case-based reasoning arise through emphasis on one or the other of the two uses of cases. In problem solving situations, we tend to emphasize the use of cases to propose solutions; in interpretive situations (e.g., law), we tend to emphasize using cases for criticism and justification. As our examples above show, however, the two styles are not completely disjoint. We use interpretive methodologies to criticize and justify a solution to a problem. And one might need to answer questions (solve problems) in order to interpret a situation.

The major processes shared by reasoners that do case-based reasoning are **case retrieval** and **case storage** (also called *memory update*). In order to make sure that poor solutions are not repeated along with the good ones, case-based reasoners must also **evaluate** their solutions.

The two styles of case use, 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 fixing an old solution to fit a new situation, and **criticism**, the process of evaluating the new solution before trying it out. 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 a process of comparing and contrasting the new situation to prior cases, and **criticism**, the process of justifying the argument, done by generating hypothetical situations and trying the argument out on those.

These steps 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. Criticism may lead to additional adaptation, so might evaluation. And when reasoning is not progressing well using one case, the whole process may need to be restarted from the top with a new case chosen. Figure 1 summarizes their relationship.

As we've shown in our examples, case-based reasoning is a natural reasoning process in people, and it has the potential to alleviate many of the complexity problems plaguing the AI endeavor. In order to build systems, however, there are several issues that must be addressed, many of them the same ones we must

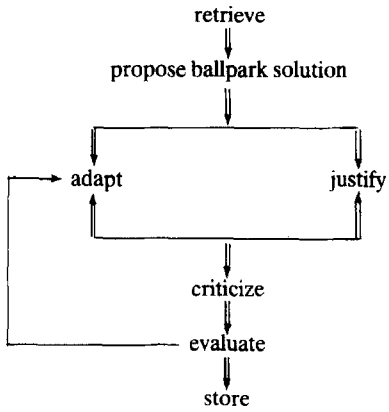

Fig. 1. The case-based reasoning cycle.

address to explain the reasoning people do. In the following sections, we give a short overview of each step and discuss the issues that must be addressed to explain it and make it work well on the computer.

3.1. *Case Retrieval*

Remembering is the process of retrieving a case or set of cases from memory. In general, it consists of two substeps:

- **Recall previous cases**. The goal of this step is to retrieve "good" cases that can support reasoning that comes in the next steps. Good cases are those that have the potential to make relevant predictions about the new case. Retrieval is done by using features of the new case as indexes into the case base. Cases labeled by subsets of those features or by features that can be derived from those features are recalled.
- **Select the best subset**. This step selects the most promising case or cases to reason with from those generated in step 1. The purpose of this step is to winnow down the set of relevant cases to a few most-on-point candidates worthy of intensive consideration. Sometimes it is a appropriate to choose one best case, sometimes a small set is needed.

One problem here is that sometimes two cases need to be judged similar even though they share few surface features. Football and chess strategies, for example, have much in common though the concrete features of the games are dissimilar. One is played on a board, the other on a field, one has pieces, the other people, one has teams competing with each other, the other individuals. What they have in common is more abstract features, shared by competitive games. There are two sides in opposition, each wants to win, each wants the other side to lose, both games involve planning and counterplanning, both involve positions on a playing field, though it is an actual field with people for

football and a board with pieces for chess. Nevertheless, you might expect a football expert who knows how to plan a fork to notice a potential fork situation in a chess game he is playing and to plan a set of moves based on that. One way to deal with this problem is to use more than just the surface representation of a case for comparison. Cases must also be compared at more abstract levels of representation. The issue we must address is which of the abstract ways of representing a case are the right ones to use for comparisons (Birnbaum and Collins 1988, Collins 1987).

Another problem is that a new situation and a case might share some derivable features while not sharing surface features. In predicting who will win a battle, for example, the ratio of defender strength to attacker strength is predictive but neither defender strength nor attacker strength by itself is. Cases need to be judged similar based on the ratio (a derived feature) rather than the individual values (surface features). Similarly in diagnosis. It is often the hypothesized disorders or conditions (derived features) that need to be compared in two cases to judge a current situation and previous case similar rather than the symptoms the patients manifest (surface features). The issue here is to come up with a way of generating derived features for cases in an efficient way. We need guidance in generating derived features because some are expensive to derive, and even if all were cheap, it would be expensive to generate all possible derived features of a case.

