**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

**BEFORE THE PATENT TRIAL AND APPEAL BOARD**

---

In re U.S. Patent No. 5,944,839

Filed: March 19, 1997

Issued: August 31, 1999

Inventor: Henri J. Isenberg

Assignee: Clouding IP, LLC

Title: System and Method for Automatically Maintaining a Computer System

---

Mail Stop PATENT BOARD, PTAB
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

# DECLARATION OF PROFESSOR TODD C. MOWRY, PH.D.

I, Prof. Todd C. Mowry, Ph.D., declare as follows:

## I. Background and Qualifications

(1) My name is Todd Mowry. I am a Professor at Carnegie Mellon University in the Computer Science Department. I have studied and practiced in

the field of computer science for almost 20 years, and have been a professor of computer science since 1993.

(2) I received my Doctor of Philosophy (Ph.D.) degree in the field of Electrical Engineering from Stanford University in 1994. I received my Masters of Science (M.S.) degree in Electrical Engineering from Stanford University and my Bachelor of Science (B.S.) degree in Electrical Engineering from the University of Virginia.

(3) Upon receiving my Ph.D. degree, I joined the faculty of the University of Toronto in the Department of Electrical and Computer Engineering and the Department of Computer Science as an Assistant Professor. I relocated and was promoted to the rank of Associate Professor (initially without tenure) at Carnegie Mellon University in 1997, was promoted to tenured Associate Professor in 2002, and I was promoted to the rank of full Professor in 2008. I was the Associate Department Head for Faculty of the Computer Science Department from 2009-2010.

(4) Since becoming a faculty member in 1993, I supervised the research of 11 Ph.D. dissertations in the field of computer science, and along with my graduate students, published over 60 technical publications in scientific journals or conferences in the field of computer science and 20 technical reports published at

Carnegie Mellon University and the University of Toronto. In addition to Ph.D. dissertations, I have also supervised several graduate student's Master's theses.

(5) As part of my research, I focus on monitoring computer systems as they run to diagnose bugs and other functionality problems. In fact, I have published a number of papers on this topic.

(6) I am a member of several professional organizations including the Institute of Electrical and Electronics Engineers (IEEE) and the Association of Computing Machinery (ACM). I am an Associate Editor of ACM Transactions on Computer Systems (TOCS). I received a Sloan Research Fellowship and the TR35 Award from MIT's Technology Review.

(7) I have served as a consultant for Intel Corporation, Silicon Graphics, Inc., SandCraft, Inc., and IBM.

(8) A copy of my latest *curriculum vitae*, which describes in further detail my qualifications, responsibilities, employment history, honors, awards, professional associations, distinguished lectures, and publications is attached to this declaration as Appendix A-1.

(9) I have reviewed U.S. Patent No. 5,944,839 ("the '839 patent," Ex. 1001) to Henri J. Isenberg. I have also reviewed the publications cited in the footnotes of this declaration and referenced in the *inter partes* review petition submitted herewith.

## II. The Person of Ordinary Skill in the Relevant Field in the Relevant Timeframe

(10) I have been informed that "a person of ordinary skill in the relevant field" is a hypothetical person to whom an expert in the relevant field could assign a routine task with reasonable confidence that the task would be successfully carried out. I have been informed that the level of skill in the art is evidenced by prior art references. The prior art discussed herein demonstrates that a person of ordinary skill in the art, at the time the '839 patent was filed, was aware of several different computer maintenance tools that used a set of sensors in combination with a case base type knowledge database to diagnose and solve computer problems.

(11) Based on my experience, I have an understanding of the capabilities of a person of ordinary skill in the relevant field. I have supervised and directed many such persons over the course of my career.

## III. State of the Art as of 1997

(12) Case-based reasoning (CBR) is an AI problem-solving technique that originated with Roger Schank and his Ph.D. students in the mid-1980s at Yale University. According to Schank, "A case-based reasoner solves new problems by

adapting solutions that were used to solve old problems."[1] The key word in this quote is "adapting." In contrast with *rule-based* reasoning, which performs a scripted action for a rule whenever the specific conditional test for that rule is satisfied, the motivation behind case-based reasoning is to take a more flexible and adaptive approach to problem-solving that draws upon *analogies* to earlier solutions of related (but somewhat different) problems. The proponents of case-based reasoning argue that drawing upon analogies to solve problems corresponds well to how humans solve problems, i.e., by recalling situations that remind them of their current problem, and by attempting to adapt the previous solution to the current circumstances.

(13) Rather than storing a set of "rules," a case-based reasoning system stores a set of "cases" to capture its previous experiences. Cases are descriptions of diagnostic situations that typically include a set of symptoms, a description of the failure and its cause, and also a repair strategy for fixing the problem.

(14) In addition to the use of analogies, another key feature of case-based reasoning is that it *adapts* its set of cases over time based upon the success or failure of its attempts to solve problems. For example, if an existing case can be successfully adapted to solve a new problem, the description of that case may be

[1] Riesbeck, C. K., et al. Inside Case-Based Reasoning. Hillsdale, NJ: L. Erlbaum Assoc. Inc., 1989.

further generalized. In contrast, if adapting existing cases fails to solve the problem, then further specialization may be required, including possibly creating a new case. This learning process where the set of cases are adapted over time is fundamental to case-based reasoning.

(15) By 1997, case-based reasoning had become a mature research area that was being actively explored by dozens of research groups around the world. In fact, there was so much published work on CBR by the early 1990's that multiple survey articles were published on this topic: Kolodner[2] and Aamodt.[3] The latter of these two articles cites 75 papers on CBR. The Aamodt survey paper also describes "The CBR Cycle" in Section 3.3 and Figure 1, which is the basic structure of nearly all CBR systems, and which has been cited numerous times.

[2] Kolodner, J. L. "An introduction to case-based reasoning." *Artificial Intelligence Review*, 6(1):3–34, March 1992.
[3] Aamodt, A., et al. "Case-based reasoning: Foundational issues, methodological variations, and system approaches." *AI Communications*, 7(1):39-59, March 1994.

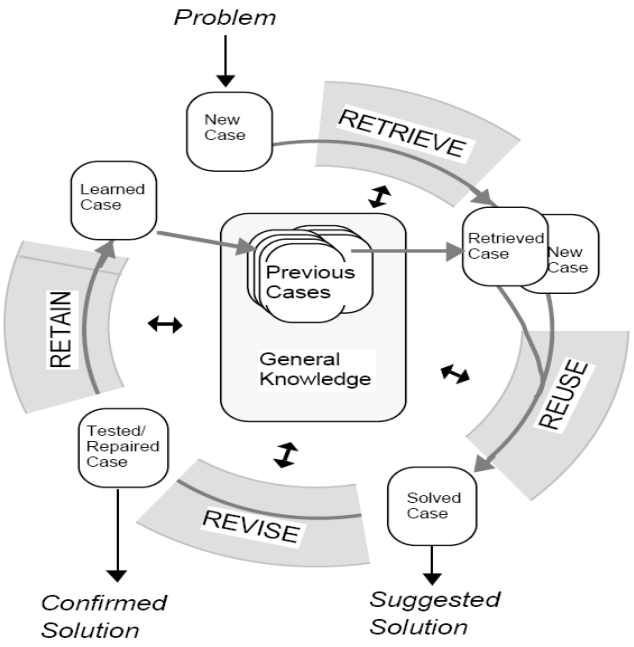



Fig. 1. The CBR Cycle

(16) Conferences and workshops were also being created that were devoted entirely to CBR: the 1st European Workshop on CBR (EWCBR)[4] began in 1993, and the 1st International Conference on CBR (ICCBR)[5] began in 1995.

## IV. The '839 Patent

(17) The '839 patent is generally directed to a tool for performing automatic maintenance, i.e., trouble-shooting, of a computer system. In order to automatically trouble-shoot the computer system, the maintenance tool collects information about the behavior of the computer system it is monitoring using

[4] Wess, S., et al. Topics in case-based reasoning: First European Workshop, EWCBR-93, Kaiserslautern, Germany, November 1-5, 1993, Selected Papers. London, UK: Springer, 1994.

[5] Veloso, M. M., et al. Case-Based Reasoning Research and Development: First International Conference, ICCBR-95, Sesimbra, Portugal, October 23-26, 1995, Proceedings. London, UK: Springer, 1995.

software routines called "sensors," and then diagnoses the problem and recommends a likely solution through the combination of an artificial intelligence ("AI") engine and a "knowledge database." *(Ex. 1001 at 1:57-59)*.

(18) The features of the trouble-shooting tool described in the '839 patent correspond in a straightforward way to conventional case-based reasoning systems at that time. In particular, after one of the sensors in the '839 patent detects a problem, it activates the AI engine. (*Ex. 1001 at 1:61-64*). The AI engine then attempts to draw analogies between the current problem and the set of cases that it has stored in its knowledge database to see whether a previous case suggests a likely solution to the current problem related to trouble-shooting the computer system. (*Id. at 1:65 – 2:1*).

(19) In the course of attempting to diagnose the problem, the AI engine may determine, based upon information stored in matching cases, that additional information is required to accurately diagnose the problem, in which case it may activate additional sensors to collect more information about the monitored system. (*Ex. 1001 at 2:5-8*). If a likely solution to the problem is determined, then the AI engine will attempt to perform the repair. (*Id. at 2:9-11*).

## V. Claim Interpretation

(20) In proceedings before the USPTO, I understand that the claims of an unexpired patent are to be given their broadest reasonable construction in view of

the specification from the perspective of one skilled in the art. I have been informed that the '839 patent has not expired. In comparing the claims of the '839 patent to the known prior art, I have carefully considered the '839 patent and the '839 file history based upon my experience and knowledge in the relevant field. In my opinion, the claim terms of the '839 patent are used in their ordinary and customary sense as one skilled in the relevant field would understand them except for those terms specifically addressed in the following paragraphs.

**"Sensors"**

(21) The '839 patent describes the sensors as follows: "[t]he sensors 112 are software programs that gather information from the computer system 300. (*See, e.g., Ex. 1001 at 3:16-17*)."

(22) Accordingly, under the broadest reasonable construction, the term "sensors" should be interpreted as including different aspects of the same software program or different components of the same application.

**"Artificial Intelligence (AI) Engine"**

(23) The term "AI engine" should be interpreted as including, under the broadest reasonable construction, a different aspect of the same software program as the "sensors" discussed above in paragraph 22. Both the sensors and the AI engine are different components of the same software program or different components of the same application.

(24) Although the prior art references discussed below, such as Gurer, Allen '218, and Barnett, may describe the "sensors" as being separate components from the "AI engine," both the sensors and the AI engine are pieces of software, and the real distinction between them is functionality. The sensors interact with the system that they are monitoring to collect information, and the AI engine reasons about the likely diagnosis of the problem and the likely solution. Hence, under the broadest reasonable construction, it would be obvious to a skilled artisan that the software that performs the sensor functionality and the software that performs the AI engine functionality might be integrated within the same larger software application such that the distinction in functionality between the sensors and the AI engine is no longer existent.

## VI. Discussion of Relevant Patents and Articles

### GURER

(25) I have reviewed the relevant patents and articles listed below, and based on this review, contributed to developing the claim charts comparing each element of claims 1, 2, 6, 8, 14, 15 and 17 of the '839 patent to the appropriate disclosures from the various prior art references in the petition from the perspective of one skilled in the art.

(26) *An Artificial Intelligence Approach to Network Fault Management* published by Gurer et al. ("Gurer," Ex. 1003) describes a system that uses case-

based reasoning to automatically diagnose and correct faults in a computer network that it is monitoring. The raw input to the system is a set of "alarms" which are produced by either the element manager software on a particular network element (e.g., an ATM switch) when it notices a hard error (e.g., a link is down), or through software that performs statistical analysis of the network when it notices a statistical error (e.g., performance degradation due to congestion). (*Id. at 1:30-34*). Since a given network problem might trigger a large number of alarms, these raw alarms are first passed through either a Neural Network or a Bayesian Belief Network to filter and correlate the alarms before they are passed into the case-based reasoning system, thereby reducing the amount of noise in the input before fault identification begins. (*Id. at 2:1-4*). Figure 1 of Gurer illustrates the fault management process described above. (*Id. at 2*).

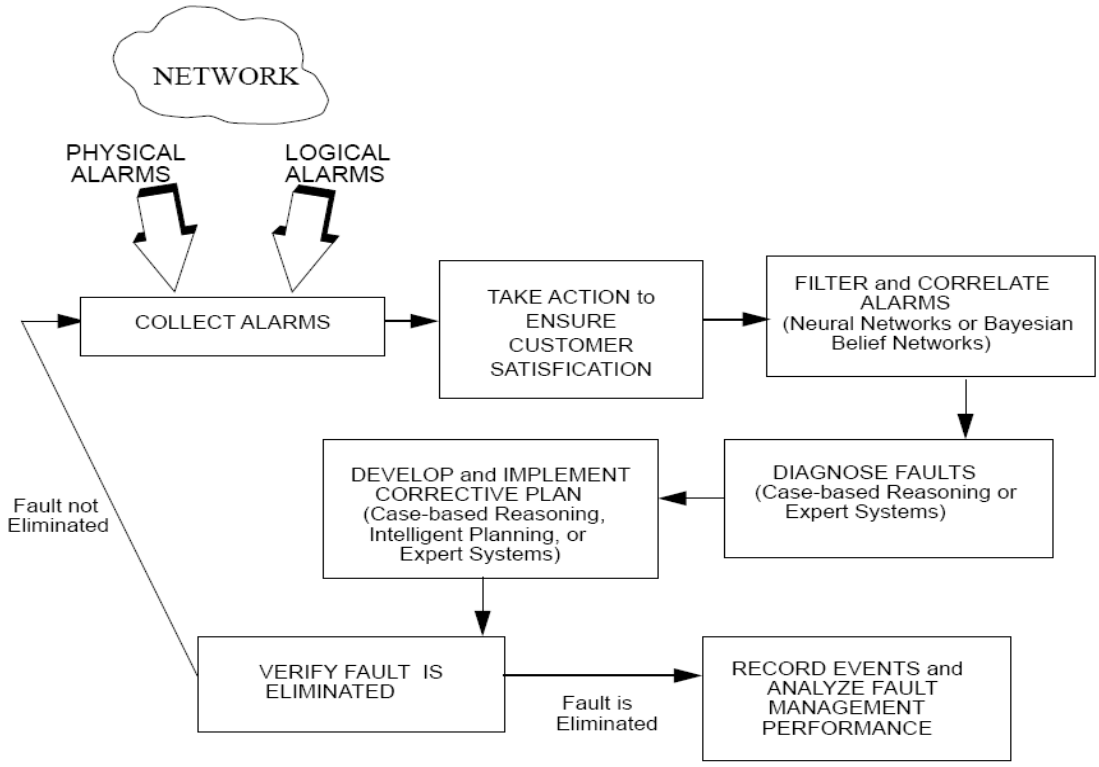



**Figure 1:** The fault management process and possible AI technologies.

(27) As illustrated in Figure 3 below, Gurer further describes how the case-based reasoning system uses its library of cases, i.e., knowledge database, to determine a likely solution to the problem, which potentially involves deciding that the sensors should collect more information regarding the state of the network, and how the gathered information is stored in the knowledge database in the form of new cases. (*Ex. 1003 at 7:1-10*).

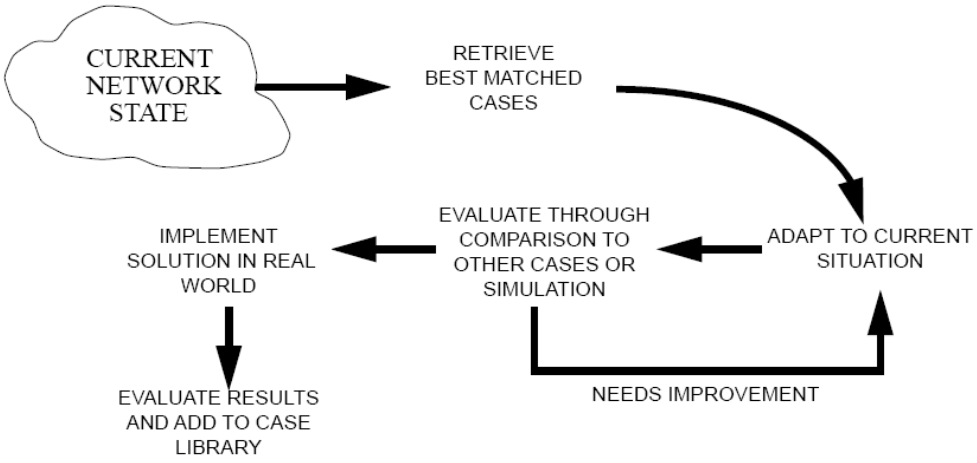



**Figure 3:** The case-based reasoning process.

(28) Gurer discloses that "[a]utomation of network management activities can benefit from the use of artificial intelligence (AI) technologies, including fault management, performance analysis, and traffic management. Here we focus on fault management, where the goal is to proactively diagnose the cause of abnormal network behavior and to propose, and if possible, take corrective actions." (*See Ex. 1003 at 1:11-14*).

(29) In the words of Gurer, "Today's high speed, heterogeneous networks represent a complex and data intensive environment . . ." (*Ex. 1003 at 1:10*). The network elements in those types of sophisticated networks included processors and memory for handling the complex and data-intensive nature of the network. Processors were needed to execute software to manage the network elements, and memory was needed both to execute the software and to store the network packets. In addition, the case-based reasoning system that is described by Gurer would

necessarily be implemented as a large and sophisticated piece of software. In order to execute the software, a processor is necessary in order to perform the instructions in the software. In addition, both the software instructions and any data values that are accessed by the software must be stored in memory. Hence, consistent with the preamble of claim 1 of the '839 patent reciting "[a] tool for automatically maintaining a computer system having a processor and a memory," it would be clear to a person of ordinary skill in the art that the system described by Gurer requires a computer system containing both a processor and a memory.

(30) Further, Gurer discloses a knowledge database ("case library") holding a plurality of cases describing potential computer problems and corresponding likely solutions. "Case-based reasoning is based on the premise that situations recur with regularity. Studies of experts and their problem solving techniques have found that experts rely quite strongly on applying their previous experiences to the current problem at hand. CBR can be thought of as such an expert that applies previous experiences stored as cases in a case library. Thus, the problem-solving process becomes one of recalling old experiences and interpreting the new situation in terms of those old experiences." (*Ex. 1003 at 6:35-39*).

(31) The structure of case-based reasoning (CBR) systems was well-known at the time of the Gurer invention. As further described by Gurer, case-based reasoning is "[a] symbolic AI technology" that operates on a case library. (*Ex.*

*1003 at 6:24*). Hence, consistent with claim 1 of the '839 patent which recites, in part, "a knowledge database stored in the memory and holding a plurality of cases describing potential computer problems and corresponding likely solutions," it would be clear to a person of ordinary skill in the art that the software necessary to implement this symbolic AI technology, i.e., case-based reasoning, must be stored in the memory of a computer system, and also its knowledge database, i.e., the case library, must also be stored in the memory of the computer system.

(32) Gurer also discloses a plurality of sensors, or alarms, adapted for gathering data about the computer system, storing the data in the knowledge database, and detecting whether a computer problem exists from the data and the plurality of cases. "The first step in fault management is to collect *monitoring and performance alarms*. Typically alarms are produced by either managed network elements (e.g., ATM switches, customer premise equipment) or by a statistical analysis of the network that monitors trends and threshold crossings. Alarms can be classified into two categories, physical and logical, where physical alarms are hard errors (e.g., a link is down), typically reported through an element manager, and logical alarms are statistical errors (e.g., performance degradation due to congestion)." (*Ex. 1003 at 1:30-34*).

(33) In addition, "[a]larm filtering is a process that analyzes the multitude of alarms received and eliminates the redundant alarms (e.g., multiple occurrences

of the same alarm). Alarm correlation is the interpretation of multiple alarms such that new conceptual meanings can be assigned to the alarms, creating derived alarms. Faults are identified by analyzing the filtered and correlated alarms and by requesting tests and status updates from the element managers, which provide additional information for diagnosis." (*Ex. 1003 at 2:1-5*).

