Automated Support for Developing Retrieve-and-Propose Systems*

Evangelos Simoudis **Peter Miller**
Lockheed Artificial Intelligence Center
O/96-20, B/254G, 3251 Hanover Street
Palo Alto, California 94304
simoudis, pmiller@aic.lockheed.com

Abstract

We have studied the retrieve-and-propose problem solving method, a specialization of the general case-based reasoning method, in terms of its tasks in the context of fault-recovery domains. We have analyzed the case retrieval and learning tasks that are primarily used in the retrieve-and-propose method, decomposed them into appropriate subtasks, associated decisions for selecting each subtask, and implemented methods for accomplishing each task. We have implemented a system, called REPRO for REtrieve and PROpose, that incorporates the results of this analysis and facilitates the selection of the appropriate algorithms for implementing a case-based expert system. We have used REPRO to develop the CABER case-based fault-recovery system.

Introduction

The development of a case-based reasoning (CBR) system is based on four complex operations: (1) selection of a case-retrieval algorithm, (2) selection of a case-adaptation algorithm, (3) selection of a learning algorithm, and (4) creation of an initial case base. The selection of the appropriate algorithms is difficult, even for knowledge engineers, because it is driven by the interrelation between the retrieval, adaptation and learning functions, the characteristics of the application domain (e.g. available knowledge, size of feature space, etc.), the tasks to be performed by the CBR system (classification, design, monitoring, etc.), and the contents of the case base. We have studied these operations while developing three case-based expert systems for fault-recovery tasks. These systems, as well as other systems that have been presented in the literature (e.g., [Goodman 1989]), were developed as monolithic structures that included predefined retrieval, adaptation, and learning functions. As a result, such CBR systems often could not scale up since the functions they included might not have been appropriate for every aspect of the domain and task they were expected to address. We have analyzed the learning and retrieval operations in fault-recovery domains. Our analysis is based on the *generic task* methodology [Chandrasekaran 1988]. Based on our analysis, we have implemented a system, called REPRO (for REtrieve and PROpose), that makes explicit the selection of case-retrieval and learning algorithms, as well as the creation of an initial case base. We have used REPRO to develop the CABER case-based fault-recovery system. REPRO is layered on top of an existing CBR shell, Remind [1] capitalizing on its capabilities.

The fault-recovery process begins with a set of symptoms describing a fault, as well as other features establishing the fault's context. The goals of fault recovery are to: (1) identify the fault's type, and (2) execute a plan whose actions can eliminate the fault. Therefore, fault recovery consists of two *generic tasks*: classification and plan instantiation. Fault classification was mapped to the retrieve-and-propose method [Kolodner 1991], a specialization of the general case-based reasoning method. In particular, given a set of cases containing preclassified faults, the retrieve-and-propose method tries to retrieve the best case and use its contents to propose a classification for the new fault. Plan instantiation was also mapped to the retrieve-and-propose method. In this instance, the method tries to retrieve the best plan for an established fault which the user must instantiate and follow verbatim. The central operations (generic tasks) that are performed by the retrieve-and-propose method are retrieval and learning.

In the next section, we analyze the retrieval and learning tasks. We then describe REPRO and how we used REPRO to build the CABER case-based ex-

*This work was supported by the Defense Advanced Research Projects Agency, administered by the U.S. Army Research Laboratory under contract #DAAA15-90-C-0004

[1] Remind is a registered trademark of Cognitive Systems, Inc.

pert system. Finally, we describe the future direction of our work and the conclusions drawn from our work.

Analysis of Retrieval and Learning

In its simplest form, the learning task performed by a CBR system consists of storing new cases in the system's case base. In the particular instance of fault recovery a case needs to be incorporated in the case base for one of three reasons: (1) a new fault type has been identified, (2) a new set of features that are indicative of a known fault type has been identified, and (3) a new repair plan for a known fault type and a known set of features has been identified.

Similarly, the simplest method for retrieving the correct case from the CBR system's case base consists of a comparison between the description (features) of a new problem and the corresponding characteristics of each case in the case base. The goal of this comparison, called nearest neighbor comparison [Duda & Hart 1973] is to identify the case(s) whose features best match those of the problem. The nearest neighbor comparison is computationally expensive for two reasons. First, all cases in the case base need to be examined before the best case is identified. Second, all provided features describing a new problem must be compared to the corresponding features of the stored cases. The cost of this comparison grows with the size of the case base and the size of each case.