And of course we need to derive fast retrieval algorithms for massive libraries of cases (Kolodner 1984, 1988b).

These problems comprise what we have called **the indexing problem**. Broadly, the indexing problem is the problem of retrieving applicable cases at appropriate times (despite all the problems cited above). In general, it has been addressed as a problem of assigning labels, called *indexes*, to cases that designate under what conditions each case can be used to make useful inferences. These labels have been treated much like indexes in a book. Old cases are indexed by those labels, one uses a new situation as a key into that index and traverses appropriate indexing paths to find relevant cases. Researchers are working on specifying what kinds of indexes are most useful, designating vocabularies for indexes, creating algorithms and heuristics for automating index choice, organizing cases based on those indexes, searching memory using those indexes, and choosing the best of the retrieved cases. The tension between using indexes to designate usefulness and direct search while at the same time not allowing them to overly dominate what can be recalled is one of the most important issues in case-based reasoning.

3.2. *Proposing a Ballpark Solution*

In this step, relevant portions of the cases selected during retrieval are extracted to form a ballpark solution to the new case. In problem solving, this step normally involves selecting the solution to the old problem, or some piece of it, as a ballpark solution to the new one. In interpretation, this step involves partitioning the retrieved cases according to the interpretations or solutions they

predict and, based on that, assigning an initial interpretation to the new problem. Alternatively, the initial interpretation might be given (as when a lawyer needs to argue for his/her client). In that situation, there is no need for this step.

For example, a problem solver attempting to plan a meal focuses on the meal plan of a recalled case and uses it as its ballpark solution. Or, if it is attempting to derive some piece of a meal plan, it focuses on the analogous piece in the old case. Thus, when JULIA is trying to design an easy-to-prepare and cheap meal for 20 people, if it remembers a meal where antipasto salad, lasagne, broccoli, and spumoni were served, that solution will be the ballpark solution after this step. When it is trying to choose a main dish, if it is reminded of the same case, lasagne (its main dish) will be its ballpark solution.

An interpretive program, in this step, decides which of the possible interpretations to begin reasoning with. PROTOS, for example, in this step uses a coarse evaluation function to distinguish which of the many possible interpretations proposed by the recalled cases is closest to its new case. HYPO, on the other hand, has its desired result given to it. Thus, its work doesn't actually begin until the next step.

There are several issues that arise in constructing a ballpark solution. First is the question of how appropriate portions of an old case can be selected for focus. An old case could be quite large, and it is important that the parts of it with no relevance to the new situation don't get in the way. On the other hand, it is possible that seemingly unrelated parts of an old case can provide guidelines. The best answer we have to this problem so far is twofold: First, the goals of the reasoner determine where to focus in the old case. The reasoner focuses attention on that part of the old case that was relevant to achieving the reasoner's current goal in the past. Thus, if the reasoner is trying to derive a solution, focus is on the previous solution. If the reasoner is trying to derive a particular part of a solution, focus is on that part of the solution in the previous case. If the reasoner is trying to interpret a situation, its classification in the old case is the focus. Second, the internal structure of the old case, especially the dependencies between different parts of the case, tell the reasoner how to expand focus in relevant ways. Thus, when the reasoner focuses on the solution or classification in an old case, those features of the case that led to selection of that solution or classification are also in focus.

Another issue has to do with how much work to do in this step before passing control to adaptation or justification processes. Often there are relatively easy and automatic, some would say *common-sense*, adaptations that can be made in an old solution before it undergoes the scrutiny of harder adaptation processes. For example, in labor mediation, adjustment of salary and other benefits based on cost of living is an expected adjustment. Such adjustments might also be made to old interpretations before creating arguments for them. Especially in interpretive reasoning, easy adjustments in this step before harder reasoning kicks in may be more advantageous than making adjustments after arguments have been mounted. This way, argumentation is based on a more realistic solution.