(34) "The more complex processes of fault management include alarm filtering and correlation, fault identification, and correction. Many of these functions involve analysis, correlation, pattern recognition, clustering or categorization, problem solving, planning, and interpreting data from a knowledge base that contains descriptions of network elements and topology." (*Ex. 1003 at 3:2-4*).

(35) Figure 1 of Gurer illustrated above in paragraph 26 clearly refers to multiple sensors, including those that collect both "physical alarms" and "logical alarms." (*Ex. 1003 at 2*). These alarms are gathered by pieces of software that are either (i) the "element manager" for a network element which reports "physical alarms" or (ii) the statistical analysis software that detects "logical alarms." (*Id. at 1:30-34*). Hence, consistent with claim 1 of the '839 patent which recites, in part, "a plurality of sensors stored in the memory and executing on the processor and adapted for gathering data about the computer system, storing the data in the knowledge database, and detecting whether a computer problem exists from the

data and the plurality of cases," it would be clear to a person of ordinary skill in the art that the instructions for executing these sensors would be stored in a memory and would execute on a processor, since this is how software is executed. It is also clear that the collected alarms must be stored somewhere in the memory, so that they can be passed into the filtering and correlating mechanism and on to the case-based reasoning system. Moreover, Gurer also talks about storing the results of tests, i.e., when the CBR system decides that additional information is needed from a sensor in order to diagnose the problem, in the knowledge database in the form of a new case: "[t]he value of the tests (i.e., useful, not useful), the steps taken, any circuitous paths or dead-ends taken, and the success of the analysis should be stored into a new case. This information, in addition to contextual information, comprise a new case which is then indexed into the case library." (*Id. at 8:6-8*).

(36) Gurer also discloses an AI system executing in response to detection of a computer problem. Gurer discloses "[a]utomation of network management activities can benefit from the use of artificial intelligence (AI) technologies, including fault management, performance analysis, and traffic management. Here we focus on fault management, where the goal is to proactively diagnose the cause of abnormal network behavior and to propose, and if possible, take corrective actions . . . AI technologies may be used to automate the fault management

process, in particular neural networks (NNs) and case-based reasoning (CBR)." (*Ex. 1003 at 1:11-16*).

(37) Further, Gurer discloses that "[a]nother area of fault management where AI technologies can have a positive impact, is fault correction. CBR systems, ESs [Expert Systems], or intelligent planning systems can develop plans or courses of action that will correct a fault that has been identified and verified." (*Ex. 1003 at 3:27-29*).

(38) As discussed above in paragraph 31, Gurer discloses that case-based reasoning (CBR) is "[a] symbolic AI technology." (*Ex. 1003 at 6:24*). At the time the invention was made, it was well known that symbolic artificial intelligence (AI) technology and case-based reasoning, in particular, was implemented through software that executed on a computer system, where the computer system would have at least a memory and a processor. Hence, consistent with claim 1 of the '839 patent which recites, in part, "an AI engine stored in the memory and executing on the processor in response to detection of a computer problem and [the AI engine] utilizing the plurality of cases to determine a likely solution to the detected computer problem," it would be clear to a person of ordinary skill in the art that the AI engine would necessarily be stored in a memory of a computer system.

(39) Gurer also discloses that "CBR problem solving can be depicted as a five-step process, as shown in Figure 3[, illustrated above in paragraph 27]:

(1) retrieval, (2) interpretation and adaptation, (3) evaluation and repair, (4) implementation, and (5) evaluation and learning. The first step is retrieving cases that best match the current situation or case. Thus it is crucial to use an appropriate indexing method, such as decision trees or nearest neighbor matching. Once a case is retrieved, it must be interpreted and then adapted. The interpretation process is a simple comparison between the retrieved cases and the current case. Adaptation is a complicated, domain-dependent process that uses rules to adapt the current case to the problem situation and propose an initial solution, based on the similarities and differences . . . " (*Ex. 1003 at 7:1-6*).

(40) Further, the '839 patent discloses that "[q]uestions belong to one or four categories: Yes/No; Numeric; Text; and List. Yes/No questions are those that have Yes or No answers. Numeric questions are those having answers that are integers. Text questions have textual answers. Finally, List questions have answers selected from a list of legal answers." (*See Ex. 1001 at 3:52-57*).

(41) Hence, consistent with the disclosure of the '839 patent and claim 2 which recites, in part, "wherein each case comprises: at least one question asking about a particular aspect of the computer system that can be answered by the data gathered by the plurality of sensors," it would be clear to a person of ordinary skill in the art that the "decision trees" used in the CBR problem solving process of

Gurer would include at least one question about a particular aspect of the computer system.

(42) Claim 6 of the ‘839 patent is similar to claim 1 discussed above except that it is broader in scope. Claim 6 is not limited to a knowledge database holding a plurality of cases. Accordingly, it would be clear to a person of ordinary skill in the art that the discussion of claim 1 applies correspondingly to claim 6.

(43) Consistent with claim 8 of the ‘839 patent which recites, in part, “wherein the determining step comprises the substeps of: inferring the likely solution to the problem from questions, actions, and rules contained in a knowledge database,” Gurer discloses that “CBR problem solving can be depicted as a five-step process: (1) retrieval, (2) interpretation and adaptation, (3) evaluation and repair, (4) implementation, and (5) evaluation and learning. The first step is retrieving cases that best match the current situation or case. Thus it is crucial to use an appropriate indexing method, such as decision trees or nearest neighbor matching. Once a case is retrieved, it must be interpreted and then adapted. The interpretation process is a simple comparison between the retrieved cases and the current case. Adaptation is a complicated, domain-dependent process that uses rules to adapt the current case to the problem situation and propose an initial solution, based on the similarities and differences. The next step is an evaluation and repair cycle where the proposed solution is evaluated through comparisons to

cases with similar solutions or through simulation, and the solution is modified accordingly. After the CBR system has found its best solution, the solution is implemented and the results are evaluated. (*Ex. 1003 at 7:1-10*).

(44) Further, claim 8 of the '839 patent also recites, in part, "wherein the AI engine utilizes the selected ones of the plurality of sensors to gather information when the knowledge database lacks information necessary to answer a question." Gurer discloses that "[t]he filtering and correlation of alarms is the first step of fault diagnosis. The second step involves further analysis and identification of the exact cause of the alarms, or the fault. This process is an iterative one where alarm data are analyzed and decisions are made whether more data should be gathered, a finer grained analysis should be executed, or problem solving should be performed. Gathering more data can consist of sending tests to network elements or requesting network performance data." (*Ex. 1003 at 6: 18-22*). Gurer also notes earlier that "[t]ypically alarms are produced by . . . network elements (e.g., ATM switches, customer premise equipment)." (*Id. at 1: 30-31*).

(45) Gurer further discloses that "[g]athering more data can consist of sending tests to network elements or requesting network performance data." (*Ex. 1003 at 6:18-22*). Gurer notes earlier that "[t]ypically alarms [or sensors] are produced by . . . network elements (e.g., ATM switches, customer premise equipment)." (*Id. at 1:30-31*). Further, "[f]aults are identified by analyzing the

filtered and correlated alarms and by requesting tests and status updates from the element managers, which provide additional information for diagnosis." (*Id. at 2:4-5*).

(46) The logical alarms described by Gurer involve performing statistical analysis of the network behavior. Hence, consistent with claim 14 of the '839 patent which recites, in part, "wherein the detecting step comprises the steps of: periodically activating selected ones of the plurality of sensors to gather information about the computer system," and similarly claim 17 which recites, in part, "wherein the sensing step comprises the substep of: periodically activating at least one sensor to gather information about the computer system," it would be clear to a person of ordinary skill in the art that the sensor that performs this statistical analysis must be periodically activated in order to recognize the logical alarms. In addition, Gurer describes how the CBR system determines that additional information is needed from the networks elements in order to diagnose a problem, which implies that selected sensors would be activated to collect this information. "After receiving the filtered and correlated alarms, the CBR system attempts to identify what information would further the diagnosis of the fault. The CBR system needs to decide on its own what additional tests need to be made on the network elements and what granularity of NN to use to further analyze the data." (*Ex. 1003 at 8:1-3*).

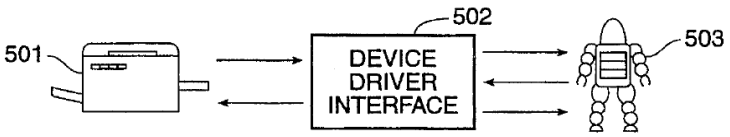
(47) Claim 15 of the ‘839 patent is similar in scope to claim 1 discussed above in with the only differences being (1) claim 1 recites a tool whereas claim 15 recites a program storage device for automatically maintaining a computer system, and (2) the limitation reciting “sensing information about the computer system by *at least one sensor*” in claim 15 versus a *plurality of sensors* in claim 1. It would be clear to a person of ordinary skill in the art that the method of Gurer may be readily executed on a computer readable storage medium. Further, the distinction between the terms “at least one sensor” and “a plurality of sensors” is merely one of form. Accordingly, the discussion of claim 1 applies correspondingly to claim 15.

(48) Thus, in my opinion, Gurer teaches each element of claims 1, 2, 6, 8, 14, 15 and 17 of the ‘839 patent or the minor modifications required would have been easily achieved by those skilled in the art.

**ALLEN ‘218**

(49) U.S. Patent No. 5,586,218 to Allen (“Allen ‘218,” Ex. 1004) describes a case-based reasoning tool that includes a sensor, an AI engine, and a knowledge database containing cases, where one of the three motivating uses of the tool is to automatically perform preventative maintenance on office equipment, e.g., printers, photocopiers, etc. (*See* FIG. 5 of Allen ‘218, entitled “Knobots In Diagnosis and Repair”). (*Ex. 1004 at 1:51 – 2:3).*

KNOBOTS IN DIAGNOSIS AND REPAIR




| | |
|---|---|
| GOALS | ELIMINATE NEED FOR USER INVOLVEMENT IN MAINTENANACE, ANTICIPATE FAILURES FOR PREVENTATIVE MAINTENANCE |
| STIMULI | DEVICE SENSOR READINGS, USER AND DEALER COMMANDS |
| ACTIONS | DEVICE DRIVER OUTPUTS, DIAGNOSTIC AND REPAIR MESSAGES |
| REINFORCEMENT | NEGATIVE IF DEVICE FAILS |
| BENEFITS | GREATLY INCREASED MEAN-TIME-BETWEEN-FAILURE |

*FIG.5*

(50) If the case-based reasoning tool described in Allen '218 determines that it needs additional information to diagnose the problem, a "queries message 119" is generated to request this additional information from the monitored system. *(Id. at 4:56-60).* Depending on the success or failure of the tool in repairing the problem, the set of cases in the knowledge database is updated accordingly based upon this "reinforcement." *(Id. at 4:28-39, 6:28-41).*

(51) Further, Allen '218 discloses "[a] software agent [101] which performs autonomous learning in a real-world environment, implemented in a case-based reasoning system and coupled to ***a sensor*** for gathering information from . . . its environment." (*See Ex.1004 at Abstract*).

(52) Although the language in the Allen '218 patent refers to a single "sensor," the preventative maintenance example illustrated in FIG. 5 above in paragraph 49 refers to the sensor providing multiple "readings" as input to the AI engine. Allen '218 further discloses "[t]he features message 110 may comprise

sensor readings from the device 501 . . .” *(Ex. 1004 at 8:3-4).* To monitor a device as complex as a printer or photocopier, a large number of different features within these computer systems would need to be monitored. It appears that while the inventor of the ‘839 patent chose to describe *separate* sensors collecting different types of information from the monitored system, the inventor of the Allen ‘218 patent instead chose to refer to a single “sensor” as collecting this wide array of different information. A skilled artisan would recognize that Allen ‘218 is simply using “sensor” to refer collectively to the comprehensive set of sensing functionality, which can be selectively activated using a “queries message 119.” *(Ex. 1004 at 4:56-60).*

(53) Hence, consistent with claim 1 of the ‘839 patent which recites, in part, “a plurality of sensors stored in the memory and executing on the processor and adapted for gathering data about the computer system, storing the data in the knowledge database, and detecting whether a computer problem exists from the data and the plurality of cases,” and similarly claim 6 which recites, in part, “utilizing, by the AI engine, selected ones of a plurality of sensors to gather information about the computer system,” it would be clear to a person of ordinary skill in the art that Allen ‘218 does teach the equivalent of activating a particular sensor among a plurality of sensors by virtue of a “queries message” describing exactly what information it would like the collective “sensor” functionality to

collect. Allen '218's "sensor" undoubtedly contains separate subroutines devoted to collecting different types of information from the monitored system, and these individual subroutines could be viewed as a plurality of "sensors." In any event, it would be clear to a skilled artisan that a plurality of "sensors" claimed in the '839 patent could be trivially substituted for the collective "sensor" in Allen '218.

(54) Allen '218 also discloses that "[t]he case database 205 may comprise its current set of cases 126, each of which may comprise a features element 301, which may generally indicate when the case 126 may be useful, and an action element 302, which may indicate the action 105 to take and an evaluation 303 of that action." (*Ex. 1004 at 6:11-15*).

(55) Further, the '839 patent discloses that "[q]uestions belong to one or four categories: Yes/No; Numeric; Text; and List. Yes/No questions are those that have Yes or No answers. Numeric questions are those having answers that are integers. Text questions have textual answers. Finally, List questions have answers selected from a list of legal answers." (*See Ex. 1001 at 3:52-57*).

(56) With case-based reasoning, the cases are structured such that each case contains at least one conditional test or "question" regarding the state of the current problem that is being diagnosed. The state of the current problem is gathered through sensor readings. Hence, consistent with the disclosure of the '839 patent and claim 2 which recites, in part, "wherein each case comprises: at

least one question asking about a particular aspect of the computer system that can be answered by the data gathered by the plurality of sensors," it would be clear to a person of ordinary skill in the art that by virtue of using a case-based reasoning system, as described by Allen '218, each case would necessarily contain at least one question regarding a particular aspect of the system that it is monitoring that can be answered by data provided by sensor input.

(57) Allen '218 also discloses activating an AI engine ("software agent 101") in response to the problem detection. "A software agent 101[, or AI engine,] may be embedded in an environment 102 so that the agent 101 may receive a stimulus 103 from the environment 102 by receiving a stimulus message 104, and may perform an action 105 which affects the environment 102 by sending an action message." (*Ex. 1004 at 3:56-60*).

(58) Further, Allen '218 discloses several explicit metrics that are stored and updated in the knowledge database to enable the AI engine to reason about the likelihood of a potential solution being successful. In particular, the "matching module 401" shown in Figure 4 below is described as generating a "match table 402." (*Ex. 1004 at 7:14-15*).

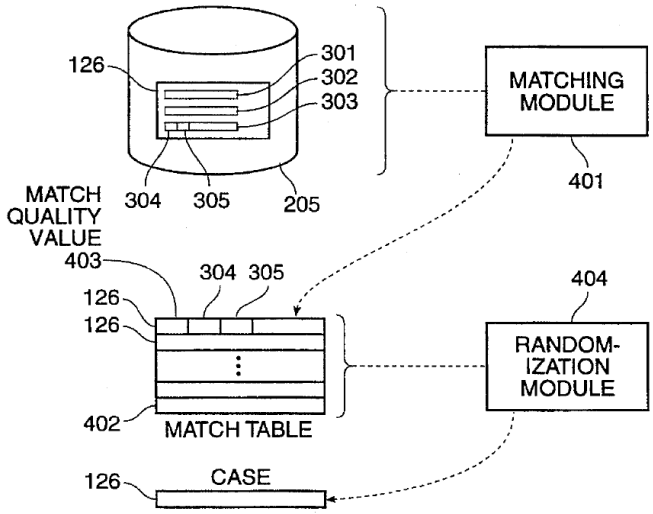



*FIG.4*

"The match table 402 may comprise a set of cases 126 (or indices of cases 126), each having a match quality value 403, the accuracy value 304 of that case 126, and the utility value 305 of that case 126." *(Id. at 7:21-24).* The purpose of this information is to help the AI engine choose a good solution amongst its set of cases and to maximize the likelihood of solving the problem effectively.

(59) Allen '218 also describes having the "evaluation module 306" adjust utility values based upon the "reinforcement," i.e., the success or failure of previous actions in solving the problem. (*Ex. 1004 at 6:28-32*). "In a preferred embodiment, the evaluation module 306 may alter the utility value 305 of each case 126 by adding the reinforcement 114 and the utility value 305 of the case 126 which is the "best match" for the next action." *(Id. at 6:42-42).* Hence, consistent with claim 6 of the '839 patent which recites, in part, "determining, by the AI engine, a likely solution to the problem from the gathered information," it would

be clear to a person of ordinary skill in the art that Allen '218 teaches determining, by the AI engine, a likely solution to the problem from the gathered information.

(60) As discussed above in paragraph 51, Allen '218 discloses "[a] software agent [101] which performs autonomous learning in a real-world environment, implemented in a case-based reasoning system and coupled to ***a sensor*** for gathering information from . . . its environment." (*See Ex.1004 at Abstract*). Further, Allen '218 discloses "[a] behavior module 118 may receive the motives message 117, the features message 110 and the reward message 113, and may generate a queries message 119, which may comprise software objects 107 which describe a request 120 by the agent 101 for further information." (*Id. at 4:56-60*).

(61) Hence, consistent with claim 8 of the '839 patent which recites, in part, "wherein the AI engine utilizes the selected ones of the plurality of sensors to gather information when the knowledge database lacks information necessary to answer a question," it would be clear to a person of ordinary skill in the art that, at least for reasons similar to those discussed above in paragraph 53, Allen '218 does teach the equivalent of activating a particular sensor among a plurality of sensors by virtue of a "queries message" describing exactly what information it would like the collective "sensor" functionality in Allen '218 to collect.

(62) Additionally, Allen ‘218 explicitly discusses the fact that the sensor gathers information from its environment, which, in the case of the preventative maintenance example illustrated in Figure 5 in paragraph 49 above, is the computer system within the office equipment, e.g., printer, photocopier, that it is monitoring. (*Ex. 1004 at 7:51-52*). Allen ‘218 describes that the purpose of this example tool is “to anticipate failures of the device 501 so that preventative maintenance may be performed so as to reduce failures and to increase meantime-between-failure.” (*Id. at 7:67-8:3)*.

(63) Hence, consistent with claim 14 of the ‘839 patent which recites, in part, “wherein the detecting step comprises the steps of: periodically activating selected ones of the plurality of sensors to gather information about the computer system,” and similarly claim 17 which recites, in part, “wherein the sensing step comprises the substep of: periodically activating at least one sensor to gather information about the computer system,” it would be clear to a person of ordinary skill in the art that because *preventative* maintenance requires that a system be monitored regularly and in advance of failure, this implicitly requires that the sensor(s) be *periodically activated* to assess the current health of the monitored system. In addition, the fact that the AI engine described by Allen ‘218 may generate a “queries message 119” to request additional information in order to

diagnose a problem represents an additional example of periodically activating a selected sensor. (*Ex. 1004 at 4:56-60*).

(64) Claim 15 of the '839 patent is similar in scope to claim 1 discussed above with the only differences being (1) claim 1 recites a tool whereas claim 15 recites a program storage device for automatically maintaining a computer system, and (2) the limitation reciting "sensing information about the computer system by *at least one sensor*" in claim 15 versus a *plurality of sensors* in claim 1. It would be clear to a person of ordinary skill in the art that the method of Gurer may be readily executed on a computer readable storage medium. Further, the distinction between the terms "at least one sensor" and "a plurality of sensors" is merely one of form. Accordingly, the discussion of claim 1 applies correspondingly to claim 15.

(65) Thus, in my opinion, Allen '218 teaches each element of claims 1, 2, 6, 8, 14, 15 and 17 of the '839 patent or the minor modifications required would have been easily achieved by those skilled in the art.