The computational and end-use cost associated with the nearest neighbor method can be reduced in two ways. First, through reasoning with appropriate domain knowledge during retrieval [Koton 1988a; Simoudis 1990]. For example, the system CASCADE [Simoudis 1991] uses knowledge about the cost of establishing each feature of a new problem when determining which case to retrieve. Second, by organizing the case base using indices and using a retrieval algorithm that utilizes these indices.

The task of case retrieval is decomposed into the following subtasks: index utilization, similarity assessment (or case comparison), and case ordering. The retrieval algorithm uses the indices to access a set of cases. While the indices might only include a subset of the features that are included in a case, the similarity assessment uses a larger subset (possibly all) of a case's features to compare to the current problems feature space. If the indices contain all of the features in the cases (or index utilization and similarity assessment use the same subset of features), then index utilization and similarity assessment collapse into a single operation. Finally, the cases are ordered; this ordering can merely be collapsed into the similarity assessment (as in the nearest neighbor comparison, which actually collapses all three subtasks into the single operation of similarity assessment) or can be based on other knowledge, for example, the cost of acquiring the values of the feature or statistics about previous experience with the case (see the description of REPRO in the next section).

The creation of indices is part of the overall learning task. After an index is created, the case to which it corresponds is incorporated into the case base. Therefore, the overall learning task is decomposed into the index-creation subtask for the identification of the appropriate feature subset to be the case's index, and the case-incorporation subtask for entering the case into the database system. Associated with index-creation are two other subtasks: index organization and index maintenance. Index organization refers to the way that indices are organized in the case base once they are created (e.g., hierarchically or in flat structures). Index maintenance refers to the modifications that can be performed on indices (e.g., specialization or generalization) in order to increase their effectiveness in organizing a particular case base.

The decision as to whether or not to generate indices is based on both domain-specific and domain-independent characteristics. For example, if the case base is small (e.g., less than 20 cases) and grows very slowly over time (e.g., less than one case per month), the number of features making up each case is small (e.g., less than six features), and acquisition of feature values is not expensive, then indexing is unnecessary (e.g., the CLAVIER case-based reasoning system [Hennessy & Hinkle 1992] does not use indices for these reasons). However, there are cases where the case base is very large and you would still choose not to index the cases (e.g., PACE [Creecy et al. 1992] and MBRtalk [Stanfill & Waltz 1986] where this is due to the nature of the task that the system is to perform and the use of massively parallel hardware). It is the interrelation between the application characteristics as well as any other choices of algorithms, rather than any direct mapping from application characteristic to choice of algorithms, that drives the selection of algorithms.

If indices are to be generated, the following questions must be answered: (1) what type of indices to generate (i.e., necessary conditions vs. necessary and sufficient conditions), (2) how to generate these indices (e.g., inductively [Lebowitz 1987; Fisher 1984], or using knowledge-based methods such as Explanation-Based Indexing [Barletta & Mark 1988]), (3) how to organize the created indices, and (4) how to maintain the created indices. For example, automatic index creation is appropriate when the application domain's feature space is large, knowledge-based index creation is appropriate when knowledge about the feature space is available, and Explanation-Based Indexing (EBI) is appropriate only when a causal model is available.

Selection of the retrieval algorithm is similarly based on domain-specific and domain-independent characteristics as well as any decisions made about the other subtasks, like index creation. For example, while CASEY [Koton 1988b] uses all the features of a case to index the case, it uses domain knowledge (previous experience with the feature and *evidence principles* as-

sociated with its causal model) to retrieve cases.

In the next section, we describe REPRO and how it helps the knowledge engineer choose the appropriate algorithms to for the retrieval and learning tasks.