A third issue has to do with choice of an interpretation in interpretive

reasoning. In programs that have been written to date, either the interpretation the program starts with is given or it is chosen doing a coarse evaluation of the alternatives. If one can get to the 'right' answer no matter where one starts, then choice of a first alternative is merely an efficiency issue. However, if all alternatives are not connected in some way, initial choice of a first alternative might impact accuracy. Thus, where to begin in doing interpretation is a real issue.

3.3. *Adaptation*

In problem solving CBR, old solutions are used as inspiration for solving new problems. Since new situations rarely match old ones exactly, however, old solutions must be fixed to fit new situations. In this step, called **adaptation**, the ballpark solution is adapted to fit the new situation. There are two major steps involved in adaptation: figuring out what needs to be adapted and doing the adaptation.

Issues in adaptation arise from both steps. We start by considering adaptation itself. For any particular domain or task, we can come up with a set of adaptation strategies or heuristics. We can implement those and create a working system. This is rather *ad hoc*, however. One big question we must address is whether there is a general set of adaptation strategies that we can start with for any domain and that provide guidelines for defining specialized adaptation strategies. Taking the meat out of a recipe to make it vegetarian is a strategy specific to recipe adaptation, for example, but it is a specialization of a more general strategy that we call *delete secondary component*. This strategy states that a secondary component of an item can be deleted if it performs no necessary function. For each type of adaptation strategy, we must also designate the knowledge necessary for its application.

Also important in adaptation is methodologies for noticing inconsistences between old solutions and new needs and, based on that, choosing what should be adapted. Some of the bookkeeping methods developed elsewhere in AI (e.g., reason-maintenance, constraint propagation) are useful here, but are used in much different ways than in their original formulations.

3.4. *Justification and Criticism*

In these steps, a solution or interpretation is justified, often before being tried out in the world. When all knowledge necessary for evaluation is known, one can think of this step as a validation step. However, in many situations, there are too many unknowns to be able to validate a solution. We can criticize solutions using all of the techniques of interpretive case-based reasoning. One way is to *compare and contrast* the proposed solution to other similar solutions. This requires a recursive call to memory processes in order to retrieve cases with similar solutions. For example, if there is an already-known instance of a similar situation failing, the reasoner must consider whether or not the new situation is subject to the same problems. Alternatively, if there is an already-known instance of a similar situation but the problem situations are very

different, the reasoner might consider whether the new situation was solved in a fair way. For example, in contract negotiations, a mediator might come up with a salary proposal and before proposing it look to see what other workers have similar salaries and whether the proposed salary fits current precedents.

We might also propose hypothetical situations to test the robustness of a solution. Yet another way to criticize a solution is to run a simulation (coarse or high fidelity) and check the results.

Criticism may require retrieval of additional cases and may result in the need for additional adaptation, this time called *repair.*

Major issues here include strategies for evaluation using cases, strategies for retrieving cases to use in interpretation, evaluation, and justification; the generation of appropriate hypotheticals and strategies for using them; and the assignment of blame or credit to old cases.

3.5 *Evaluation*

In this step, the results of reasoning are tried out in the real world. Feedback about the real things that happened during or as a result of executing the solution are obtained and analyzed. If results were as expected, further analysis is not necessary in this step, but if they were different than expected, explanation of the anomalous results is necessary. This requires figuring out what caused the anomaly and what could have been done to prevent it. Explanation can sometimes be done by case-based reasoning.

This step is one of the most important for a case-based reasoner. It gives it a way of evaluating its decisions in the real world, allowing it to collect feedback that enables it to learn. Feedback allows it to notice the consequences of its reasoning. This in turn facilitates analysis of its reasoning and explanation of things that didn't go exactly as planned. This analysis, in turn, allows a reasoner both to anticipate and avoid mistakes it has been able to explain sufficiently and to notice previously unforeseen opportunities that it might have a chance to reuse.

Evaluation is the process of judging the goodness of a proposed solution. Sometimes evaluation is done in the context of previous cases, sometimes it is based on feedback from the world, sometimes it is based on a mental or real simulation. Evaluation includes explaining differences (e.g., between what is expected and what actually happens), justifying differences (e.g., between a proposed solution and one used in the past), projecting outcomes, and comparing and ranking of alternative possibilities. The result of evaluation can be additional adaptation, or repair, of the proposed solution.