**BARNETT**

(66) U.S. Patent No. 5,664,093 to Barnett et al. ("Barnett," Ex. 1005) describes a rule-based artificial intelligence (AI) engine that uses a plurality of sensors and a knowledge database to identify faults and propose solutions to those faults in a distributed computer system. As illustrated in Figure 2 of Barnett, the

AI engine is depicted by reference numeral (26), which the specification refers to as the "diagnostic system," in combination with the inference engine (32), the plurality of sensors are the multiple nodes, each labeled "agent process" (24) or referred to as the "measurement agents" in the specification, and the knowledge database is labeled "rule base" (34). (*Ex. 1005 at 2:58-64, 4:8-10*).

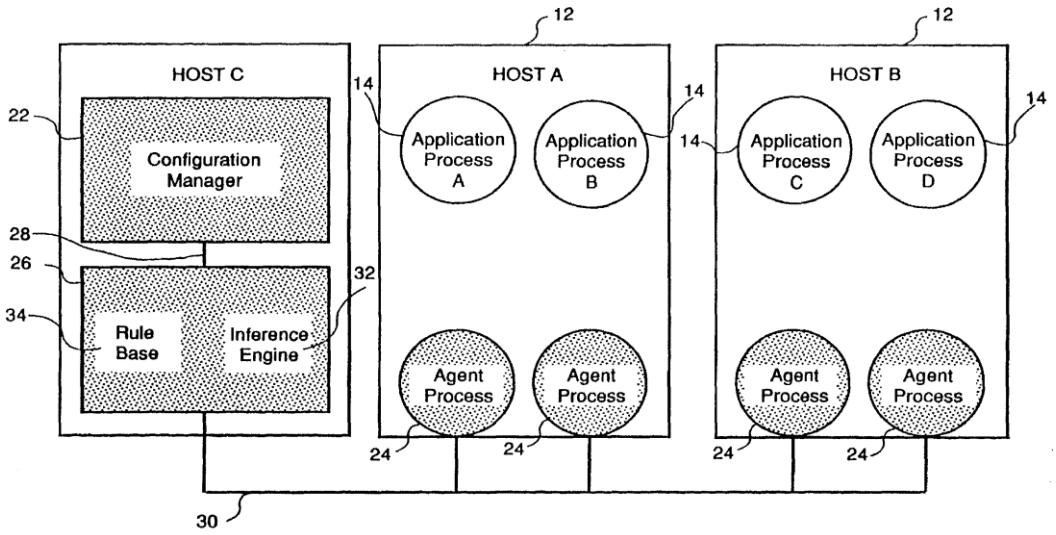



FIG. 2

(67) In addition, the AI engine described by Barnett also uses a "configuration manager" (22) to model abstract relationships between components in the distributed system. (*Ex. 1005 at 6:8-10*). If the AI engine in Barnett decides that it needs additional information to diagnose the fault, it can query its sensors, i.e., measurement agents (24), to obtain that information. (*Id. at 4:51-52, 5:48-50*).

(68) Barnett discloses that "[t]he diagnostic system receives the configuration information from the configuration manager and the performance

information from the plurality of measurement agents and uses the configuration and performance information with the plurality of rules to identify faults and provide solutions for the faults." (*Ex. 1005 at 2:4-9*). Barnett, however, is silent on the subject of what happens when its rule-based AI engine fails to determine a likely solution to the fault.

(69) Although rule-based systems do not automatically update their rules when they fail, Barnett does teach that the precision of the AI engine can be improved by adding additional rules. (*Ex. 1005 at 7:1-6*). Hence, consistent with claim 6 of the '839 patent which recites, in part, "when a likely solution cannot be determined, saving a state of the computer system.," it would be clear to a person of ordinary skill in the art that in order for the programmer or user to understand how and when these additional rules should be added, a state of the computer system should be saved when the AI engine fails to determine a likely solution, so that the programmer or user can later determine the context when the failure took place. Since actions such as this are implemented via rules in a rule-based system, it would be natural and obvious to a skilled artisan to add a rule to the rule base that states that when a likely solution cannot be found, such a state of the computer system should be saved.

(70) In addition, U.S. Patent No. 5,581,664 to Allen ("Allen '664," Ex. 1006) describes a case-based reasoning system where "[i]n the case-matching step

202 or in the best-case step 203, the inference engine 111 may determine that there is no case 105 [or solution] which is a good match for the case template 312. The inference engine 111 may create a new case 105 which partly or fully copies the case template 312, and may ask the user 119 . . . what the prescribed action 309 for the case 105 should be . . . The inference engine 111 may also add new cases 105 to the case base 104 without asking the user 119. When the inference engine 111 determines that there is no case 105 which is a good match for the case template 312, it may invoke a rule 103 or a procedural structure 117 which creases a new case 105 . . .” (*Id. at 8:21-36*).

(71) A skilled artisan would be motivated to combine the teachings of Barnett and Allen ‘664 because the case-based reasoning system described by Allen ‘664, which is smoothly integrated into a rule-based reasoning system, could be easily substituted for the rule-based AI engine described in Barnett. It would be clear to a skilled artisan at that time that Allen ‘664’s hybrid case-based/rule-based approach would provide advantages over Barnett’s simple rule-based approach, e.g., it could handle cases that did not match specific rules, it could learn and adapt its strategies over time, etc., which were well-documented in the many articles that had been published at that time on case-based reasoning. Hence this “upgrade” of the AI engine from rule-based to a hybrid case-based/rule-based approach would be an obvious step for improving Barnett’s system.

(72) Consistent with claim 8 of the '839 patent which recites, in part, "wherein the determining step comprises the substeps of: inferring the likely solution to the problem from questions, actions, and rules contained in a knowledge database," Barnett discloses "[t]he querying operation is performed by an interference engine 32. The diagnostic system also includes a rule base 34 comprising a plurality of rules for the various objects within the distributed application. By using the abstract relationships learned from the configuration manager and the rules from the rule base, the diagnostic system is able to monitor performance and diagnose several types of failures that are reported from any of the plurality of measurement agents." (*Ex. 1005 at 4:6-14*).

(73) Further, claim 8 of the '839 patent also recites, in part, "wherein the AI engine utilizes the selected ones of the plurality of sensors to gather information when the knowledge database lacks information necessary to answer a question." Barnett discloses that "the diagnostic system 26 does not have information about the static or dynamic model, but can obtain all of the information from the configuration manager 22 and the measurement agents 24." (*Ex. 1005 at 4:25-28*). Barnett further discloses "[t]here are several mechanisms which permit the diagnostic system to ask the configuration manager and the measurement agents for information. For example, there may be a coordinator which the diagnostic system uses to communicate with the configuration manager and the agents. The

agent may in turn talk to other agents if they need to abstract and encapsulate information to the diagnostic system." (*Id. at 3:2-9*).

(74) Further, Barnett discloses that "the fault and performance analysis of the present invention can be used for any system composed of hardware and software components, and even abstract components like actions, tasks, and deliverables. It can be integrated into network management applications and capacity planning tools. It can do load balancing and predictive fault analysis. In general it can be applied to any distributed application on a computer network that needs to react to changes in a dynamic or static environment." (*Ex. 1005 at 7:7-15*).

(75) Barnett also teaches that as part of diagnosing a fault, the AI engine can decide to activate specific sensors, i.e., measurement agents (24), to collect more information about the state of the computer system. (*Ex. 1005 at 4:50-52, 5:47-50*). Hence, consistent with claim 14 of the '839 patent which recites, in part, "wherein the detecting step comprises the steps of: periodically activating selected ones of the plurality of sensors to gather information about the computer system," it would be clear to a person of ordinary skill in the art that that the diagnostic system of Barnett would periodically activate at least one of the measurement agents (24) to gather information about the computer system, i.e., performance information related to the components in the distributed system, particularly when

the fault and performance analysis of Barnett is integrated into network management applications and capacity planning tools and can do load balancing and predictive fault analysis. Further, the decision regarding which specific sensors to activate is informed by the information on abstract relationships between components that is stored in the configuration manager. (*Id. at 5:41-44*).

(76) Thus, in my opinion, Barnett teaches each element of claims 6, 8 and 14 of the '839 patent or the minor modifications required would have been easily achieved by those skilled in the art.

## VII. Miscellaneous Matters

(77) I have been informed that a patent claim can be found unpatentable as obvious where the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the relevant field. I understand that an obviousness analysis involves a consideration of (1) the scope and content of the prior art; (2) the differences between the claimed invention and the prior art; (3) the level of ordinary skill in the pertinent field; and (4) secondary considerations of non-obviousness.

(78) For my efforts in connection with the preparation of this declaration I have been compensated at my standard hourly rate for this type of consulting

activity. My compensation is in no way contingent on the results of these or any other proceedings relating to the above-captioned patent.

(79) I reserve the right to supplement my opinions in the future to respond to any arguments that Patentee raises and to take into account new information as it becomes available to me.

(80) I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the results of these proceedings.

Date: 12/21/12

Prof. Todd C. Mowry, Ph.D.

## MOWRY DECLARATION: APPENDIX LIST

**A-1** Curriculum Vitae of Dr. Todd C. Mowry, Ph.D.

**A-2** Riesbeck, C. K., et al. Inside Case-Based Reasoning. Hillsdale: L. Erlbaum Assoc. Inc., 1989.

**A-3** Kolodner, J. L. "An introduction to case-based reasoning." *Artificial Intelligence Review*, 6(1):3-34, March 1992.

**A-4** Aamodt, A., et al. "Case-based reasoning: Foundational issues, methodological variations, and system approaches." *AI Communications*, 7(1):39-59, March 1994.

**A-5** Wess, S., et al. Topics in case-based reasoning: First European Workshop, EWCBR-93, Kaiserslautern, Germany, November 1-5, 1993, Selected Papers. London, UK: Springer, 1994.

**A-6** Veloso, M. M., et al. Case-Based Reasoning Research and Development: First International Conference, ICCBR-95, Sesimbra, Portugal, October 23-26, 1995, Proceedings. London, UK: Springer, 1995.

# Todd C. Mowry

Curriculum Vitae

August, 2012

**Contact Information**


Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Phone: (412) 268-3725
Fax: (412) 268-5576
Email: tcm@cs.cmu.edu
URL: http://www.cs.cmu.edu/~tcm


**Education**

Ph.D. in Electrical Engineering, Stanford University, March 1994.

- Thesis: "Tolerating Latency Through Software-Controlled Data Prefetching."
- Supervisors: Anoop Gupta and Monica Lam.

M.S. in Electrical Engineering, Stanford University, June 1989.

B.S. in Electrical Engineering with Highest Distinction, University of Virginia, May 1988.

**Academic Appointments**

*7/08–Present:* Professor, Computer Science Department (with a courtesy appointment in the Department of Electrical and Computer Engineering), Carnegie Mellon University.

*8/09–6/10:* Associate Department Head for Faculty, Computer Science Department

*7/97–6/08:* Associate Professor, Computer Science Department (with a courtesy appointment in the Department of Electrical and Computer Engineering), Carnegie Mellon University.

*12/93–6/97:* Assistant Professor, Department of Electrical and Computer Engineering (with a courtesy appointment in the Department of Computer Science), University of Toronto.

*9/89–11/93:* Graduate Research Assistant, Stanford University.

**Honors and Awards**

- Second Place, Audience Choice Award for Best Demo (for Claytronics) at the Intel Developers Forum, 2006. (Chosen among several hundred demos of Intel's latest research and technology.)
- SCS Doctoral Dissertation Award co-won by Angela Demke Brown (Ph.D. advisee), 2005.
- Most Thought-Provoking Idea Award (for Claytronics), the Wild and Crazy Idea Session IV held at ASPLOS-XI, 2004.
- Best Paper Award, the 20th International Conference on Data Engineering (ICDE), 2004.
- SCS Doctoral Dissertation Award co-won by J. Gregory Steffan (Ph.D. advisee), 2003.
- Runner-Up for Best Paper Award, the ACM SIGMOD Conference, 2001.
- Alfred P. Sloan Research Fellow, 1999-2001.
- TR100 Award (awarded by MIT's Technology Review magazine to the top 100 most promising young innovators in science and technology), 1999.
- Best Paper Award, the Second Symposium on Operating Systems Design and Implementation, 1996.
- IBM Faculty Development Awards, 1996, 1997, 1998, 2000, 2001, 2002, 2003.
- Arthur Samuel Thesis Award (awarded by the Stanford Computer Science department to the top two Ph.D. theses in a given year), 1994.

**Doctoral Thesis Supervision**

| Student | Thesis Title (or Topic) | Graduation Date |
|---|---|---|
| Chi-Keung Luk | Optimizing the Cache Performance of Non-Numeric Applications. *(Nominated for the ACM Thesis Award by the University of Toronto Department of Computer Science.)* | January, 2000 |
| J. Gregory Steffan | Hardware Support for Thread-Level Speculation. *(Co-winner of the SCS Doctoral Dissertation Award.)* | April, 2003 |
| Antonia Zhai | Compiler Optimization of Value Communication for Thread-Level Speculation. | January, 2005 |
| Angela Demke Brown | Explicit Compiler-based Memory Management for Out-of-core Applications. *(Co-winner of the SCS Doctoral Dissertation Award.)* | May, 2005 |
| Christopher Colohan | Applying Thread-Level Speculation to Database Transactions. | November, 2005 |
| Shimin Chen | Redesigning Database Systems in Light of CPU Cache Prefetching. | December, 2005 |
| Amit Manjhi | Increasing the Scalability of Dynamic Web Applications | March, 2008 |
| Olatunji Ruwase | Dynamic Monitoring of Device Driver Code | December, 2012 *(expected)* |
| Michelle Goodstein | Dynamic Monitoring of Parallel Software | May, 2013 *(expected)* |
| Vivek Seshadri | Multicore Cache Hierchies | May, 2014 *(expected)* |
| Gennady Pekhimenko | Monitoring to Confirm Software Patch Safety | May, 2015 *(expected)* |

**Masters Thesis Supervision**

*NOTE: these students were all at the University of Toronto.*

| Student | Thesis Title (or Topic) | Graduation Date |
|---|---|---|
| Matthew Lam | Multicast Support in Mesh Networks | July, 1996 |
| Robert Ho | Doacross Parallelism in Single-Chip Multiprocessors | September, 1996 |
| Angela Demke | Compiler-Inserted I/O Prefetching for Out-of-Core Applications | January, 1997 |
| Gregory Steffan | The Potential for Thread-Level Data Speculation | January, 1997 |
| Charles Chan | Prefetching in Networks of Workstations | February, 1998 |
| Adley Lo | Multithreading in Networks of Workstations | February, 1998 |
| Antonia Zhai | A Compiler Algorithm for Scheduling Non-Numeric Code with Explicitly Forwarded Data on a Speculative Multiprocessor | September, 1998 |
| Daniel Sladic | Exploiting Fast Comunication in Single-Chip Multiprocessing | September, 1998 |
| Sherwyn Ramkissoon | Software-Controlled Multithreading | January, 1999 |

**Teaching Experience**

**Graduate Courses Taught**

| Course Number | Title | Year | Enrollment | Instructor Rating |
|---|---|---|---|---|
| 15-740 | Computer Architecture | 2012 | 60 | N/A |
| | | 2009 | 35 | 3.90/5.00, 4.45/5.00 |
| | | 2007 | 26 | 4.33/5.00 |
| | | 2001 | 31 | 4.48/5.00 |
| | | 2000 | 28 | 4.46/5.00 |
| | | 1999 | 26 | 4.58/5.00 |
| 15-745 | Optimizing Compilers for Modern Architectures | 2012 | 39 | 4.09/5.00 |
| | | 2011 | 22 | 4.86/5.00 |
| | | 2003 | 23 | 4.74/5.00 |
| | | 2003 | 23 | 4.74/5.00 |
| | | 2001 | 14 | 4.56/5.00 |
| 15-740 | Basic Computer Systems | 1998 | 27 | 4.47/5.00 |
| ECE1755S | Parallel Computer Architecture | 1997 | 6 | N/A |
| | | 1996 | 12 | N/A |
| | | 1995 | 15 | N/A |
| | | 1994 | 9 | N/A |
| ECE1760S | HW/SW Tradeoffs in High-Performance Computing | 1996 | 8 | N/A |
| | | 1995 | 12 | N/A |
| ECE1753F | Optimizing Compilers | 1994 | 20 | N/A |

*(NOTE: The University of Toronto does not collect course or instructor ratings for graduate courses.)*

**Undergraduate Courses Taught**

| Course Number | Title | Year | Enrollment | Instructor Rating |
|---|---|---|---|---|
| 15-213 | Intro to Computer Systems | 2012 | 230 | 4.35/5.00 |
| | | 2007 | 147 | 4.06/5.00 |
| | | 2000 | 140 | 3.91/5.00 |
| | | 1999 | 144 | 4.34/5.00 |
| 15-418/ 15-495 | Parallel Computer Architecture and Programming | 2011 | 26 | 4.29/5.00 |
| | | 2008 | 14 | 4.20/5.00 |
| | | 2004 | 16 | 3.92/5.00 |
| | | 2002 | 23 | 4.80/5.00 |
| | | 2002 | 14 | 4.56/5.00 |
| 15-347 | Computer Architecture | 1998 | 116 | 3.85/5.00 |
| CSC372S | Microprocessor Software | 1997 | 54 | 6.0/7.0 |
| | | 1996 | 13 | 6.2/7.0 |
| | | 1995 | 32 | 5.5/7.0 |
| ECE540F | Optimizing Compilers | 1996 | 34 | 5.68/7.0 |
| | | 1995 | 35 | 5.44/7.0 |

**Publications**

**Refereed Journal Articles**

- Shimin Chen, Phillip B. Gibbons, Michael Kozuch, and Todd C. Mowry. Log-Based Architectures: Using Multicore to Help Software Behave Correctly. In *Operating Systems Review*, 45(1), January 2011.
- Shimin Chen, Michael Kozuch, Phillip B. Gibbons, Michael Ryan, Theodoros Strigkos, Todd C. Mowry, Olatunji Ruwase, Evangelos Vlachos, Babak Falsafi, and Vijaya Ramachandran. Flexible Hardware Acceleration for Instruction-Grain Lifeguards. In *IEEE Micro (Top Picks from 2008 Computer Architecture Conferences)*, 29(1), January 2009.
- Seth C. Goldstein, Todd C. Mowry, Jason D. Campbell, Michael P. Ashley-Rollman, Michael De Rosa, Stanislav Funiak, James F. Hoburg, Mustafa E. Karagozler, Brian Kirby, Peter Lee, Padmananabhan Pillai, J. Robert Reid, Daniel D. Stancil, and Michael P. Weller. Beyond Audio and Video: Using Claytronics to Enable Pario. In *AI Magazine*, 30(2), March 2009.
- Antonia Zhai, J. Gregory Steffan, Christopher B. Colohan, and Todd C. Mowry. Compiler and Hardware Support for Reducing the Synchronization of Speculative Threads. In *ACM Transactions on Architecture and Code Optimization (TACO)*, 5(1), May 2008.
- Christopher B. Colohan, Anastassia Ailamaki, J. Gregory Steffan, and Todd C. Mowry. Incrementally parallelizing database transactions with thread-level speculation. In ACM Transactions on Computer Systems (TOCS), 26(1), February 2008.
- Shimin Chen, Anastassia Ailamaki, Phillip B. Gibbons, and Todd C. Mowry. Improving Hash Join Performance through Prefetching. In *ACM Transactions on Database Systems*, 32(3):1-32, September 2007.
- Christopher B. Colohan, Anastasia Ailamaki, J. Gregory Steffan, and Todd C. Mowry. CMP Support for Large and Dependent Speculative Threads. In *IEEE Transactions on Parallel and Distributed Systems*, 18(8):1041-1054, August 2007.
- J. Gregory Steffan, Christopher B. Colohan, Antonia Zhai and Todd C. Mowry. The STAMPede Approach to Thread-Level Speculation. In *ACM Transactions on Computer Systems*, 23(3):253-300, August 2005.
- Seth Copen Goldstein, Jason Campbell and Todd C. Mowry. Programmable Matter. In *IEEE Computer*, 38(6):99-101, June 2005.
- Angela Demke Brown, Todd C. Mowry and Orran Krieger. Compiler-Based I/O Prefetching for Out-of-Core Applications. In *ACM Transactions on Computer Systems*, 19(2):111-170, May 2001.
- Chi-Keung Luk and Todd C. Mowry. Architectural and Compiler Support for Effective Instruction Prefetching: A Cooperative Approach. In *ACM Transactions on Computer Systems*, 19(1):71-109, February 2001.
- Todd C. Mowry and Chi-Keung Luk. Understanding Why Correlation Profiling Improves the Predictability of Data Cache Misses in Nonnumeric Applications. In *IEEE Transactions on Computers*, 49(4), April 2000.
- Chi-Keung Luk and Todd C. Mowry. Automatic Compiler-Inserted Prefetching for Pointer-Based Applications. In *IEEE Transactions on Computers*, 48(2), February 1999.
- Mark Horowitz, Margaret Martonosi, Todd C. Mowry, and Michael D. Smith. Informing Memory Operations: Memory Performance Feedback Mechanisms and their Applications. In *ACM Transactions on Computer Systems*, 16(2):170-205, May 1998.
- Todd C. Mowry. Tolerating Latency in Multiprocessors through Compiler-Inserted Prefetching. In *ACM Transactions on Computer Systems*, 16(1):55-92, February 1998.