REPRO

We have represented the knowledge we obtained from our analyses of the retrieval and learning tasks, and a set of methods to perform these tasks in a tool called REPRO that is used by a knowledge engineer as a shell for developing case-based retrieve-and-propose expert systems. The knowledge engineer is expected to cooperate with domain experts during the expert system development process. REPRO can be layered on top of existing CBR shells to take advantage of their retrieval, indexing, and data storage capabilities, while providing a greater set of capabilities to the knowledge engineer.

REPRO provides the knowledge engineer with:

1. The capability to define the feature space of a domain and create a seed case base.
2. A structured way of selecting an initial set of index-creation, index-organization, index-maintenance, and case-retrieval methods.
3. A structured way of satisfying the requirements of each selected method for explicit domain-specific knowledge. This knowledge expresses the interrelations among features and methods for their acquisition.
4. The ability to experiment with the chosen methods.
5. The ability to create a runtime system that can be used as a stand-alone application by novice users.

REPRO supports cases consisting of a feature set describing a particular fault, a repair plan for addressing this fault, the set of features expected after the repair plan is applied, and a set of variables necessary to keep statistics on the use of the case (currently, the number of times a case is retrieved and the number of times that the case successfully repaired the fault). REPRO has a graphical user-interface that provides a set of structured editors and viewers that allow the user to easily build up the feature space and case base.

In REPRO a feature consists of: an attribute name, a value, and an action for acquiring that value. Currently, REPRO supports nominal and numeric attribute values. A feature space is defined in REPRO by naming each attribute, specifying the valid values (enumerating the nominal values or in the case of numeric values, specifying whether the attribute takes integer or real values and optionally specifying the range of allowable values) for each attribute, and defining the action that will be used to obtain the values. The actions are associated with the attributes and consist of a set of instructions for the user to follow in obtaining the value and a cost associated with obtaining that value. The cost is entered manually by the knowledge engineer and is based on domain-dependent factors (e.g., can the value be obtained automatically or is human intervention necessary; if human intervention is necessary, what is the level of expertise necessary and what is the availability of the individual(s) with the appropriate level of expertise).

REPRO organizes a feature space into three types of features: (1) complaint features, i.e., a fault's surface features, (2) diagnostic features, i.e., derived features that are causally linked to a fault, and (3) repair features, i.e., the features that describe the outcome of applying a repair plan.

In addition to the different feature types and the actions associated with them, the knowledge engineer can define preconditions of each action and interrelations among the features. For example, in the CABER system described below, a precondition of the action for checking the condition of a milling machine's axis drive tachometer establishes that the machine's axis drive uses a direct current motor.

REPRO uses the cost associated with the actions and the knowledge of the interrelations between the attributes (and their associated actions) to order the cases that are retrieved and place them on an agenda. While the default case sorter uses cost to order the cases, the changes to allow the use of the case statistics or some combination of cost and usage statistics are trivial. The next implementation of REPRO will allow the user to choose the algorithm for sorting the cases on the agenda.

The ability to experiment with different methods of index-creation, index-organization, and case-retrieval (as well as the method of ordering the accessed cases) allows the knowledge engineer to make changes in the CBR system being developed; these changes can reflect an evolving understanding of the application domain. In addition, the ability to simultaneously use more than one method for index creation and for case retrieval, while not yet fully implemented, will allow the system to deal with incomplete knowledge. For example, assume that a causal model has been captured and represented for only a part of a domain. Furthermore, assume that because of representing the causal model the knowledge engineer selected the EBI index-creation algorithm. Finally, assume that, since the model is incomplete, the knowledge engineer also selected an inductive index-creation algorithm. Due to the incompleteness of the captured knowledge a new case will be indexed using both selected index-creation methods. During retrieval, if the features of the new problem cannot match the features in the indices created through EBI, then the case retriever that corresponds to the inductive index-creation method is used to access a case that is appropriate to the problem being solved.

The ability to create a separate runtime system for each case-based expert system developed with REPRO

has two major advantages. First, it saves resources by shedding those parts of the REPRO system that are just used in development (e.g., the unused algorithms and the user interface for the knowledge engineer). Second, by eliminating the knowledge engineer's tools, it prevents a novice user from making changes to the functionality of the expert system. Each runtime system allows its user to work on more than one problem at a time (as does REPRO itself, the knowledge-engineering environment allows simulation of the runtime environment); the user can switch back and forth between problems (this is very useful for a fault-recovery system such as the CABER system described below, where a centrally located fault-recovery system might be helping more than one engineer who will be making repairs at a remote site).