3.6. *Memory Update*

In this step, the new case is stored appropriately in the case memory for future use. A case is comprised of the problem, its solution, plus any underlying facts and supporting reasoning that the system knows how to make use of, and its outcome. The most important process that happens at this time is choosing the

ways to 'index' the new case in memory. Indexes must be chosen such that the new case can be recalled during later reasoning at times when it can be most helpful. It should not be over-indexed, since we would not want it recalled indiscriminately. This means that the reasoner must be able to anticipate the importance of the case to later reasoning. Memory's indexing structure and organization are also adjusted in this step.

This problem shares all of its issues with the first: we must choose appropriate indexes for the new case using the right vocabulary, and we must at the same time make sure that all other items remain accessible as we add to the case library's store.

4. APPLICABILITY OF CASE-BASED REASONING

4.1 *Range of Applicability and Real-World Usefulness*

We start by considering why a doctor, or anybody else trained in the practice of making logical decisions, would make case-based inferences. After all, the doctor is trained to use facts and knowledge, and case-based reasoning looks like it is based on hearsay. The answer is simple. The doctor is trained to recognize disorders in isolation and to recognize common combinations of disorders. He also knows the etiology of disorders, i.e., how they progress. But he cannot be trained to recognize every combination of disorders, and the knowledge he has of disease processes is time consuming to use for generation of plausible diagnoses. If he has used his knowledge of disease process to solve a hard problem once, it makes sense to cache the solution in such a way that it can be reused. That is, once he has learned to recognize a novel combination of disorders, if he remembers that experience, he will be able to recognize it again, just as he recognizes more common combinations, without the difficult reasoning necessary the first time. The logical medical judgment comes in later in deciding whether or not the patient does indeed have the proposed set of diseases.

Similarly, we can't expect a computer program to be seeded with all the possible combinations of problems it might encounter. Nor can we expect it to have efficient algorithms for generating plausible solutions from scratch all the time. A model-based trouble shooting system, for example, might know very well how something functions. That doesn't necessarily mean that it can generate solutions to problems easily, especially when more then fault could be possible at any time. Similarly, while a causal model may be helpful in verifying a design, it may not provide enough information to be able to generate designs in underconstrained or overconstrained situations. Just as case-based reasoning provides a way for people to generate solutions easily, it also provides a way for a computer program to propose solutions efficiently when previous similar situations have been encountered. This doesn't mean that causal reasoning is without merit. On the contrary, it must play the role that the medical doctor's logic plays after a solution is proposed. The causal-model-based system needs to

work along with the case-based system to identify changes that must be made in an old solution, to ensure valid adaptations, and to verify proposed solutions. Indeed, CASEY (Koton 1988) and KRITIK (Goel and Chandrasekaran 1989) do just that, CASEY for the task of heart failure diagnosis and KRITIK for design of simple mechanical objects.

So case-based reasoning is useful to people and machines that know a lot about a task and domain because it gives them a way of reusing hard reasoning they've done in the past. It is equally useful, however, to those who know little about a task or domain. Consider, for example, a person who has never done any entertaining yet has to plan the meal specified in the introduction. His own entertaining experience won't help him. But if he has been to dinner parties, he has a place to start. If he remembered meals he'd been served under circumstances similar to those he has to deal with, he could use one of those to get started. For example, if he could generate a list of large dinner parties he has attended, he could, for each one, figure out whether it was easy to make and inexpensive, and when he remembered one, adapt it to fit.

Case-based reasoning is also useful when knowledge is incomplete and/or evidence is sparse. Logical systems have trouble dealing with either of these situations because they want to base their answers on what is well-known and sound. More traditional AI systems use certainty factors and other methods of inexact reasoning to counter these problems, all of which require considerable effort on the part of the computer and none of which seem intuitively very plausible. Case-based reasoning provides another method for dealing with incomplete knowledge. A case-based reasoner makes assumptions to fill in incomplete or missing knowledge based on what his experience tells him, and goes on from there. Solutions generated this way won't always be optimal, or even right, but if the reasoner is careful about evaluating proposed answers, the case-based methodology gives him a way to generate answers easily.