- Todd Mowry and Anoop Gupta. Tolerating Latency Through Software-Controlled Prefetching in Shared-Memory Multiprocessors. In *Journal of Parallel and Distributed Computing*, 12(2):87-106, 1991.

**Refereed Conference Papers**

- Michelle L. Goodstein, Shimin Chen, Phillip B. Gibbons, Michael A. Kozuch, and Todd C. Mowry. Chrysalis Analysis: Incorporating Synchronization Arcs in Dataflow-Analysis-based Parallel Monitoring. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT-2012)*, September 2012.
- Vivek Seshadri, Onur Mutlu, Todd C. Mowry, and Michael A. Kozuch. The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT-2012)*, September 2012.
- Gennady Pekhimenko, Vivek Seshadri, Onur Mutlu, Todd C. Mowry, Phillip B. Gibbons, and Michael A. Kozuch. Base-Delta-Immediate Compression: A Practical Data Compression Mechanism for On-Chip Caches. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT-2012)*, September 2012.
- Olatunji Ruwase, Shimin Chen, Phillip Gibbons, and Todd C. Mowry. Decoupled Lifeguards: Enabling Path Optimizations for Dynamic Correctness Checking Tools. In *Proceedings of the ACM SIGPLAN 2010 Conference on Programming Language Design and Implementation (PLDI)*, June 2010.
- F. Ryan Johnson, Radu Stoica, Anastasia Ailamaki, and Todd C. Mowry. Decoupling Contention Management from Scheduling. In *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010)*, March 2010.
- Michelle Goodstein, Evangelos Vlachos, Shimin Chen, Phillip B. Gibbons, Michael Kozuch, and Todd C. Mowry. Butterfly Analysis: Adapting Dataflow Analysis to Dynamic Parallel Monitoring. In *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010)*, March 2010.
- Evangelos Vlachos, Michelle Goodstein, Michael Kozuch, Shimin Chen, Babak Falsafi, Phillip B. Gibbons, and Todd C. Mowry. ParaLog: Enabling and Accelerating Online Parallel Monitoring of Multithreaded Applications. In *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010)*, March 2010.
- Amit Manjhi, Charles Garrod, Bruce M. Maggs, Todd C. Mowry, Anthony Tomasic. Holistic Query Transformations for Dynamic Web Applications. In *Proceedings of the 2009 IEEE 25th International Conference on Data Engineering (ICDE)*, March-April 2009.
- Daniel J. Dewey, Michael P. Ashley-Rollman, Michael DeRosa, Seth Copen Goldstein, Todd C. Mowry, Siddhartha S. Srinivasa, Padmanabhan Pillai, and Jason Campbell. Generalizing metamodules to simplify planning in modular robotic systems. In *Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems (IROS)*, September 2008.
- Lei Li, Wenjie Fu, Fan Guo, Todd C. Mowry, and Christos Faloutsos. Cut-and-stitch: efficient parallel learning of linear dynamical systems on SMPs. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, August 2008.
- Shimin Chen, Michael Kozuch, Theodoros Strigkos, Babak Falsafi, Phillip B. Gibbons, Todd C. Mowry, Michael Ryan, Olatunji Ruwase, and Evangelos Vlachos. Flexible Hardware Acceleration for Instruction-Grain Program Monitoring. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA)*, June 2008.

- Olatunji Ruwase, Phillip B. Gibbons, Todd C. Mowry, Vijaya Ramachandran, Shimin Chen, Michael Kozuch, and Michael Ryan. Parallelizing dynamic information flow tracking. In *Proceedings of the 20th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, June 2008.
- Brian Kirby, Burak Aksak, James Hoburg, Todd C. Mowry, and Padmanabhan Pillai. A Modular Robotic System Using Magnetic Force Effectors. In *Proceedings of the IEEE/RSJ 2007 International Conference on Intelligent Robots and Systems (IROS)*, October 2007.
- Michael Ashley-Rollman, Seth Goldstein, Peter Lee, Todd C. Mowry, and Padmanabhan Pillai. Meld: A Declarative Approach to Programming Ensembles. In *Proceedings of the IEEE/RSJ 2007 International Conference on Intelligent Robots and Systems (IROS)*, October 2007.
- Amit Manjhi, Phillip B. Gibbons, Anastassia Ailamaki, Charles Garrod, Bruce M. Maggs, Todd C. Mowry, Christopher Olston, Anthony Tomasic, and Haifeng Yu. Invalidation Clues for Database Scalability Services. In *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering (ICDE)*, pages 316-325, April 2007.
- Michael De Rosa, Peter Lee, Seth Goldstein, Jason Campbell, Padmanabhan Pillai, and Todd C. Mowry. Distributed Watchpoints: Debugging Large Multi-Robot Systems. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3723-3729, April 2007.
- Benjamin D. Rister, Jason Campbell, Padmanabhan Pillai, and Todd C. Mowry. Integrated Debugging of Large Modular Robot Ensembles. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2227-2234, April 2007.
- Shimin Chen, Phillip Gibbons, Michael Kozuch, Vasileios Liaskovitis, Anastassia Ailamaki, Guy Blelloch, Babak Falsafi, Limor Fix, Nikos Hardavellas, Todd C. Mowry and Chris Wilkerson. Scheduling Threads for Constructive Cache Sharing on CMPs. In *Proceedings of 18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 105-115, June 2007.
- Shimin Chen, Babak Falsafi, Phillip B. Gibbons, Michael Kozuch, Todd C. Mowry, Radu Teodorescu, Anastassia Ailamaki, Limor Fix, Gregory R. Ganger, Bin Lin, and Steven W. Schlosser. Log-Based Architectures for General-Purpose Monitoring of Deployed Code. In *Proceedings of the Workshop on Architectural and System Support for Improving Software Dependability (ASID), held with ASPLOS XII*, October 2006.
- Christopher B. Colohan, Anastassia Ailamaki, J. Gregory Steffan, and Todd C. Mowry. Tolerating Dependences Between Large Speculative Threads Via Sub-Threads. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA)*, pages 216-226, June 2006.
- Amit Manjhi, Anastassia Ailamaki, Bruce M. Maggs, Todd C. Mowry, Christopher Olston, and Anthony Tomasic. Simultaneous Scalability and Security for Data-Intensive Web Applications. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 241-252, June 2006.
- Vasileios Liaskovitis, Shimin Chen, Phillip B. Gibbons, Anastassia Ailamaki, Guy E. Blelloch, Babak Falsafi, Limor Fix, Nikos Hardavellas, Michael Kozuch, Todd C. Mowry, and Chris Wilkerson. Parallel Depth First vs. Work Stealing Schedulers on CMP Architectures. In *Proceedings of 18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, August 2006.
- Brian Kirby, Jason Campbell, Burak Aksak, Padmanabhan Pillai, James F. Hoburg, Todd C. Mowry, and Seth Copen Goldstein. Catoms: Moving Robots without Moving Parts. In *Proceedings of the Twentieth National Conference qon Artificial Intelligence (AAAI)*, pages 1730-1731, July 2005.

- Christopher Olston, Amit Manjhi, Charles Garrod, Anastassia Ailamaki, Bruce M. Maggs, and Todd C. Mowry. A Scalability Service for Dynamic Web Applications. In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 56-69, January 2005.
- Christopher B. Colohan, Anastassia Ailamaki, J. Gregory Steffan, and Todd C. Mowry. Optimistic Intra-Transaction Parallelism on Chip Multiprocessors. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, pages 73-84, September 2005.
- Shimin Chen, Anastassia Ailamaki, Phillip B. Gibbons, and Todd C. Mowry. Inspector Joins. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, pages 817-828, September 2005.
- Antonia Zhai, Christopher B. Colohan, J. Gregory Steffan, and Todd C. Mowry. Compiler Optimization of Memory-Resident Value Communication Between Speculative Threads. In *Proceedings of the 2007 International Symposium on Code Generation and Optimization (CGO)*, pages 39-52, March 2004.
- Shimin Chen, Anastassia Ailamaki, Phillip B. Gibbons and Todd C. Mowry. Improving Hash Join Performance through Prefetching. In *Proceedings of the 20th International Conference on Data Engineering (ICDE)*, pages 116-127, April 2004. *(This paper received the Best Paper Award.)*
- Antonia Zhai, Christopher B. Colohan, J. Gregory Steffan and Todd C. Mowry. Compiler Optimization of Scalar Value Communication Between Speculative Threads. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, October 2002.
- Shimin Chen, Philip B. Gibbons, Todd C. Mowry and Gary Valentin. Fractal Prefetching B+-Trees: Optimizing Both Cache and Disk Performance. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, June 2002.
- J. Gregory Steffan, Christopher B. Colohan, Antonia Zhai and Todd C. Mowry. Improving Value Communication for Thread-Level Speculation. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA)*, February 2002.
- Shimin Chen, Philip B. Gibbons and Todd C. Mowry. Improving Index Performance through Prefetching. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 235-246, May 2001. *(This paper received the Runner-Up for Best Paper Award.)*
- Angela Demke Brown and Todd C. Mowry. Taming the Memory Hogs: Using Compiler-Inserted Releases to Manage Physical Memory Intelligently. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)*, pages 31-44, October 2000.
- J. Gregory Steffan, Christopher B. Colohan, Antonia Zhai and Todd C. Mowry. A Scalable Approach to Thread- Level Speculation. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 1-12, June 2000.
- Todd C. Mowry and Sherwyn R. Ramkissoon. Software-Controlled Multithreading Using Informing Memory Operations. In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, January, 2000.
- Chi-Keung Luk and Todd C. Mowry. Memory Forwarding: Enabling Aggressive Layout Optimizations by Guaranteeing the Safety of Data Relocation. In *Proceedings of the 26th Annual International Symposium on Computer Architecture (ISCA)*, pages 88-99, May 1999.
- Chi-Keung Luk and Todd C. Mowry. Cooperative Prefetching: Compiler and Hardware Support for Effective Instruction Prefetching in Modern Microprocessors. In *Proceedings of the 31st Annual International Symposium on Microarchitecture*, pages 182-193, December 1998.

- J. Gregory Steffan and Todd C. Mowry. The Potential for Using Thread-Level Data Speculation to Facilitate Automatic Parallelization. In *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture*, pages 2-13, February, 1998.
- Charles Chan, Adley Lo, and Todd C. Mowry. Comparative Evaluation of Latency Tolerance Techniques for Software Distributed Shared Memory. In *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture*, pages 300-311, February, 1998.
- Todd C. Mowry and Chi-Keung Luk. Predicting Data Cache Misses in Non-Numeric Applications Through Correlation Profiling. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 314-320, December 1997.
- Todd C. Mowry, Angela K. Demke and Orran Krieger. Automatic Compiler-Inserted I/O Prefetching for Out-of-Core Applications. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation (OSDI '96)*, pages 3-17, October 1996. *(This paper received the Best Paper Award.)*
- Chi-Keung Luk and Todd C. Mowry. Compiler-Based Prefetching for Recursive Data Structures. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 222-233, October 1996.
- Edouard Bugnion, Jennifer M. Anderson, Todd C. Mowry, Mendel Rosenblum and Monica S. Lam. Compiler- Directed Page Coloring for Multiprocessors. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 244-255, October 1996.
- Mark Horowitz, Margaret Martonosi, Todd C. Mowry, and Michael D. Smith. Informing Memory Operations: Providing Memory Performance Feedback in Modern Processors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 260-270, May 1996.
- Todd C. Mowry, Monica S. Lam and Anoop Gupta. Design and Evaluation of a Compiler Algorithm for Prefetching. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 62-73, October 1992.
- Anoop Gupta, John Hennessy, Kourosh Gharachorloo, Todd Mowry, and Wolf-Dietrich Weber. Comparative Evaluation of Latency Reducing and Tolerating Techniques. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 254-263, May 1991.
- Anoop Gupta, Wolf-Dietrich Weber, and Todd Mowry. Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes. In *Proceedings of International Conference on Parallel Processing*, pages 312-321, August 1990.

**Technical Reports**

- Evangelos Vlachos, Michelle Goodstein, Michael Kozuch, Shimin Chen, Babak Falsafi, Phillip B. Gibbons, Todd C. Mowry, and Olatunji Ruwase. Parallel LBA: Conherence-based Parallel Monitoring of Multithreaded Applications. Carnegie Mellon University Technical Report CMU-CS-09-108, March 2009.
- Michelle Goodstein, Evangelos Vlachos, Shimin Chen, Phillip Gibbons, Michael Kozuch, and Todd C. Mowry. The Butterfly Model: Theoretical Foundations. Carnegie Mellon University Technical Report CMU-CS-08-170, February 2009.
- Amit Manjhi, Anastassia Ailamaki, Bruce M. Maggs, Todd C. Mowry, Christopher Olston, and Anthony Tomasic. Simultaneous Scalability and Security for Data-Intensive Web Applications. Carnegie Mellon University Technical Report CMU-CS-06-116, March 2006.
- Charles Garrod, Amit Manjhi, Anastassia Ailamakiii, Phil Gibbons, Bruce Maggs, Todd Mowry, Christopher Olston, and Anthony Tomasic. Scalable Consistency Management for Web Database Caches. Carnegie Mellon University Technical Report CMU-CS-06-128, July 2006.

- Amit Manjhi, Phillip B. Gibbons, Anastassia Ailamaki, Charles Garrod, Bruce M. Maggs, Todd C. Mowry, Christopher Olston, Anthony Tomasic, and Haifeng Yu. Invalidation Clues for Database Scalability Services. Carnegie Mellon University Technical Report CMU-CS-06-139, July 2006.
- Christopher B. Colohan, Anastassia Ailamaki, J. Gregory Steffan, and Todd C. Mowry. Supporting Large Speculative Threads for Databases and Beyond. Carnegie Mellon University Technical Report CMU-CS-05-109, July 2005.
- Christopher B. Colohan, Anastassia Ailamaki, J. Gregory Steffan, and Todd C. Mowry. Optimistic Intra-Transaction Parallelism on Chip Multiprocessors. Carnegie Mellon University Technical Report CMU-CS-05-118, March 2005.
- Shimin Chen, Anastassia Ailamaki, Phillip B. Gibbons, and Todd C. Mowry. Improving Hash Join Performance through Prefetching. Carnegie Mellon University Technical Report CMU-CS-03-157, October 2003.
- Shimin Chen, Phillip B. Gibbons, Todd C. Mowry, and Gary Valentin. Fractal Prefetching B+-Trees: Optimizing Both Cache and Disk Performance. Carnegie Mellon University Technical Report CMU-CS-02-115, March 2002.
- Spiros Papadimitrious and Todd C. Mowry. Exploring Thread-Level Speculation in Software: The Effects of Memory Access Tracking Granularity. Carnegie Mellon University Technical Report CMU-CS-01-145, July 2001.
- Shimin Chen, Phillip B. Gibbons and Todd C. Mowry. Improving Index Performance through Prefetching. Carnegie Mellon University Technical Report CMU-CS-00-177, December 2000.
- J. Gregory Steffan, Christopher B. Colohan and Todd C. Mowry. Extending Cache Coherence to Support Thread-Level Data Speculation on a Single Chip and Beyond. Carnegie Mellon University Technical Report CMU-CS-98-171, December 1998.
- Todd C. Mowry and Sherwyn R. Ramkissoon. Software-Controlled Multithreading Using Informing Memory Operations. Carnegie Mellon University Technical Report CMU-CS-98-169, October 1998.
- Chi-Keung Luk and Todd C. Mowry. Compiler and Hardware Support for Automatic Instruction Prefetching: A Cooperative Approach. Carnegie Mellon University Technical Report CMU-CS-98-140, June 1998.
- J. Gregory Steffan, Christopher B. Colohan and Todd C. Mowry. Architectural Support for Thread-Level Data Speculation. Carnegie Mellon University Technical Report CMU-CS-97-188, November 1997.
- Todd C. Mowry and Chi-Keung Luk. Predicting Data Cache Misses in Non-Numeric Applications Through Correlation Profiling. Carnegie Mellon University Technical Report CMU-CS-97-175, September 1997.
- J. Gregory Steffan and Todd C. Mowry. The Potential for Thread-Level Data Speculation in Tightly-Coupled Multiprocessors. University of Toronto Technical Report CSRI-TR-350, February 1997.
- Chi-Keung Luk and Todd C. Mowry. Predicting Data Cache Misses in Non-Numeric Applications Through Correlation Profiling. University of Toronto Technical Report CSRI-TR-359, February, 1997.
- Mark Horowitz, Margaret Martonosi, Todd C. Mowry, and Michael D. Smith. Informing Loads: Enabling Software to Observe and React to Memory Behavior. Stanford University Technical Report CSL-TR-95-673, July 1995.
- Todd C. Mowry. Tolerating Latency Through Software-Controlled Data Prefetching. Technical Report CSL-TR-94-628, Stanford University, June 1994.

**Patents Held**

- Todd C. Mowry. U.S. Patent 7,127,586: Prefetching hints. Issued October, 2006.
- Shimin Chen, Phillip B. Gibbons, and Todd C. Mowry. U.S. Patent 6,772,179: System and method for improving index performance through prefetching. Issued August, 2004.
- Todd C. Mowry. U.S. Patent 6,240,488: Prefetching hints. Issued May, 2001.
- Todd C. Mowry. U.S. Patent 5,732,242: Consistently specifying way destinations through prefetching hints. Issued March, 1998.
- Todd C. Mowry and Earl A. Killian. U.S. Patent 5,696,958: Method and apparatus for reducing delays following the execution of a branch instruction in an instruction pipeline. Issued December, 1997.

**Distinguished Lectures**

*2/7/08: "Pario: the Next Step Beyond Audio and Video"*, University of Toronto, Department of Electrical and Computer Engineering Distinguished Lecture Series.

## Professional Activities

### Professional Societies

- Member of the Institute of Electrical and Electronics Engineers (IEEE)
- Member of the Association of Computing Machinery (ACM)

### Industrial Employment

*5/04–6/07:* Intel Corporation, Pittsburgh, Pennsylvania: Director of the Intel Research Pittsburgh Lab.

*6/89–11/93:* Silicon Graphics, Inc., Mountain View, California (formerly MIPS Computer Systems, Sunnyvale California): Computer Architect (part-time).

### Consulting

*7/07–2/11:* Intel Corporation, Pittsburgh, Pennsylvania: Research Advisor.

*8/96–4/04:* SandCraft, Inc., Santa Clara, California: Member of the Technical Advisory Board.

*1/96–4/04:* IBM, Toronto: Visiting Scientist.

*12/93–12/96:* Silicon Graphics, Inc., Mountain View, California: Computer Architecture Consultant.