REPRO could be easily extended to keep track of the runtime systems created and their specifications (e.g., size of seed case-base, indexing method, index organization, retrieval method, etc.), so that experience with these systems could help future designers choose the appropriate algorithms.

REPRO is currently implemented on an Apple Macintosh II in Macintosh Common LISP. We have used REPRO to implement a case-based expert system, called CABER, which helps the operator of a milling machine recover from machine faults. In the following section we examine the building of CABER.

CABER

Milling machines are complex pieces of equipment that consist of mechanical, electric, and hydraulic components. Furthermore, recent milling machine models are equipped with advanced computer hardware that is controlled by sophisticated software. Recovering from faults that occur during the normal operation of a milling machine is a knowledge-based task that is difficult to perform even for experienced operators because of the equipment's complexity. However, rapid fault recovery is essential in manufacturing operations. For example, the growing use of just-in-time material-handling in manufacturing requires that downtime for any piece of equipment be minimized. Today, fault recovery operations are performed by field-service engineers who are dispatched to the customer's site by the milling machine manufacturer. After a problem is solved the field-service engineer completes a report that includes a description of the problem (provided by the customer), additional problem-related information obtained by the engineer, and the repair plan executed by the engineer to recover from the fault.

Discussions with milling machine operators established the need for a decision-support system to assist them in recovering from faults and decrease their reliance on the field service engineers. After being exposed to the fault recovery process and examining samples of problem reports provided by Cincinnati Milacron, a milling machine manufacturing company, we decided to use REPRO to develop the CABER case-based decision-support system.

```
(case1 tachometer-dirty
  (complaint-features
    (machine-type t-30)
    (control-type cnc)
    (year-of-manafacture 1989)
    (tool-changer-arm-acceleration sluggish))
  (diagnostic-features
    (tach-condition dirty))
  (repair-plan
    (replace-tachometer))
  (outcome
    (tach-condition clean))
  (statistics
    (times-retrieved 3)
    (times-successful 2)))
```

Figure 1: A sample case used by the CABER system

Creating the seed case base

Cincinnati Milacron initially provided us with a set of 100 problem reports. We analyzed these reports with an expert field-service engineer and organized the information they included into 20 prototypical cases which we represented using REPRO's case structure. An example case from the CABER system is shown in Figure 1.

In addition to the cases, the expert provided us with knowledge about the parts of the milling machine that were referenced in the cases (e.g., tachometer, tool changer, etc.). In this way we developed a good understanding of the initial feature space. Since the initial set of cases was small, as was the number of the features included in these cases, we elected not to index the cases in the case base. Therefore, case incorporation was performed by directly storing each case in the case base. We selected the nearest neighbor retrieval algorithm that is included in the Remind shell for searching the case base.

We proceeded to use REPRO's feature acquisition facilities to represent the set of features extracted from the provided problem reports. The represented features were used to encode each of the 20 cases, that were subsequently stored in Remind's database.

Refining CABER

After an initial experimentation phase during which CABER's performance was evaluated by the expert, Cincinnati Milacron provided us with an additional set of 250 problem reports to enhance CABER's case base. With the expert's assistance we created an additional 40 prototypical cases. Encoding these cases required that we expand the represented feature space, which grew from 30 to 100 features. Due to the increase in the size of the augmented case base and the corresponding feature space, we decided to index cases.

Since we had acquired, from the domain expert and milling machine operation manuals, domain-specific knowledge about the features and their interrelations, we selected one of REPRO's knowledge-based index-creation methods. In particular, we selected to use the EBI method since the acquired knowledge was causal. EBI requires knowledge in the form of a structure and behavior model of the system that is referred to in the cases. Once such knowledge about milling machines was encoded in REPRO, the entire case base was indexed.

Due to the relatively small size of the case base, we also selected a case organization algorithm that arranges the indices and the cases into a flat structure. Finally, the selected retrieval algorithm could search such a structure. The selected methods, as well as the updated and organized case base were included in the second prototype of CABER. This system is currently under evaluation.