While the advantages of case-based reasoning are easily evident when an old solution is fairly close to that needed in the new case, case-based reasoning also can provide advantage even if the old solution is far from what is needed. There are two possibilities. Those features of the remembered case that must be ruled out in the new situation can be added to its description and a new case recalled, or the recalled case can be used as a starting point for coming up with a new solution. When there is considerable interaction between the parts of a solution, then even if large amounts of adaptation are required to derive an acceptable solution, it may still be easier than generating a solution from scratch. And the case provides something concrete to base reasoning on. For many people, this is a preferred reasoning style.

4.2. *Advantages of CBR*

Case-based reasoning provides many advantages for a reasoner.

- Case-based reasoning allows the reasoner to propose solutions to problems quickly, avoiding the time necessary to derive those answers from scratch.

The doctor remembering an old diagnosis or treatment experiences this benefit.

While the case-based reasoner has to evaluate proposed solutions like any reasoner does, it gets a head start on solving problems because it can generate proposals easily. There is considerable advantage in not having to redo time-consuming computations and inferences. This advantage is helpful for almost all reasoning tasks, including problem solving, planning, explanation, and diagnosis. Indeed, evaluation of CASEY (Koton 1988) shows two orders of magnitude speedup when a problem had been seen in the past.

- Case-based reasoning allows a reasoner to propose solutions in domains that he/she/it doesn't understand completely.

Many domains are impossible to understand completely, often because much depends on unpredictable human behavior, e.g., the economy. Others nobody understands yet, e.g., how some medications and diseases operate. Other times, we simply find ourselves in situations that we don't understand well, but in which we must act anyway, e.g., choosing which graduate students to accept into a program. Case-based reasoning allows us to make assumptions and predictions based on what worked in the past without having a complete understanding.

- Case-based reasoning gives a reasoner a means of evaluating solutions when no algorithmic method is available for evaluation.

Using cases to aid in evaluation is particularly helpful when there are many unknowns, making any other kind of evaluation impossible or hard. Instead, solutions are evaluated in the context of previous similar situations. Again, the reasoner does his/her evaluation based on what worked in the past.

- Cases are particularly useful for use in interpreting open-ended and ill-defined concepts.

As stated above, this is one use attorneys put cases to extensively. Bur it is also important in everyday situations. In the example above, cases were used to determine what was included in the concept 'fish Anne won't eat'. The performance of PROTOS (Bareiss 1989) in classifying hearing disorders when little information is known shows that a case-based methodology for interpretation can be more accurate that a generalization-based method when classifications are ill-defined. PROTOS is significantly more capable and accurate than classification systems based on more traditional classification methods.

- Remembering previous experiences is particularly useful in warning of the potential for problems that have occurred in the past, alerting a reasoner to take actions to avoid repeating past mistakes.

How can this work? Remembered experiences can be successful or failure episodes, i.e., situations in which things did not turn out exactly as planned. Consider again the reasoner trying to plan a meal. He/she can be helped considerably, for example, if he/she remembers a meal that was supposed to be easy-to-prepare and inexpensive and instead was hard to make because some of the ingredients were hard to obtain in manufactured form and had to be made from scratch. The reasoner is warned, by this case, to avoid those ingredients or to make sure they are available before committing to a menu.

- Cases help a reasoner to focus its reasoning on important parts of a problem by pointing out what features of a problem are the important ones.

What was important in previous situations will tend to be important in new ones. Thus, if in a previous case, some set of features was implicated in a failure, the reasoner focuses on those features to insure that the failure will not be repeated. Similarly, if some features are implicated in a success, the reasoner knows to focus on those features. Such focus plays a role in both problem solving and interpretive case-based reasoning. In interpretive case-based reasoning, justifications and critiques are built based on those features that have proven responsible for failures and successes in the past. An attorney, for example focuses on those aspects of a new situation that mattered in previous cases. In problem solving, a reasoner might attempt to adapt his solution so that it includes more of what was responsible for previous successes and less of what was responsible for failures.