## University Committee Work and Other Service Activities

### Committee Work

- Faculty Hiring Committees:
  - Computer Science Department: 2008, 2010.
  - Computer Systems Area: 2000, 2001, 2002, 2003, 2004.
- Chair of the Computer Science Department Head Selection Committee, 2007.
- CSD Ph.D. Admissions Committee, 2001, 2002, 2003.
- School of Computer Science Council, 2001-2004.
- Carnegie Mellon University Faculty Senate (Presidential Appointeee), 2000-2001.
- Computer Systems Area Advocate, 2000-2004.
- Doctoral Review Committee, 1998-2004.
- ACM Thesis Award Nomination Committee, 1999.

### Journal, Conference and Workshop Organization

- Associate Editor, ACM Transactions on Computer Systems, 2001-present.
- Program Chair of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2011.
- Sponsorship Chair, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2010.
- Member of the ISCA program committee, 1998, 2000, and 2011.
- Member of the ASPLOS program committee, 2000, 2004.
- Member of the ASPLOS external review committee, 2012.
- Co-Program Chair (with John Shen) of the International Conference on Parallel Architectures and Compilation Techniques (PACT), 2001.
- Member of the International Symposuim on Microarchitecture program committee, 1998.
- Member of the Workshop on Architectural and System Support for Improving Software Dependability (ASID) program committee, 2005.
- Member of the Workshop on Memory System Performance (MSP) program committee (in conjunction with PLDI), 2002 and 2004.
- Member of the IBM CASCON program committee, 1995, 1996, 1997.
- Member of the First SUIF Workshop Program Committee, 1996.
- Organized the CASCON 97 workshop on Compiler Optimization, 1997.
- Initiated and organized the 1st Toronto / Rochester Workshop on Shared-Memory Multiprocessors, 1994.

### Miscellaneous

- Organizing the Computer Systems faculty lunches, 1999-2008, 2009-present.
- Organized the Computer Systems seminar series in the CMU CS department, 1998-2001.

*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, or 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

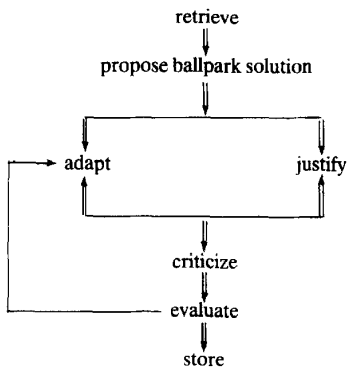



*Fig. 1.* The case-based reasoning cycle.

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.

Published in: *AI Communications*, Vol. 7 Nr. 1, March 1994, pp 39-59

# Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches

Agnar Aamodt
*University of Trondheim, College of Arts and Science, Department of Informatics, N-7055 Dragvoll, Norway.*
*Phone: +47 73 591838; fax: +47 73 591733;*
*e-mail: agnar@ifi.unit.no*

Enric Plaza
*Institut d'Investigació en Intel·ligència Artificial, CSIC, Camí de Santa Bàrbara,17300 Blanes, Catalonia, Spain.*
*e-mail: plaza@ceab.es*

Case-based reasoning is a recent approach to problem solving and learning that has got a lot of attention over the last few years. Originating in the US, the basic idea and underlying theories have spread to other continents, and
we are now within a period of highly active research in case-based reasoning in Europe, as well. This paper gives an overview of the foundational issues related to case-based reasoning, describes some of the leading methodological approaches within the field, and exemplifies the current state through pointers to some systems. Initially, a general framework is defined, to which the subsequent descriptions and discussions will refer. The framework is influenced by recent methodologies for knowledge level descriptions of intelligent systems. The methods for case retrieval, reuse, solution testing, and learning are summarized, and their actual realization is discussed in the light of a few example systems that represent different CBR approaches. We also discuss the role of case-based methods as one type of reasoning and learning method within an integrated system architecture.

## 1. Introduction

Over the last few years, case-based reasoning (CBR) has grown from a rather specific and iso-lated research area to a field of widespread interest. Activities are rapidly growing - as seen by the increased rate of research papers, availability of commercial products, and also reports on applications in regular use. In Europe, researchers and application developers recently met at the First European Workshop on Case-based reasoning, which took place in Germany, November 1993. It gath-ered around 120 people and more than 80 papers on scientific and application-oriented research were presented.

### *1.1. Background and motivation.*

Case-based reasoning is a problem solving paradigm that in many respects is fundamentally differ-ent from other major AI approaches. Instead of re-lying solely on general knowledge of a problem domain, or making associations along generalized relationships between problem descriptors and conclusions, CBR is able to utilize the *specific* knowledge of previously experienced, concrete problem situations (cases). A new problem is solved by finding a similar past case, and reusing it in the new problem situation. A second important difference is that CBR also is an approach to incre-mental, sustained learning, since a new experience is retained each time a problem has been solved, making it immediately available for future prob-lems.

This paper presents an overview of the field, in terms of its underlying foundation, its current state-of-the-art, and future trends. The description of CBR principles, methods, and systems is made within a general analytic scheme. Other authors have recently given overviews of case-based rea-soning (Ch. 1 in [51], Introductory section of [18], [36,61]). Our overview differs in four major ways from these accounts: First, we initially specify a general descriptive framework to which the subse-quent method descriptions will refer. Second, we put a strong emphasis on the methodological issues of case-based reasoning, and less on a discussion of suitable application types and on the advantages of CBR over rule-based systems. (This has been taken very well care of in the documents cited above). Third, we strive to maintain a neutral view of existing CBR ap-

proaches, unbiased by a particu-lar 'school'[1]. And finally, we include results from the European CBR arena, which unfortunately have been missing in American CBR reports.

What is case-based reasoning? Basically: To solve a new problem by remembering a previous similar situation and by reusing information and knowledge of that situation. Let us illustrate this by looking at some typical problem solving situations:

- A physician - after having examined a particu-lar patient in his office - gets a reminding to a patient that he treated two weeks ago. Assuming that the reminding was caused by a similarity of important symptoms (and not the patient's hair-color, say), the physician uses the diagnosis and treatment of the previous patient to determine the disease and treatment for the patient in front of him.
- A drilling engineer, who have experienced two dramatic blow out situations, is quickly re-minded of one of these situations (or both) when the combination of critical measurements matches those of a blow out case. In particular, he may get a reminding to a mistake he made during a previous blow-out, and use this to avoid repeating the error once again.
- A financial consultant working on a difficult credit decision task, uses a reminding to a pre-vious case, which involved a company in simi-lar trouble as the current one, to recommend that the loan application should be refused.

### *1.2. Case-based problem solving.*

As the above examples indicate, reasoning by re-using past cases is a powerful and frequently app-lied way to solve problems for humans. This claim is also supported by results from cognitive psychological research. Part of the foundation for the case-based approach, is its psychological plau-sibility. Several studies have given empirical evi-dence for the dominating role of specific, pre-viously experienced situations (what we call cases) in human problem solving (e.g. [53]). Schank [54] developed a theory of learning and reminding based on retaining of experience in a dynamic, evolving memory[2] structure. Anderson [6] has shown that people use past cases as models when learning to solve problems, particularly in early learning. Other results (e.g. by W.B. Rouse [75]) indicate that the use of past cases is a predominant problem solving method among experts as well. Studies of problem solving by analogy (e.g. [16,22]) also shows the frequent use of past experience in solving new and different problems. Case-based reasoning and analogy are sometimes used as synonyms (e.g. in [16]). Case-based reasoning can be considered a form of *intra-domain analogy*. However, as will be discussed later, the main body of analogical research [14,23,30] has a differ-ent focus, namely analogies across domains.

In CBR terminology, a *case* usually denotes a *problem situation*. A previously experienced situation, which has been captured and learned in such way that it can be reused in the solving of future problems, is referred to as a past case, previous case, stored case, or retained case. Correspond-ingly, a new case or unsolved case is the description of a new problem to be solved. Case-based reasoning is - in effect - a cyclic and integrated process of solving a problem, learning from this experience, solving a new problem, etc.

Note that the term problem solving is used here in a wide sense, coherent with common practice within the area of knowledge-based systems in general. This means that problem solving is not necessarily the finding of a concrete solution to an application problem, it may be any problem put forth by the user. For example, to justify or criti-cize a solution proposed by the user, to interpret a problem situation, to generate a set of possible so-lutions, or generate expectations in observable data are also problem solving situations.

### *1.3. Learning in Case-based Reasoning.*

A very important feature of case-based reason-ing is its coupling to learning. The driving force behind case-based methods has to a large extent come from the machine learning community, and case-based reasoning is also regarded as a subfield of machine

[1]Our own experience from active CBR research over the last 5 years started out from different backgrounds and motivations, and we may have developed different views to some of the major issues involved. We will give examples of our respective priorities and concerns related to CBR research as part of the discussion about future trends towards the end of the paper.

[2]The term 'memory' is often used to refer to the storage structure that holds the existing cases, i.e. to the case base. A memory, thus, refers to what is remembered from previous experiences. Correspondingly, a reminding is a pointer structure to some part of memory.

learning[3]. Thus, the notion of case-based reasoning does not only denote a particular reasoning method, irrespective of how the cases are acquired, it also denotes a machine learning paradigm that enables sustained learning by updating the case base after a problem has been solved. Learning in CBR occurs as a natural by-product of problem solving. When a problem is successfully solved, the experience is retained in order to solve similar problems in the future. When an attempt to solve a problem fails, the reason for the failure is identified and remembered in order to avoid the same mistake in the future.

Case-based reasoning favours learning from experience, since it is usually easier to learn by retaining a concrete problem solving experience than to generalize from it. Still, effective learning in CBR requires a well worked out set of methods in order to extract relevant knowledge from the expe-rience, integrate a case into an existing knowledge structure, and index the case for later matching with similar cases.

### *1.4. Combining cases with other knowledge.*

By examining theoretical and experimental results from cognitive psychology, it seems clear that human problem solving and learning in general are processes that involve the representation and utili-zation of several types of knowledge, and the com-bination of several reasoning methods. If cognitive plausibility is a guiding principle, an architecture for intelligence where the reuse of cases is at the centre, should also incorporate other and more general types of knowledge in one form or another. This is an issue of current concern in CBR research [67].

The rest of this paper is structured as follows: The next section gives a brief historical overview of the CBR field. This is followed by a grouping of CBR methods into a set of characteristic types, and a presentation of the descriptive framework which will be used throughout the paper to discuss CBR methods. Sections 4 to 8 discuss representation is-sues and methods related to the four main tasks of case-based reasoning, respectively. In section 9 we look at CBR in relation to integrated architectures and multistrategy problem solving and learning. This is followed by a short description of some fielded applications, and a few words about CBR development tools. The conclusion part briefly summarizes the paper, and point out some possi-ble trends.

[3]The learning approach of case-based reasoning is sometimes referred to as case-based learning. This term is sometimes also used synonymous with example-based learning, and may therefore point to classical induction and other generalization-driven learning methods. Hence, we will here use the term case-based reasoning both for the problem solving and learning part, and explicitly state which part we talk about whenever necessary.

## 2. History of the CBR field

The roots of case-based reasoning in AI are found in the works of Roger Schank on dynamic memory and the central role that a reminding of earlier situations (episodes, cases) and situation patterns (scripts, MOPs) have in problem solving and learning [54]. Other trails into the CBR field have come from the study of analogical reasoning [22], and - further back - from theories of concept formation, problem solving and experiential learning within philosophy and psychology (e.g. [62,69,72]). For example, Wittgenstein observed that ‘natural concepts’, i.e., concepts that are part of the natural world - such as bird, orange, chair, car, etc. - are polymorphic. That is, their instances may be categorized in a variety of ways, and it is not possible to come up with a useful classical definition in terms of a set of necessary and sufficient features for such concepts. An answer to this problem is to represent a concept extensionally, defined by its set of instances - or cases.

The first system that might be called a case-based reasoner was the CYRUS system, developed by Janet Kolodner [33, 34], at Yale University (Schank's group). CYRUS was based on Schank's dynamic memory model and MOP theory of prob-lem solving and learning [54]. It was basically a question-answering system with knowledge of the various travels and meetings of former US Secre-tary of State Cyrus Vance. The case memory model developed for this system has later served as basis for several other case-based reasoning systems (in-cluding MEDIATOR [59], PERSUADER [68], CHEF [24], JULIA [27], CASEY [38]).

Another basis for CBR, and another set of models, were developed by Bruce Porter and his group [48] at the University of Texas, Austin. They initially addressed the machine learning problem of concept learning for classification tasks. This lead to the development of the PROTOS system [9], which emphasized integrating general domain knowledge and specific case knowledge into a unified representation structure. The combination of cases with general domain

knowledge was pushed further in GREBE [12], an application in the domain of law. Another early significant contribution to CBR was the work by Edwina Rissland and her group at the University of Massachusetts, Amhearst. With sev-eral law scientists in the group, they were interested in the role of precedence reasoning in legal judgements [52]. Cases (precedents) are here not used to produce a single answer, but to interpret a situation in court, and to produce and assess argu-ments for both parties. This resulted in the HYPO system [10], and later the combined case-based and rule-based system CABARET [60]. Phyllis Koton at MIT studied the use of case-based reason-ing to optimize performance in an existing knowledge based system, where the domain (heart fail-ure) was described by a deep, causal model. This resulted in the CASEY system [38], in which case-based and deep model-based reasoning was com-bined.

In Europe, research on CBR was taken up a little later than in the US. The CBR work seems to have been stronger coupled to expert systems develop-ment and knowledge acquisition research than in the US. Among the earliest results was the work on CBR for complex technical diagnosis within the MOLTKE system, done by Michael Richter to-gether with Klaus Dieter Althoff and others at the University of Kaiserslautern [4]. This lead to the PATDEX system [50], with Stefan Wess as the main developer, and later to several other systems and methods [5]. At IIIA in Blanes, Enric Plaza and Ramon López de Mántaras developed a case-based learning apprentice system for medical diag-nosis [46], and Beatrice Lopez investigated the use of case-based methods for strategy-level reasoning [39]. In Aberdeen, Derek Sleeman's group studied the use of cases for knowledge base refinement. An early result was the REFINER system, developed by Sunil Sharma [57]. Another result is the IULIAN system for theory revision by Ruediger Oehlman [44]. At the University of Trondheim, Agnar Aamodt and colleagues at Sintef studied the learning aspect of CBR in the context of knowl-edge acquisition in general, and knowledge main-tenance in particular. For problem solving, the combined use of cases and general domain knowl-edge was focused [1]. This lead to the development of the CREEK system and integration framework [2], and to continued work on knowledge-intensive case-based reasoning. On the cognitive science side, early work was done on analogical reasoning by Mark Keane, at Trinity College, Dublin, [29], a group that has developed into a strong environment for this type of CBR. In Gerhard Strube's group at the University of Freiburg, the role of episodic knowledge in cognitive models was investigated in the EVENTS project [66], which lead to the group"s current research profile of cognitive science and CBR.

Currently, the CBR activities in the United States as well as in Europe are spreading out (see, e.g. [8,19,20,28], and the rapidly growing number of papers on CBR in almost any AI journal). Germany seems to have taken a leading position in terms of number of active researchers, and several groups of significant size and activity level have been established recently. From Japan and other Asian countries, there are also activity points, for example in India [71]. In Japan, the interest is to a large ex-ent focused on the parallel computation approach to CBR [32].

## 3. Fundamentals of case-based reasoning methods

Central tasks that all case-based reasoning meth-ods have to deal with are to identify the current problem situation, find a past case similar to the new one, use that case to suggest a solution to the current problem, evaluate the proposed solution, and update the system by learning from this experi-ence. How this is done, what part of the process is focused, what type of problems drives the methods, etc. varies considerably, however. Below is an at-tempt to classify CBR methods into types with roughly similar properties in this respect.

### *3.1. Main types of CBR methods.*

The CBR paradigm covers a range of different methods for organizing, retrieving, utilizing and indexing the knowledge retained in past cases. Cases may be kept as concrete experiences, or a set of similar cases may form a generalized case. Cases may be stored as separate knowledge units, or splitted up into subunits and distributed within the knowledge structure. Cases may be indexed by a prefixed or open vocabulary, and within a flat or hierarchical index structure. The solution from a previous case may be directly applied to the present problem, or modified according to differ-ences between the two cases. The matching of cases, adaptation of solutions, and learning from an

experience may be guided and supported by a deep model of general domain knowledge, by more shallow and compiled knowledge, or be based on an apparent, syntactic similarity only. CBR meth-ods may be purely self-contained and automatic, or they may interact heavily with the user for support and guidance of its choices. Some CBR method assume a rather large amount of widely distributed cases in its case base, while others are based on a more limited set of typical ones. Past cases may be retrieved and evaluated sequentially or in parallel.

Actually, "case-based reasoning" is just one of a set of terms used to refer to systems of this kind. This has lead to some confusions, particularly since case-based reasoning is a term used both as a gen-eric term for several types of more specific ap-proaches, as well as for one such approach. To some extent, this can also be said for analogy rea-soning. An attempt of a clarification, although not resolving the confusions, of the terms related to case-based reasoning are given below.

*Exemplar-based reasoning.* The term is derived from a classification of different views to concept definition into "the classical view", "the proba-bilistic view", and "the exemplar view" (see [62]). In the exemplar view, a concept is defined *exten-sionally*, as the set of its exemplars. CBR methods that address the learning of concept definitions (i.e. the problem addressed by most of the research in machine learning), are sometimes referred to as exemplar-based. Examples are early papers by Kibler and Aha [31], and Bareiss and Porter [48]. In this approach, solving a problem is a *classifica-tion task*, i.e., finding the right class for the unclas-sified exemplar. The class of the most similar past case becomes the solution to the classification problem. The set of classes constitutes the set of possible solutions. Modification of a solution found is therefore outside the scope of this method.

*Instance-based reasoning.* This is a specializa-tion of exemplar-based reasoning into a highly *syn-tactic* CBR-approach. To compensate for lack of guidance from general background knowledge, a relatively large number of instances are needed in order to close in on a concept definition. The representation of the instances are usually simple (e.g. feature vectors), since the major focus is on studying *automated learning* with no user in the loop. Instance-based reasoning labels recent work by Kibler and Aha and colleagues [7], and serves to distinguish their methods from more knowledge-intensive exemplar-based approaches (e.g. Protos' methods). Basically, this is a non-generalization approach to the concept learning problem ad-dressed by classical, inductive machine learning methods.

*Memory-based reasoning.* This approach em-phasizes a collection of cases as a *large memory*, and reasoning as a process of accessing and searching in this memory. Memory organization and access is a focus of the case-based methods. The utilization of *parallel processing* techniques is a characteristic of these methods, and distinguishes this approach from the others. The access and stor-age methods may rely on purely syntactic criteria, as in the MBR-Talk system [63], or they may at-tempt to utilize general do-main knowledge, as the work done in Japan on massive parallel memories [32].

*Case-based reasoning.* Although case-based reasoning is used as a generic term in this paper, the *typical* case-based reasoning methods have some characteristics that distinguish them from the other approaches listed here. First, a typical case is usually assumed to have a certain degree of rich-ness of information contained in it, and a certain complexity with respect to its internal organization. That is, a feature vector holding some values and a corresponding class is not what we would call a typical case description. What we refer to as typi-cal case-based methods also has another charac-teristic property: They are able to modify, or adapt, a re-trieved solution when applied in a different problem solving context. Paradigmatic case-based methods also utilizes general background knowl-edge - although its richness, degree of explicit re-resentation, and role within the CBR processes varies. Core methods of typical CBR systems bor-row a lot from cognitive psychology theories.

*Analogy-based reasoning.* This term is some-times used, as a synonym to case-based reasoning, to describe the typical case-based approach just described [70]. However, it is also often used to characterize methods that solve new problems based on past cases from a *different domain*, while typical case-based methods focus on indexing and matching strategies for single-domain cases. Re-search on analogy reasoning is therefore a subfield concerned with mechanisms for identification and utilization of cross-domain analogies [23, 30]. The major focus of study has been on the *reuse* of a past case, what is called the mapping problem: Finding a way to transfer, or map, the solution of an identified analogue (called source or base) to the present problem (called target).