Future Work

We have identified several areas to further our work in supporting the development of retrieve-and-propose systems. First, we will complete the implementation of the mechanisms to allow the simultaneous use of multiple index-creation and case retrieval algorithms. We will then test our hypothesis that multiple index-creation and retrieval algorithms will allow a CBR system to deal with incomplete knowledge.

Second, we will extend REPRO to keep track of the domain characteristics and algorithms that were selected for each generated CBR system. Our attempts to organize the decisions into a simple reasoning structure such as a hierarchy, has convinced us that the development process of CBR systems is a problem that is well-suited to case-based reasoning. REPRO will help the knowledge engineer add into a case base the domain characteristics and algorithm selections associated with each created runtime system, which could help future designers to fine tune their systems.

Finally, we will integrate REPRO with tools that support the representation of causal models so that we can expand REPRO's repertoire of knowledge-based index-creation and case-retrieval algorithms.

Conclusions

We have studied the retrieve-and-propose problem-solving method, a specialization of the general case-based reasoning method, in terms of its tasks and in the context of fault recovery domains. We analyzed the case retrieval and learning tasks and decomposed them into appropriate subtasks. We associated decisions for selecting each subtask, established criteria for reaching each decision, and implemented methods for accomplishing each task. Finally, we have incorporated all the established knowledge and developed methods into REPRO. We have used REPRO to develop two prototypes of the CABER system.

Our work has allowed us to reach the following conclusions:

1. The task decomposition and characterization of the operations performed by case-based reasoning systems facilitates their development and maintenance phases.
2. Task interrelations make method selection difficult and partially explaining why most CBR systems have been developed from scratch.
3. A shell like REPRO, which makes the choice of methods explicit, can help to alleviate the problems associated with task interrelations.

Acknowledgements

We thank Albert Mendall for his considerable contribution to the design and implementation of REPRO and David Hinkle for his work on CABER.

References

Barletta, R., and Mark, W.S. 1988. Explanation-Based Indexing of Cases. In Proceedings of AAAI-88, 541-546, Morgan Kaufmann Publishers, Inc., San Mateo, CA.

Chandrasekaran, B. 1988. Generic Tasks as Building Blocks for Knowledge-Based Systems: The Diagnosis and Routine Design Examples. *Knowledge Engineering Review*, 3(3):183-219.

Creecy, R., et. al. 1992. Trading MIPS and Memory for Knowledge Engineering. *Communications of the ACM* 35(8):48-64.

Duda, R., and Hart, P. 1973. *Pattern Classification and Scene Analysis*, John Wiley and Sons.

Fisher, D. 1984. A Hierarchical Conceptual Clustering Algorithm, Technical report, Department of Computer and Information Science, University of California Irvine.

Goodman, M. 1989. CBR in Battle Planning. In Proceedings: Case-Based Reasoning Workshop, 264-269, Morgan Kaufman Publishers, Inc., San Mateo, May.

Hennessy, D., and Hinkle, D. 1992. Applying Case-Based Reasoning to Autoclave Loading. *IEEE Expert*, 7(5):21-26.

Kolodner, J. 1983. Reconstructive Memory: A Computer Model. *Cognitive Science Journal* 7:281-328.

Kolodner, J. 1991. Improving Human Decision Making through Case-Based Decision Aiding. *AI Magazine* 12:52-68.

Koton, P. 1988a. Using Experience in Learning and Problem Solving, Ph.D. diss., Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Koton, P. 1988b. Reasoning about Evidence in Causal Explanations. In Proceedings of a Workshop on Case-Based Reasoning, Janet Kolodner, ed., 260-270. Mor-

gan Kaufmann Publishers, Inc., San Mateo.

Lebowitz, M. 1987. Experiments with incremental concept formation: Unimem. *Machine Learning*, 2:103-138.

Simoudis, E., and Miller, J. 1990. Validated Retrieval in Case-Based Reasoning. In Proceedings AAAI-90, 310-315, MIT Press, Cambridge, August.

Simoudis, E. 1991. Retrieving Justifiably Relevant Cases from a Case Base Using Validation Models. PhD diss., Dept. of Computer Science, Brandeis Univ.