4.3. *Pitfalls*

Of course, there are also pitfalls in using cases to reason. A case-based reasoner might be tempted to use old cases blindly, relying on previous experience without validating it in the new situation. A case-based reasoner might allow cases to bias him/her/it too much in solving a new problem. And, often people, especially novices, are not reminded of the most appropriate sets of cases when they are reasoning (Holyoak 1985, Gentner 1989). People do find case-based reasoning a natural way to reason, however. The endeavor of explaining the processes involved in case-based reasoning might help us to learn how to teach people to reason better using cases. In addition, the case memory technology we develop might allow us to build decision aiding systems that augment human memory by providing the appropriate cases while still allowing the human to reason in a natural and familiar way. And we can make sure our programs avoid this type of behavior.

5. COGNITIVE MODEL OR METHODOLOGY FOR BUILDING EXPERT SYSTEMS?

In case-based reasoning a model of people, or is it a methodology for building intelligent systems? We've been somewhat schizophrenic about this issue up to now, sometimes referring to people doing case-based reasoning, sometimes referring to machines that use the method to reason. Case-based reasoning is both, and explorations in case-based reasoning have been of both the ways people use cases to solve problems and the ways we can make machines use them.

5.1. *Case-Based Reasoning and People*

There is much evidence that people do, in fact, use case-based reasoning in their daily reasoning. Some of that evidence is hearsay — we have observed it. Other evidence is experimental. Ross (Ross 1986, Ross 1989), for example, has

shown that people learning a new skill often refer back to previous problems to refresh their memories on how to do the task. Research conducted in our lab shows that both novice and experienced car mechanics use their own experiences and those of others to help them generate hypotheses about what is wrong with a car, recognize problems (e.g., a testing instrument not working), and remember how to test for different diagnoses (Lancaster and Kolodner 1988, Redmond 1989). Other research in our lab shows that physicians use previous cases extensively to generate hypotheses about what is wrong with a patient, to help them interpret test results, and to select therapies when several are available and none are understood very well (Kolodner, unpublished). We have also observed architects and caterers recalling, merging, and adapting old design plans to create new ones.

The programs we build are an attempt to understand the processes involved in reasoning in a case-based way. There are several important potential applications of an understanding of the way people solve problems in a natural way. First, we might build decision aiding systems for people that can help them retrieve cases better. Psychologists have found that people are comfortable using cases to make decisions but don't always remember the right ones. The computer could be used as a retrieval tool to augment people's memories. Second, we might 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 ones. Third, if we understand which parts of this natural process are difficult to do well, we can teach people better how to do 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.

5.2. *Building a Case-Based Reasoner*

As a method for building intelligent reasoning systems, case-based reasoning has appeal because it seems relatively simple and natural. While it is hard to get experts to tell you all the knowledge they use to solve problems, it is easy to get them to recount their war stories. In fact, several people building expert systems that know how to reason using cases have found it easier to build case-based expert systems than traditional ones (Barletta and Hennessy 1989, Goodman 1989). A big problem in reasoning in expert domains is the high degree of uncertainty and incompleteness of knowledge involved. Case-based reasoning addresses those problems by having the reasoner rely on what has worked in the past. Case-based systems also provide efficiency. While we find first-princi-

ples problem solving systems spending large amounts of time solving their problems from scratch, case-based systems have been found to be several orders of magnitude faster (Koton 1988).

There are several different kinds of case-based reasoning systems one might build. At the two extremes are fully-automated systems and retrieval-only systems. Fully automated systems are those that solve problems completely by themselves and have some means of interacting with the world to receive feedback on their decisions. Retrieval-only systems work interactively with a person to solve a problem. They act to augment a person's memory, providing cases for the person to consider that the person might not be aware of himself, but the person will be responsible for hard decisions. Then there is the whole range of systems in between, some requiring more on the part of the person using the system, some requiring less.