Throughout the paper we will continue to use the term case-based reasoning in the generic sense, although our examples, elaborations, and discussions will lean towards CBR in the more typical sense. The fact that a system is described as an example of some other approach, does not exclude it from being a typical CBR system as well. To the degree that more special examples of, e.g., instance-based, memory-based, or analogy-based methods will be discussed, this will be stated explicitly.

### *3.2. A descriptive framework.*

Our framework for describing CBR methods and systems has two main parts:

- A process model of the CBR cycle
- A task-method structure for case-based reasoning

The two models are complementary and represent two views on case-based reasoning. The first is a dynamic model that identifies the main subprocesses of a CBR cycle, their interdependencies and products. The second is a task-oriented view, where a task decomposition and related problem solving methods are described. The framework will be used in subsequent parts to identify and discuss important problem areas of CBR, and means of dealing with them.

### *3.3. The CBR cycle*

At the highest level of generality, a general CBR cycle may be described by the following four processes[4]:

1. RETRIEVE the most similar case or cases
2. REUSE the information and knowledge in that case to solve the problem
3. REVISE the proposed solution
4. RETAIN the parts of this experience likely to be useful for future problem solving

A new problem is solved by *retrieving* one or more previously experienced cases, *reusing* the case in one way or another, *revising* the solution based on reusing a previous case, and *retaining* the new experience by incorporating it into the exist-ing knowledge-base (case-base). The four proc-esses each involve a number of more specific steps, which will be described in the task model. In fFg. 1, this cycle is illustrated.

An initial description of a problem (top of Fig. 1) defines a *new case*. This new case is used to RETRIEVE a case from the collection of *previous cases*. The *retrieved case* is combined with the new case - through REUSE - into a *solved case*, i.e. a proposed solution to the initial problem. Through the REVISE process this solution is tested for success, e.g. by being applied to the real world environment or evaluated by a teacher, and repaired if failed. During RETAIN, useful experience is retained for future reuse, and the case base is updated by a new *learned case*, or by modification of some existing cases.

As indicated in the figure, general knowledge usually plays a part in this cycle, by supporting the CBR processes. This support may range from very weak (or none) to very strong, depending on the type of CBR method. By general knowledge we here mean general domain-dependent knowledge, as opposed to specific knowledge embodied by cases. For example, in diagnosing a patient by re-trieving and reusing the case of a previous patient, a model of anatomy together with causal relation-ships between pathological states may constitute the general knowledge used by a CBR system. A set of rules may have the same role.

### *3.4. A hierarchy of CBR tasks*

The process view just described was chosen in order to emphasize on CBR as a cycle of sequential steps. To further decompose and describe the four top-level steps, we switch to a task-oriented view, where each step, or subprocess, is viewed as a task that the CBR reasoner has to achieve. While a process-oriented view enables a global, external

[4]As a mnemonic, try "the four REs".

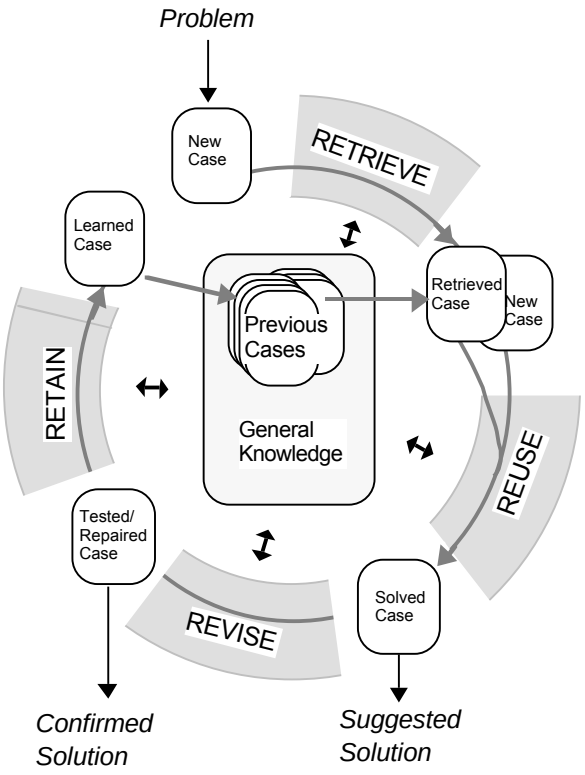



Fig. 1. The CBR Cycle

view to what is happening, a task oriented view is suitable for describing the detailed mechanisms from the perspective of the CBR reasoner itself. This is coherent with a task-oriented view of knowledge level modeling [73]. At the knowledge level, a system is viewed as an agent which has goals, and means to achieve its goals. A system de-scription can be made from three perspectives: Tasks, methods and domain knowledge models. Tasks are set up by the goals of the system, and a task is performed by applying one or more meth-ods. For a method to be able to accomplish a task, it needs knowledge about the general application do-main as well as information about the current prob-lem and its context. Our framework and analysis approach is strongly influenced by knowledge level modeling methods, particularly the Components of Expertise methodology [64.65].

The task-method structure we will refer to in subsequent parts of the paper is shown in Fig. 2. Tasks have node names in bold letters, while meth-ods are written in italics. The links between task nodes (plain lines) are *task decompositions*, i.e., part-of relations, where the direction of the rela-tionship is downwards. The top-level task is prob-lem solving and learning from experience and the method to accomplish the task is *case-based rea-soning* (indicated in a special way by a stippled ar-row). This splits the top-level task into the four major CBR tasks corresponding to the four proc-esses of Fig.1, retrieve, reuse, revise, and retain. All the four tasks are necessary in order to perform the top-level task. The retrieve task is, in turn, partitioned in the same manner (by a retrieval method) into the tasks identify (relevant descriptors), search (to find a set of past cases), initial match (the relevant descriptors to past cases), and select (the most similar case).

All task partitions in the figure are complete, i.e. the set of subtasks of a task are intended to be sufficient to accomplish the task, at this level of description. The figure does not show any control structure over the subtasks, although a rough sequencing of them is indicated by having put ear-lier subtasks higher up on the page than those that follow (for a particular set of subtasks). The actual control is specified as part of the problem solving method. The relation between tasks and methods (stippled lines) identify alternative methods ap-plicable for solving a task. A method specifies the algorithm that identifies and controls the execution of subtasks, and accesses and utilizes the knowl-edge and information needed to do this. The meth-ods shown are high level method classes, from which one or more specific methods should be cho-sen. The method set as shown is incomplete, i.e. one of the methods indicated may be sufficient to solve the task, several methods may be combined, or there may be other methods that can do the job. The methods shown in the figure are task decom-position and control methods. At the bottom level of the task hierarchy (not shown), a task is solved directly, i.e. by what may be referred to as task execution methods.

### *3.5. CBR Problem Areas*

As for AI in general, there are no universal CBR methods suitable for every domain of application. The challenge in CBR as elsewhere is to come up with methods that are suited for problem solving and learning in particular subject domains and for particular application environments. In line with the task model just shown, core problems addressed

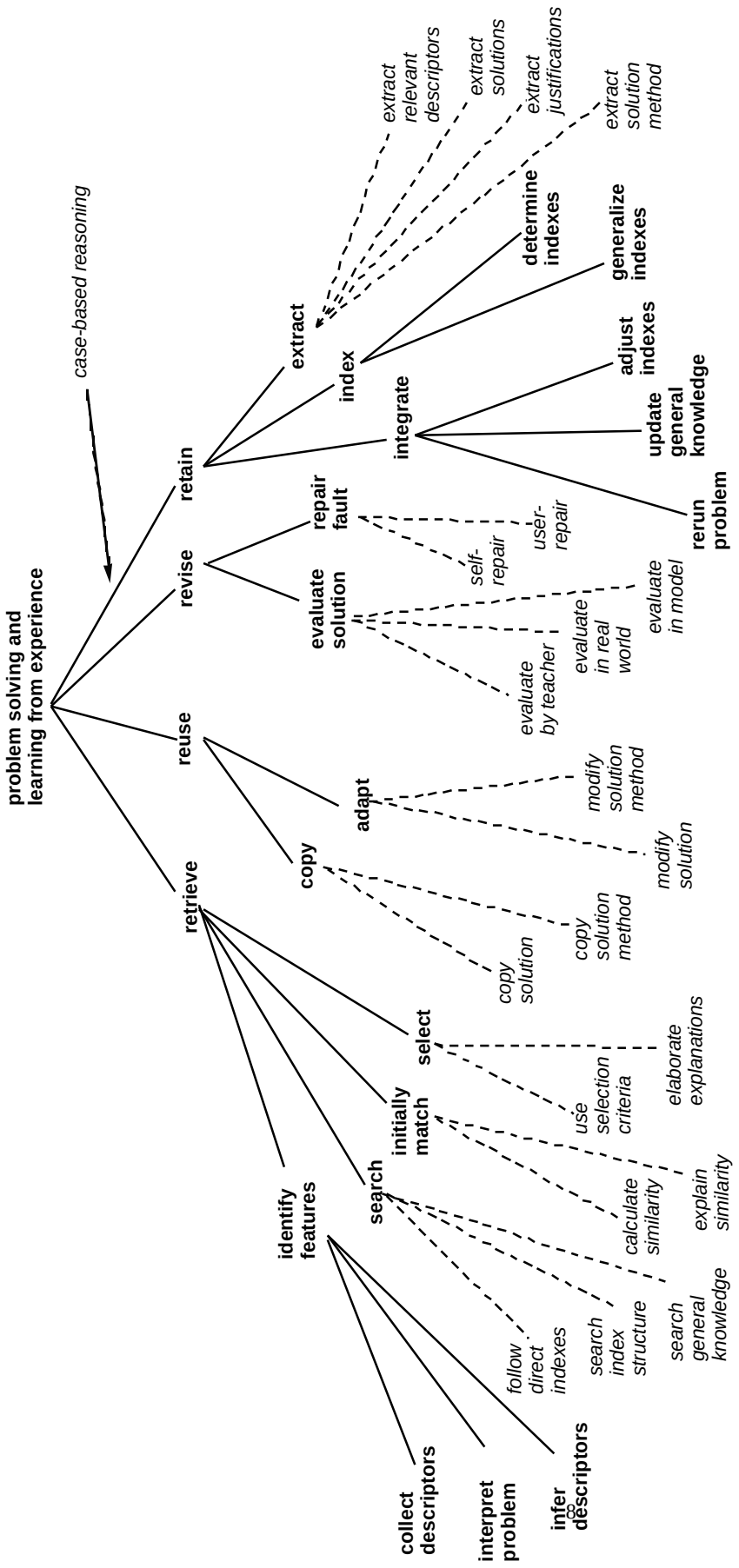



**Figure 2. A task-method decomposition of CBR**

by CBR research can be grouped into five areas. A set of coherent solutions to these problems constitutes a CBR method:

- *Knowledge representation*
- *Retrieval methods*
- *Reuse methods*
- *Revise methods*
- *Retain methods*

In the next five sections, we give an overview of the main problem issues related to these five areas, and exemplify how they are solved by some exist-ing methods. Our examples will be drawn from the six systems PROTOS, CHEF, CASEY, PATDEX, BOLERO, and CREEK. In the recently published book by Janet Kolodner [37] these problems are discussed and elaborated to substantial depth, and hints and guidelines on how to deal with them are given.

## 4. Representation of Cases

A case-based reasoner is heavily dependent on the structure and content of its collection of cases - often referred to as its case memory. Since a problem is solved by recalling a previous experience suitable for solving the new problem, the case search and matching processes need to be both effective and reasonably time efficient. Further, since the experience from a problem just solved has to be retained in some way, these requirements also apply to the method of integrating a new case into the memory. The representation problem in CBR is primarily the problem of deciding what to store in a case, finding an appropriate structure for describing case contents, and deciding how the case memory should be organized and indexed for effective retrieval and reuse. An additional problem is how to integrate the case memory structure into a model of general domain knowledge, to the extent that such knowledge is incorporated.

In the following subsection, two influential case memory models are briefly reviewed: The dynamic memory model of Schank and Kolodner, and the category-exemplar model of Porter and Bareiss[5].

[5]Other early models include Rissland and Ashley's HYPO system [52] in which cases are grouped under a set of domain-specific dimensions, and Stanfill and Waltz' MBR model, designed for parallel computation rather than knowledge-based matching.

### *4.1. The Dynamic Memory Model*

As previously mentioned, the first system that may be referred to as a case-based reasoner was Kolodner's CYRUS system, based on Schank's dynamic memory model [54]. The case memory in this model is a hierarchical structure of what is called 'episodic memory organization packets' (E-MOPs [33,34]), also referred to as generalized episodes [38]. This model was developed from Schank's more general MOP theory. The basic idea is to organize specific cases which share similar properties under a more general structure (a generalized episode - GE). A generalized episode contains three different types of objects: *Norms*, *cases* and *indices*. Norms are features common to all cases indexed under a GE. Indices are features which discriminate between a GE's cases. An index may point to a more specific generalized episode, or directly to a case. An index is composed of two terms: An index name and an index value.

Fig. 3 illustrates this structure. The figure illustrates a complex generalized episode, with its underlying cases and more specific GE. The entire case memory is a discrimination network where a node is either a generalized episode (containing the norms), an index name, index value or a case. Each index-value pair points from a generalized episode to another generalized episode or to a case. An index value may only point to a single case or a single generalized episode. The indexing scheme is redundant, since there are multiple paths to a particular case or GE. This is illustrated in the figure by the indexing of case1.

When a new case description is given and the best matching is searched for, the input case structure is 'pushed down' the network structure, starting at the root node. The search procedure is similar for case retrieval as for case storing. When one or more features of the case matches one or more features of a GE, the case is further discriminated based on its remaining features. Eventually, the case with most features in common with the input case is found[6]. During storing of a new case,

[6]This may not be the right similarity criterion, and is mentioned just to illustrate the method. Similarity criteria may favour matching of a particular subset of features, or there may be other means of assessing case similarity. Similarity assessment criteria can in turn can be used to guide the search - for example by identifying which indexes to follow first if there is a choice to be made.

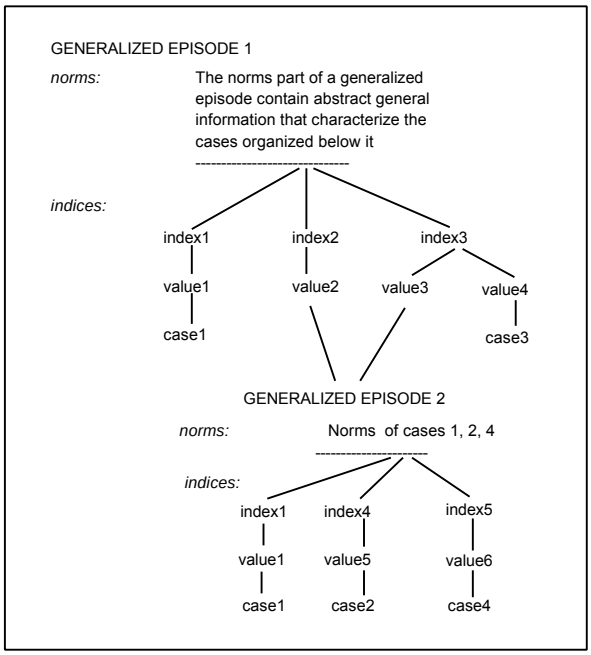



Fig. 3: Structure of cases and generalized episodes.

then discriminated by indexing them under different indices below this generalized episode. If - during the storage of a case - two cases (or two GEs) end up under the same index, a new generalized episode is automatically created. Hence, the memory structure is *dynamic* in the sense that similar parts of two case descriptions are dynamically generalized into a GE, and the cases are indexed under this GE by their difference features.

A case is retrieved by finding the GE with most norms in common with the problem description. Indices under that GE are then traversed in order to find the case which contains most of the additional problem features. Storing a new case is performed in the same way, with the additional process of dynamically creating generalized episodes, as described above. Since the index structure is a discrimination network, a case (or pointer to a case) is stored under each index that discriminates it from other cases. This may easily lead to an explosive growth of indices with increased number of cases. Most systems using this indexing scheme therefore put some limits to the choice of indices for the cases. In CYRUS, for example, only a small vocabulary of indices is permitted.

CASEY stores a large amount of information in its cases. In addition to all observed features, it retains the causal explanation for the diagnosis found, as well as the list of states in the heart failure model for which there was evidence in the patient. These states, referred to as generalized causal states, are also the primary indices to the cases.

The primary role of a generalized episode is as an indexing structure for matching and retrieval of cases. The dynamic properties of this memory organization, however, may also be viewed as an attempt to build a memory structure which integrates knowledge from specific episodes with knowledge generalized from the same episodes. It is therefore claimed that this knowledge organization structure is suitable for learning generalized knowledge as well as case specific knowledge, and that it is a plausible - although simplified - model of human reasoning and learning.

### *4.2. The category & exemplar model*

The PROTOS system, built by Ray Bareiss and Bruce Porter [9,49], proposes an alternative way to organize cases in a case memory. Cases are also referred to as *exemplars*. The psychological and philosophical basis of this method is the view that 'real world', natural concepts should be defined extensionally. Further, different features are assigned different importances in describing a case's membership to a category. Any attempt to generalize a set of cases should - if attempted at all - be done very cautiously. This fundamental view of concept representation forms the basis for this memory model.

The case memory is embedded in a network structure of *categories*, *semantic relations*,*cases*, and *index pointers*. Each case is associated with a category. An index may point to a case or a category. The indices are of three kinds: Feature links pointing from problem descriptors (features) to cases or categories (called remindings), case links pointing from categories to its associated cases (called exemplar links), and difference links pointing from cases to the neighbour cases that only differs in one or a small number of features. A feature is, generally, described by a name and a value. A category's exemplars are sorted according

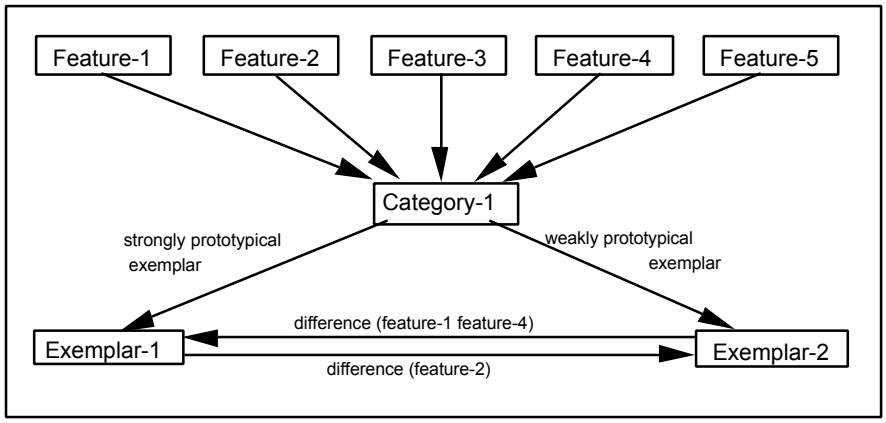



Fig. 4: The Structure of Categories, Features and Exemplars

to their degree of prototypicality in the category.

Fig. 4 illustrates a part of this memory structure, i.e. the linking of features and cases (exemplars) to categories. The unnamed indices are remindings from features to a category.

Within this memory organization, the categories are inter-linked within a semantic network, which also contains the features and intermediate states (e.g. subclasses of goal concepts) referred to by other terms. This network represents a background of general domain knowledge, which enables explanatory support to some of the CBR tasks. For example, a core mechanism of case matching is a method called 'knowledge-based pattern matching'.

Finding a case in the case base that matches an input description is done by combining the input features of a problem case into a pointer to the case or category that shares most of the features. If a reminding points directly to a category, the links to its most prototypical cases are traversed, and these cases are returned. As indicated above, general domain knowledge is used to enable matching of features that are semantically similar. A new case is stored by searching for a matching case, and by establishing the appropriate feature indices. If a case is found with only minor differences to the input case, the new case may not be retained or the two cases may be merged by following taxonomic links in the semantic network.

## 5. Case Retrieval

The Retrieve task starts with a (partial) problem description, and ends when a best matching previous case has been found. Its subtasks are referred to as Identify Features, Initially Match, Search, and Select, executed in that order. The identification task basically comes up with a set of relevant problem descriptors, the goal of the matching task is to return a set of cases that are sufficiently similar to the new case - given a similarity threshold of some kind, and the selection task works on this set of cases and chooses the best match (or at least a first case to try out).

While some case-based approaches retrieve a previous case largely based on superficial, *syntactical similarities* among problem descriptors (e.g. the CYRUS system [33], ARC [46], and PATDEX-1 [50] systems), some approaches attempt to retrieve cases based on features that have deeper, *semantical similarities* (e.g. the PROTOS [9], CASEY [38], GREBE [12], CREEK [3], and MMA [47] systems). Ongoing work in the FABEL project, aimed to develop a decision support system for architects, explores various methods for combined reasoning and mutual support of different knowledge types [21]. In order to match cases based on semantic similarities and relative importance of features, an extensive body of general domain knowledge is needed to produce an explanation of why two cases match and how strong the match is. Syntactic similarity assessment - sometimes referred to as a "knowledge-poor" approach - has its advantage in domains where general domain knowledge is very difficult or impossible to acquire. On the other hand, semantical oriented approaches - referred to as "knowledge-intensive"[7] - are able to use the contextual

[7]Note that syntactic oriented methods may also contain a lot of general domain knowledge, implicit in their matching methods. The distinction between knowledge-poor and knowledge-intensive is therefore related to *explicitly represented* domain knowledge. Further, it refers to *generalized* domain knowledge, since cases also contain explicit knowledge, but this

meaning of a problem description in its matching, for domains where general domain knowledge is available.

A question that should be asked when deciding on a retrieval strategy, is the purpose of the retrieval task. If the purpose is to retrieve a case which is to be adapted for reuse, this can be accounted for in the retrieval method. Approaches to 'retrieval for adaptation' have for example been suggested for retrieval of cases for design problem solving [15], and for analogy reasoning [17,45].

### *5.1. Identify Feature*

To identify a problem may involve simply noticing its input descriptors, but often - and particularly for knowledge-intensive methods - a more elaborate approach is taken, in which an attempt is made to 'understand' the problem within its context. Unknown descriptors may be disregarded or requested to be explained by the user. In PROTOS, for example, if an input feature is unknown to the system, the user is asked to supply an explanation that links the feature into the existing semantic network (category structure). To understand a problem involves to filter out noisy problem descriptors, to infer other relevant problem features, to check whether the feature values make sense within the context, to generate expectations of other features, etc. Other descriptors than those given as input, may be inferred by using a general knowledge model, or by retrieving a similar problem description from the case base and use features of that case as expected features. Checking of expectations may be done within the knowledge model (cases and general knowledge), or by asking the user.

### *5.2. Initially Match*

The task of finding a good match is typically split into two subtasks: An initial matching process which retrieves a set of plausible candidates, and a more elaborate process of selecting the best one among these. The latter is the Select task, described below. Finding a set of matching cases is done by using the problem descriptors (input features) as indexes to the case memory in a direct or indirect way. There are in principle three ways of retrieving a case or a set of cases: By following direct index pointers from problem features, by searching an index structure, or by searching in a model of general domain knowledge. PATDEX implements the first strategy for its diagnostic reasoning, and the second for test selection. A domain-dependent, but global similarity metric is used to assess similarity based on surface match. Dynamic memory based systems takes the second approach, but general domain knowledge may be used in combination with search in the discrimination network. PROTOS and CREEK combines one and three, since direct pointers are used to hypothesize a candidate set which in turn is justified as plausible matches by use of general knowledge.

Cases may be retrieved solely from input features, or also from features inferred from the input. Cases that match all input features are, of course, good candidates for matching, but - depending on the strategy - cases that match a given fraction of the problem features (input or inferred) may also be retrieved. PATDEX uses a global similarity metric, with several parameters that are set as part of the domain analysis. Some tests for relevance of a retrieved case is often executed, particularly if cases are retrieved on the basis of a subset of features. For example, a simple relevance test may be to check if a retrieved solution conforms with the expected solution type of the new problem. A way to assess the degree of similarity is needed, and several 'similarity metrics' have been proposed, based on surface similarities of problem and case features.

Similarity assessment may also be more knowledge-intensive, for example by trying to understand the problem more deeply, and using the goals, constraints, etc. from this elaboration process to guide the matching [3]. Another option is to weigh the problem descriptors according to their importance for characterizing the problem, during the learning phase. In PROTOS, for example, each feature in a stored case has assigned to it a degree of importance for the solution of the case. A similar mechanism is adopted by CREEK, which stores both the predictive strength (discriminatory value) of a feature with respect to the set of cases, as well as a feature's criticality, i.e. what influence the lack of a feature has on the case solution.

### *5.3. Select*

From the set of similar cases, a best match is chosen. This may have been done during the initial match process, but more often a set of cases are returned from that task. The best matching case is usually determined by evaluating the degree of initial match more closely. This is done by an attempt to generate explanations to justify non-identical features,

is *specialized* (specific) domain knowledge.

based on the knowledge in the semantic network. If a match turns out not to be strong enough, an attempt to find a better match by following difference links to closely related cases is made. This subtask is usually a more elaborate one than the retrieval task, although the distinction between retrieval and elaborate matching is not distinct in all systems. The selection process typically generate consequences and expectations from each retrieved case, and attempts to evaluate consequences and justify expectations. This may be done by using the system's own model of general domain knowledge, or by asking the user for confirmation and additional information. The cases are eventually ranked according to some metric or ranking criteria. Knowledge-intensive selection methods typically generate explanations that support this ranking process, and the case that has the strongest explanation for being similar to the new problem is chosen. Other properties of a case that are considered in some CBR systems include relative importance and discriminatory strengths of features, prototypicality of a case within its assigned class, and difference links to related cases.

## 6. Case Reuse

The reuse of the retrieved case solution in the context of the new case focuses on two aspects: (a) the differences among the past and the current case and (b) what part of retrieved case can be transferred to the new case. The possible subtasks of Reuse are Copy and Adapt.

### *6.1. Copy*

In simple classification tasks the differences are abstracted away (they are considered non relevant while similarities are relevant) and the solution class of the retrieved case is transferred to the new case as its solution class. This is a trivial type of reuse. However, other systems have to take into account differences in (a) and thus the reused part (b) cannot be directly transferred to the new case but requires an *adaptation* process that takes into account those differences.

### *6.2. Adapt*

There are two main ways to reuse past cases[8]: (1) reuse the past case solution (transformational reuse), and (2) reuse the past method that constructed the solution (derivational reuse). In transformational reuse the past case solution is not directly a solution for the new case but there exists some knowledge in the form of transformational operators {T} such that applied to the old solution they transform it into a solution for the new case. A way to organize this T operators is to index them around the differences detected among the retrieved and current cases. An example of this is CASEY, where a new causal explanation is built from the old causal explanations by rules with condition-part indexing differences and with a transformational operator T at the action part of the rule. Transformational reuse does not look how a problem is solved but focuses on the equivalence of solutions, and this is one requires a strong domain-dependent model in the form of transformational operators {T} plus a control regime to organize the operators application.

Derivational reuse looks at how the problem was solved in the retrieved case. The retrieved case holds information about the method used for solving the retrieved problem including a justification of the operators used, subgoals considered, alternatives generated, failed search paths, etc. Derivational reuse then reinstantiates the retrieved method to the new case and "replays" the old plan into the new context (usually general problem solving systems are seen here as planning systems). During the replay successful alternatives, operators, and paths will be explored first while filed paths will be avoided; new subgoals are pursued based on the old ones and old subplans can be recursively retrieved for them. An example of derivational reuse is the Analogy/Prodigy system [70] that reuses past plans guided by commonalties of goals and initial situations, and resumes a means-ends planning regime if the retrieved plan fails or is not found.

## 7. Case Revision

When a case solution generated by the reuse phase is not correct, an opportunity for learning from failure arises. This phase is called case revision and consists of two tasks: (1) evaluate the case solution generated

[8]We here adapt the distinction between transformational and derivational analogy, put forth in [16].

by reuse. If successful, learn from the success (case retainment, see next section), (2) otherwise repair the case solution using domain-specific knowledge or user input.

*7.1. Evaluate solution*

The evaluation task takes the result from applying the solution in the real environment (asking a teacher or performing the task in the real world). This is usually a step outside the CBR system, since it - at least for a system in normal operation - involves the application of a suggested solution to the real problem. The results from applying the solution may take some time to appear, depending on the type of application. In a medical decision support system, the success or failure of a treatment may take from a few hours up to several months. The case may still be learned, and be available in the case base in the intermediate period, but it has to be marked as a non-evaluated case. A solution may also be applied to a simulation program that is able to generate a correct solution. This is used in CHEF, where a solution (i.e. a cooking recipe) is applied to an internal model assumed to be strong enough to give the necessary feedback for solution repair.

*7.2. Repair fault*

Case repair involves detecting the errors of the current solution and retrieving or generating explanations for them. The best example is the CHEF system, where causal knowledge is used to generate an explanation of why certain goals of the solution plan were not achieved. CHEF learns the general situations that will cause the failures using an explanation-based learning technique. This is included into a failure memory that is used in the reuse phase to predict possible shortcomings of plans. This form of learning moves detection of errors in a pot hoc fashion to the elaboration plan phase were errors can be predicted, handled and avoided. A second step of the revision phase uses the failure explanations to modify the solution in such a way that failures do not occur. For instance, the failed plan in the CHEF system is modified by a repair module that adds steps to the plan that will assure that the causes of the errors will not occur. The repair module possesses general causal knowledge and domain knowledge about how to disable or compensate causes of errors in the domain. The revised plan can then be retained directly (if the revision phase assures its correctness) or it can be evaluated and repaired again.

## 8. Case Retainment - Learning

This is the process of incorporating what is useful to retain from the new problem solving episode into the existing knowledge. The learning from success or failure of the proposed solution is triggered by the outcome of the evaluation and possible repair. It involves selecting which information from the case to retain, in what form to retain it, how to index the case for later retrieval from similar problems, and how to integrate the new case in the memory structure.

*8.1. Extract*

In CBR the case base is updated no matter how the problem was solved. If it was solved by use of a previous case, a new case may be built or the old case may be generalized to subsume the present case as well. If the problem was solved by other methods, including asking the user, an entirely new case will have to be constructed. In any case, a decision need to be made about what to use as the source of learning. Relevant problem descriptors and problem solutions are obvious candidates. But an explanation or another form of justification of why a solution is a solution to the problem may also be marked for inclusion in a new case. In CASEY and CREEK, for example, explanations are included in retained cases, and reused in later modification of the solution. CASEY uses the previous explanation structure to search for other states in the diagnostic model which explains the input data of the new case, and to look for causes of these states as answers to the new problem. This focuses and speeds up the explanation process, compared to a search in the entire domain model. The last type of structure that may be extracted for learning is the problem solving method, i.e. the strategic reasoning path, making the system suitable for derivational reuse.

Failures, i.e. information from the Revise task, may also be extracted and retained, either as separate failure cases or within total-problem cases. When a failure is encountered, the system can then get a reminding to a previous similar failure, and use the failure case to improve its understanding of - and correct - the present failure.

*8.2. Index*

The 'indexing problem' is a central and much focused problem in case-based reasoning. It amounts to deciding what type of indexes to use for future retrieval, and how to structure the search space of indexes. Direct indexes, as previously mentioned, skips the latter step, but there is still the problem of identifying what type of indexes to use. This is actually a knowledge acquisition problem, and should be analyzed as part of the domain knowledge analysis and modeling step. A trivial solution to the problem is of course to use all input features as indices. This is the approach of syntax-based methods within instance-based and memory-based reasoning. In the memory-based method of CBR-Talk [63], for example, relevant features are determined by matching, in parallel, all cases in the case-base, and filtering out features that belong to cases with few features in common with the problem case.

In CASEY, a two-step indexing method is used. Primary index features are - as referred to in the section on representation - general causal states in the heart failure model that are part of the explanation of the case. When a new problem enters, the features are propagated in the heart failure model, and the states that explain the features are used as indices to the case memory. The observed features themselves are used as secondary features only.

### *8.3. Integrate*

This is the final step of updating the knowledge base with new case knowledge. If no new case and index set has been constructed, it is the main step of Retain. By modifying the indexing of existing cases, CBR systems learn to become better similarity assessors. The tuning of existing indexes is an important part of CBR learning. Index strengths or importances for a particular case or solution are adjusted due to the success or failure of using the case to solve the input problem. For features that have been judged relevant for retrieving a successful case, the association with the case is strengthened, while it is weakened for features that lead to unsuccessful cases being retrieved. In this way, the index structure has a role of tuning and adapting the case memory to its use. PATDEX has a special way to learn feature relevance: A relevance matrix links possible features to the diagnosis for which they are relevant, and assign a weight to each such link. The weights are updated, based on feedback of success or failure, by a connectionist method.

In knowledge-intensive approaches to CBR, learning may also take place within the general conceptual knowledge model, for example by other machine learning methods (see next section) or through interaction with the user. Thus, with a proper interface to the user (whether a competent end user or an expert) a system may incrementally extend and refine its general knowledge model, as well as its memory of past cases, in the normal course of problem solving. This is an inherent method in the PROTOS system, for example. All general knowledge in PROTOS is assumed to be acquired in such a bottom-up interaction with a competent user.

The case just learned may finally be tested by re-entering the initial problem and see whether the system behaves as wanted.

## 9. Integrated approaches

Most CBR systems make use of general domain knowledge in addition to knowledge represented by cases. Representation and use of that domain knowledge involves integration of the case-based method with other methods and representations of problem solving, for instance rule-based systems or deep models like causal reasoning. The overall architecture of the CBR *system* has to determine the interactions and control regime between the CBR *method* and the other components. Note that the type of integration we address here involves case based reasoning as a core part of the target system's reasoning method, and does not include case-oriented approaches for acquiring general domain knowledge (e.g. [74]). For instance, the CASEY system integrates a model-based causal reasoning program to diagnose heart diseases. When the case-based method fails to provide a correct solution CASEY executes the model-based method to solve the problem and stores the solution as a new case for future use. Since the model-based method is complex and slow, the case-based method in CASEY is essentially a way to achieve speed-up learning. The integration of model-based reasoning is also important for the case-based method itself: the causal model of the disease of a case is what is retrieved and reused in CASEY.

An example of integrating rules and cases is the BOLERO system [40]. BOLERO is a meta-level architecture where the base-level is composed of rules embodying knowledge to diagnose the plausible pneumonias of a patient, while the meta-level is a case-based planner that, at every moment, is able to dictate which diagnoses are worthwhile to consider.

Thus in BOLERO the rule-based level contains domain knowledge (how to deduce plausible diagnosis from patient facts) while the meta-level contains strategic knowledge (it plans, from all possible goals, which are likely to be successfully achieved). The case-based planner is therefore used to control the space searched by the rule-based level, achieving a form of speed-up learning. The control regime between the two components is interesting: the control passes to the meta-level whenever some new information is known at the base level, assuring that the system is dynamically able to generate a more appropriate strategic plan. This control regime in the meta-level architecture assures that the case-based planner is capable of *reactive behaviour*, i.e. of modifying plans reacting to situation changes. Also the clear separation of rule-based and case-based methods in two different levels of computation is important: it clarifies their distinction and their interaction.

The integration of CBR with other reasoning paradigms is closely related to the general issue of architectures for unified problem solving and learning. These approach is a current trend in machine learning with architectures such as Soar, Theo, or Prodigy. CBR as such is a form of combining problem solving (through retrieval and reuse) and learning (through retainment). However, as we have seen, other forms of representation and reasoning are usually integrated into a CBR system and thus the general issue is an important dimension into CBR research. In the CREEK architecture, the cases, heuristic rules, and deep models are integrated into a unified knowledge structure. The main role of the general knowledge is to provide explanatory support to the case-based processes [3], rules or deep models may also be used to solve problems on their own if the case-based method fails. Usually the domain knowledge used in a CBR system is acquired through knowledge acquisition in the normal way for knowledge-based systems. Another option would be to also learn that knowledge from the cases. In this situation it can be learnt in a case-based way or by induction. This line of work is currently being developed in Europe by systems like the Massive Memory Architecture and INRECA [41]. These systems are closely related to the *multistrategy learning systems* [42]: the issues of integrating different problem solving and learning methods are essential to them.

The Massive Memory Architecture (MMA) [47] is an integrated architecture for learning and problem solving based on reuse of case experiences retained in the systems memory. A goal of MMA is understanding and implementing the relationship between learning and problem solving into a reflective or introspective framework: the system is able to inspect its own past behaviour in order to learn how to change its structure so as to improve is future performance. Case-based reasoning methods are implemented by retrieval methods (to retrieve past cases), a language of preferences (to select the best case) and a form of derivational analogy (to reuse the retrieved method into the current problem). A problem in the MMA does not use one CBR method, since several CBR methods can be programmed for different subgoals by means of specific retrieval methods and domain-dependent preferences. Learning in MMA is viewed as a form of introspective inference, where the reasoning is not about a domain but about the past behaviour of the system and about ways to modify and improve this behaviour. This view supports integration of case-based learning as well as of other forms of learning from examples, like inductive methods, which are also integrated into the MMA and combined with case-based methods.

## 10. Example applications and tools

As a relatively young field, CBR can not yet brag about a lot of fielded applications. But there are some. We briefly summarize two of these systems, to illustrate how CBR methods can successfully realize knowledge-based decision support systems.

### *10.1. Two Fielded Applications*

At Lockheed, Palo Alto, the first fielded CBR system was developed. The problem domain is optimization of autoclave loading for heat treatment of composite materials [26]. The autoclave is a large convection oven, where airplane parts are treated in order to get the right properties. Different material types need different heating and cooling procedures, and the task is to load the autoclave for optimized throughput, i.e. to select the parts that can be treated together, and distribute them in the oven so that their required heating profiles are taken care of. There are always more parts to be cured than the autoclave can take in one load. The knowledge needed to perform this task reasonably well used to reside in the head of a just a few experienced people. There is no theory and very few generally applicable schemes for doing this job, so to build up experience in the form of previously

successful and unsuccessful situations is important. The motivation for developing this application was to be able to remember the relevant earlier situations. Further, a decision support system would enable other people than the experts to do the job, and to help training new personnel. The development of the system started in 1987, and it has been in regular use since the fall 1990. The results so far are very positive. The current system handles the configuration of one loading operation in isolation, and an extended system to handle the sequencing of several loads is under testing. The development strategy of the application has been to hold a low-risk profile, and to include more advanced functionalities and solutions as experience with the system has been gained over some time.

The second application has been developed at General Dynamics, Electric Boat Division [13]. During construction of ships, a frequently re-occurring problem is the selection of the most appropriate mechanical equipment, and to fit it to its use. Most of these problems can be handled by fairly standard procedures, but some problems are harder and occur less frequently. These type of problems - referred to as "non-conformances" - also repeat over time, and because regular procedures are missing, they consume a lot of resources to get solved . General Dynamics wanted to see whether a knowledge-based decision support tool could reduce the cost of these problems. The application domain chosen was the selection and adjustment of valves for on-board pipeline systems. The development of the first system started in 1986, using a rule-based systems approach. The testing of the system on real problems initially gave positive results, but problems of brittleness and knowledge maintenance soon became apparent. In 1988 a feasibility study was made of the use of case-based reasoning methods instead of rules, and a prototype CBR system was developed. The tests gave optimistic results, and an operational system was fielded in 1990. The rule-base was taken advantage of in structuring the case knowledge and filling the initial case base. IN the fall of 1991 the system was continually used in three out of four departments involved with mechanical construction. A quantitative estimate of cost reductions has been made: The rule-based system took 5 man-years to develop, and the same for the CBR system (2 man-years of studies and experimental development and 3 man-years for the prototype and operational system). This amounts to $750.000 in total costs. In the period December 90 - September 91 20.000 non-conformances were handled. The cost reduction, compared to previous costs of manual procedures, was about 10%, which amounts to a saving of $240.000 in less than one year.