There are also several purposes one might create a case-based reasoning system to serve. We might want it to solve problems, to suggest concrete answers to problems, to be suggestive without providing answers (i.e., to give abstract advice), or to just act as a database that can retrieve partially-matching cases. Much as has been the case with database systems, we can foresee case-based systems interacting with a person or another program. Interacting with a person, we can see an executive doing strategic planning asking for cases to help in deriving or evaluating a solution. The CSI Battle Planner (Goodman 1989) provides this type of capability now for battle planning. Or, we can imagine a tutoring system that accesses a library of examples to use in teaching.

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.

NOTES

* This article is excerpted from *Case-Based Reasoning* by Janet Kolodner, to be published by Morgan-Kaufmann Publishers, Inc. in 1992.

** This work was partially funded by DARPA under Contract No. F49620-88-C-0058 monitored by AFOSR, by NSF under Grant No. IST-8608362, and by ARI under Contract No. MDA-903-86-C-173. Address correspondence to the author at the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, email: jlk@cc.gatech.edu.

1 Thanks to Kevin Ashley for the example.

2 Example due to Kevin Ashley (Ashley, 1989).

3 In a previous section, we presented PROTO as a problem solving system. Here we present it as an interpretive system. PROTOS' task, diagnosis, includes aspects of both uses of cases. It interprets open-ended concepts (classifications) using a case-based classification algorithm, and it uses an indexing scheme based on differences to avoid previously-made diagnostic mistakes.

REFERENCES

Alterman, R. (1988). Adaptive planning. *Cognitive Science* **12**, 393—422.

Ashley, K. D. (1987). Distinguishing — a reasoner's wedge. *Proceedings of the 1987 Conference of the Cognitive Science Society.* Lawrence Erlbaum Assoc., Hillsdale, NJ, pp. 737—747.

Ashley, K. D. (1988). *Modelling Legal Argument: Reasoning with Cases and Hypotheticals* Ph.D. Dissertation, COINS Technical Report No. 88—01, Department of Computer and Information Science, University of Massachusetts, Amherst.

Bain, W. (1986). *Case-Based Reasoning: A Computer Model of Subjective Assessment.* Ph.D. Thesis, Dept of Computer Science, Yale University, New Haven, CT.

Bareiss, E. R. (1989). *Exemplar-Based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning.* Academic Press, Boston, MA.

Barletta, R. and Hennessy, D. (1989). Case adaptation in autoclave layout design. In Hammond, K. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA), II* Morgan-Kaufmann, Pensacola Beach, FL.

Birnbaum, Lawrence and Collins, Gregg (1988). The transfer of experience across planning domains through the acquisition of abstract strategies. In Kolodner, J. L. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA).* Morgan-Kaufmann Publishers, Inc., San Mateo, CA.

Branting, L. K. (1989). Integrating generalizations with exemplar-based reasoning. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society,* Ann Arbor.

Charniak, E. and McDermott, D. (1985). *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, MA.

Collins, G. (1987). *Plan Creation: Using Strategies as Blueprints.* Ph. D. Thesis. Department of Computer Science, Yale University, New Haven, CT.

Gentner, D. (1989). Finding the needle: Accessing and reasoning from prior cases. In Hammond, K. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA), II,* Morgan-Kaufmann, Pensacola Beach, FL.

Goel, A. (1989). *Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving.* Ph.D. Thesis. Department of Computer and Information Science. The Ohio State University.

Goel, A. and Chandrasekaran, B. (1989). Use of device models in adaptation of design cases. In Hammond, K. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA) II,* Morgan-Kaufmann, Pensacola Beach, FL.

Goodman, M. (1989). CBR in battle planning. In Hammond, K. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA), II,* Morgan-Kaufmann, Pensacola Beach, FL.

Hammond, K. J. (1989a). *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press, Boston, MA.

Hammond, K. (1989b). Opportunistic memory. *Proceedings of IJCAI-89,* Detroit.

Hammond, K. (1989c). *Proceedings: Second Case-Based Reasoning Workshop (DARPA), II* Morgan Kaufmann Publishers, Inc., San Mateo, CA.

Hinrichs, Thomas R., (1988). Towards an architecture for open world problem solving. In Kolodner, J. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA).* Morgan-Kaufmann Publishers, Inc., San Mateo, CA.