There are many any other applications in test use or more or less regular use. A rapidly growing application type is "help desk systems" [36,58], where basically case-based indexing and retrieval methods are used to retrieve cases, which then are viewed as information chunks for the user, instead of sources of knowledge for reasoning[9]. Such a system can also be a first step towards a more full-fledged CBR system.

### *10.2. Tools*

Several commercial companies offer shells for building CBR systems. Just as for rule-based systems shells, they enable you to quickly develop applications, but at the expense of flexibility of representation, reasoning approach and learning methods. In [25] Paul Harmon reviewed four such shells: ReMind from Cognitive Systems Inc., CBR Express/ART-IM from Inference Corporation, Esteem from Esteem Software Inc., and Induce-it (later renamed to CasePower) from Inductive Solutions, Inc. The first three of these were reviewed more thoroughly by Thomas Schult for the German AI journal [56]. The example of CBR Express and ART-IM is typical, since many vendors offer CBR extensions to an existing tool. On the European scene Acknosoft in Paris offers the shell KATE-CBR as part of their CaseCraft Toolbox, Isoft, also in Paris, has a shell called ReCall. TechInno in Kaiserslautern has S3-Case, a PATDEX-derived tool that is part of their S3 environment for technical systems maintenance.

As an example of functionality, the ReMind shell offers an interactive environment for acquisition of cases, domain vocabulary, indexes and prototypes. The user may define hierarchical relations among attributes and a similarity measure based on them. Indexing is done inductively by building a decision tree and allowing the user to graphically edit the importance of attributes. Several retrieval methods are supported: (1) inductive retrieval matching the most specific prototype in a prototype hierarchy, (2) nearest neighbour retrieval, and (3) SQL-like template retrieval. Case adaptation is based on formulas that adjust values based on retrieved vs. new case

[9]A gradual transition from such a system, starting with some help desk and information filtering [74] functions, and moving to an advice-giving case-based decision support tool, is the approach taken in a system currently being specified for the Norwegian oil industry [43].

differences. ReMind also has the capability of representing causal relationships using a qualitative model. The first commercial products appeared in 1991 including Help-Desk systems, technical diagnosis, classification and prediction, control and monitoring, planning, and design applications. ReMind is a trade mark of Cognitive Systems Inc. and was developed with the DARPA support.

ReCall is a CBR system trademark of ISoft, a Paris based AI company, and applications include help desk systems, fault diagnosis, bank loan analysis, control and monitoring. Retrieval methods are a combination of methods (1) and (2) in ReMind, but offer standard adaptation mechanisms such as vote and analogy, and a library of adaptation methods.

The KATE-CBR tool, named CaseWork, integrates an instance-based CBR approach within a tool for inductive learning of, and problem solving from, decision trees. The inductive and case-based methods can be used separately, or integrated into a single combined method. There are editor facilities to graphically build parts of the case/index structure, and to generate user dialogues. The tool has incorporated initial results on integration of case-based and inductive methods from the INRECA project [41].

Some academic CBR tools are freely available, e.g. by anonymous ftp, or via contacting the developers.[10]

## 11. Conclusions and Future trends

Summarizing the paper, we can say that case-based reasoning (CBR) puts forward a paradigmatic way to attack AI issues, namely problem solving, learning, usage of general and specific knowledge, combining different reasoning methods, etc. In particular we have seen that CBR emphasizes problem solving and learning as two sides of the same coin: problem solving uses the results of past learning episodes while problem solving provides the backbone of the experience from which learning advances. The current state of the art in Europe regarding CBR is characterized by a strong influence of the USA ideas and CBR systems, although Europe is catching up and provides a somewhat different approach to CBR, particularly in its many activities related to integration of CBR and other approaches and by its movement toward the development of application-oriented CBR systems.

The development trends of CBR *methods* can be grouped around four main topics: Integration with other learning methods, integration with other reasoning components, incorporation into massive parallel processing, and method advances by focusing on new cognitive aspects. The first trend, integration of other learning methods into CBR, forms part of the current trend in ML research toward *multistrategy learning* systems. This research aims at achieving an integration of different learning methods (for instance case-based learning and induction as is done in the MMA and INRECA systems) into a coherent framework, where each learning method fulfills a specific and distinct role in the system. The second trend, integration of several reasoning methods aims at using the different sources of knowledge in a more thorough, principled way, like what is done in the CASEY system with the use of causal knowledge. This trend emphasizes the increasing importance of knowledge acquisition issues and techniques in the development of knowledge-intensive CBR systems, and the European Workshop on CBR showed a strong European commitment towards the utilization of knowledge level modeling in CBR systems design.

The massive memory parallelism trend applies case-based reasoning to domains suitable for shallow, instance-based retrieval methods on a very large amount of data. This direction may also benefit from integration with neural network methods, as several Japanese projects currently are investigating [32]. By the fourth trend, method advances from focusing on the cognitive aspects, what we particularly have in mind is the follow-up of work initiated on creativity (e.g. [55]) as a new focus for CBR methods. It is not just an 'application type', but a way to view CBR in general, which may have significant impact on our methods.

The trends of CBR *applications* are clearly that we initially will see a lot of help desk applications around. This type of systems may open up for a more general coupling of CBR - and AI in general - to information systems. The use of cases for human browsing and decision making, is also likely to lead to increased interest in intelligent computer-aided learning, training, and teaching. The strong role of user interaction, of flexible user control, and the drive towards total interactiveness of systems (of 'situatedness', if you like) favours a case-based approach to intelligent computer assistance, since CBR systems are able to continually learn from, and evolve through, the capturing and retainment of past

[10] Protos, for example, is available from the University of Texas, and code for implementing a simple version of dynamic memory, as described in [51], is available from the Institute of Learning Sciences at Northwestern University.

experiences.

Case-based reasoning has blown a fresh wind and a well justified degree of optimism into AI in general and knowledge based decision support systems in particular. The growing amount of ongoing CBR research - within an AI community that has learned from its previous experiences - has the potential of leading to significant breakthroughs of AI methods and applications.

## Acknowledgements

We thank Stefan Wess, Ramon Lopez de Mantaras, and Pinar Ozturk for comments on drafts of this paper.

## References

[1] Aamodt, A., (1989) Towards robust expert systems that learn from experience - an architectural framework. In John Boose, Brian Gaines, Jean-Gabriel Ganascia (eds.): *EKAW-89; Third European Knowledge Acquisition for Knowledge-Based Systems Workshop* , Paris, July 1989. pp 311-326.

[2] Aamodt, A. (1991). *A knowledge-intensive approach to problem solving and sustained learning,* Ph.D. dissertation, University of Trondheim, Norwegian Institute of Technology, May 1991. (University Microfilms PUB 92-08460)

[3] Aamodt, A. (1993) Explanation-driven retrieval, reuse, and learning of cases, In *EWCBR-93: First European Workshop on Case-Based Reasoning*. University of Kaiserslautern SEKI Report SR-93-12 (SFB 314) (Kaiserslautern, Germany, 1993) 279-284.

[4] Althoff, K.D (1989). Knowledge acquisition in the domain of CNC machine centers; the MOLTKE approach. In John Boose, Brian Gaines, Jean-Gabriel Ganascia (eds.): *EKAW-89; Third European Workshop on Knowledge-Based Systems,* Paris, July 1989. pp 180-195.

[5] Althoff, K-D. (1992). Machine learning and knowledge acquisition in a computational architecture for fault diagnosis in engineering systems. *Proceedings of the ML-92 Workshop on Computational Architectures for Machine Learning and Knowledge Acquisition*. Aberdeen, Scotland, July 1992.

[6] Anderson, J. R., (1983). *The architecture of cognition*. Harvard University Press, Cambridge.

[7] Aha, D. , Kibler, D. , and Albert, M. K. (1991). Instance-Based Learning Algorithms. *Machine Learning,* vol.6 (1).

[8] Allemang, Dean (1994) Review of EWCBR-93, *AI Communication*, this issue.

[9] Bareiss, R. (1989). *Exemplar-based knowledge acquisition: A unified approach to concept representation, classification, and learning.* Boston, Academic Press.

[10] K. Ashley (1991). *Modeling legal arguments: Reasoning with cases and hypotheticals.* MIT Press, Bradford Books, Cambridge.

[11] Bareiss, R. (1988): PROTOS; a unified approach to concept representation, classification and learning. Ph.D. Dissertation, University of Texas at Austin, Dep. of Computer Sciences 1988. Technical Report AI88-83.

[12] Branting, K. (1991): Exploiting the complementarity of rules and precedents with reciprocity and fairness. In: *Proceedings from the Case-Based Reasoning Workshop 1991,* Washington DC, May 1991. Sponsored by DARPA. Morgan Kaufmann, 1991. pp 39-50.

[13] Brown, B. and Lewis, L. (1991): A case-based reasoning solution to the problem of redundant resolutions of non-conformances in large scale manufacturing. In: R. Smith, C. Scott (eds.): *Innovative Applications for Artificial Intelligence 3*. MIT Press.

[14] Burstein, M.H. (1989): Analogy vs. CBR; The purpose of mapping. Proceedings from the Case-Based Reasoning Workshop, Pensacola Beach, Florida, May-June 1989. Sponsored by DARPA. Morgan Kaufmann. pp 133-136.

[15] Börner, K. (1993): Structural similarity as guidance in case-based design. In: *First European Workshop on Case-based Reasoning, Posters and Presentations,* 1-5 November 1993. Vol. I. University of Kaiserslautern, pp. 14-19.

[16] Carbonell, J. (1986): Derivational analogy; A theory of reconstructive problem solving and expertise acquisition. In R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds.): *Machine Learning - An artificial Intelligence Approach, Vol.II,* Morgan Kaufmann, pp. 371-392.

[17] Cunningham, P. and Slattery, S. (1993): Knowledge enigneering requirements in derivational analogy. In: *First European Workshop on Case-based Reasoning, Posters and Presentations,* 1-5 November 1993. Vol. I. University of Kaiserslautern, pp. 108-113.

[18] *Proceedings from the Case-Based Reasoning Workshop*, Pensacola Beach, Florida, May-June 1989. Sponsored by DARPA. Morgan Kaufmann.

[19] *Proceedings from the Case-Based Reasoning Workshop*, Washington D.C., May 8-10, 1991. Sponsored by DARPA. Morgan Kaufmann.

[20] *First European Workshop on Case-based Reasoning, Posters and Presentations,* 1-5 November 1993. Vol. I-II. University of Kaiserslautern.

[21] The FABEL Consortium (1993): *Survey of FABEL*. FABEL Report No. 2, GMD, Sankt Augustin.

[22] Gentner, D. (1983): Structure mapping - a theoretical framework for analogy. Cognitive Science, Vol.7. s.155-170.

[23] Hall, R. P. (1989): Computational approaches to analogical reasoning; A comparative analysis. Artificial Intelligence, Vol. 39, no. 1, 1989. pp 39-120.

[24] Hammond, K.J (1989): Case-based planning. Academic Press.

[25] Harmon, P. (1992): Case-based reasoning III, *Intelligent Software Strategies*, VIII (1).

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

[27] Hinrichs, T.R. (1992): Problem solving in open worlds. Lawrence Erlbaum Associates.

[28] IEEE Expert (1992): 7(5), Special issue on case-based reasoning. October 1992

[29] Keane, M. (1988): Where's the Beef? The Absence of Pragmatic Factors in Pragmatic Theories of Analogy In: *Proc. ECAI-88,* pp. 327-332

[30] Kedar-Cabelli, S. (1988): Analogy - from a unified perspective. In: D.H. Helman (ed.), Analogical reasoning. Kluwer Academic, 1988. pp 65-103.

[31] Kibler, D. and Aha, D. !987): Learning representative exemplars of concepts; An initial study. Proceedings of the fourth international workshop on Machine Learning, UC-Irvine, June 1987. pp 24-29.

[32] Kitano, H. (1993): Challenges for massive parallelism. IJCAI-93, Proceedings of the Thirteenth International Conference on Artificial Intelligence, Chambery, France, 1993. Morgan Kaufman 1993. pp. 813-834.

[33] Kolodner, J. (1983a): Maintaining organization in a dynamic long-term memory. Cognitive Science, Vol.7, s.243-280.

[34] KolodnerJ. (1983b): Reconstructive memory, a computer model. *Cognitive Science*, Vol.7, s.281-328.

[35] Kolodner, J. (1988): Retrieving events from case memory: A parallel implementation. In: *Proceedings from the Case-based Reasoning Workshop, DARPA*, Clearwater Beach, 1988, pp. 233-249.

[36] Kolodner, J. (1992): An introduction to case-based reasoning. *Artificial Intelligence Review* 6(1), pp. 3-34.

[37] Kolodner, J. (1993): Case-based reasoning. Morgan Kaufmann, 1993.

[38] Koton, P. (1989): Using experience in learning and problem solving. Massachusetts Institute of Technology, Laboratory of Computer Science (Ph.D. diss, October 1988). MIT/LCS/TR-441. 1989.

[39] López, B. and Plaza, E. (1990): Case-based learning of strategic knowledge. Centre d'Estudis Avançats de Blanes, CSIC, Report de Recerca GRIAL 90/14. Blanes, Spain, October 1990.

[40] Lopez, B. and Plaza, E. (1993):Case-based planning for medical diagnosis, In: *Methodologies for intelligent systems: 7th International Symposium, ISMIS '93*, Trondheim, June 1993 (Springer 1993) 96-105.

[41] Manago, M., Althoff, K-D. and Traphöner, R. (1993): Induction and reasoning from cases. In: *ECML - European Conference on Machine Learning, Workshop on Intelligent Learning Architectures.* Vienna, April 1993.

[42] Michalski, R and Tecuci, G (1992): Proceedings Multistrategy Learning Workshop, George Mason University.

[43] Nordbø, I., Skalle, P., Sveen, J., Aakvik, G., and Aamodt, A. (1992): *Reuse of experience in drilling - Phase 1 Report*. SINTEF DELAB and NTH, Div. of Petroleum Engineering. STF 40 RA92050 and IPT 12/92/PS/JS. Trondheim.

[44] Oehlmann, R. (1992): Learning causal models by self-questioning and experimentation. *AAAI-92 Workshop on Communicating Sientific and Technical Knowledge.* American Association of Artificial Intelligence.

[45] O'Hara, S. and Indurkhya, B. (1992): Incorporating (re)-interpretation in case-based reasoning. In: *First European Workshop on Case-based Reasoning, Posters and Presentations*, 1-5 November 1993. Vol. I. University of Kaiserslautern, pp. 154-159

[46] Plaza, E. and López de Mántaras, R (1990): A case-based apprentice that learns from fuzzy examples. Proceedings, ISMIS, Knoxville, Tennessee, 1990. pp 420-427.

[47] Plaza, E. and Arcos J. L. (1993):, Reflection and Analogy in Memory-based Learning, Proceedings Multistrategy Learning Workshop, George Mason University. p. 42-49.

[48] Porter, B. and Bareiss, R. (1986): PROTOS: An experiment in knowledge acquisition for heuristic classification tasks. In: *Proceedings of the First International Meeting on Advances in Learning (IMAL)*, Les Arcs, France, pp. 159-174.

[49] Porter, B., Bareiss, R. and Holte, R. (1990): Concept learning and heuristic classification in weak theory domains. Artificial Intelligence, vol. 45, no. 1-2, September 1990. pp 229-263.

[50] Richter, A.M. and Weiss, S. (1991): Similarity, uncertainty and case-based reasoning in PATDEX. In R.S. Boyer (ed.): Automated reasoning, essays in honour of Woody Bledsoe. Kluwer, pp. 249-265.

[51] Riesbeck, C. and Schank, R. (1989): Inside case-based reasoning. Lawrence Erlbaum.

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

[53] Ross, B.H (1989): Some psychological results on case-based reasoning. Case-Based Reasoning Workshop , DARPA 1989. Pensacola Beach. Morgan Kaufmann. pp. 144-147).

[54] Schank, R. (1982): Dynamic memory; a theory of reminding and learning in computers and people. Cambridge University Press.

[55] Schank, R. and Leake, D. (1989): Creativity and learning in a case-based explainer. Artificial Intelligence, Vol. 40, no 1-3. pp 353-385.

[56] Schult, T. (1992): Werkzeuge für fallbaseierte systeme. *Künstliche Intelligenz* 3(92).

[57] Sharma, S., Sleeman, D (1988): REFINER; a case-based differential diagnosis aide for knowledge acquisition and knowledge refinement. In: *EWSL 88; Proceedings of the Third European Working Session on Learning*, Pitman. pp 201-210.

[58] Simoudis, E. (1992): Using case-based reasoning for customer techical support. *IEEE Expert* 7(5), pp. 7-13.

[59] Simpson, R.L. (1985): A computer model of case-based reasoning in problem solving: An investigation in the domain of dispute mediation. Technical Report GIT-ICS-85/18, Georgia Institute of Technology.

[60] Skalak, C.B, and Rissland, E. (1992): Arguments and cases: An inevitable twining. *Artificial Intelligence and Law, An International Journal*, 1(1), pp.3-48.

[61] Slade, S. (1991): Case-based reasoning: A research paradigm. *AI Magazine* Spring 1991, pp. 42-55.

[62] Smith, E. and Medin, D. (1981): Categories and concepts. Harvard University Press.

[63] Stanfill, C and Waltz, D. (1988): The memory based reasoning paradigm. In: *Case based reasoning. Proceedings from a workshop*, Clearwater Beach, Florida, May 1988. Morgan Kaufmann Publ. pp.414-424.

[64] Steels, L. (1990): Components of expertise, *AI Magazine*, 11 (2) (Summer 1990) 29-49.

[65] Steels, L. (1993): The componential framework and its role in reusability, In J-M. David, J-P. Krivine, R. Simmons (eds.), *Second generation expert systems* (Spinger, 1993) 273-298.

[66] Strube, G. and Janetzko, D. (1990): Episodishes Wissen und Fallbasierte Schliessen: Aufgabe fur die Wissendsdiagnostik und die Wissenspsychologie. *Schweizerische Zeitschrift fur Psychologie*, 49, 211-221.

[67] Strube, G. (1991): The role of cognititve science in knowledge engineering, In: F. Schmalhofer, G. Strube (eds.), *Contemporary knowledge engineering and cognition: First joint workshop, proceedings*, (Springer 1991) 161-174.

[68] Sycara, K. (1988): Using case-based reasoning for plan adaptation and repair. *Proceedings Case-Based Reasoning Workshop, DARPA.* Clearwater Beach, Florida. Morgan Kaufmann, pp. 425-434.

[69] Tulving, E. (1977): Episodic and semantic memory. In E. Tulving and W. Donaldson: Organization of memory, Academic Press, 1972. pp. 381-403.

[70] Veloso, M.M. and Carbonell, J. (1993): Derivational analogy in PRODIGY. In *Machine Learning* 10(3), pp. 249-278.

[71] Venkatamaran, S., Krishnan, R. and Rao, K.K. (1993): A rule-rule-case based system for image analysis. In: *First European Workshop on Case-based Reasoning, Posters and Presentations*, 1-5 November 1993. Vol. II. University of Kaiserslautern, pp. 410-415.

[72] Wittgenstein, L. (1953): Philosophical investigations. Blackwell, pp. 31-34.

[73]Van de Velde, W. (1993): Issues in knowledge level modelling, In J-M. David, J-P. Krivine, R. Simmons (eds.), *Second generation expert systems* , Spinger, 211-231.

[74]Schmalhofer, F and Thoben, J. (1992): The model-based construction of a case-oriented expert system, *AI Communications* 5(1), 3-18.

[75]Rouse, W.B and Hurt, R.M (1982): Human problem solving in fault diagnosis tasks, Georgia Institute of Technology, Center for Man-Machine Systems Research, Research Report no 82-3.