Hinrichs, T. R. (1989). Strategies for adaptation and recovery in a design problem solver. In Hammond, K. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA), II,* Morgan-Kaufmann, Pensacola Beach, FL.

Holyoak, K. J. (1985). The pragmatics of analogical transfer. In Bower, G. (Ed.), *The Psychology of Learning and Motivation* Academic Press, New York, NY.

Kass, Alex M. and Leake, David B. (1988). Case-based reasoning applied to constructing explanations. In Kolodner, J. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA).* Morgan-Kaufmann Publishers, Inc., San Mateo, CA.

Kolodner, J. L. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model.* Lawrence Erlbaum Associates, Inc., Hillsdale, NJ.

Kolodner, J. L. (1987). Capitalizing on failure through case-based inference. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society.*

Kolodner, J. L. (1988a). *Proceedings: Case-Based Reasoning Workshop (DARPA).* Morgan Kaufmann Publishers, Inc., San Mateo, CA.

Kolodner, Janet L. (1988b). Retrieving events from a case memory: a parallel implementation. In Kolodner, J. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA).* Morgan-Kaufmann Publishers, Inc., San Mateo, CA.

Kolodner, J. L. and Kolodner, R. M. (1987). Using experience in clinical problem solving. *IEEE Transactions on Systems, Man, and Cybernetics* **17**(3), 420—431.

Kolodner, J. L. and Simpson, R. L. (1989). The MEDIATOR: analysis of an early case-based problem solver. *Cognitive Science* **13**(4), 507—549.

Koton, P. (1988). *Using Experience in Learning and Problem Solving.* Ph.D. Thesis. Computer Science. MIT.

Lancaster, J. S. and Kolodner, J. L. (1988). Varieties of learning from problem solving experience. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.*

Marks, Mitchell, Hammond, Kristian A., and Converse, Tim, (1988). Planning in an open world: a pluralistic approach. In Kolodner, J. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA).* Morgan-Kaufmann Publishers, Inc., San Mateo, CA.

Navinchandra, D. (1988). Case-based reasoning in CYCLOPS, a design problem solver. In Kolodner, J. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA).* Morgan-Kaufmann Publishers, Inc., San Mateo, CA.

Redmond, M. (1989). Combining explanation types for learning by understanding instructional examples. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society,* Ann Arbor.

Riesbeck, C. K. and Schank, R. S. (1989). *Inside Case-Based Reasoning.* Lawrence Erlbaum Assoc., Inc., Hillsdale, NJ.

Rissland, E. L. (1983). Examples in legal reasoning: legal hypotheticals. *Proceedings of IJCAI-83,* Karlsruhe, West Germany.

Rissland, E. L. (1986). Learning how to argue: using hypotheticals. In Kolodner, J. L. and Riesbeck, C. K. (Eds.), *Experience, Memory, and Reasoning.* Lawrence Erlbaum Ass., Inc., Hillsdale, NJ.

Rissland, E. and Ashley, K. (1987). HYPO: A case-based reasoning system. *Proceedings of IJCAI-87,* Milan, Italy.

Ross, B. H. (1986). Remindings in learning: objects and tools. In Vosniadou, S. and Ortony, A. (Eds.), *Similarity and Analogical Reasoning,* Cambridge University Press, New York, NY.

Ross, B. H. (1989). some psychological results on case-based reasoning. In Hammond, K. (Ed.), *Proceedings: Case-Based Reasoning Workshop (DARPA, II),* Morgan-Kaufmann, Pensacola Beach, FL.

Schank, R. (1986). *Explanation Patterns.* Lawrence Erlbaum Associates, Inc., Hillsdale, NJ.

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

Sycara, E. P. (1987). *Resolving Adversarial Conflicts: An Approach to Integrating Case-Based and Analytic Methods.* Ph.D. Thesis. Technical Report No. GIT-ICS-87/26, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA.

Turner, R. M. (1989). *A Schema-Based Model of Adaptive Problem Solving.* Ph.D. Thesis. Technical Report No. GIT-ICS-89/42. School of Information and Computer Science. Georgia Institute of Technology, Atlanta, GA.